MINIMUM-RESIDUAL METHODS
FOR SPARSE LEAST-SQUARES
USING GOLUB-KAHAN BIDIAGONALIZATION

A DISSERTATION
SUBMITTED TO THE INSTITUTE FOR
COMPUTATIONAL AND MATHEMATICAL
ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

David Chin-lung Fong
December 2011

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Michael Saunders, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Margot Gerritsen

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Walter Murray

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

ABSTRACT

For 30 years, LSQR and its theoretical equivalent CGLS have been the standard iterative solvers for large rectangular systems $Ax = b$ and least-squares problems $\min \|Ax - b\|$. They are analytically equivalent to symmetric CG on the normal equation $A^TAx = A^Tb$, and they reduce $\|r_k\|$ monotonically, where $r_k = b - Ax_k$ is the k-th residual vector. The techniques pioneered in the development of LSQR allow better algorithms to be developed for a wider range of problems.

We derive LSMR, an algorithm that is similar to LSQR but exhibits better convergence properties. LSMR is equivalent to applying MINRES to the normal equation, so that the error $\|x^* - x_k\|$, the residual $\|r_k\|$, and the residual of the normal equation $\|A^Tr_k\|$ all decrease monotonically. In practice we observe that the Stewart backward error $\|A^Tr_k\|/\|r_k\|$ is usually monotonic and very close to optimal. LSMR has essentially the same computational cost per iteration as LSQR, but the Stewart backward error is always smaller. Thus if iterations need to be terminated early, it is safer to use LSMR.

LSQR and LSMR are based on Golub-Kahan bidiagonalization. Following the analysis of LSMR, we leverage the techniques used there to construct algorithm AMRES for negatively-damped least-squares systems $(A^TA - \delta^2 I)x = A^Tb$, again using Golub-Kahan bidiagonalization. Such problems arise in total least-squares, Rayleigh quotient iteration (RQI), and Curtis-Reid scaling for rectangular sparse matrices. Our solver AMRES provides a stable method for these problems. AMRES allows caching and reuse of the Golub-Kahan vectors across RQIs, and can be used to compute any of the singular vectors of A, given a reasonable estimate of the singular vector or an accurate estimate of the singular value.

Acknowledgements

First and foremost, I am extremely fortunate to have met Michael Saunders during my second research rotation in ICME. He designed the optimal research project for me from the very beginning, and patiently guided me through every necessary step. He devotes an enormous amount of time and consideration to his students. Michael is the best advisor that any graduate student could ever have.

Our work would not be possible without standing on the shoulder of a giant like Chris Paige, who designed MINRES and LSQR together with Michael 30 years ago. Chris gave lots of helpful comments on reorthogonalization and various aspects of this work. I am thankful for his support.

David Titley-Peloquin is one of the world experts in analyzing iterative methods like LSQR and LSMR. He provided us with great insights on various convergence properties of both algorithms. My understanding of LSMR would be left incomplete without his ideas.

Jon Claerbout has been the most enthusiatic supporter of the LSQR algorithm, and is constantly looking for applications of LSMR in geophysics. His idea of “computational success” aligned with our goal in designing LSMR and gave us strong motivation to develop this algorithm.

James Nagy’s enthusiasm in experimenting with LSMR in his latest research also gave us much confidence in further developing and polishing our work. I feel truly excited to see his work on applying LSMR to image processing.

AMRES would not be possible without the work that Per Christian Hansen and Michael started. The algorithm and some of the applications are built on top of their insights.

I would like to thank Margot Gerritsen and Walter Murray for serving on both my reading and oral defense committees. They provided many valuable suggestions that enhanced the completeness of the experimental results. I would also like to thank Eric Darve for chairing my oral defense and Peter Kitanidis for serving on my oral defense committee and giving me objective feedback on the algorithm from a user's perspective.

I am also grateful to the team of ICME staff, especially Indira Choudhury and Brian Tempero. Indira helped me get through all the paperwork, and Brian assembled and maintained all the ICME computers. His support allowed us to focus fully on number-crunching.

I have met lots of wonderful friends here at Stanford. It is hard to acknowledge all of them here. Some of them are Santiago Akle, Michael Chan, Felix Yu, and Ka Wai Tsang.

Over the past three years, I have been deeply grateful for the love that Amy Lu has brought to me. Her smile and encouragement enabled me to go through all the tough times in graduate school.

I thank my parents for creating the best possible learning environment for me over the years. With their support, I feel extremely priviledged to be able to fully devote myself to this intellectual venture.

I would like to acknowledge my funding, the Stanford Graduate Fellowship (SGF). For the first half of my SGF, I was supported by the Hong Kong Alumni Scholarship with generous contributions from Dr Franklin and Mr Lee. For the second half of my SGF, I was supported by the Office of Technology Licensing Fellowship with generous contributions from Professor Charles H. Kruger and Ms Katherine Ku.

Contents

List of Tables

List of Figures

List of Algorithms

List of Matlab Code

1

INTRODUCTION

The quest for the solution of linear equations is a long journey. The earliest known work is in 263 AD [64]. The book Jiuzhang Suanshu (Nine Chapters of the Mathematical Art) was published in ancient China with a chapter dedicated to the solution of linear equations.[1] The modern study of linear equations was picked up again by Newton, who wrote unpublished notes in 1670 on solving system of equations by the systematic elimination of variables [33].

[1] Procedures for solving systems of three linear equations in three variables were discussed.

Cramer's Rule was published in 1750 [16] after Leibniz laid the work for determinants in 1693 [7]. In 1809, Gauss invented the method of least squares by solving the normal equation for an over-determined system for his study of celestial orbits. Subsequently, in 1826, he extended his method to find the minimum-norm solution for underdetermined systems, which proved to be very popular among cartographers [33].

For linear systems with dense matrices, Cholesky factorization, LU factorization and QR factorization are the popular methods for finding solutions. These methods require access to the elements of matrix.

We are interested in the solution of linear systems when the matrix is large and sparse. In such circumstances, direct methods like the ones mentioned above are not practical because of memory constraints. We also allow the matrix to be a linear operator defined by a procedure for computing matrix-vector products. We focus our study on the class of iterative methods, which usually require only a small amount of auxiliary storage beyond the storage for the problem itself.

1.1 PROBLEM STATEMENT

We consider the problem of solving a system of linear equations. In matrix notation we write

$$Ax = b, \tag{1.1}$$

where A is an $m \times n$ real matrix that is typically large and sparse, or is available only as a linear operator, b is a real vector of length m, and x is a real vector of length n.

We call an x that satisfies (1.1) a solution of the problem. If such an x does not exist, we have an inconsistent system. If the system is inconsistent, we look for an optimal x for the following least-squares problem instead:

$$\min_x \|Ax - b\|_2 . \tag{1.2}$$

We denote the exact solution to the above problems by x^*, and ℓ denotes the number of iterations that any iterative method takes to converge to this solution. That is, we have a sequence of approximate solutions $x_0, x_1, x_2, \ldots, x_\ell$, with $x_0 = 0$ and $x_\ell = x^*$. In Section 1.2 and 1.3 we review a number of methods for solving (1.1) when A is square ($m = n$). In Section 1.4 we review methods that handle the general case when A is rectangular, which is also the main focus of this thesis.

1.2 Basic iterative methods

Jacobi iteration, Gauss-Seidel iteration [31, p510] and successive over-relaxation (SOR) [88] are three early iterative methods for linear equations. These methods have the common advantage of minimal memory requirement compared with the Krylov subspaces methods that we focus on hereafter. However, unlike Krylov subspace methods, these methods will not converge to the exact solution in a finite number of iterations even with exact arithmetic, and they are applicable to only narrow classes of matrices (e.g., diagonally dominant matrices). They also require explicit access to the nonzeros of A.

1.3 Krylov subspace methods for square $Ax = b$

Sections 1.3 and 1.4 describe a number of methods that can regard A as an operator; i.e. only matrix-vector multiplication with A (and sometimes A^T) is needed, but not direct access to the elements of A.[2] Section 1.3 focuses on algorithms for the case when A is square. Section 1.4 focuses on algorithms that handle both rectangular and square A. Krylov subspaces of increasing dimensions are generated by the matrix-vector products, and an optimal solution within each subspace is found at each iteration of the methods (where the measure of optimality differs with each method).

[2] These methods are also know as matrix-free iterative methods.

Algorithm 1.1 Lanczos process Tridiag(A, b)

1: $\beta_1 v_1 = b$ (i.e. $\beta_1 = \|b\|_2$, $v_1 = b/\beta_1$)
2: **for** $k = 1, 2, \dots$ **do**
3: $\quad w = Av_k$
4: $\quad \alpha_k = v_k^T w_k$
5: $\quad \beta_{k+1} v_{k+1} = w - \alpha_k v_k - \beta_k v_{k-1}$
6: **end for**

1.3.1 THE LANCZOS PROCESS

In this section, we focus on symmetric linear systems. The Lanczos process [40] takes a symmetric matrix A and a vector b, and generates a sequence of Lanczos vectors v_k and scalars α_k, β_k for $k = 1, 2, \dots$ as shown in Algorithm 1.1. The process can be summarized in matrix form as

$$AV_k = V_k T_k + \beta_{k+1} v_{k+1} e_k^T = V_{k+1} H_k \tag{1.3}$$

with $V_k = \begin{pmatrix} v_1 & v_2 & \cdots & v_k \end{pmatrix}$ and

$$T_k = \begin{pmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_k \\ & & \beta_k & \alpha_k \end{pmatrix}, \qquad H_k = \begin{pmatrix} T_k \\ \beta_{k+1} e_k^T \end{pmatrix}.$$

An important property of the Lanczos vectors in V_k is that they lie in the Krylov subspace $\mathcal{K}_k(A, b) = \text{span}\{b, Ab, A^2 b, \dots, A^{k-1} b\}$. At iteration k, we look for an approximate solution $x_k = V_k y_k$ (which lies in the Krylov subspace). The associated residual vector is

$$r_k = b - Ax_k = \beta_1 v_1 - AV_k y_k = V_{k+1}(\beta_1 e_1 - H_k y_k).$$

By choosing y_k in various ways to make r_k small, we arrive at different iterative methods for solving the linear system. Since V_k is theoretically orthonormal, we can achieve this by solving various *subproblems* to make

$$H_k y_k \approx \beta_1 e_1. \tag{1.4}$$

Three particular choices of subproblem lead to three established methods (CG, MINRES and SYMMLQ) [52]. Each method has a different minimization property that suggests a particular factorization of H_k. Certain auxiliary quantities can be updated efficiently without the

Algorithm 1.2 Algorithm CG

```
r_0 = b, p_1 = r_0, ρ_0 = r_0^T r_0
for k = 1, 2, ... do
    q_k = A p_k
    α_k = ρ_{k−1} / p_k^T q_k
    x_k = x_{k−1} + α_k p_k
    r_k = r_{k−1} − α_k q_k
    ρ_k = r_k^T r_k
    β_k = ρ_k / ρ_{k−1}
    p_{k+1} = r_k + β_k p_k
end for
```

need for y_k itself (which in general is completely different from y_{k+1}).

With exact arithmetic, the Lanczos process terminates with $k = \ell$ for some $\ell \le n$. To ensure that the approximations $x_k = V_k y_k$ improve by some measure as k increases toward ℓ, the Krylov solvers minimize some convex function within the expanding Krylov subspaces [27].

CG

CG was introduced in 1952 by Hestenes and Stiefel [38] for solving $Ax = b$ when A is symmetric positive definite (spd). The quadratic form $\phi(x) \equiv \frac{1}{2}x^T Ax - b^T x$ is bounded below, and its unique minimizer solves $Ax = b$. CG iterations are characterized by minimizing the quadratic form within each Krylov subspace [27], [46, §2.4], [84, §§8.8–8.9]:

$$x_k = V_k y_k, \qquad \text{where} \qquad y_k = \arg\min_y \phi(V_k y). \tag{1.5}$$

With $b = Ax$ and $2\phi(x_k) = x_k^T A x_k - 2x^T A x_k$, this is equivalent to minimizing the function $\|x^* - x_k\|_A \equiv (x^* - x_k)^T A (x^* - x_k)$, known as the *energy norm of the error*, within each Krylov subspace. A version of CG adapted from van der Vorst [79, p42] is shown in Algorithm 1.2.

CG has an equivalent Lanczos formulation [52]. It works by deleting the last row of (1.4) and defining y_k by the subproblem $T_k y_k = \beta_1 e_1$. If A is positive definite, so is each T_k, and the natural approach is to employ the Cholesky factorization $T_k = L_k D_k L_k^T$. We define W_k and z_k from the lower triangular systems

$$L_k W_k^T = V_k^T, \qquad L_k D_k z_k = \beta_1 e_1.$$

It then follows that $z_k = L_k^T y_k$ and $x_k = V_k y_k = W_k L_k^T y_k = W_k z_k$, where the elements of W_k and z_k do not change when k increases. Simple recursions follow. In particular, $x_k = x_{k-1} + \zeta_k w_k$, where $z_k = \left(\begin{smallmatrix} z_{k-1} \\ \zeta_k \end{smallmatrix}\right)$ and $W_k = \begin{pmatrix} W_{k-1} & w_k \end{pmatrix}$. This formulation requires one more n-vector than the non-Lanczos formulation.

When A is not spd, the minimization in (1.5) is unbounded below, and the Cholesky factorization of T_k might fail or be numerically unstable. Thus, CG cannot be recommended in this case.

MINRES

MINRES [52] is characterized by the following minimization:

$$x_k = V_k y_k, \qquad \text{where} \qquad y_k = \arg\min_y \|b - AV_k y\|. \tag{1.6}$$

Thus, MINRES minimizes $\|r_k\|$ within the kth Krylov subspace. Since this minimization is well-defined regardless of the definiteness of A, MINRES is applicable to both positive definite and indefinite systems. From (1.4), the minimization is equivalent to

$$\min \|H_k y_k - \beta_1 e_1\|_2 .$$

Now it is natural to use the QR factorization

$$Q_k \begin{pmatrix} H_k & \beta_1 e_1 \end{pmatrix} = \begin{pmatrix} R_k & z_k \\ 0 & \bar{\zeta}_{k+1} \end{pmatrix},$$

from which we have $R_k y_k = z_k$. We define W_k from the lower triangular system $R_k^T W_k^T = V_k^T$ and then $x_k = V_k y_k = W_k R_k y_k = W_k z_k = x_{k-1} + \zeta_k w_k$ as before. (The Cholesky factor L_k is lower bidiagonal but R_k^T is lower tridiagonal, so MINRES needs slightly more work and storage than CG.)

Stiefel's Conjugate Residual method (CR) [72] for spd systems also minimizes $\|r_k\|$ in the same Krylov subspace. Thus, CR and MINRES must generate the same iterates on spd systems. We will use the two algorithms interchangeably in the spd case to prove a number of properties in Chapter 2. CR is shown in Algorithm 1.3.

Note that MINRES is reliable for any symmetric matrix A, whereas CR was designed for positive definite systems. For example it will fail

Algorithm 1.3 Algorithm CR

1: $x_0 = 0$, $r_0 = b$, $s_0 = Ar_0$, $\rho_0 = r_0^T s_0$, $p_0 = r_0$, $q_0 = s_0$
2: **for** $k = 1, 2, \ldots$ **do**
3: ($q_{k-1} = Ap_{k-1}$ holds, but not explicitly computed)
4: $\alpha_k = \rho_{k-1}/\|q_{k-1}\|^2$
5: $x_k = x_{k-1} + \alpha_k p_{k-1}$
6: $r_k = r_{k-1} - \alpha_k q_{k-1}$
7: $s_k = Ar_k$
8: $\rho_k = r_k^T s_k$
9: $\beta_k = \rho_k/\rho_{k-1}$
10: $p_k = r_k + \beta_k p_{k-1}$
11: $q_k = s_k + \beta_k q_{k-1}$
12: **end for**

on the following nonsingular but indefinite system:

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix} x = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

In this case, r_1 and s_1 are nonzero, but $\rho_1 = 0$ and CR fails after 1 iteration. Luenberger extended CR to indefinite systems [43; 44]. The extension relies on testing whether $\alpha_k = 0$ and switching to a different update rule. In practice it is difficult to judge whether α_k should be treated as zero. MINRES is free of such a decision (except when A is singular and $Ax = b$ is inconsistent, in which case MINRES-QLP [14; 13] is recommended).

Since the pros and cons of CG and MINRES are central to the design of the two new algorithms in this thesis (LSMR and AMRES), a more in-depth discussion of their properties is given in Chapter 2.

SYMMLQ

SYMMLQ [52] solves the minimum 2-norm solution of an underdetermined subproblem obtained by deleting the last 2 rows of (1.4):

$$\min \|y_k\| \quad \text{s.t.} \quad H_{k-1}^T y_k = \beta_1 e_1.$$

This is solved using the LQ factorization $H_{k-1}^T Q_{k-1}^T = \begin{pmatrix} L_{k-1} & 0 \end{pmatrix}$. A benefit is that x_k is computed as steps along a set of theoretically orthogonal directions (the columns of $V_k Q_{k-1}^T$).

Algorithm 1.4 Unsymmetric Lanczos process

1: $\beta_1 = \|b\|_2$, $v_1 = b/\beta_1$, $\delta_1 = 0$, $v_0 = w_0 = 0$, $w_1 = v_1$.
2: **for** $k = 1, 2, \ldots$ **do**
3: $\alpha_k = w_k^T A v_k$
4: $\delta_{k+1} v_{k+1} = A v_k - \alpha_k v_k - \beta_k v_{k-1}$
5: $\bar{w}_{k+1} = A^T w_k - \alpha_k w_k - \delta_k w_{k-1}$
6: $\beta_{k+1} = \bar{w}_{k+1}^T v_{k+1}$
7: $w_{k+1} = \bar{w}_{k+1}/\beta_{k+1}$
8: **end for**

1.3.2 UNSYMMETRIC LANCZOS

The symmetric Lanczos process from Section 1.3.1 transforms a symmetric matrix A into a symmetric tridiagonal matrix T_k, and generates a set of orthonormal[3] vectors v_k using a 3-term recurrence. If A is not symmetric, there are two other popular strategies, each of which sacrifices some properties of the symmetric Lanczos process.

[3]Orthogonality holds only under exact arithmetic. In finite precision, orthogonality is quickly lost.

If we don't enforce a short-term recurrence, we arrive at the Arnoldi process presented in Section 1.3.4. If we relax the orthogonality requirement, we arrive at the unsymmetric Lanczos process[4], the basis of BiCG and QMR. The unsymmetric Lanczos process is shown in Algorithm 1.4.

[4]The version of unsymmetric Lanczos presented here is adapted from [35].

The scalars δ_{k+1} and β_{k+1} are chosen so that $\|v_{k+1}\| = 1$ and $v_{k+1}^T w_{k+1} = 1$. In matrix terms, if we define

$$T_k = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \delta_{k-1} & \alpha_{k-1} & \beta_k \\ & & & \delta_k & \alpha_k \end{pmatrix}, \quad H_k = \begin{pmatrix} T_k \\ \delta_{k+1} e_k^T \end{pmatrix}, \quad \bar{H}_k = \begin{pmatrix} T_k & \beta_{k+1} e_k \end{pmatrix},$$

then we have the relations

$$AV_k = V_{k+1} H_k \qquad \text{and} \qquad A^T W_k = W_{k+1} \bar{H}_k. \tag{1.7}$$

As with symmetric Lanczos, by defining $x_k = V_k y_k$ to search for some optimal solution within the Krylov subspace, we can write

$$r_k = b - Ax_k = \beta_1 v_1 - AV_k y_k = V_{k+1}(\beta_1 e_1 - H_k y_k).$$

For unsymmetric Lanczos, the columns of V_k are not orthogonal even in exact arithmetic. However, we pretend that they are orthogonal, and using similar ideas from Lanczos CG and MINRES, we arrive at the following two algorithms by solving

$$H_k y_k \approx \beta_1 e_1. \tag{1.8}$$

BiCG

BiCG [23] is an extension of CG to the unsymmetric Lanczos process. As with CG, BiCG can be derived by deleting the last row of (1.8), and solving the resultant square system with LU decomposition [79].

QMR

QMR [26], the quasi-minimum residual method, is the MINRES analog for the unsymmetric Lanczos process. It is derived by solving the least-squares subproblem $\min \|H_k y_k - \beta_1 e_1\|$ at every iteration with QR decomposition. Since the columns of V_k are not orthogonal, QMR doesn't give a minimum residual solution for the original problem in the corresponding Krylov subspace, but the residual norm does tend to decrease.

1.3.3 Transpose-free methods

One disadvantage of BiCG and QMR is that matrix-vector multiplication by A^T is needed. A number of algorithm have been proposed to remove this multiplication. These algorithms are based on the fact that in BiCG, the residual vector lies in the Krylov subspace and can be written as

$$r_k = P_k(A)b,$$

where $P_k(A)$ is a polynomial of A of degree k. With the choice

$$r_k = Q_k(A)P_k(A)b,$$

where $Q_k(A)$ is some other polynomial of degree k, all the coefficients needed for the update at every iteration can be computed without using the multiplication by A^T [79].

CGS [66] is the extension of BiCG with $Q_k(A) \equiv P_k(A)$. CGS has been shown to exhibit irregular convergence behavior. To achieve smoother convergence, BiCGStab [78] was designed with some optimal polynomial $Q_k(A)$ that minimizes the residual at each iteration.

Algorithm 1.5 Arnoldi process

$\beta v_1 = b$ (i.e. $\beta = \|b\|_2$, $v_1 = b/\beta$)
for $k = 1, 2, \ldots, n$ **do**
  $w = Av$
  for $i = 1, 2, \ldots, k$ **do**
    $h_{ik} = w^T v_i$
    $w = w - h_{ik} v_i$
  end for
  $\beta_{k+1} v_{k+1} = w$
end for

1.3.4 The Arnoldi process

Another variant of the Lanczos process for an unsymmetric matrix A is the Arnoldi process. Compared with unsymmetric Lanczos, which preserves the tridiagonal property of H_k and loses the orthogonality among columns of V_k, the Arnoldi process transforms A into an upper Hessenberg matrix H_k with an orthogonal transformation V_k. A short-term recurrence is no longer available for the Arnoldi process. All the Arnoldi vectors must be kept to generate the next vector, as shown in Algorithm 1.5.

The process can be summarized by

$$AV_k = V_{k+1} H_k, \tag{1.9}$$

with

$$H_k = \begin{pmatrix} h_{11} & h_{12} & \ldots & \ldots & h_{1k} \\ \beta_2 & h_{22} & \ldots & \ldots & h_{2k} \\ & \beta_3 & \ldots & \ldots & h_{3k} \\ & & \ddots & \vdots & \vdots \\ & & & \beta_k & h_{kk} \\ & & & & \beta_{k+1} \end{pmatrix}.$$

As with symmetric Lanczos, this allows us to write

$$r_k = b - Ax_k = \beta_1 v_1 - AV_k y_k = V_{k+1}(\beta_1 e_1 - H_k y_k),$$

and our goal is again to find approximate solutions to

$$H_k y_k \approx \beta_1 e_1. \tag{1.10}$$

Note that at the k-th iteration, the amount of memory needed to store H_k and V_k is $O(k^2 + kn)$. Since iterative methods primarily focus on matrices that are large and sparse, the storage cost will soon overwhelm other costs and render the computation infeasible. Most Arnoldi-based methods adopt a strategy of restarting to handle this issue, trading storage cost for slower convergence.

FOM

FOM [62] is the CG analogue for the Arnoldi process, with y_k defined by deleting the last row of (1.10) and solving the truncated system.

GMRES

GMRES [63] is the MINRES counterpart for the Arnoldi process, with y_k defined by the least-squares subproblem $\min \|H_k y_k - \beta_1 e_1\|_2$. Like the methods we study (CG, MINRES, LSQR, LSMR, AMRES), GMRES does not break down, but it might require significantly more storage.

1.3.5 Induced dimension reduction

Induced dimension reduction (IDR) is a class of transpose-free methods that generate residuals in a sequence of nested subspaces of decreasing dimension. The original IDR [85] was proposed by Wesseling and Sonneveld in 1980. It converges after at most $2n$ matrix-vector multiplications under exact arithmetic. Theoretically, this is the same complexity as the unsymmetric Lanczos methods and the transpose-free methods. In 2008, Sonneveld and van Gijzen published IDR(s) [67], an improvement over IDR that takes advantage of extra memory available. The memory required increases linearly with s, while the maximum number of matrix-vector multiplications needed becomes $n + n/s$.

We note that in some informal experiments on square unsymmetric systems $Ax = b$ arising from a convection-diffusion-reaction problem involving several parameters [81], IDR(s) performed significantly better than LSQR or LSMR for some values of the parameters, but for certain other parameter values the reverse was true [82]. In this sense the solvers complement each other.

Algorithm 1.6 Golub-Kahan process Bidiag(A, b)

1: $\beta_1 u_1 = b$, $\alpha_1 v_1 = A^T u_1$.
2: **for** $k = 1, 2, \dots$ **do**
3: $\quad \beta_{k+1} u_{k+1} = A v_k - \alpha_k u_k$
4: $\quad \alpha_{k+1} v_{k+1} = A^T u_{k+1} - \beta_{k+1} v_k$.
5: **end for**

1.4 Krylov subspace methods for rectangular $Ax = b$

In this section, we introduce a number of Krylov subspace methods for the matrix equation $Ax = b$, where A is an m-by-n square or rectangular matrix. When $m > n$, we solve the least-squares problem $\min \|Ax - b\|_2$. When $m < n$, we find the minimum 2-norm solution $\min_{Ax=b} \|x\|_2$. For any m and n, if $Ax = b$ is inconsistent, we solve the problem $\min \|x\|$ s.t. $x = \arg\min \|Ax - b\|$.

1.4.1 The Golub-Kahan process

In the dense case, we can construct orthogonal matrices U and V to transform $\begin{pmatrix} b & A \end{pmatrix}$ to upper bidiagonal form as follows:

$$
\begin{aligned}
U^T \begin{pmatrix} b & A \end{pmatrix} \begin{pmatrix} 1 & \\ & V \end{pmatrix} &= \begin{pmatrix} \times & \times & & \\ & \times & \ddots & \\ & & \ddots & \times \\ & & & \times \end{pmatrix} \\
\Rightarrow \quad \begin{pmatrix} b & AV \end{pmatrix} &= U \begin{pmatrix} \beta_1 e_1 & B \end{pmatrix},
\end{aligned}
$$

where B is a lower bidiagonal matrix. For sparse matrices or linear operators, Golub and Kahan [29] gave an iterative version of the bidiagonalization as shown in Algorithm 1.6.

After k steps, we have $AV_k = U_{k+1} B_k$ and $A^T U_{k+1} = V_{k+1} L_{k+1}^T$, where

$$
B_k = \begin{pmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_k & \alpha_k \\ & & & \beta_{k+1} \end{pmatrix}, \qquad L_{k+1} = \begin{pmatrix} B_k & \alpha_{k+1} e_{k+1} \end{pmatrix}.
$$

This is equivalent to what would be generated by the symmetric Lanczos process with matrix A^TA and starting vector A^Tb. The Lanczos vectors V_k are the same, and the Lanczos tridiagonal matrix satisfies $T_k = B_k^T B_k$. With $x_k = V_k y_k$, the residual vector r_k can be written as

$$r_k = b - AV_k y_k = \beta_1 u_1 - U_{k+1} B_k y_k = U_{k+1}(\beta_1 e_1 - B_k y_k),$$

and our goal is to find an approximate solution to

$$B_k y_k \approx \beta_1 e_1. \tag{1.11}$$

CRAIG

CRAIG [22; 53] is defined by deleting the last row from (1.11), so that y_k satisfies $L_k y_k = \beta_1 e_1$ at each iteration. It is an efficient and reliable method for consistent square or rectangular systems $Ax = b$, and it is known to minimize the error norm $\|x^* - x_k\|$ within each Krylov subspace [51].

LSQR

LSQR [53] is derived by solving $\min \|r_k\| \equiv \min \|\beta_1 e_1 - B_k y_k\|$ at each iteration. Since we are minimizing over a larger Krylov subspace at each iteration, this immmediately implies that $\|r_k\|$ is monotonic for LSQR.

LSMR

LSMR is derived by minimizing $\left\|A^T r_k\right\|$ within each Krylov subspace. LSMR is a major focus in this thesis. It solves linear systems $Ax = b$ and least-squares problems $\min \|Ax - b\|_2$, with A being sparse or a linear operator. It is analytically equivalent to applying MINRES to the normal equation $A^TAx = A^Tb$, so that the quantities $\|A^T r_k\|$ are monotonically decreasing. We have proved that $\|r_k\|$ also decreases monotonically. As we will see in Theorem 4.1.1, this means that a certain backward error measure (the Stewart backward error $\|A^T r_k\|/\|r_k\|$) is always smaller for LSMR than for LSQR. Hence it is safer to terminate LSMR early.

CGLS

LSQR has an equivalent formulation named CGLS [38; 53], which doesn't involve the computation of Golub-Kahan vectors. See Algorithm 1.7.[5]

[5] This version of CGLS is adapted from [53].

Algorithm 1.7 Algorithm CGLS

```
r_0 = b, s_0 = A^T b, p_1 = s_0
ρ_0 = ||s_0||^2, x_0 = 0
for k = 1,2,... do
    q_k = A p_k
    α_k = ρ_{k-1} / ||q_k||^2
    x_k = x_{k-1} + α_k p_k
    r_k = r_{k-1} - α_k q_k
    s_k = A^T r_k
    ρ_k = ||s_k||^2
    β_k = ρ_k / ρ_{k-1}
    p_{k+1} = s_k + β_k p_k
end for
```

1.5 OVERVIEW

Chapter 2 compares the performance of CG and MINRES on various symmetric problems $Ax = b$. The results suggested that MINRES is a superior algorithm even for solving positive definite linear systems. This provides motivation for LSMR, the first algorithm developed in this thesis, to be based on MINRES. Chapter 3 focuses on a mathematical background of constructing LSMR, the derivation of a computationally efficient algorithm, as well as stopping criteria. Chapter 4 describes a number of numerical experiments designed to compare the performance of LSQR and LSMR in solving overdetermined, consistent, or underdetermined linear systems. Chapter 5 derives an iterative algorithm AMRES for solving the negatively damped least-squares problem. Chapter 6 focuses on applications of AMRES to Curtis-Reid scaling, improving approximate singular vectors, and computing singular vectors when the singular value is known. Chapter 7 summarizes the contributions of this thesis and gives a summary of interesting problems available for future research.

The notation used in this thesis is summarized in Table 1.1.

Table 1.1 Notation

A	matrix, sparse matrix or linear operator
A_{ij}	the element of matrix A in i-th row and j-th column
$b, p, r, t, u, v, x, y, \ldots$	vectors
k	subscript index for iteration number. E.g. x_k is the approximate solution generated at the k-th iteration of an iterative solver such as MINRES. In Chapters 3 to 6, k represents the number of Golub-Kahan bidiagonalization iterations.
q	subscript index for RQI iteration number, used in Section 6.2.
c_k, s_k	non-identity elements $\left(\begin{smallmatrix} c_k & s_k \\ -s_k & c_k \end{smallmatrix}\right)$ in a Givens rotation matrix
B_k	bidiagonal matrix generated at the k-th step of Golub-Kahan bidiagonalization.
Greek letters	scalars
$\\|\cdot\\|$	vector 2-norm or the induced matrix 2-norm.
$\\|\cdot\\|_F$	Frobenius norm
$\\|x\\|_A$	energy norm of vector x with repect to positive definite matrix A: $\sqrt{x^T A x}$
cond(A)	condition number of A
e_k	k-th column of an identity matrix
$\mathbb{1}$	a vector with all entries being 1.
R(A)	range of matrix A.
N(A)	null space of matrix A.
$A \succ 0$	A is symmetric positive definite.
x^*	the unique solution to a nonsingular square system $Ax = b$, or more generally the pseudoinverse solution of a rectangular system $Ax \approx b$.

2 A tale of two algorithms

The conjugate gradient method (CG) [38] and the minimum residual method (MINRES) [52] are both Krylov subspace methods for the iterative solution of symmetric linear equations $Ax = b$. CG is commonly used when the matrix A is positive definite, while MINRES is generally reserved for indefinite systems [79, p85]. We reexamine this wisdom from the point of view of early termination on positive-definite systems. This also serves as the rationale for why MINRES is chosen as the basis for the development of LSMR.

In this Chapter, we study the application of CG and MINRES to real symmetric positive-definite (spd) systems $Ax = b$, where A is of dimension $n \times n$. The unique solution is denoted by x^*. The initial approximate solution is $x_0 \equiv 0$, and $r_k \equiv b - Ax_k$ is the residual vector for an approximation x_k within the kth Krylov subspace.

From Section 1.3.1, we know that CG and MINRES use the same information V_{k+1} and H_k to compute solution estimates $x_k = V_k y_k$ within the Krylov subspace $\mathcal{K}_k(A, b) \equiv \text{span}\{b, Ab, A^2b, \ldots, A^{k-1}b\}$ (for each k). It is commonly thought that the number of iterations required will be similar for each method, and hence CG should be preferable on spd systems because it requires less storage and fewer floating-point operations per iteration. This view is justified if an accurate solution is required (stopping tolerance τ close to machine precision ϵ). We show that with looser stopping tolerances, MINRES is sure to terminate sooner than CG when the stopping rule is based on the backward error for x_k, and by numerical examples we illustrate that the difference in iteration numbers can be substantial.

Section 2.1 describes a number of monotonic convergence properties that make CG and MINRES favorable iterative solvers for linear systems. Section 2.2 introduces the concept of backward error and how it is used in designing stopping rules for iterative solvers, and the reason why MINRES is more favorable for applications where backward error is important. Section 2.3 compares experimentally the behavior of CG and MINRES in terms of energy norm error, backward error, residual norm, and solution norm.

2.1 Monotonicity of norms

In designing iterative solvers for linear equations, we often gauge convergence by computing some norms from the current iterate x_k. These norms[1] include $\|x_k - x^*\|$, $\|x_k - x^*\|_A$, $\|r_k\|$. An iterative method might sometimes be stopped by an iteration limit or a time limit. It is then highly desirable that some or all of the above norms converge monotonically.

[1]For any vector x, the *energy norm* with repect to spd matrix A is defined as
$$\|x\|_A = \sqrt{x^T A x}.$$

It is also desirable to have monotonic convergence for $\|x_k\|$. First, some applications such as trust-region methods [68] depend on that property. Second, when convergence is measure by backward error (Section 2.2), monotonicity in $\|x_k\|$ (together with monotonicity in $\|r_k\|$) gives monotonic convergence in backward error. More generally, if $\|x_k\|$ is monotonic, there cannot be catastrophic cancellation error in stepping from x_k to x_{k+1}.

2.1.1 Properties of CG

A number of monotonicity properties have been found by various authors. We summarize them here for easy reference.

Theorem 2.1.1. *[68, Thm 2.1] For CG on an spd system $Ax = b$, $\|x_k\|$ is strictly increasing.*

Theorem 2.1.2. *[38, Thm 4:3] For CG on an spd system $Ax = b$, $\|x^* - x_k\|_A$ is strictly decreasing.*

Theorem 2.1.3. *[38, Thm 6:3] For CG on an spd system $Ax = b$, $\|x^* - x_k\|$ is strictly decreasing.*

$\|r_k\|$ is not monotonic for CG. Examples are shown in Figure 2.4.

2.1.2 Properties of CR and MINRES

Here we prove a number of monotonicity properties for CR and MINRES on an spd system $Ax = b$. Some known properties are also included for completeness. Relations from Algorithm 1.3 (CR) are used extensively in the proofs. Termination of CR occurs when $r_k = 0$ for some index $k = \ell \le n$ ($\Rightarrow \rho_\ell = \beta_\ell = 0$, $r_\ell = s_\ell = p_\ell = q_\ell = 0$, $x_\ell = x^*$, where $Ax^* = b$). Note: This ℓ is the same ℓ at which the Lanczos process theoretically terminates for the given A and b.

Theorem 2.1.4. *The following properties hold for Algorithm CR:*

(a) $q_i^T q_j = 0 \qquad (0 \le i, j \le \ell - 1,\ i \neq j)$

(b) $r_i^T q_j = 0 \qquad (0 \le i, j \le \ell - 1,\ i \geq j + 1)$

(c) $r_i \neq 0 \Rightarrow p_i \neq 0 \qquad (0 \le i \le \ell - 1)$

Proof. Given in [44, Theorem 1]. □

Theorem 2.1.5. *The following properties hold for Algorithm* CR *on an spd system $Ax = b$:*

(a) $\alpha_i > 0 \qquad (i = 1, \dots, \ell)$

(b) $\beta_i > 0 \qquad (i = 1, \dots, \ell - 1)$
$\beta_\ell = 0$

(c) $p_i^T q_j > 0 \qquad (0 \le i, j \le \ell - 1)$

(d) $p_i^T p_j > 0 \qquad (0 \le i, j \le \ell - 1)$

(e) $x_i^T p_j > 0 \qquad (1 \le i \le \ell, 0 \le j \le \ell - 1)$

(f) $r_i^T p_j > 0 \qquad (0 \le i, j \le \ell - 1)$

Proof. (a) Here we use the fact that A is spd. Since $r_i \neq 0$ for $0 \le i \le \ell - 1$, we have for $1 \le i \le \ell$,

$$\begin{aligned} \rho_{i-1} &= r_{i-1}^T s_{i-1} = r_{i-1}^T A r_{i-1} > 0 \qquad (A \succ 0) \qquad (2.1) \\ \alpha_i &= \rho_{i-1} / \|q_{i-1}\|^2 > 0, \end{aligned}$$

where $q_{i-1} \neq 0$ follows from $q_{i-1} = A p_{i-1}$ and Theorem 2.1.4 (c).

(b) For $1 \le i \le \ell - 1$, we have

$$\beta_i = \rho_i / \rho_{i-1} > 0, \qquad \text{(by (2.1))}$$

and $r_\ell = 0$ implies $\beta_\ell = 0$.

(c) For any $0 \le i, j \le \ell - 1$, we have

Case I: $i = j$

$$p_i^T q_i = p_i^T A p_i > 0 \qquad (A \succ 0)$$

where $p_i \neq 0$ from Theorem 2.1.4 (c). Next, we prove the cases where $i \neq j$ by induction.

Case II: $i - j = k > 0$

$$\begin{aligned} p_i^T q_j = p_i^T q_{i-k} &= r_i^T q_{i-k} + \beta_i p_{i-1}^T q_{i-k} \\ &= \beta_i p_{i-1}^T q_{i-k} && \text{(by Thm 2.1.4 (b))} \\ &> 0, \end{aligned}$$

where $\beta_i > 0$ by (b)[2] and $p_{i-1}^T q_{i-k} > 0$ by induction as $(i-1) - (i-k) = k - 1 < k$.

[2] Note that $i - j = k > 0$ implies $i \geq 1$.

Case III: $j - i = k > 0$

$$\begin{aligned} p_i^T q_j = p_i^T q_{i+k} &= p_i^T A p_{i+k} \\ &= p_i^T A(r_{i+k} + \beta_{i+k} p_{i+k-1}) \\ &= q_i^T r_{i+k} + \beta_{i+k} p_i^T q_{i+k-1} \\ &= \beta_{i+k} p_i^T q_{i+k-1} && \text{(by Thm 2.1.4 (b))} \\ &> 0, \end{aligned}$$

where $\beta_{i+k} = \beta_j > 0$ by (b) and $p_i^T q_{i+k-1} > 0$ by induction as $(i + k - 1) - i = k - 1 < k$.

(d) Define $\mathcal{P} \equiv \text{span}\{p_0, p_1, \dots, p_{\ell-1}\}$ and $\mathcal{Q} \equiv \text{span}\{q_0, \dots, q_{\ell-1}\}$ at termination. By construction, $\mathcal{P} = \text{span}\{b, Ab, \dots, A^{\ell-1}b\}$ and $\mathcal{Q} = \text{span}\{Ab, \dots, A^{\ell}b\}$ (since $q_i = Ap_i$). Again by construction, $x_\ell \in \mathcal{P}$, and since $r_\ell = 0$ we have $b = Ax_\ell \Rightarrow b \in \mathcal{Q}$. We see that $\mathcal{P} \subseteq \mathcal{Q}$. By Theorem 2.1.4(a), $\{q_i/\|q_i\|\}_{i=0}^{\ell-1}$ forms an orthonormal basis for $\mathcal{Q}$. If we project $p_i \in \mathcal{P} \subseteq \mathcal{Q}$ onto this basis, we have

$$p_i = \sum_{k=0}^{\ell-1} \frac{p_i^T q_k}{q_k^T q_k} q_k,$$

where all coordinates are positive from (c). Similarly for any other p_j. Therefore $p_i^T p_j > 0$ for any $0 \leq i, j < \ell$.

(e) By construction,

$$x_i = x_{i-1} + \alpha_i p_{i-1} = \cdots = \sum_{k=1}^{i} \alpha_k p_{k-1} \qquad (x_0 = 0)$$

Therefore $x_i^T p_i > 0$ by (d) and (a).

(f) Note that any r_i can be expressed as a sum of q_i:

$$\begin{aligned} r_i &= r_{i+1} + \alpha_{i+1} q_i \\ &= \cdots \\ &= r_l + \alpha_l q_{l-1} + \cdots + \alpha_{i+1} q_i \\ &= \alpha_l q_{l-1} + \cdots + \alpha_{i+1} q_i. \end{aligned}$$

Thus we have

$$r_i^T p_j = (\alpha_l q_{l-1} + \cdots + \alpha_{i+1} q_i)^T p_j > 0,$$

where the inequality follows from (a) and (c).

□

We are now able to prove our main theorem about the monotonic increase of $\|x_k\|$ for CR and MINRES. A similar result was proved for CG by Steihaug [68].

Theorem 2.1.6. *For CR (and hence MINRES) on an spd system $Ax = b$, $\|x_k\|$ is strictly increasing.*

Proof. $\|x_i\|^2 - \|x_{i-1}\|^2 = 2\alpha_i x_{i-1}^T p_{i-1} + p_{i-1}^T p_{i-1} > 0$, where the last inequality follows from Theorem 2.1.5 (a), (d) and (e). Therefore $\|x_i\| > \|x_{i-1}\|$. □

The following theorem is a direct consequence of Hestenes and Stiefel [38, Thm 7:5]. However, the second half of that theorem, $\|x^* - x_{k-1}^{\mathrm{CG}}\| > \|x^* - x_k^{\mathrm{MINRES}}\|$, rarely holds in machine arithmetic. We give here an alternative proof that does not depend on CG.

Theorem 2.1.7. *For CR (and hence MINRES) on an spd system $Ax = b$, the error $\|x^* - x_k\|$ is strictly decreasing.*

Proof. From the update rule for x_k, we can express x^* as

$$\begin{aligned} x^* = x_l &= x_{l-1} + \alpha_l p_{l-1} \\ &= \cdots \\ &= x_k + \alpha_{k+1} p_k + \cdots + \alpha_l p_{l-1} &\quad (2.2)\\ &= x_{k-1} + \alpha_k p_{k-1} + \alpha_{k+1} p_k + \cdots + \alpha_l p_{l-1}. &\quad (2.3) \end{aligned}$$

Using the last two equalities above, we can write

$$\begin{aligned}
&\|x^* - x_{k-1}\|^2 - \|x^* - x_k\|^2 \\
&= (x_l - x_{k-1})^T(x_l - x_{k-1}) - (x_l - x_k)^T(x_l - x_k) \\
&= 2\alpha_k p_{k-1}^T(\alpha_{k+1}p_k + \cdots + \alpha_l p_{l-1}) + \alpha_k^2 p_{k-1}^T p_{k-1} \\
&> 0,
\end{aligned}$$

where the last inequality follows from Theorem 2.1.5 (a), (d). □

The following theorem is given in [38, Thm 7:4]. We give an alternative proof here.

Theorem 2.1.8. *For CR (and hence MINRES) on an spd system $Ax = b$, the energy norm error $\|x^* - x_k\|_A$ is strictly decreasing.*

Proof. From (2.2) and (2.3) we can write

$$\begin{aligned}
&\|x_l - x_{k-1}\|_A^2 - \|x_l - x_k\|_A^2 \\
&= (x_l - x_{k-1})^T A(x_l - x_{k-1}) - (x_l - x_k)^T A(x_l - x_k) \\
&= 2\alpha_k p_{k-1}^T A(\alpha_{k+1}p_k + \cdots + \alpha_{l-1} p_{l-1}) + \alpha_k^2 p_{k-1}^T A p_{k-1} \\
&= 2\alpha_k q_{k-1}^T(\alpha_{k+1}p_k + \cdots + \alpha_{l-1} p_{l-1}) + \alpha_k^2 q_{k-1}^T p_{k-1} \\
&> 0,
\end{aligned}$$

where the last inequality follows from Theorem 2.1.5 (a), (c). □

The following theorem is available from [52] and [38, Thm 7:2], and is the characterizing property of MINRES. We include it here for completeness.

Theorem 2.1.9. *For MINRES on any system $Ax = b$, $\|r_k\|$ is decreasing.*

Proof. This follows immediately from (1.6). □

2.2 Backward error analysis

> "The data frequently contains uncertainties due to measurements, previous computations, or errors committed in storing numbers on the computer. If the backward error is no larger than these uncertainties then the computed solution can hardly be criticized — it may be the solution we are seeking, for all we know." Nicholas J. Higham, *Accuracy and Stability of Numerical Algorithms* (2002)

For many physical problems requiring numerical solution, we are given inexact or uncertain input data. Examples include model estimation in geophysics [45], system identification in control theory [41], and super-resolution imaging [80]. For these problems, it is not justifiable to seek a solution beyond the accuracy of the data [19]. Computation time may be wasted in the extra iterations without yielding a more desirable answer [3]. Also, rounding errors are introduced during computation. Both errors in the original data and rounding errors can be analyzed in a common framework by applying the *Wilkinson principle*, which considers any computed solution to be the exact solution of a *nearby problem* [10; 25]. The measure of "nearby" should match the error in the input data. The design of stopping rules from this viewpoint is an important part of *backward error analysis* [4; 39; 48; 61].

For a consistent linear system $Ax = b$, there may be uncertainty in A and/or b. From now on we think of x_k coming from the kth iteration of one of the iterative solvers. Following Titley-Peloquin [75] we say that x_k is an *acceptable solution* if and only if there exist perturbations E and f satisfying

$$(A+E)x_k = b+f, \qquad \frac{\|E\|}{\|A\|} \le \alpha, \qquad \frac{\|f\|}{\|b\|} \le \beta \tag{2.4}$$

for some tolerances $\alpha \ge 0$, $\beta \ge 0$ that reflect the (preferably known) accuracy of the data. We are naturally interested in minimizing the size of E and f. If we define the optimization problem

$$\min_{\xi,E,f} \xi \ \text{ s.t. } \ (A+E)x_k = b+f, \quad \frac{\|E\|}{\|A\|} \le \alpha\xi, \quad \frac{\|f\|}{\|b\|} \le \beta\xi$$

to have optimal solution ξ_k, E_k, f_k (all functions of x_k, α, and β), we see that x_k is an acceptable solution if and only if $\xi_k \le 1$. We call ξ_k the *normwise relative backward error* (NRBE) for x_k.

With $r_k = b - Ax_k$, the optimal solution ξ_k, E_k, f_k is shown in [75] to be given by

$$\phi_k = \frac{\beta\|b\|}{\alpha\|A\|\|x_k\| + \beta\|b\|}, \qquad E_k = \frac{(1-\phi_k)}{\|x_k\|^2} r_k x_k^T, \tag{2.5}$$

$$\xi_k = \frac{\|r_k\|}{\alpha\|A\|\|x_k\| + \beta\|b\|}, \qquad f_k = -\phi_k r_k. \tag{2.6}$$

(See [39, p12] for the case $\beta = 0$ and [39, §7.1 and p336] for the case $\alpha = \beta$.)

2.2.1 STOPPING RULE

For general tolerances α and β, the condition $\xi_k \leq 1$ for x_k to be an acceptable solution becomes

$$\|r_k\| \leq \alpha\|A\|\|x_k\| + \beta\|b\|, \tag{2.7}$$

the stopping rule used in LSQR for consistent systems [53, p54, rule S1].

2.2.2 MONOTONIC BACKWARD ERRORS

Of interest is the *size* of the perturbations to A and b for which x_k is an exact solution of $Ax = b$. From (2.5)–(2.6), the perturbations have the following norms:

$$\|E_k\| = (1-\phi_k)\frac{\|r_k\|}{\|x_k\|} = \frac{\alpha\|A\|\|r_k\|}{\alpha\|A\|\|x_k\| + \beta\|b\|}, \tag{2.8}$$

$$\|f_k\| = \phi_k\|r_k\| = \frac{\beta\|b\|\|r_k\|}{\alpha\|A\|\|x_k\| + \beta\|b\|}. \tag{2.9}$$

Since $\|x_k\|$ is monotonically increasing for CG and MINRES (when A is spd), we see from (2.5) that ϕ_k is monotonically decreasing for both solvers. Since $\|r_k\|$ is monotonically decreasing for MINRES (but not for CG), we have the following result.

Theorem 2.2.1. *Suppose $\alpha > 0$ and $\beta > 0$ in (2.4). For CR and MINRES (but not CG), the relative backward errors $\|E_k\|/\|A\|$ and $\|f_k\|/\|b\|$ decrease monotonically.*

Proof. This follows from (2.8)–(2.9) with $\|x_k\|$ increasing for both solvers and $\|r_k\|$ decreasing for CR and MINRES but not for CG. □

2.2.3 OTHER CONVERGENCE MEASURES

Error $\|x^* - x_k\|$ and energy norm error $\|x^* - x_k\|_A$ are two possible measures of convergence. In trust-region methods [68], some eigenvalue problems [74], finite element approximations [1], and some other applications [46; 84], it is desirable to minimize $\|x^* - x_k\|_A$, which makes CG a sensible algorithm to use.

We should note that since x^* is not known, neither $\|x^* - x_k\|$ nor $\|x^* - x_k\|_A$ can be computed directly from the CG algorithm. However, bounds and estimates have been derived for $\|x^* - x_k\|_A$ [9; 30] and they can be used for stopping rules based on the energy norm error.

An alternative stopping criterion is derived for MINRES by Calvetti et al. [8] based on an L-curve defined by $\|r_k\|$ and $\text{cond}(H_k)$.

2.3 NUMERICAL RESULTS

Here we compare the convergence of CG and MINRES on various spd systems $Ax = b$ and some associated indefinite systems $(A - \delta I)x = b$. The test examples are drawn from the University of Florida Sparse Matrix Collection (Davis [18]). We experimented with all 26 cases for which A is real spd and b is supplied. We compute the condition number for each test matrix by finding the largest and smallest eigenvalue using `eigs(A,1,'LM')` and `eigs(A,1,'SM')` respectively. For this test set, the condition numbers range from 1.7E+03 to 3.1E+13.

Since A is spd, we applied diagonal preconditioning by redefining A and b as follows: $d = \text{diag}(A)$, $D = \text{diag}(1./\text{sqrt}(d))$, $A \leftarrow DAD$, $b \leftarrow Db$, $b \leftarrow b/\|b\|$. Thus in the figures below we have $\text{diag}(A) = I$ and $\|b\| = 1$. With this preconditioning, the condition numbers range from 1.2E+01 to 2.2E+11. The distribution of condition number of the test set matrices before and after preconditioning is shown in Figure 2.1.

The stopping rule used for CG and MINRES was (2.7) with $\alpha = 0$ and $\beta = 10^{-8}$ (that is, $\|r_k\| \le 10^{-8}\|b\| = 10^{-8}$).

2.3.1 POSITIVE-DEFINITE SYSTEMS

In defining backward errors, we assume for simplicity that $\alpha > 0$ and $\beta = 0$ in (2.4)–(2.6), even though it doesn't match the choice $\alpha = 0$ and $\beta = 10^{-8}$ in the stopping rule. This gives $\phi_k = 0$ and $\|E_k\| = \|r_k\|/\|x_k\|$ in (2.8). Thus, as in Theorem 2.2.1, we expect $\|E_k\|$ to decrease monotonically for CR and MINRES but not for CG.

We also compute $\|x^* - x_k\|$ and $\|x^* - x_k\|_A$ at each iteration for both algorithms, where x^* is obtained by MATLAB's backslash function `A\b`, which uses a sparse Cholesky factorization of A [12].

In Figure 2.2 and 2.3, we plot $\|r_k\|/\|x_k\|$, $\|x^* - x_k\|$, and $\|x^* - x_k\|_A$ for CG and MINRES for four different problems. For CG, the plots confirm the properties in Theorem 2.1.2 and 2.1.3 that $\|x^* - x_k\|$ and $\|x^* - x_k\|_A$ are monotonic. For MINRES, the plots confirm the properties in Theorem 2.2.1, 2.1.8, and 2.1.7 that $\|r_k\|/\|x_k\|$, $\|x^* - x_k\|$, and $\|x^* - x_k\|_A$ are monotonic.

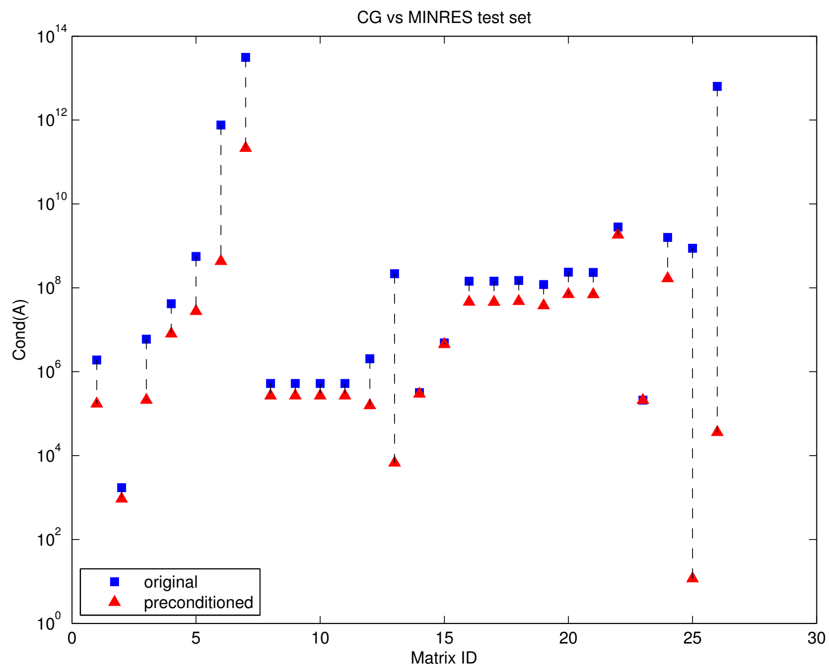


Figure 2.1: Distribution of condition number for matrices used for CG vs MINRES comparison, before and after diagonal preconditioning

Figure 2.2 (left) shows problem `Schenk_AFE_af_shell8` with A of size 504855×504855 and cond(A) = 2.7E+05. From the plot of backward errors $\|r_k\|/\|x_k\|$, we see that both CG and MINRES converge quickly at the early iterations. Then the backward error of MINRES plateaus at about iteration 80, and the backward error of CG stays about 1 order of magnitude behind MINRES. A similar phenomenon of fast convergence at early iterations followed by slow convergence is also observed in the energy norm error and 2-norm error plots.

Figure 2.2 (right) shows problem `Cannizzo_sts4098` with A of size 19779×19779 and cond(A) = 6.7E+03. MINRES converges slightly faster in terms of backward error, while CG converges slightly faster in terms of energy norm error and 2-norm error.

Figure 2.3 (left) shows problem `Simon_raefsky4` with A of size 19779×19779 and cond(A) = 2.2E+11. Because of the high condition number, both algorithms hit the $5n$ iteration limit that we set. We see that the backward error for MINRES converges faster than for CG as expected. For the energy norm error, CG is able to decrease over 5 orders of magnitude while MINRES plateaus after a 2 orders of magnitude decrease.

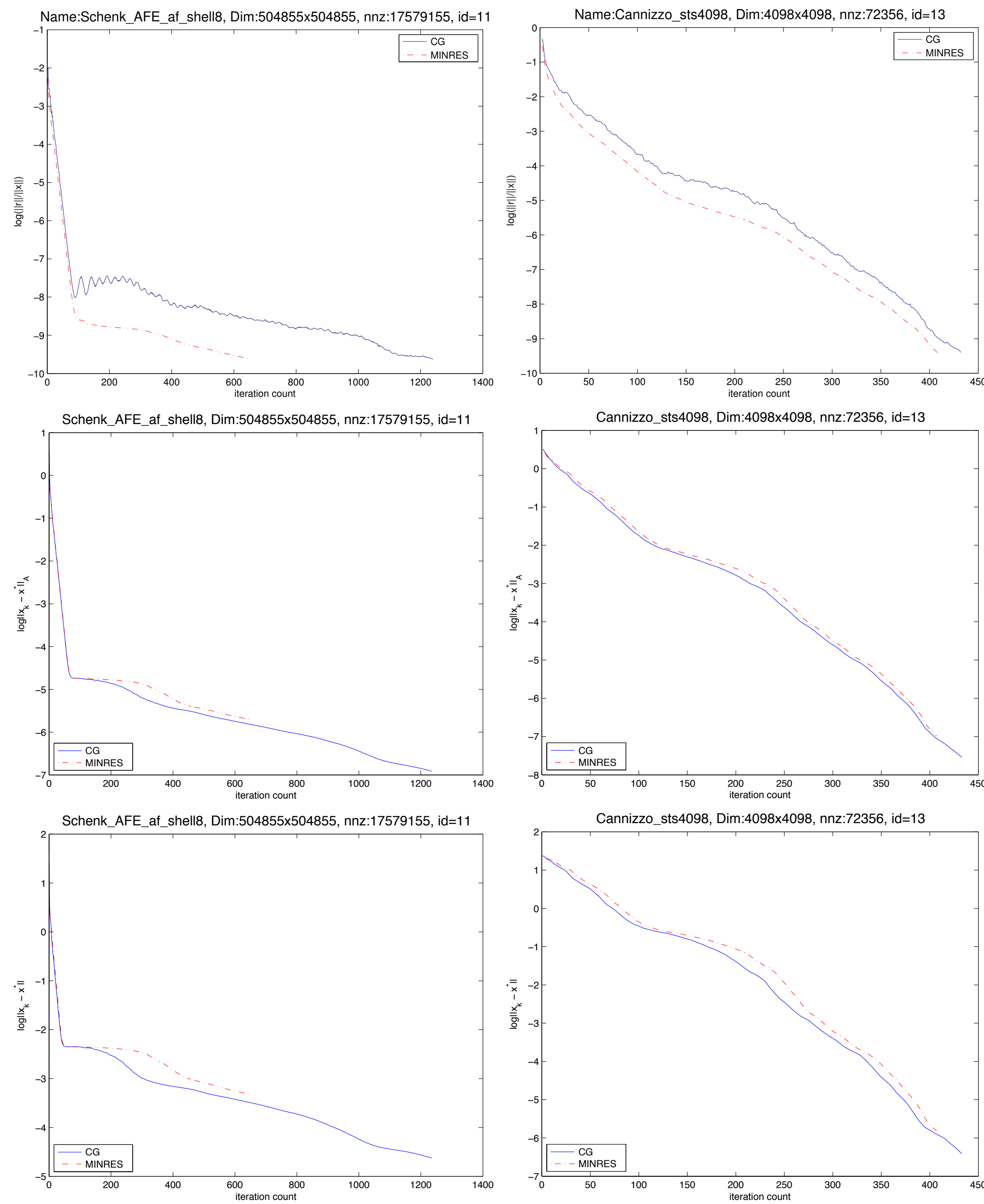


Figure 2.2: Comparison of backward and forward errors for CG and MINRES solving two spd systems $Ax = b$.
Top: The values of $\log_{10}(\|r_k\|/\|x_k\|)$ are plotted against iteration number k. These values define $\log_{10}(\|E_k\|)$ when the stopping tolerances in (2.7) are $\alpha > 0$ and $\beta = 0$. **Middle**: The values of $\log_{10} \|x_k - x^*\|_A$ are plotted against iteration number k. This is the quantity that CG minimizes at each iteration. **Bottom**: The values of $\log_{10} \|x_k - x^*\|$.
Left: Problem `Schenk_AFE_af_shell8`, with $n = 504855$ and $\mathrm{cond}(A)$ = 2.7E+05.
Right: `Cannizzo_sts4098`, with $n = 19779$ and $\mathrm{cond}(A)$ = 6.7E+03.

For both the energy norm error and 2-norm error, MINRES reaches a lower point than CG for some iterations. This must be due to numerical error in CG and MINRES (a result of loss of orthogonality in V_k).

Figure 2.3 (right) shows problem `BenElechi_BenElechi1` with A of size 245874×245874 and cond(A) = 1.8E+09. The backward error of MINRES stays ahead of CG by 2 orders of magnitude for most iterations. Around iteration 32000, the backward error of both algorithms goes down rapidly and CG catches up with MINRES. Both algorithms exhibit a plateau on energy norm error for the first 20000 iterations. The error norms for CG start decreasing around iteration 20000 and decreases even faster after iteration 30000.

Figure 2.4 shows $\|r_k\|$ and $\|x_k\|$ for CG and MINRES on two typical spd examples. We see that $\|x_k\|$ is monotonically increasing for both solvers, and the $\|x_k\|$ values rise fairly rapidly to their limiting value $\|x^*\|$, with a moderate delay for MINRES.

Figure 2.5 shows $\|r_k\|$ and $\|x_k\|$ for CG and MINRES on two spd examples in which the residual decrease and the solution norm increase are somewhat slower than typical. The rise of $\|x_k\|$ for MINRES is rather more delayed. In the second case, if the stopping tolerance were $\beta = 10^{-6}$ rather than $\beta = 10^{-8}$, the final MINRES $\|x_k\|$ ($k \approx 10000$) would be less than half the exact value $\|x^*\|$. It will be of future interest to evaluate this effect within the context of trust-region methods for optimization.

WHY DOES $\|r_k\|$ FOR CG LAG BEHIND MINRES?

It is commonly thought that even though MINRES is known to minimize $\|r_k\|$ at each iteration, the cumulative minimum of $\|r_k\|$ for CG should approximately match that of MINRES. That is,

$$\min_{0 \le i \le k} \|r_i^{\mathrm{CG}}\| \approx \|r_k^{\mathrm{MINRES}}\|.$$

However, in Figure 2.2 and 2.3 we see that $\|r_k\|$ for MINRES is often smaller than for CG by 1 or 2 orders of magnitude. This phenomenon can be explained by the following relations between $\|r_k^{\mathrm{CG}}\|$ and $\|r_k^{\mathrm{MINRES}}\|$ [76; 35, Lemma 5.4.1]:

$$\|r_k^{\mathrm{CG}}\| = \frac{\|r_k^{\mathrm{MINRES}}\|}{\sqrt{1 - \|r_k^{\mathrm{MINRES}}\|^2 / \|r_{k-1}^{\mathrm{MINRES}}\|^2}}. \tag{2.10}$$

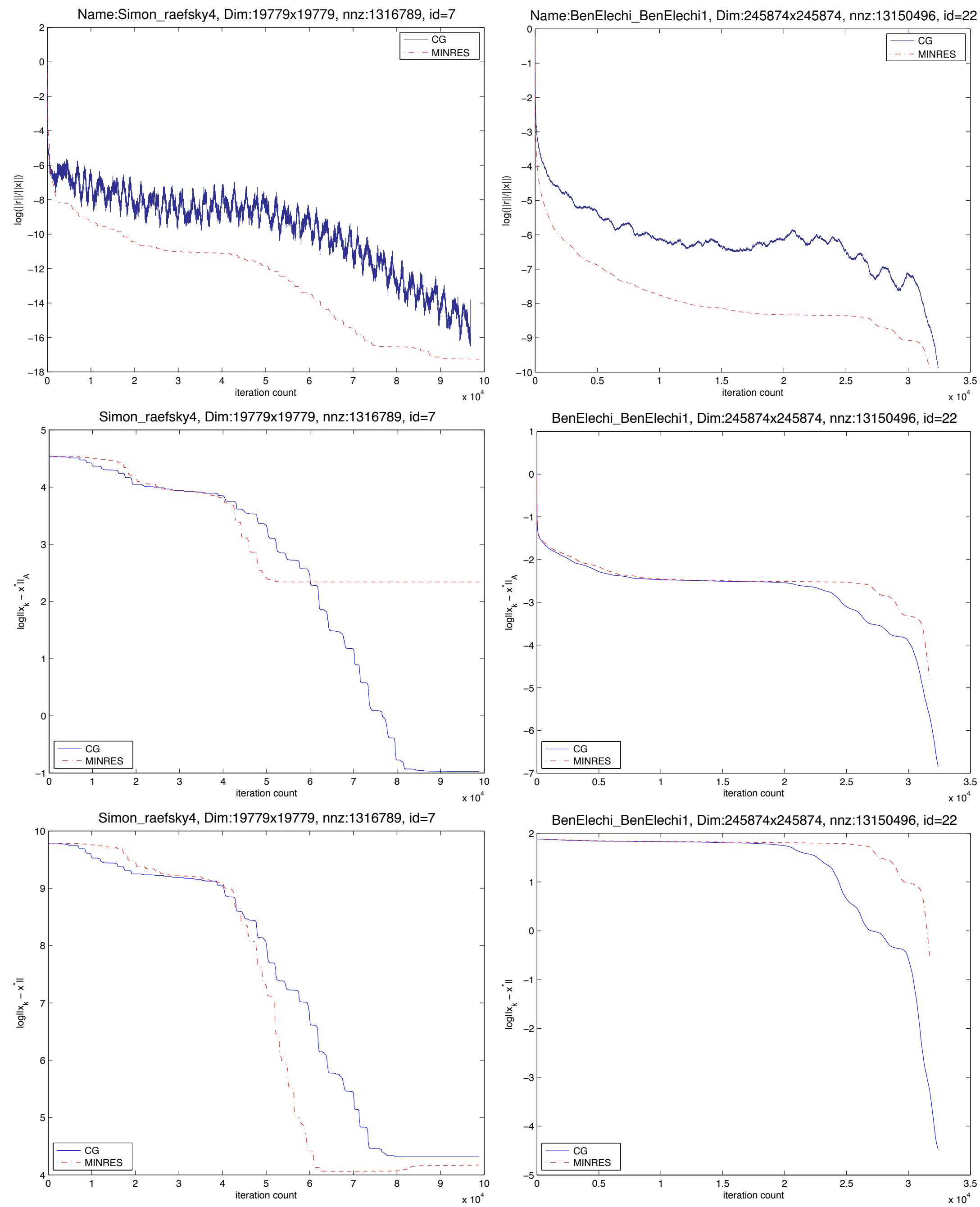


Figure 2.3: Comparison of backward and forward errors for CG and MINRES solving two spd systems $Ax = b$.
Top: The values of $\log_{10}(\|r_k\|/\|x_k\|)$ are plotted against iteration number k. These values define $\log_{10}(\|E_k\|)$ when the stopping tolerances in (2.7) are $\alpha > 0$ and $\beta = 0$. **Middle**: The values of $\log_{10}\|x_k - x^*\|_A$ are plotted against iteration number k. This is the quantity that CG minimizes at each iteration. **Bottom**: The values of $\log_{10}\|x_k - x^*\|$.
Left: Problem `Simon_raefsky4`, with $n = 19779$ and cond(A) = 2.2E+11.
Right: `BenElechi_BenElechi1`, with $n = 245874$ and cond(A) = 1.8E+09.

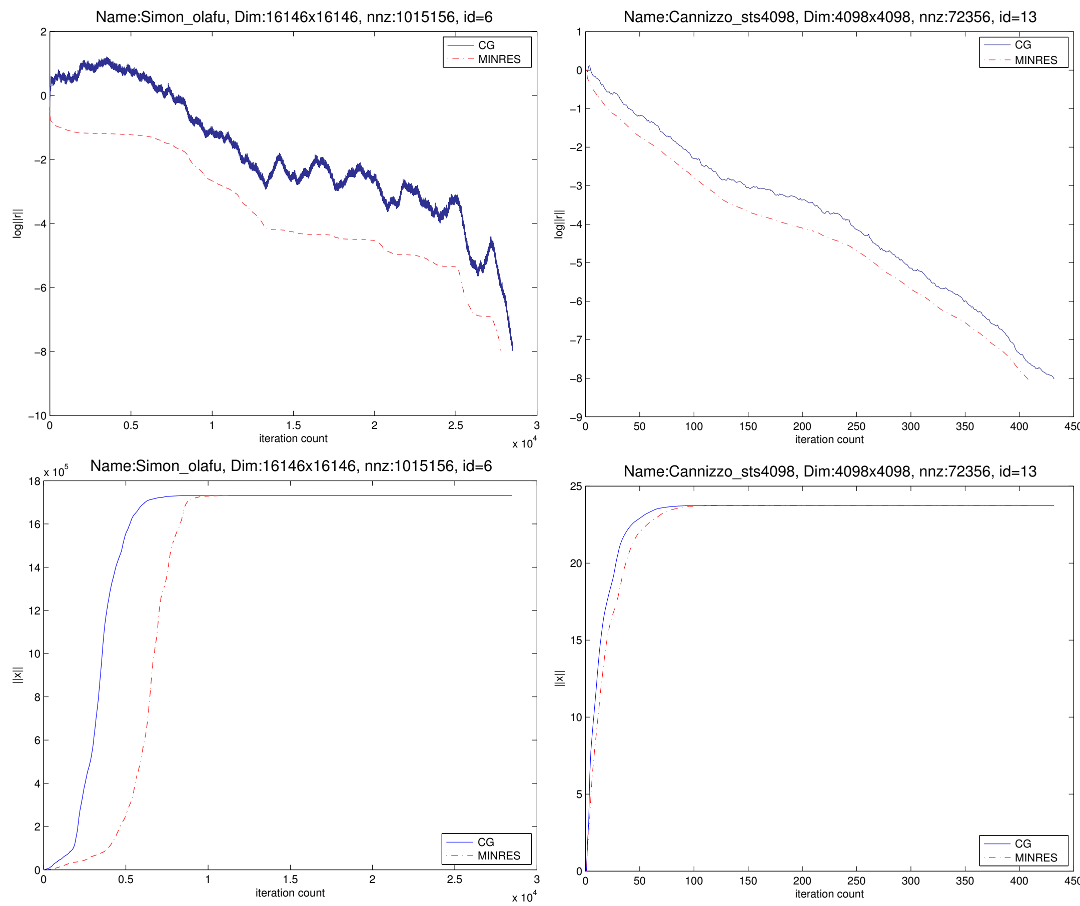


Figure 2.4: Comparison of residual and solution norms for CG and MINRES solving two spd systems $Ax = b$ with $n = 16146$ and 4098.
Top: The values of $\log_{10} \|r_k\|$ are plotted against iteration number k. **Bottom**: The values of $\|x_k\|$ are plotted against k. The solution norms grow somewhat faster for CG than for MINRES. Both reach the limiting value $\|x^*\|$ significantly before x_k is close to x.

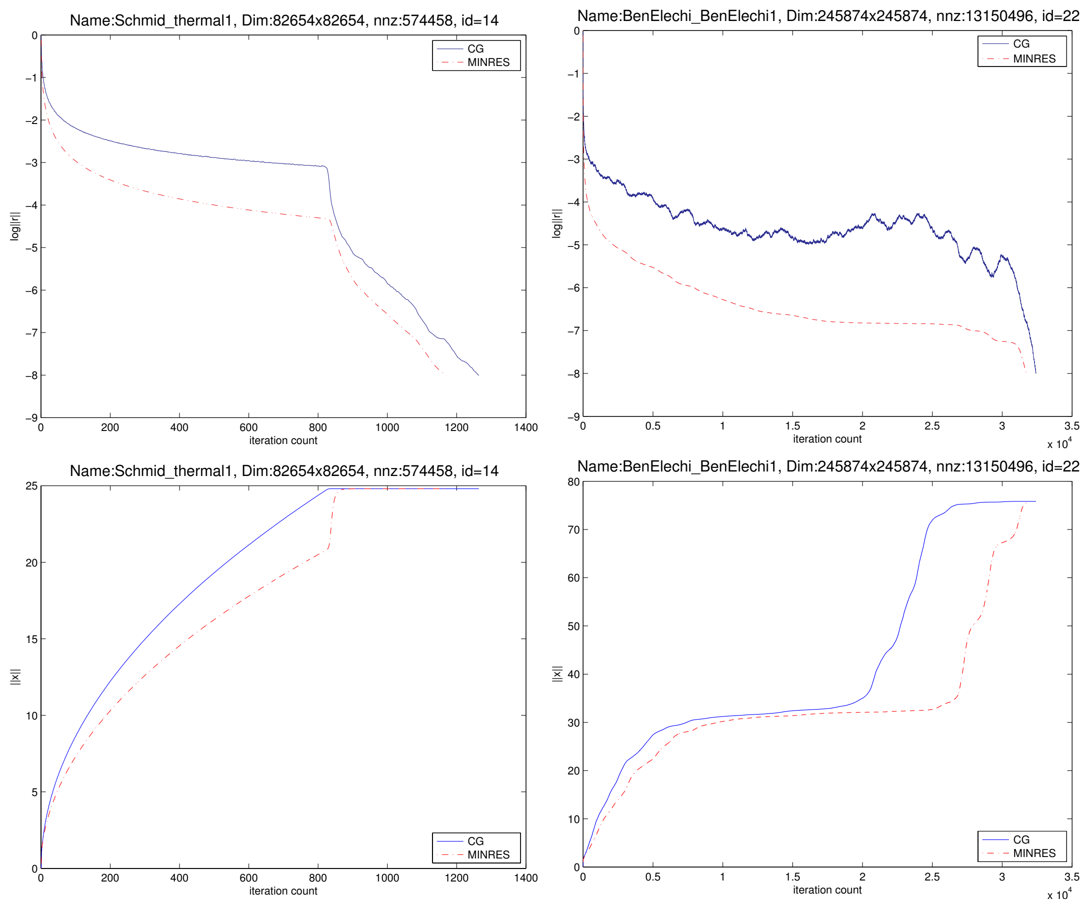


Figure 2.5: Comparison of residual and solution norms for CG and MINRES solving two spd systems $Ax = b$ with $n = 82654$ and 245874. Sometimes the solution norms take longer to reach the limiting value $\|x\|$.
Top: The values of $\log_{10}\|r_k\|$ are plotted against iteration number k. **Bottom**: The values of $\|x_k\|$ are plotted against k. Again the solution norms grow faster for CG.

From (2.10), one can infer that if $\|r_k^{\text{MINRES}}\|$ decreases a lot between iterations $k-1$ and k, then $\|r_k^{\text{CG}}\|$ would be roughly the same as $\|r_k^{\text{MINRES}}\|$. The converse also holds, in that $\|r_k^{\text{CG}}\|$ will be much larger than $\|r_k^{\text{MINRES}}\|$ if MINRES is almost stalling at iteration k (i.e., $\|r_k^{\text{MINRES}}\|$ did not decrease much relative to the previous iteration). The above analysis was pointed out by Titley-Peloquin [76] in comparing LSQR and LSMR. We repeat the analysis here for CG vs MINRES and extend it to demonstrate why there is a lag in general for large problems.

With a residual-based stopping rule, CG and MINRES stop when

$$\|r_k\| \leq \beta \|r_0\|.$$

When CG and MINRES stop at iteration ℓ, we have

$$\prod_{k=1}^{\ell} \frac{\|r_k\|}{\|r_{k-1}\|} = \frac{\|r_\ell\|}{\|r_0\|} \approx \beta.$$

Thus on average, $\|r_k^{\text{MINRES}}\|/\|r_{k-1}^{\text{MINRES}}\|$ will be closer to 1 if ℓ is large. This means that for systems that take more iterations to converge, CG will lag behind MINRES more (a bigger gap between $\|r_k^{\text{CG}}\|$ and $\|r_k^{\text{MINRES}}\|$).

2.3.2 Indefinite systems

A key part of Steihaug's trust-region method for large-scale unconstrained optimization [68] (see also [15]) is his proof that when CG is applied to a symmetric (possibly indefinite) system $Ax = b$, the solution norms $\|x_1\|, \ldots, \|x_k\|$ are strictly increasing as long as $p_j^T A p_j > 0$ for all iterations $1 \leq j \leq k$. (We are using the notation in Algorithm 1.3.)

From our proof of Theorem 2.1.5, we see that the same property holds for CR and MINRES as long as both $p_j^T A p_j > 0$ and $r_j^T A r_j > 0$ for all iterations $1 \leq j \leq k$. Since MINRES might be a useful solver in the trust-region context, it is of interest now to offer some empirical results about the behavior of $\|x_k\|$ when MINRES is applied to indefinite systems.

First, on the nonsingular indefinite system

$$\begin{pmatrix} 2 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 2 \end{pmatrix} x = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \tag{2.11}$$

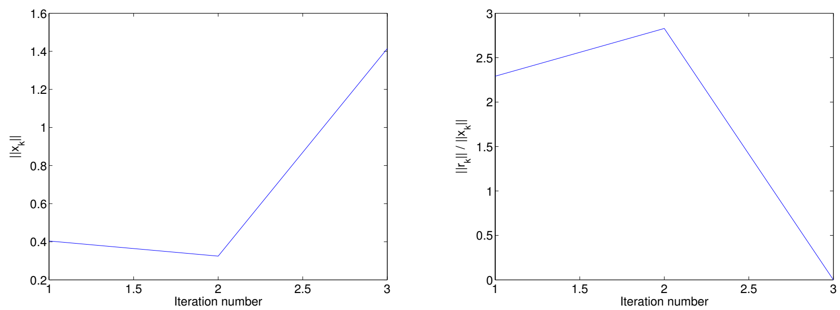


Figure 2.6: For MINRES on indefinite problem (2.11), $\|x_k\|$ and the backward error $\|r_k\|/\|x_k\|$ are both slightly non-monotonic.

MINRES gives non-monotonic solution norms, as shown in the left plot of Figure 2.6. The decrease in $\|x_k\|$ implies that the backward errors $\|r_k\|/\|x_k\|$ may not be monotonic, as illustrated in the right plot.

More generally, we can gain an impression of the behavior of $\|x_k\|$ by recalling from Choi et al. [14] the connection between MINRES and MINRES-QLP. Both methods compute the iterates $x_k^M = V_k y_k^M$ in (1.6) from the subproblems

$$y_k^M = \arg\min_{y \in \mathbb{R}^k} \|H_k y - \beta_1 e_1\| \qquad \text{and possibly} \qquad T_\ell y_\ell^M = \beta_1 e_1,$$

where $k = \ell$ is the last iteration. When A is nonsingular or $Ax = b$ is consistent (which we now assume), y_k^M is uniquely defined for each $k \le \ell$ and the methods compute the same iterates x_k^M (but by different numerical methods). In fact they both compute the expanding QR factorizations

$$Q_k \begin{bmatrix} H_k & \beta_1 e_1 \end{bmatrix} = \begin{bmatrix} R_k & t_k \\ 0 & \phi_k \end{bmatrix},$$

(with R_k upper tridiagonal) and MINRES-QLP also computes the orthogonal factorizations $R_k P_k = L_k$ (with L_k lower tridiagonal), from which the kth solution estimate is defined by $W_k = V_k P_k$, $L_k u_k = t_k$, and $x_k^M = W_k u_k$. As shown in [14, §5.3], the construction of these quantities is such that the first $k-3$ columns of W_k are the same as in W_{k-1}, and the first $k-3$ elements of u_k are the same as in u_{k-1}. Since W_k has orthonormal columns, $\|x_k^M\| = \|u_k\|$, where the first $k-2$ elements of u_k are unaltered by later iterations. As shown in [14, §6.5], it means that certain quantities can be cheaply updated to give norm estimates in the form

$$\chi^2 \leftarrow \chi^2 + \hat{\mu}_{k-2}^2, \qquad \|x_k^M\|^2 = \chi^2 + \tilde{\mu}_{k-1}^2 + \bar{\mu}_k^2,$$

where it is clear that χ^2 increases monotonically. Although the last two terms are of unpredictable size, $\|x_k^M\|^2$ tends to be dominated by the monotonic term χ^2 and we can expect that $\|x_k^M\|$ will be *approximately* monotonic as k increases from 1 to ℓ.

Experimentally we find that for most MINRES iterations on an indefinite problem, $\|x_k\|$ does increase. To obtain indefinite examples that were sensibly scaled, we used the four spd (A, b) cases in Figures 2.4–2.5, applied diagonal scaling as before, and solved $(A - \delta I)x = b$ with $\delta = 0.5$ and where A and b are now scaled (so that $\text{diag}(A) = I$). The number of iterations increased significantly but was limited to n.

Figure 2.7 shows $\log_{10} \|r_k\|$ and $\|x_k\|$ for the first two cases (where A is the spd matrices in Figure 2.4). The values of $\|x_k\|$ are *essentially* monotonic. The backward errors $\|r_k\|/\|x_k\|$ (not shown) were even closer to being monotonic.

Figure 2.7 shows the values of $\|x_k\|$ for the first two cases (MINRES applied to $(A - \delta I)x = b$, where A is the spd matrices used in Figure 2.4 and $\delta = 0.5$ is large enough to make the systems indefinite). The number of iterations increased significantly but again was limited to n. These are typical examples in which $\|x_k\|$ is monotonic as in the spd case.

Figure 2.8 shows $\|x_k\|$ and $\log_{10} \|r_k\|$ for the second two cases (where A is the spd matrices in Figure 2.5). The left example reveals a definite period of decrease in $\|x_k\|$. Nevertheless, during the $n = 82654$ iterations, $\|x_k\|$ increased 83% of the time and the backward errors $\|r_k\|/\|x_k\|$ decreased 91% of the time. The right example is more like those in Figure 2.8. During $n = 245874$ iterations, $\|x_k\|$ increased 83% of the time, the backward errors $\|r_k\|/\|x_k\|$ decreased 91% of the time, and any nonmonotonicity was very slight.

2.4 Summary

Our experimental results here provide empirical evidence that MINRES can often stop much sooner than CG on spd systems when the stopping rule is based on backward error norms $\|r_k\|/\|x_k\|$ (or the more general norms in (2.8)–(2.9)). On the other hand, CG generates iterates x_k with smaller $\|x^* - x_k\|_A$ and $\|x^* - x_k\|$, and is recommended in applications

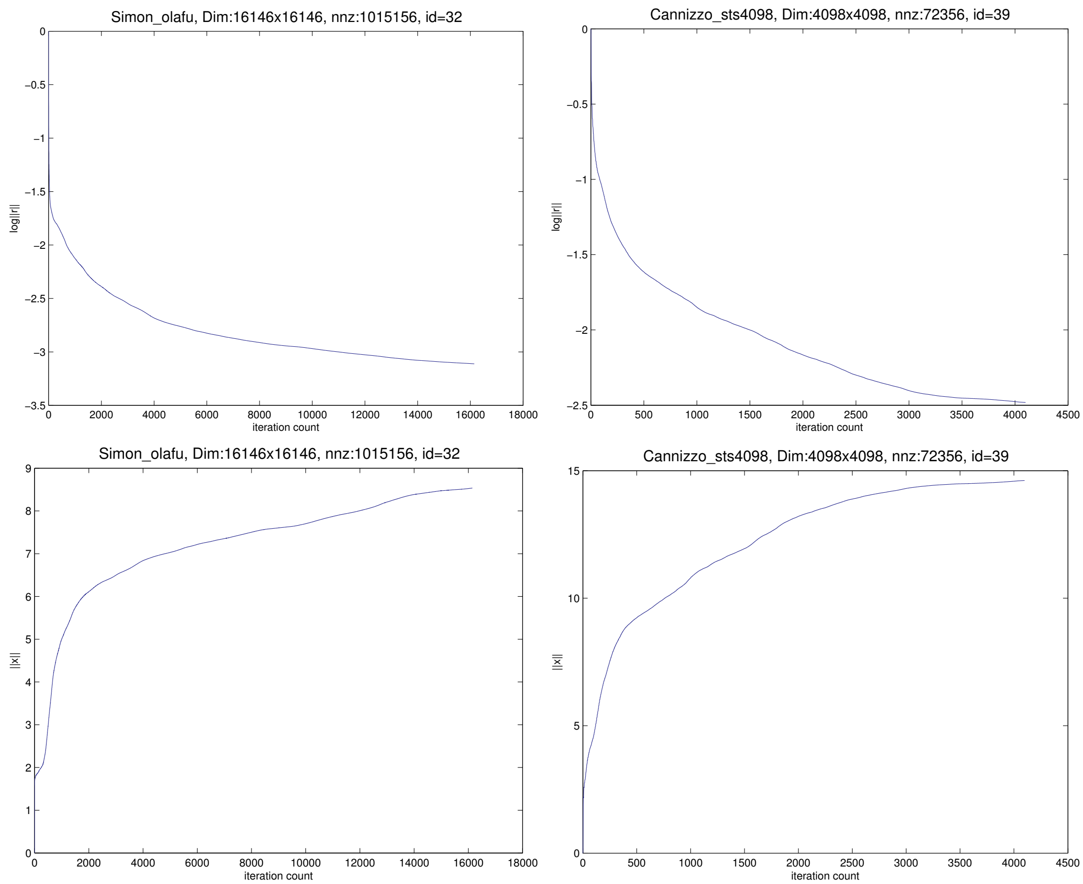


Figure 2.7: Residual norms and solution norms when MINRES is applied to two indefinite systems $(A - \delta I)x = b$, where A is the spd matrices used in Figure 2.4 ($n = 16146$ and 4098) and $\delta = 0.5$ is large enough to make the systems indefinite.
Top: The values of $\log_{10} \|r_k\|$ are plotted against iteration number k for the first n iterations.
Bottom left: The values of $\|x_k\|$ are plotted against k. During the $n = 16146$ iterations, $\|x_k\|$ increased 83% of the time and the backward errors $\|r_k\|/\|x_k\|$ (not shown here) decreased 96% of the time.
Bottom right: During the $n = 4098$ iterations, $\|x_k\|$ increased 90% of the time and the backward errors $\|r_k\|/\|x_k\|$ (not shown here) decreased 98% of the time.

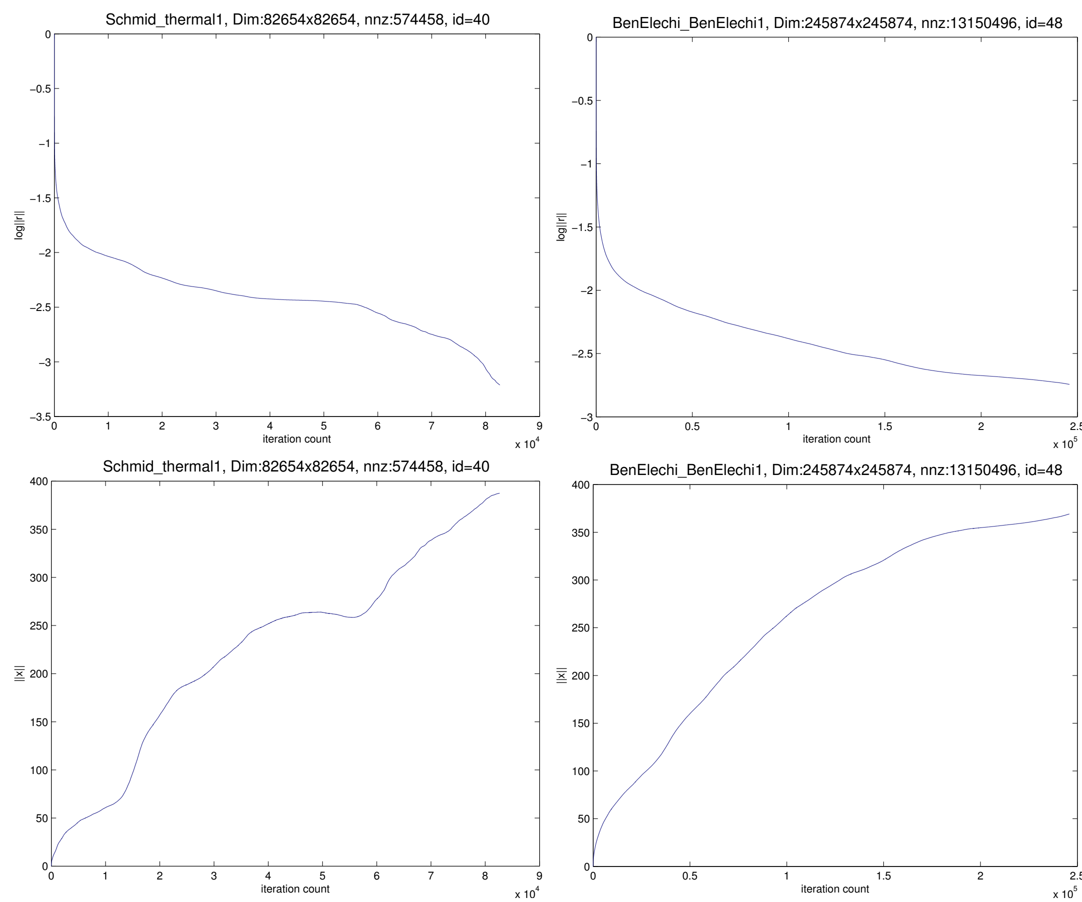


Figure 2.8: Residual norms and solution norms when MINRES is applied to two indefinite systems $(A-\delta I)x = b$, where A is the spd matrices used in Figure 2.5 ($n = 82654$ and 245874) and $\delta = 0.5$ is large enough to make the systems indefinite.
Top: The values of $\log_{10} \|r_k\|$ are plotted against iteration number k for the first n iterations.
Bottom left: The values of $\|x_k\|$ are plotted against k. There is a mild but clear decrease in $\|x_k\|$ over an interval of about 1000 iterations. During the $n = 82654$ iterations, $\|x_k\|$ increased 83% of the time and the backward errors $\|r_k\|/\|x_k\|$ (not shown here) decreased 91% of the time.
Bottom right: The solution norms and backward errors are essentially monotonic. During the $n = 245874$ iterations, $\|x_k\|$ increased 88% of the time and the backward errors $\|r_k\|/\|x_k\|$ (not shown here) decreased 95% of the time.

where these quantities should be minimized.

For full-rank least-squares problems $\min \|Ax - b\|$, the solver LSQR [53; 54] is equivalent to CG on the (spd) normal equation $A^TAx = A^Tb$. This suggests that a MINRES-based algorithm for least-squares may share the same advantage as MINRES for symmetric systems, especially in the case of early termination. LSMR is designed on such a basis. The derivation of LSMR is presented in Chapter 3. Numerical experiments in Chapter 4 show that LSMR has more good properties than our original expectation.

Theorem 2.1.5 shows that MINRES shares a known property of CG: that $\|x_k\|$ increases monotonically when A is spd. This implies that $\|x_k\|$ is monotonic for LSMR (as conjectured in [24]), and suggests that MINRES might be a useful alternative to CG in the context of trust-region methods for optimization.

3 LSMR

We present a numerical method called LSMR for computing a solution x to the following problems:

Unsymmetric equations:	minimize $\|x\|_2$ subject to $Ax = b$
Linear least squares (LS):	minimize $\|Ax - b\|_2$
Regularized least squares:	minimize $\left\| \begin{pmatrix} A \\ \lambda I \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|_2$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $\lambda \geq 0$, with $m \leq n$ or $m \geq n$. The matrix A is used as an operator for which products of the form Av and $A^T u$ can be computed for various v and u. (If A is symmetric or Hermitian and $\lambda = 0$, MINRES [52] or MINRES-QLP [14] are applicable.)

LSMR is similar in style to the well known method LSQR [53; 54] in being based on the Golub-Kahan bidiagonalization of A [29]. LSQR is equivalent to the conjugate-gradient (CG) method applied to the normal equation $(A^TA + \lambda^2 I)x = A^Tb$. It has the property of reducing $\|r_k\|$ monotonically, where $r_k = b - Ax_k$ is the residual for the approximate solution x_k. (For simplicity, we are letting $\lambda = 0$.) In contrast, LSMR is equivalent to MINRES applied to the normal equation, so that the quantities $\|A^T r_k\|$ are monotonically decreasing. We have also proved that $\|r_k\|$ is monotonically decreasing, and in practice it is never very far behind the corresponding value for LSQR. Hence, although LSQR and LSMR ultimately converge to similar points, it is safer to use LSMR in situations where the solver must be terminated early.

Stopping conditions are typically based on *backward error*: the norm of some perturbation to A for which the current iterate x_k solves the perturbed problem exactly. Experiments on many sparse LS test problems show that for LSMR, a certain *cheaply computable* backward error for each x_k is close to the *optimal* (smallest possible) backward error. This is an unexpected but highly desirable advantage.

Overview

Section 3.1 derives the basic LSMR algorithm with $\lambda = 0$. Section 3.2 derives various norms and stopping criteria. Section 3.3.2 discusses singular systems. Section 3.4 compares the complexity of LSQR and LSMR. Section 3.5 derives the LSMR algorithm with $\lambda \geq 0$. Section 3.6 proves one of the main lemmas.

3.1 Derivation of LSMR

We begin with the Golub-Kahan process Bidiag(A, b) [29], an iterative procedure for transforming $\begin{pmatrix} b & A \end{pmatrix}$ to upper-bidiagonal form $\begin{pmatrix} \beta_1 e_1 & B_k \end{pmatrix}$. The process was introduced in 1.4.1. Since it is central to the development of LSMR, we will restate it here in more detail.

3.1.1 The Golub-Kahan process

1. Set $\beta_1 u_1 = b$ (that is, $\beta_1 = \|b\|$, $u_1 = b/\beta_1$) and $\alpha_1 v_1 = A^T u_1$.
2. For $k = 1, 2, \ldots,$ set

$$\begin{aligned} \beta_{k+1} u_{k+1} &= A v_k - \alpha_k u_k \\ \alpha_{k+1} v_{k+1} &= A^T u_{k+1} - \beta_{k+1} v_k. \end{aligned} \tag{3.1}$$

After k steps, we have

$$A V_k = U_{k+1} B_k \qquad \text{and} \qquad A^T U_{k+1} = V_{k+1} L_{k+1}^T,$$

where we define $V_k = \begin{pmatrix} v_1 & v_2 & \ldots & v_k \end{pmatrix}$, $U_k = \begin{pmatrix} u_1 & u_2 & \ldots & u_k \end{pmatrix}$, and

$$B_k = \begin{pmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_k & \alpha_k \\ & & & \beta_{k+1} \end{pmatrix}, \qquad L_{k+1} = \begin{pmatrix} B_k & \alpha_{k+1} e_{k+1} \end{pmatrix}.$$

Now consider

$$\begin{aligned} A^T A V_k = A^T U_{k+1} B_k = V_{k+1} L_{k+1}^T B_k &= V_{k+1} \begin{pmatrix} B_k^T \\ \alpha_{k+1} e_{k+1}^T \end{pmatrix} B_k \\ &= V_{k+1} \begin{pmatrix} B_k^T B_k \\ \alpha_{k+1} \beta_{k+1} e_k^T \end{pmatrix}. \end{aligned} \tag{3.2}$$

This is equivalent to what would be generated by the symmetric Lanczos process with matrix A^TA and starting vector A^Tb,[1] and the columns of V_k lie in the Krylov subspace $\mathcal{K}_k(A^TA, A^Tb)$.

[1]For this reason we define $\bar{\beta}_k \equiv \alpha_k\beta_k$ below.

3.1.2 Using Golub-Kahan to solve the normal equation

Krylov subspace methods for solving linear equations form solution estimates $x_k = V_k y_k$ for some y_k, where the columns of V_k are an expanding set of theoretically independent vectors.[2]

[2]In this case, V_k and also U_k are theoretically orthonormal.

For the equation $A^TAx = A^Tb$, any solution x has the property of minimizing $\|r\|$, where $r = b - Ax$ is the corresponding residual vector. Thus, in the development of LSQR it was natural to choose y_k to minimize $\|r_k\|$ at each stage. Since

$$r_k = b - AV_k y_k = \beta_1 u_1 - U_{k+1}B_k y_k = U_{k+1}(\beta_1 e_1 - B_k y_k), \qquad (3.3)$$

where U_{k+1} is theoretically orthonormal, the subproblem $\min_{y_k} \|\beta_1 e_1 - B_k y_k\|$ easily arose. In contrast, for LSMR we wish to minimize $\|A^T r_k\|$. Let $\bar{\beta}_k \equiv \alpha_k\beta_k$ for all k. Since $A^T r_k = A^Tb - A^TAx_k = \beta_1\alpha_1 v_1 - A^TAV_k y_k$, from (3.2) we have

$$A^T r_k = \bar{\beta}_1 v_1 - V_{k+1}\begin{pmatrix} B_k^T B_k \\ \alpha_{k+1}\beta_{k+1}e_k^T \end{pmatrix} y_k = V_{k+1}\left(\bar{\beta}_1 e_1 - \begin{pmatrix} B_k^T B_k \\ \bar{\beta}_{k+1}e_k^T \end{pmatrix} y_k\right),$$

and we are led to the subproblem

$$\min_{y_k} \|A^T r_k\| = \min_{y_k} \left\| \bar{\beta}_1 e_1 - \begin{pmatrix} B_k^T B_k \\ \bar{\beta}_{k+1}e_k^T \end{pmatrix} y_k \right\|. \qquad (3.4)$$

Efficient solution of this LS subproblem is the heart of algorithm LSMR.

3.1.3 Two QR factorizations

As in LSQR, we form the QR factorization

$$Q_{k+1}B_k = \begin{pmatrix} R_k \\ 0 \end{pmatrix}, \qquad R_k = \begin{pmatrix} \rho_1 & \theta_2 & & \\ & \rho_2 & \ddots & \\ & & \ddots & \theta_k \\ & & & \rho_k \end{pmatrix}. \qquad (3.5)$$

If we define $t_k = R_k y_k$ and solve $R_k^T q_k = \bar{\beta}_{k+1} e_k$, we have $q_k = (\bar{\beta}_{k+1}/\rho_k) e_k = \varphi_k e_k$ with $\rho_k = (R_k)_{kk}$ and $\varphi_k \equiv \bar{\beta}_{k+1}/\rho_k$. Then we perform a second QR factorization

$$\bar{Q}_{k+1} \begin{pmatrix} R_k^T & \bar{\beta}_1 e_1 \\ \varphi_k e_k^T & 0 \end{pmatrix} = \begin{pmatrix} \bar{R}_k & z_k \\ 0 & \bar{\zeta}_{k+1} \end{pmatrix}, \qquad \bar{R}_k = \begin{pmatrix} \bar{\rho}_1 & \bar{\theta}_2 & & \\ & \bar{\rho}_2 & \ddots & \\ & & \ddots & \bar{\theta}_k \\ & & & \bar{\rho}_k \end{pmatrix}. \quad (3.6)$$

Combining what we have with (3.4) gives

$$\begin{aligned} \min_{y_k} \|A^T r_k\| = \min_{y_k} \left\| \bar{\beta}_1 e_1 - \begin{pmatrix} R_k^T R_k \\ q_k^T R_k \end{pmatrix} y_k \right\| &= \min_{t_k} \left\| \bar{\beta}_1 e_1 - \begin{pmatrix} R_k^T \\ \varphi_k e_k^T \end{pmatrix} t_k \right\| \\ &= \min_{t_k} \left\| \begin{pmatrix} z_k \\ \bar{\zeta}_{k+1} \end{pmatrix} - \begin{pmatrix} \bar{R}_k \\ 0 \end{pmatrix} t_k \right\|. \end{aligned} \quad (3.7)$$

The subproblem is solved by choosing t_k from $\bar{R}_k t_k = z_k$.[3]

[3] Since every element of t_k changes in each iteration, it is never constructed explicitly. Instead, the recurrences derived in the following sections are used.

3.1.4 Recurrence for x_k

Let W_k and $\bar{W}_k$ be computed by forward substitution from $R_k^T W_k^T = V_k^T$ and $\bar{R}_k^T \bar{W}_k^T = W_k^T$. Then from $x_k = V_k y_k$, $R_k y_k = t_k$, and $\bar{R}_k t_k = z_k$, we have $x_0 \equiv 0$ and

$$x_k = W_k R_k y_k = W_k t_k = \bar{W}_k \bar{R}_k t_k = \bar{W}_k z_k = x_{k-1} + \zeta_k \bar{w}_k.$$

3.1.5 Recurrence for W_k and $\bar{W}_k$

If we write

$$\begin{aligned} V_k &= \begin{pmatrix} v_1 & v_2 & \cdots & v_k \end{pmatrix}, & W_k &= \begin{pmatrix} w_1 & w_2 & \cdots & w_k \end{pmatrix}, \\ \bar{W}_k &= \begin{pmatrix} \bar{w}_1 & \bar{w}_2 & \cdots & \bar{w}_k \end{pmatrix}, & z_k &= \begin{pmatrix} \zeta_1 & \zeta_2 & \cdots & \zeta_k \end{pmatrix}^T, \end{aligned}$$

an important fact is that when k increases to $k+1$, all quantities remain the same except for one additional term.

The first QR factorization proceeds as follows. At iteration k we construct a plane rotation operating on rows l and $l+1$:

$$P_l = \begin{pmatrix} I_{l-1} & & & \\ & c_l & s_l & \\ & -s_l & c_l & \\ & & & I_{k-l} \end{pmatrix}.$$

Now if $Q_{k+1} = P_k \dots P_2 P_1$, we have

$$Q_{k+1}B_{k+1} = Q_{k+1}\begin{pmatrix} B_k & \alpha_{k+1}e_{k+1} \\ & \beta_{k+2} \end{pmatrix} = \begin{pmatrix} R_k & \theta_{k+1}e_k \\ 0 & \bar{\alpha}_{k+1} \\ & \beta_{k+2} \end{pmatrix},$$

$$Q_{k+2}B_{k+1} = P_{k+1}\begin{pmatrix} R_k & \theta_{k+1}e_k \\ 0 & \bar{\alpha}_{k+1} \\ & \beta_{k+2} \end{pmatrix} = \begin{pmatrix} R_k & \theta_{k+1}e_k \\ 0 & \rho_{k+1} \\ 0 & 0 \end{pmatrix},$$

and we see that $\theta_{k+1} = s_k\alpha_{k+1} = (\beta_{k+1}/\rho_k)\alpha_{k+1} = \bar{\beta}_{k+1}/\rho_k = \varphi_k$. Therefore we can write θ_{k+1} instead of φ_k.

For the second QR factorization, if $\bar{Q}_{k+1} = \bar{P}_k \dots \bar{P}_2 \bar{P}_1$ we know that

$$\bar{Q}_{k+1}\begin{pmatrix} R_k^T \\ \theta_{k+1}e_k^T \end{pmatrix} = \begin{pmatrix} \bar{R}_k \\ 0 \end{pmatrix},$$

and so

$$\bar{Q}_{k+2}\begin{pmatrix} R_{k+1}^T \\ \theta_{k+2}e_{k+1}^T \end{pmatrix} = \bar{P}_{k+1}\begin{pmatrix} \bar{R}_k & \bar{\theta}_{k+1}e_k \\ & \bar{c}_k\rho_{k+1} \\ & \theta_{k+2} \end{pmatrix} = \begin{pmatrix} \bar{R}_k & \bar{\theta}_{k+1}e_k \\ & \bar{\rho}_{k+1} \\ & 0 \end{pmatrix}. \tag{3.8}$$

By considering the last row of the matrix equation $R_{k+1}^T W_{k+1}^T = V_{k+1}^T$ and the last row of $\bar{R}_{k+1}^T \bar{W}_{k+1}^T = W_{k+1}^T$ we obtain equations that define w_{k+1} and $\bar{w}_{k+1}$:

$$\begin{aligned} \theta_{k+1}w_k^T + \rho_{k+1}w_{k+1}^T &= v_{k+1}^T, \\ \bar{\theta}_{k+1}\bar{w}_k^T + \bar{\rho}_{k+1}\bar{w}_{k+1}^T &= w_{k+1}^T. \end{aligned}$$

3.1.6 The two rotations

To summarize, the rotations P_k and $\bar{P}_k$ have the following effects on our computation:

$$\begin{pmatrix} c_k & s_k \\ -s_k & c_k \end{pmatrix}\begin{pmatrix} \bar{\alpha}_k & \\ \beta_{k+1} & \alpha_{k+1} \end{pmatrix} = \begin{pmatrix} \rho_k & \theta_{k+1} \\ 0 & \bar{\alpha}_{k+1} \end{pmatrix}$$

$$\begin{pmatrix} \bar{c}_k & \bar{s}_k \\ -\bar{s}_k & \bar{c}_k \end{pmatrix}\begin{pmatrix} \bar{c}_{k-1}\rho_k & & \bar{\zeta}_k \\ \theta_{k+1} & \rho_{k+1} & \end{pmatrix} = \begin{pmatrix} \bar{\rho}_k & \bar{\theta}_{k+1} & \zeta_k \\ 0 & \bar{c}_k\rho_{k+1} & \bar{\zeta}_{k+1} \end{pmatrix}.$$

Algorithm 3.1 Algorithm LSMR

1: (Initialize)

$$\beta_1 u_1 = b \quad \alpha_1 v_1 = A^T u_1 \quad \bar{\alpha}_1 = \alpha_1 \quad \bar{\zeta}_1 = \alpha_1 \beta_1 \quad \rho_0 = 1 \quad \bar{\rho}_0 = 1$$
$$\bar{c}_0 = 1 \quad \bar{s}_0 = 0 \quad h_1 = v_1 \quad \bar{h}_0 = 0 \quad x_0 = 0$$

2: **for** $k = 1, 2, 3 \ldots$ **do**

3: (Continue the bidiagonalization)

$$\beta_{k+1} u_{k+1} = A v_k - \alpha_k u_k, \qquad \alpha_{k+1} v_{k+1} = A^T u_{k+1} - \beta_{k+1} v_k$$

4: (Construct and apply rotation P_k)

$$\rho_k = \left(\bar{\alpha}_k^2 + \beta_{k+1}^2\right)^{\frac{1}{2}} \qquad c_k = \bar{\alpha}_k / \rho_k \qquad s_k = \beta_{k+1}/\rho_k \tag{3.9}$$
$$\theta_{k+1} = s_k \alpha_{k+1} \qquad \bar{\alpha}_{k+1} = c_k \alpha_{k+1} \tag{3.10}$$

5: (Construct and apply rotation $\bar{P}_k$)

$$\bar{\theta}_k = \bar{s}_{k-1} \rho_k \qquad \bar{\rho}_k = \left((\bar{c}_{k-1}\rho_k)^2 + \theta_{k+1}^2\right)^{\frac{1}{2}}$$
$$\bar{c}_k = \bar{c}_{k-1}\rho_k/\bar{\rho}_k \qquad \bar{s}_k = \theta_{k+1}/\bar{\rho}_k \tag{3.11}$$
$$\zeta_k = \bar{c}_k \bar{\zeta}_k \qquad \bar{\zeta}_{k+1} = -\bar{s}_k \bar{\zeta}_k \tag{3.12}$$

6: (Update h, $\bar{h}$ x)

$$\bar{h}_k = h_k - (\bar{\theta}_k \rho_k / (\rho_{k-1} \bar{\rho}_{k-1})) \bar{h}_{k-1}$$
$$x_k = x_{k-1} + (\zeta_k / (\rho_k \bar{\rho}_k)) \bar{h}_k$$
$$h_{k+1} = v_{k+1} - (\theta_{k+1}/\rho_k) h_k$$

7: **end for**

3.1.7 Speeding up forward substitution

The forward substitutions for computing w and $\bar{w}$ can be made more efficient if we define $h_k = \rho_k w_k$ and $\bar{h}_k = \rho_k \bar{\rho}_k \bar{w}_k$. We then obtain the updates described in part 6 of the pseudo-code below.

3.1.8 Algorithm LSMR

Algorithm 3.1 summarizes the main steps of LSMR for solving $Ax \approx b$, excluding the norms and stopping rules developed later.

3.2 Norms and stopping rules

For any numerical computation, it is impossible to obtain a result with less relative uncertainty than the input data. Thus, in solving linear systems with iterative methods, it is important to know when we have arrived at a solution with the best level of accuracy permitted, in order to terminate before we waste computational effort.

Sections 3.2.1 and 3.2.2 discusses various criteria used in stopping LSMR at the appropriate number of iterations. Sections 3.2.3, 3.2.4, 3.2.5 and 3.2.6 derive efficient ways to compute $\|r_k\|$, $\|A^T r_k\|$, $\|x_k\|$ and estimate $\|A\|$ and $\mathrm{cond}(A)$ for the stopping criteria to be implemented efficiently. All quantities require $O(1)$ computation each iteration.

3.2.1 Stopping criteria

With exact arithmetic, the Golub-Kahan process terminates when either $\alpha_{k+1} = 0$ or $\beta_{k+1} = 0$. For certain data b, this could happen in practice when k is small (but is unlikely later because of rounding error). We show that LSMR will have solved the problem at that point and should therefore terminate.

When $\alpha_{k+1} = 0$, with the expression of $\|A^T r_k\|$ from section 3.2.4, we have

$$\|A^T r_k\| = |\bar{\zeta}_{k+1}| = |\bar{s}_k \bar{\zeta}_k| = |\theta_{k+1} \bar{\rho}_k^{-1} \bar{\zeta}_k| = |s_k \alpha_{k+1} \bar{\rho}_k^{-1} \bar{\zeta}_k| = 0,$$

where (3.12), (3.11), (3.10) are used. Thus, a least-squares solution has been obtained.

When $\beta_{k+1} = 0$, we have

$$s_k = \beta_{k+1} \rho_k^{-1} = 0. \qquad \text{(from (3.9))} \tag{3.13}$$

$$\ddot{\beta}_{k+1} = -s_k \ddot{\beta}_k = 0. \qquad \text{(from (3.19), (3.13))} \tag{3.14}$$

$$\begin{aligned}
\dot{\beta}_k &= \tilde{c}_k^{-1} \left(\tilde{\beta}_k - \tilde{s}_k (-1)^k s^{(k)} c_{k+1} \beta_1 \right) && \text{(from (3.30))} \\
&= \tilde{c}_k^{-1} \tilde{\beta}_k && \text{(from (3.13))} \\
&= \dot{\rho}_k^{-1} \tilde{\rho}_k \tilde{\beta}_k && \text{(from (3.20))} \\
&= \dot{\rho}_k^{-1} \tilde{\rho}_k \tilde{\tau}_k && \text{(from Lemma 3.2.1)} \\
&= \dot{\tau}_k. && \text{(from (3.26), (3.27))}
\end{aligned} \tag{3.15}$$

By using (3.21) (derived in Section 3.2.3), we conclude that $\|r_k\| = 0$ from (3.15) and (3.14), so the system is consistent and $Ax_k = b$.

3.2.2 PRACTICAL STOPPING CRITERIA

For LSMR we use the same stopping rules as LSQR [53], involving dimensionless quantities ATOL, BTOL, CONLIM:

S1: Stop if $\|r_k\| \le \text{BTOL}\|b\| + \text{ATOL}\|A\|\|x_k\|$

S2: Stop if $\|A^T r_k\| \le \text{ATOL}\|A\|\|r_k\|$

S3: Stop if $\text{cond}(A) \ge \text{CONLIM}$

S1 applies to consistent systems, allowing for uncertainty in A and b [39, Theorem 7.1]. S2 applies to inconsistent systems and comes from Stewart's backward error estimate $\|E_2\|$ assuming uncertainty in A; see Section 4.1.1. S3 applies to any system.

3.2.3 COMPUTING $\|r_k\|$

We transform $\bar{R}_k^T$ to upper-bidiagonal form using a third QR factorization: $\widetilde{R}_k = \widetilde{Q}_k \bar{R}_k^T$ with $\widetilde{Q}_k = \widetilde{P}_{k-1} \dots \widetilde{P}_1$. This amounts to one additional rotation per iteration. Now let

$$\tilde{t}_k = \widetilde{Q}_k t_k, \qquad \tilde{b}_k = \begin{pmatrix} \widetilde{Q}_k & \\ & 1 \end{pmatrix} Q_{k+1} e_1 \beta_1. \tag{3.16}$$

Then from (3.3), $r_k = U_{k+1}(\beta_1 e_1 - B_k y_k)$ gives

$$\begin{aligned} r_k &= U_{k+1} \left(\beta_1 e_1 - Q_{k+1}^T \begin{pmatrix} R_k \\ 0 \end{pmatrix} y_k \right) \\ &= U_{k+1} \left(\beta_1 e_1 - Q_{k+1}^T \begin{pmatrix} t_k \\ 0 \end{pmatrix} \right) \\ &= U_{k+1} \left(Q_{k+1}^T \begin{pmatrix} \widetilde{Q}_k^T & \\ & 1 \end{pmatrix} \tilde{b}_k - Q_{k+1}^T \begin{pmatrix} \widetilde{Q}_k^T \tilde{t}_k \\ 0 \end{pmatrix} \right) \\ &= U_{k+1} Q_{k+1}^T \begin{pmatrix} \widetilde{Q}_k^T & \\ & 1 \end{pmatrix} \left(\tilde{b}_k - \begin{pmatrix} \tilde{t}_k \\ 0 \end{pmatrix} \right). \end{aligned}$$

Therefore, assuming orthogonality of U_{k+1}, we have

$$\|r_k\| = \left\| \tilde{b}_k - \begin{pmatrix} \tilde{t}_k \\ 0 \end{pmatrix} \right\|. \tag{3.17}$$

Algorithm 3.2 Computing $\|r_k\|$ in LSMR

1: (Initialize)

$$\ddot{\beta}_1 = \beta_1 \qquad \dot{\beta}_0 = 0 \qquad \dot{\rho}_0 = 1 \qquad \tilde{\tau}_{-1} = 0 \qquad \tilde{\theta}_0 = 0 \qquad \zeta_0 = 0$$

2: **for** the kth iteration **do**

3: (Apply rotation P_k)

$$\hat{\beta}_k = c_k \ddot{\beta}_k \qquad \ddot{\beta}_{k+1} = -s_k \ddot{\beta}_k \tag{3.19}$$

4: (If $k \geq 2$, construct and apply rotation $\widetilde{P}_{k-1}$)

$$\begin{aligned}
\tilde{\rho}_{k-1} &= \left(\dot{\rho}_{k-1}^2 + \bar{\theta}_k^2\right)^{\frac{1}{2}} & & \\
\tilde{c}_{k-1} &= \dot{\rho}_{k-1}/\tilde{\rho}_{k-1} & \tilde{s}_{k-1} &= \bar{\theta}_k/\tilde{\rho}_{k-1} \\
\tilde{\theta}_k &= \tilde{s}_{k-1}\bar{\rho}_k & \dot{\rho}_k &= \tilde{c}_{k-1}\bar{\rho}_k \\
\tilde{\beta}_{k-1} &= \tilde{c}_{k-1}\dot{\beta}_{k-1} + \tilde{s}_{k-1}\hat{\beta}_k & \dot{\beta}_k &= -\tilde{s}_{k-1}\dot{\beta}_{k-1} + \tilde{c}_{k-1}\hat{\beta}_k
\end{aligned} \tag{3.20}$$

5: (Update $\tilde{t}_k$ by forward substitution)

$$\tilde{\tau}_{k-1} = (\zeta_{k-1} - \tilde{\theta}_{k-1}\tilde{\tau}_{k-2})/\tilde{\rho}_{k-1} \qquad \dot{\tau}_k = (\zeta_k - \tilde{\theta}_k\tilde{\tau}_{k-1})/\dot{\rho}_k$$

6: (Form $\|r_k\|$)

$$\gamma = (\dot{\beta}_k - \dot{\tau}_k)^2 + \ddot{\beta}_{k+1}^2, \qquad \|r_k\| = \sqrt{\gamma} \tag{3.21}$$

7: **end for**

The vectors $\tilde{b}_k$ and $\tilde{t}_k$ can be written in the form

$$\begin{aligned}
\tilde{b}_k &= \begin{pmatrix} \tilde{\beta}_1 & \cdots & \tilde{\beta}_{k-1} & \dot{\beta}_k & \ddot{\beta}_{k+1} \end{pmatrix}^T \\
\tilde{t}_k &= \begin{pmatrix} \tilde{\tau}_1 & \cdots & \tilde{\tau}_{k-1} & \dot{\tau}_k \end{pmatrix}^T.
\end{aligned} \tag{3.18}$$

The vector $\tilde{t}_k$ is computed by forward substitution from $\widetilde{R}_k^T \tilde{t}_k = z_k$.

Lemma 3.2.1. *In* (3.17)–(3.18), $\tilde{\beta}_i = \tilde{\tau}_i$ *for* $i = 1, \ldots, k-1$.

Proof. Section 3.6 proves the lemma by induction. □

Using this lemma we can estimate $\|r_k\|$ from just the last two elements of $\tilde{b}_k$ and the last element of $\tilde{t}_k$, as shown in (3.21).

Algorithm 3.2 summarizes how $\|r_k\|$ may be obtained from quantities arising from the first and third QR factorizations.

3.2.4 Computing $\|A^T r_k\|$

From (3.7), we have $\|A^T r_k\| = |\bar{\zeta}_{k+1}|$.

3.2.5 Computing $\|x_k\|$

From Section 3.1.4 we have $x_k = V_k R_k^{-1} \bar{R}_k^{-1} z_k$. From the third QR factorization $\widetilde{Q}_k \bar{R}_k^T = \widetilde{R}_k$ in Section 3.2.3 and a fourth QR factorization $\hat{Q}_k (\widetilde{Q}_k R_k)^T = \hat{R}_k$ we can write

$$\begin{aligned} x_k &= V_k R_k^{-1} \bar{R}_k^{-1} z_k = V_k R_k^{-1} \bar{R}_k^{-1} \bar{R}_k \widetilde{Q}_k^T \tilde{z}_k \\ &= V_k R_k^{-1} \widetilde{Q}_k^T \widetilde{Q}_k R_k \hat{Q}_k^T \hat{z}_k = V_k \hat{Q}_k^T \hat{z}_k, \end{aligned}$$

where $\tilde{z}_k$ and $\hat{z}_k$ are defined by forward substitutions $\widetilde{R}_k^T \tilde{z}_k = z_k$ and $\hat{R}_k^T \hat{z}_k = \tilde{z}_k$. Assuming orthogonality of V_k we arrive at the estimate $\|x_k\| = \|\hat{z}_k\|$. Since only the last diagonal of $\widetilde{R}_k$ and the bottom 2×2 part of $\hat{R}_k$ change each iteration, this estimate of $\|x_k\|$ can again be updated cheaply. The pseudo-code, omitted here, can be derived as in Section 3.2.3.

3.2.6 Estimates of $\|A\|$ and cond(A)

It is known that the singular values of B_k are interlaced by those of A and are bounded above and below by the largest and smallest nonzero singular values of A [53]. Therefore we can estimate $\|A\|$ and $\mathrm{cond}(A)$ by $\|B_k\|$ and $\mathrm{cond}(B_k)$ respectively. Considering the Frobenius norm of B_k, we have the recurrence relation

$$\|B_{k+1}\|_F^2 = \|B_k\|_F^2 + \alpha_k^2 + \beta_{k+1}^2.$$

From (3.5)–(3.6) and (3.8), we can show that the following QLP factorization [71] holds:

$$Q_{k+1} B_k \bar{Q}_k^T = \begin{pmatrix} \bar{R}_{k-1}^T & \\ \bar{\theta}_k e_{k-1}^T & \bar{c}_{k-1} \rho_k \end{pmatrix}$$

(the same as $\bar{R}_k^T$ except for the last diagonal). Since the singular values of B_k are approximated by the diagonal elements of that lower-bidiagonal matrix [71], and since the diagonals are all positive, we can estimate $\mathrm{cond}(A)$ by the ratio of the largest and smallest values in $\{\bar{\rho}_1, \ldots, \bar{\rho}_{k-1}, \bar{c}_{k-1}\rho_k\}$. Those values can be updated cheaply.

3.3 LSMR Properties

With x^* denoting the pseudoinverse solution of $\min \|Ax - b\|$, we have the following theorems on the norms of various quantities for LSMR.

3.3.1 Monotonicity of norms

A number of monotonic properties for LSQR follow directly from the corresponding properties for CG in Section 2.1.1. We list them here from completeness.

Theorem 3.3.1. *$\|x_k\|$ is strictly increasing for LSQR.*

Proof. LSQR on $\min \|Ax - b\|$ is equivalent to CG on $A^T Ax = A^T b$. By Theorem 2.1.1, $\|x_k\|$ is strictly increasing. □

Theorem 3.3.2. *$\|x^* - x_k\|$ is strictly decreasing for LSQR.*

Proof. This follows from Theorem 2.1.2 for CG. □

Theorem 3.3.3. *$\|x^* - x_k\|_{A^T A} = \|A(x^* - x_k)\| = \|r^* - r_k\|$ is strictly decreasing for LSQR.*

Proof. This follows from Theorem 2.1.3 for CG. □

We also have the characterizing property for LSQR [53].

Theorem 3.3.4. *$\|r_k\|$ is strictly decreasing for LSQR.*

Next, we prove a number of monotonic properties for LSMR. We would like to emphasize that LSMR has all the above monotonic properties that LSQR enjoys. In addition, $\|A^T r_k\|$ is monotonic for LSMR. This gives LSMR a much smoother convergence behavior in terms of the Stewart backward error, as shown in Figure 4.2.

Theorem 3.3.5. *$\|A^T r_k\|$ is monotonically decreasing for LSMR.*

Proof. From Section 3.2.4 and (3.12), $\|A^T r_k\| = |\bar{\zeta}_{k+1}| = |\bar{s}_k||\bar{\zeta}_k| \le |\bar{\zeta}_k| = \|A^T r_{k-1}\|$. □

Theorem 3.3.6. *$\|x_k\|$ is strictly increasing for LSMR on $\min \|Ax - b\|$ when A has full column rank.*

Proof. LSMR on $\min \|Ax - b\|$ is equivalent to MINRES on $A^T Ax = A^T b$. When A has full column rank, $A^T A$ is symmetric positive definite. By Theorem 2.1.6, $\|x_k\|$ is strictly increasing. □

Algorithm 3.3 Algorithm CRLS

1: $x_0 = 0$, $\bar{r}_0 = A^T b$, $s_0 = A^T A \bar{r}_0$, $\rho_0 = \bar{r}_0^T s_0$, $p_0 = \bar{r}_0$, $q_0 = s_0$
2: **for** $k = 1, 2, \ldots$ **do**
3: $\quad \alpha_k = \rho_{k-1}/\|q_{k-1}\|^2$
4: $\quad x_k = x_{k-1} + \alpha_k p_{k-1}$
5: $\quad \bar{r}_k = \bar{r}_{k-1} - \alpha_k q_{k-1}$
6: $\quad s_k = A^T A \bar{r}_k$
7: $\quad \rho_k = \bar{r}_k^T s_k$
8: $\quad \beta_k = \rho_k/\rho_{k-1}$
9: $\quad p_k = \bar{r}_k + \beta_k p_{k-1}$
10: $\quad q_k = s_k + \beta_k q_{k-1}$
11: **end for**

Theorem 3.3.7. *The error $\|x^* - x_k\|$ is strictly decreasing for LSMR on $\min \|Ax - b\|$ when A has full column rank.*

Proof. This follows directly from Theorem 2.1.7 for MINRES. □

Theorem 3.3.8. *$\|x^* - x_k\|_{A^T A} = \|r^* - r_k\|$ is strictly decreasing for LSMR on $\min \|Ax - b\|$ when A has full column rank.*

Proof. This follows directly from Theorem 2.1.8 for MINRES. □

Since LSMR is equivalent to MINRES on the normal equation, and CR is a non-Lanczos equivalent of MINRES, we can apply CR to the normal equation to derive a non-Lanczos equivalent of LSMR, which we will call CRLS. We start from CR (Algorithm 1.3) and apply the substitutions $A \to A^T A$, $b \to A^T b$. Since r_k in CR would correspond to $A^T r_k$ in CRLS, we rename r_k to $\bar{r}_k$ in the algorithm to avoid confusion. With these substitutions, we arrive at Algorithm 3.3.

In CRLS, the residual $r_k = b - Ax_k$ is not computed. However, we know that $r_0 = b_0$ and that r_k satisfies the recurrence relation:

$$r_k = b - Ax_k = b - Ax_{k-1} - \alpha_k A p_{k-1} = r_{k-1} - \alpha_k A p_{k-1}. \tag{3.22}$$

Also, as mentioned, we define

$$\bar{r}_k = A^T r_k. \tag{3.23}$$

Let ℓ denote the iteration at which LSMR terminates; i.e. $A^T r_\ell = \bar{r}_\ell = 0$ and $x_\ell = x^*$. Then Theorem 2.1.4 for CR translates to the following.

Theorem 3.3.9. *These properties hold for Algorithm* CRLS*:*

(a) $q_i^T q_j = 0 \qquad (0 \le i, j \le \ell - 1,\ i \ne j)$

(b) $r_i^T q_j = 0 \qquad (0 \le i, j \le \ell - 1,\ i \ge j + 1)$

(c) $\bar{r}_i \ne 0 \Rightarrow p_i \ne 0 \qquad (0 \le i \le \ell - 1)$

Also, Theorem 2.1.5 for CR translates to the following.

Theorem 3.3.10. *These properties hold for Algorithm* CRLS *on a least-squares system* $\min \|Ax - b\|$ *when* A *has full column rank:*

(a) $\alpha_i > 0 \qquad (i = 1, \ldots, \ell)$

(b) $\beta_i > 0 \qquad (i = 1, \ldots, \ell - 1)$
$\beta_\ell = 0$

(c) $p_i^T q_j > 0 \qquad (0 \le i, j \le \ell - 1)$

(d) $p_i^T p_j > 0 \qquad (0 \le i, j \le \ell - 1)$

(e) $x_i^T p_j > 0 \qquad (1 \le i \le \ell,\ 0 \le j \le \ell - 1)$

(f) $\bar{r}_i^T p_j > 0 \qquad (0 \le i \le \ell - 1,\ 0 \le j \le \ell - 1)$

Theorem 3.3.11. *For* CRLS *(and hence* LSMR*) on* $\min \|Ax - b\|$ *when* A *has full column rank,* $\|r_k\|$ *is strictly decreasing.*

Proof.

$$
\begin{aligned}
\|r_{k-1}\|^2 - \|r_k\|^2 &= r_{k-1}^T r_{k-1} - r_k^T r_k \\
&= 2\alpha_k r_k^T A p_{k-1} + \alpha_k^2 p_{k-1}^T A^T A p_{k-1} && \text{by (3.22)} \\
&= 2\alpha_k (A^T r_k)^T p_{k-1} + \alpha_k^2 \|A p_{k-1}\|^2 \\
&> 2\alpha_k (A^T r_k)^T p_{k-1} && \text{by Thm 3.3.9 (c)} \\
& && \text{and Thm 3.3.10 (a)} \\
&= 2\alpha_k \bar{r}_k^T p_{k-1} && \text{by (3.23)} \\
&\ge 0,
\end{aligned}
$$

where the last inequality follows from Theorem 3.3.10 (a) and (f), and is strict except at $k = \ell$. The strict inequality on line 4 requires the fact that A has full column rank. Therefore we have $\|r_{k-1}\| > \|r_k\|$. □

3.3.2 Characteristics of the solution on singular systems

The least-squares problem $\min \|Ax - b\|$ has a unique solution when A has full column rank. If A does not have full column rank, infinitely many points x give the minimum value of $\|Ax - b\|$. In particular, the normal equation $A^TAx = A^Tb$ is singular but consistent. We show that LSQR and LSMR both give the minimum-norm LS solution. That is, they both solve the optimization problem $\min \|x\|_2$ such that $A^TAx = A^Tb$. Let $\mathrm{N}(A)$ and $\mathrm{R}(A)$ denote the nullspace and range of a matrix A.

Lemma 3.3.12. *If $A \in \mathbb{R}^{m\times n}$ and $p \in \mathbb{R}^n$ satisfy $A^TAp = 0$, then $p \in N(A)$.*

Proof. $A^TAp = 0 \Rightarrow p^TA^TAp = 0 \Rightarrow (Ap)^TAp = 0 \Rightarrow Ap = 0.$ □

Theorem 3.3.13. *LSQR returns the minimum-norm solution.*

Proof. The final LSQR solution satisfies $A^TAx_\ell^{\mathrm{LSQR}} = A^Tb$, and any other solution $\widehat{x}$ satisfies $A^TA\widehat{x} = A^Tb$. With $p = \widehat{x} - x_\ell^{\mathrm{LSQR}}$, the difference between the two normal equations gives $A^TAp = 0$, so that $Ap = 0$ by Lemma 3.3.12. From $\alpha_1 v_1 = A^Tu_1$ and $\alpha_{k+1}v_{k+1} = A^Tu_{k+1} - \beta_{k+1}v_k$ (3.1), we have $v_1, \ldots, v_\ell \in \mathrm{R}(A^T)$. With $Ap = 0$, this implies $p^TV_\ell = 0$, so that

$$\begin{aligned}\|\widehat{x}\|_2^2 - \|x_\ell^{\mathrm{LSQR}}\|_2^2 = \|x_\ell^{\mathrm{LSQR}} + p\|_2^2 - \|x_\ell^{\mathrm{LSQR}}\|_2^2 &= p^Tp + 2p^Tx_\ell^{\mathrm{LSQR}} \\ &= p^Tp + 2p^TV_\ell y_\ell^{\mathrm{LSQR}} \\ &= p^Tp \geq 0.\end{aligned}$$

□

Corollary 3.3.14. *LSMR returns the minimum-norm solution.*

Proof. At convergence, $\alpha_{\ell+1} = 0$ or $\beta_{\ell+1} = 0$. Thus $\bar{\beta}_{\ell+1} = \alpha_{\ell+1}\beta_{\ell+1} = 0$, which means equation (3.4) becomes $\min \|\bar{\beta}_1 e_1 - B_\ell^TB_\ell y_\ell\|$ and hence $B_\ell^TB_\ell y_\ell = \bar{\beta}_1 e_1$, since B_ℓ has full rank. This is the normal equation for $\min \|B_\ell y_\ell - \beta_1 e_1\|$, the same LS subproblem solved by LSQR. We conclude that at convergence, $y_\ell = y_\ell^{\mathrm{LSQR}}$ and thus $x_\ell = V_\ell y_\ell = V_\ell y_\ell^{\mathrm{LSQR}} = x_\ell^{\mathrm{LSQR}}$, and Theorem 3.3.13 applies. □

3.3.3 BACKWARD ERROR

For completeness, we state a final desirable result about LSMR. The Stewart backward error $\|A^Tr_k\|/\|r_k\|$ for LSMR is always less than or equal to that for LSQR. See Chapter 4, Theorem 4.1.1 for details.

Table 3.1 Storage and computational cost for various least-squares methods

	Storage		Work	
	m	n	m	n
LSMR	$p = Av, u$	$x, v = A^T u, h, \bar{h}$	3	6
LSQR	$p = Av, u$	$x, v = A^T u, w$	3	5
MINRES on $A^T Ax = A^T b$	$p = Av$	$x, v_1, v_2 = A^T p, w_1, w_2, w_3$		8

3.4 Complexity

We compare the storage requirement and computational complexity for LSMR and LSQR on $Ax \approx b$ and MINRES on the normal equation $A^T Ax = A^T b$. In Table 3.1, we list the vector storage needed (excluding storage for A and b). Recall that A is $m \times n$ and for LS systems m may be considerably larger than n. Av denotes the working storage for matrix-vector products. Work represents the number of floating-point multiplications required at each iteration.

From Table 3.1, we see that LSMR requires storage of one extra vector, and also n more scalar floating point multiplication when compared to LSQR. This difference is negligible compared to the cost of performing Av and $A^T u$ multiplication for most problems. Thus, the computational and storage complexity of LSMR is comparable to LSQR.

3.5 Regularized least squares

In this section we extend LSMR to the regularized LS problem

$$\min \left\| \begin{pmatrix} A \\ \lambda I \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|_2, \tag{3.24}$$

where λ is a nonnegative scalar. If $\bar{A} = \left(\begin{smallmatrix} A \\ \lambda I \end{smallmatrix}\right)$ and $\bar{r}_k = \left(\begin{smallmatrix} b \\ 0 \end{smallmatrix}\right) - \bar{A} x_k$, then

$$\begin{aligned} \bar{A}^T \bar{r}_k = A^T r_k - \lambda^2 x_k &= V_{k+1} \left(\bar{\beta}_1 e_1 - \begin{pmatrix} B_k^T B_k \\ \bar{\beta}_{k+1} e_k^T \end{pmatrix} y_k - \lambda^2 \begin{pmatrix} y_k \\ 0 \end{pmatrix} \right) \\ &= V_{k+1} \left(\bar{\beta}_1 e_1 - \begin{pmatrix} R_k^T R_k \\ \bar{\beta}_{k+1} e_k^T \end{pmatrix} y_k \right) \end{aligned}$$

and the rest of the main algorithm follows the same as in the unregularized case. In the last equality, R_k is defined by the QR factorization

$$Q_{2k+1}\begin{pmatrix} B_k \\ \lambda I \end{pmatrix} = \begin{pmatrix} R_k \\ 0 \end{pmatrix}, \qquad Q_{2k+1} \equiv P_k\hat{P}_k \dots P_2\hat{P}_2P_1\hat{P}_1,$$

where $\hat{P}_l$ is a rotation operating on rows l and $l+k+1$. The effects of $\hat{P}_1$ and P_1 are illustrated here:

$$\hat{P}_1\begin{pmatrix} \alpha_1 & \\ \beta_2 & \alpha_2 \\ & \beta_3 \\ \lambda & \\ & \lambda \end{pmatrix} = \begin{pmatrix} \hat{\alpha}_1 & \\ \beta_2 & \alpha_2 \\ & \beta_3 \\ 0 & \\ & \lambda \end{pmatrix}, \qquad P_1\begin{pmatrix} \hat{\alpha}_1 & \\ \beta_2 & \alpha_2 \\ & \beta_3 \\ & \\ & \lambda \end{pmatrix} = \begin{pmatrix} \rho_1 & \theta_2 \\ & \bar{\alpha}_2 \\ & \beta_3 \\ & \\ & \lambda \end{pmatrix}.$$

3.5.1 Effects on $\|\bar{r}_k\|$

Introduction of regularization changes the residual norm as follows:

$$\begin{aligned}
\bar{r}_k = \begin{pmatrix} b \\ 0 \end{pmatrix} - \begin{pmatrix} A \\ \lambda I \end{pmatrix} x_k &= \begin{pmatrix} u_1 \\ 0 \end{pmatrix}\beta_1 - \begin{pmatrix} AV_k \\ \lambda V_k \end{pmatrix} y_k \\
&= \begin{pmatrix} u_1 \\ 0 \end{pmatrix}\beta_1 - \begin{pmatrix} U_{k+1}B_k \\ \lambda V_k \end{pmatrix} y_k \\
&= \begin{pmatrix} U_{k+1} & \\ & V_k \end{pmatrix}\left(e_1\beta_1 - \begin{pmatrix} B_k \\ \lambda I \end{pmatrix} y_k\right) \\
&= \begin{pmatrix} U_{k+1} & \\ & V_k \end{pmatrix}\left(e_1\beta_1 - Q_{2k+1}^T\begin{pmatrix} R_k \\ 0 \end{pmatrix} y_k\right) \\
&= \begin{pmatrix} U_{k+1} & \\ & V_k \end{pmatrix}\left(e_1\beta_1 - Q_{2k+1}^T\begin{pmatrix} t_k \\ 0 \end{pmatrix}\right) \\
&= \begin{pmatrix} U_{k+1} & \\ & V_k \end{pmatrix} Q_{2k+1}^T\begin{pmatrix} \widetilde{Q}_k^T & \\ & 1 \end{pmatrix}\left(\tilde{b}_k - \begin{pmatrix} \tilde{t}_k \\ 0 \end{pmatrix}\right)
\end{aligned}$$

with $\tilde{b}_k = \begin{pmatrix} \widetilde{Q}_k & \\ & 1 \end{pmatrix} Q_{2k+1}e_1\beta_1$, where we adopt the notation

$$\tilde{b}_k = \begin{pmatrix} \tilde{\beta}_1 & \cdots & \tilde{\beta}_{k-1} & \dot{\beta}_k & \ddot{\beta}_{k+1} & \check{\beta}_1 & \cdots & \check{\beta}_k \end{pmatrix}^T.$$

We conclude that

$$\|\bar{r}_k\|^2 = \check{\beta}_1^2 + \cdots + \check{\beta}_k^2 + (\dot{\beta}_k - \tau_k)^2 + \ddot{\beta}_{k+1}^2.$$

The effect of regularization on the rotations is summarized as

$$\begin{pmatrix} \hat{c}_k & \hat{s}_k \\ -\hat{s}_k & \hat{c}_k \end{pmatrix} \begin{pmatrix} \bar{\alpha}_k & \ddot{\beta}_k \\ \lambda & \end{pmatrix} = \begin{pmatrix} \hat{\alpha}_k & \acute{\beta}_k \\ & \check{\beta}_k \end{pmatrix}$$

$$\begin{pmatrix} c_k & s_k \\ -s_k & c_k \end{pmatrix} \begin{pmatrix} \hat{\alpha}_k & & \acute{\beta}_k \\ \beta_{k+1} & \alpha_{k+1} & \end{pmatrix} = \begin{pmatrix} \rho_k & \theta_{k+1} & \hat{\beta}_k \\ & \bar{\alpha}_{k+1} & \ddot{\beta}_{k+1} \end{pmatrix}.$$

3.5.2 Pseudo-code for regularized LSMR

Algorithm 3.4 summarizes LSMR for solving the regularized problem (3.24) with given λ. Our MATLAB implementation is based on these steps.

3.6 Proof of Lemma 3.2.1

The effects of the rotations P_k and $\widetilde{P}_{k-1}$ can be summarized as

$$\widetilde{R}_k = \begin{pmatrix} \tilde{\rho}_1 & \tilde{\theta}_2 & & \\ & \ddots & \ddots & \\ & & \tilde{\rho}_{k-1} & \tilde{\theta}_k \\ & & & \dot{\rho}_k \end{pmatrix},$$

$$\begin{pmatrix} c_k & s_k \\ -s_k & c_k \end{pmatrix} \begin{pmatrix} \ddot{\beta}_k \\ 0 \end{pmatrix} = \begin{pmatrix} \hat{\beta}_k \\ \ddot{\beta}_{k+1} \end{pmatrix},$$

$$\begin{pmatrix} \tilde{c}_k & \tilde{s}_k \\ -\tilde{s}_k & \tilde{c}_k \end{pmatrix} \begin{pmatrix} \dot{\rho}_{k-1} & & \dot{\beta}_{k-1} \\ \bar{\theta}_k & \bar{\rho}_k & \hat{\beta}_k \end{pmatrix} = \begin{pmatrix} \tilde{\rho}_{k-1} & \tilde{\theta}_k & \tilde{\beta}_{k-1} \\ 0 & \dot{\rho}_k & \dot{\beta}_k \end{pmatrix},$$

where $\ddot{\beta}_1 = \beta_1$, $\dot{\rho}_1 = \bar{\rho}_1$, $\dot{\beta}_1 = \hat{\beta}_1$ and where c_k, s_k are defined in section 3.1.6.

We define $s^{(k)} = s_1 \dots s_k$ and $\bar{s}^{(k)} = \bar{s}_1 \dots \bar{s}_k$. Then from (3.18) and (3.6) we have $\widetilde{R}_k^T \tilde{t}_k = z_k = \begin{pmatrix} I_k & 0 \end{pmatrix} \bar{Q}_{k+1} e_{k+1} \bar{\beta}_1$. Expanding this and (3.16) gives

$$\widetilde{R}_k^T \tilde{t}_k = \begin{pmatrix} \bar{c}_1 \\ -\bar{s}_1 \bar{c}_2 \\ \vdots \\ (-1)^{k+1} \bar{s}^{(k-1)} \bar{c}_k \end{pmatrix} \bar{\beta}_1, \qquad \tilde{b}_k = \begin{pmatrix} \widetilde{Q}_k & \\ & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ -s_1 c_2 \\ \vdots \\ (-1)^{k+1} s^{(k-1)} c_k \\ (-1)^{k+2} s^{(k)} \end{pmatrix} \beta_1,$$

Algorithm 3.4 Regularized LSMR (1)

(Initialize)

$$
\begin{array}{cccccc}
\beta_1 u_1 = b & \alpha_1 v_1 = A^T u_1 & \bar{\alpha}_1 = \alpha_1 & \bar{\zeta}_1 = \alpha_1 \beta_1 & \rho_0 = 1 & \bar{\rho}_0 = 1 \\
\bar{c}_0 = 1 & \bar{s}_0 = 0 & \ddot{\beta}_1 = \beta_1 & \dot{\beta}_0 = 0 & \dot{\rho}_0 = 1 & \tilde{\tau}_{-1} = 0 \\
\tilde{\theta}_0 = 0 & \zeta_0 = 0 & d_0 = 0 & h_1 = v_1 & \bar{h}_0 = 0 & x_0 = 0
\end{array}
$$

for $k = 1, 2, 3, \dots$ **do**

(Continue the bidiagonalization)

$$
\beta_{k+1} u_{k+1} = A v_k - \alpha_k u_k \qquad \alpha_{k+1} v_{k+1} = A^T u_{k+1} - \beta_{k+1} v_k
$$

(Construct rotation $\hat{P}_k$)

$$
\hat{\alpha}_k = \left(\bar{\alpha}_k^2 + \lambda^2\right)^{\frac{1}{2}} \qquad \hat{c}_k = \bar{\alpha}_k / \hat{\alpha}_k \qquad \hat{s}_k = \lambda / \hat{\alpha}_k
$$

(Construct and apply rotation P_k)

$$
\begin{array}{lll}
\rho_k = \left(\hat{\alpha}_k^2 + \beta_{k+1}^2\right)^{\frac{1}{2}} & c_k = \hat{\alpha}_k / \rho_k & s_k = \beta_{k+1} / \rho_k \\
\theta_{k+1} = s_k \alpha_{k+1} & \bar{\alpha}_{k+1} = c_k \alpha_{k+1} &
\end{array}
$$

(Construct and apply rotation $\bar{P}_k$)

$$
\begin{array}{ll}
\bar{\theta}_k = \bar{s}_{k-1} \rho_k & \bar{\rho}_k = \left((\bar{c}_{k-1} \rho_k)^2 + \theta_{k+1}^2\right)^{\frac{1}{2}} \\
\bar{c}_k = \bar{c}_{k-1} \rho_k / \bar{\rho}_k & \bar{s}_k = \theta_{k+1} / \bar{\rho}_k \\
\zeta_k = \bar{c}_k \bar{\zeta}_k & \bar{\zeta}_{k+1} = -\bar{s}_k \bar{\zeta}_k
\end{array}
$$

(Update $\bar{h}$, x, h)

$$
\begin{aligned}
\bar{h}_k &= h_k - (\bar{\theta}_k \rho_k / (\rho_{k-1} \bar{\rho}_{k-1})) \bar{h}_{k-1} \\
x_k &= x_{k-1} + (\zeta_k / (\rho_k \bar{\rho}_k)) \bar{h}_k \\
h_{k+1} &= v_{k+1} - (\theta_{k+1} / \rho_k) h_k
\end{aligned}
$$

(Apply rotation $\hat{P}_k, P_k$)

$$
\acute{\beta}_k = \hat{c}_k \ddot{\beta}_k \qquad \check{\beta}_k = -\hat{s}_k \ddot{\beta}_k \qquad \hat{\beta}_k = c_k \acute{\beta}_k \qquad \ddot{\beta}_{k+1} = -s_k \acute{\beta}_k
$$

if $k \geq 2$ **then**

(Construct and apply rotation $\widetilde{P}_{k-1}$)

$$
\begin{array}{ll}
\tilde{\rho}_{k-1} = \left(\dot{\rho}_{k-1}^2 + \bar{\theta}_k^2\right)^{\frac{1}{2}} & \\
\tilde{c}_{k-1} = \dot{\rho}_{k-1} / \tilde{\rho}_{k-1} & \tilde{s}_{k-1} = \bar{\theta}_k / \tilde{\rho}_{k-1} \\
\tilde{\theta}_k = \tilde{s}_{k-1} \bar{\rho}_k & \dot{\rho}_k = \tilde{c}_{k-1} \bar{\rho}_k \\
\tilde{\beta}_{k-1} = \tilde{c}_{k-1} \dot{\beta}_{k-1} + \tilde{s}_{k-1} \hat{\beta}_k & \dot{\beta}_k = -\tilde{s}_{k-1} \dot{\beta}_{k-1} + \tilde{c}_{k-1} \hat{\beta}_k
\end{array}
$$

end if

Algorithm 3.5 Regularized LSMR (2)

12: (Update $\tilde{t}_k$ by forward substitution)

$$\tilde{\tau}_{k-1} = (\zeta_{k-1} - \tilde{\theta}_{k-1}\tilde{\tau}_{k-2})/\tilde{\rho}_{k-1} \qquad \dot{\tau}_k = (\zeta_k - \tilde{\theta}_k\tilde{\tau}_{k-1})/\dot{\rho}_k$$

13: (Compute $\|\bar{r}_k\|$)

$$d_k = d_{k-1} + \check{\beta}_k^2 \qquad \gamma = d_k + (\dot{\beta}_k - \dot{\tau}_k)^2 + \ddot{\beta}_{k+1}^2 \qquad \|\bar{r}_k\| = \sqrt{\gamma}$$

14: (Compute $\|\bar{A}^T\bar{r}_k\|$, $\|x_k\|$, estimate $\|\bar{A}\|$, cond$(\bar{A})$)

$\|\bar{A}^T\bar{r}_k\| = |\bar{\zeta}_{k+1}|$ (section 3.2.4)
Compute $\|x_k\|$ (section 3.2.5)
Estimate $\sigma_{\max}(B_k), \sigma_{\min}(B_k)$ and hence $\|\bar{A}\|, \mathrm{cond}(\bar{A})$ (section 3.2.6)

15: Terminate if any of the stopping criteria in Section 3.2.2 are satisfied.
16: **end for**

and we see that

$$\tilde{\tau}_1 = \tilde{\rho}_1^{-1}\bar{c}_1\bar{\beta}_1 \tag{3.25}$$
$$\tilde{\tau}_{k-1} = \tilde{\rho}_{k-1}^{-1}((-1)^k\bar{s}^{(k-2)}\bar{c}_{k-1}\bar{\beta}_1 - \tilde{\theta}_{k-1}\tilde{\tau}_{k-2}) \tag{3.26}$$
$$\dot{\tau}_k = \dot{\rho}_k^{-1}((-1)^{k+1}\bar{s}^{(k-1)}\bar{c}_k\bar{\beta}_1 - \tilde{\theta}_k\tilde{\tau}_{k-1}). \tag{3.27}$$
$$\dot{\beta}_1 = \hat{\beta}_1 = c_1\beta_1 \tag{3.28}$$
$$\dot{\beta}_k = -\tilde{s}_{k-1}\dot{\beta}_{k-1} + \tilde{c}_{k-1}(-1)^{k-1}s^{(k-1)}c_k\beta_1 \tag{3.29}$$
$$\tilde{\beta}_k = \tilde{c}_k\dot{\beta}_k + \tilde{s}_k(-1)^k s^{(k)}c_{k+1}\beta_1. \tag{3.30}$$

We want to show by induction that $\tilde{\tau}_i = \tilde{\beta}_i$ for all i. When $i = 1$,

$$\tilde{\beta}_1 = \tilde{c}_1c_1\beta_1 - \tilde{s}_1s_1c_2\beta_1 = \frac{\beta_1}{\tilde{\rho}_1}(c_1\bar{\rho}_1 - \bar{\theta}_2s_1c_2) = \frac{\beta_1}{\tilde{\rho}_1}\frac{\alpha_1}{\rho_1}\frac{\rho_1^2}{\bar{\rho}_1} = \frac{\bar{\beta}_1}{\tilde{\rho}_1}\frac{\rho_1}{\bar{\rho}_1} = \frac{\bar{\beta}_1}{\tilde{\rho}_1}\bar{c}_1 = \tilde{\tau}_1$$

where the third equality follows from the two lines below:

$$c_1\bar{\rho}_1 - \bar{\theta}_2s_1c_2 = c_1\bar{\rho}_1 - \bar{\theta}_2s_1\frac{c_1\alpha_2}{\rho_2} = \bar{\rho}_1 - \bar{\theta}_2s_1\frac{\alpha_2}{\rho_2} = \frac{\alpha_1}{\rho_1}(\bar{\rho}_1 - \frac{1}{\rho_2}\bar{\theta}_2s_1\alpha_2)$$
$$\bar{\rho}_1 - \frac{1}{\rho_2}\bar{\theta}_2s_1\alpha_2 = \bar{\rho}_1 - \frac{1}{\rho_2}(\bar{s}_1\rho_2)\theta_2 = \bar{\rho}_1 - \frac{\theta_2}{\bar{\rho}_1}\theta_2 = \frac{\bar{\rho}_1^2 - \theta_2^2}{\bar{\rho}_1} = \frac{\rho_1^2 + \theta_2^2 - \theta_2^2}{\bar{\rho}_1}.$$

Suppose $\tilde{\tau}_{k-1} = \tilde{\beta}_{k-1}$. We consider the expression

$$
\begin{aligned}
s^{(k-1)} c_k \bar{\rho}_k^{-1} \bar{c}_{k-1}^2 \rho_k^2 \beta_1 &= \frac{\bar{c}_{k-1}\rho_k}{\bar{\rho}_k}(s^{(k-1)}c_k)\bar{c}_{k-1}\rho_k\beta_1 \\
= \bar{c}_k \frac{\theta_2 \cdots \theta_k \alpha_1}{\rho_1 \cdots \rho_k} \frac{\rho_1 \cdots \rho_{k-1}}{\bar{\rho}_1 \cdots \bar{\rho}_{k-1}} \rho_k \beta_1 &= \bar{c}_k \frac{\theta_2}{\bar{\rho}_1} \cdots \frac{\theta_k}{\bar{\rho}_{k-1}} \bar{\beta}_1 \\
= \bar{c}_k \bar{s}_1 \cdots \bar{s}_{k-1} \bar{\beta}_1 = \bar{c}_k \bar{s}^{(k-1)} \bar{\beta}_1. & \qquad (3.31)
\end{aligned}
$$

Applying the induction hypothesis on $\tilde{\tau}_k = \tilde{\rho}_k^{-1}\left((-1)^{k+1}\bar{s}^{(k-1)}\bar{c}_k\bar{\beta}_1 - \tilde{\theta}_k\tilde{\tau}_{k-1}\right)$ gives

$$
\begin{aligned}
\tilde{\tau}_k &= \tilde{\rho}_k^{-1}\left((-1)^{k+1}\bar{s}^{(k-1)}\bar{c}_k\bar{\beta}_1 - \tilde{\theta}_k\left(\tilde{c}_{k-1}\dot{\beta}_{k-1} + \tilde{s}_{k-1}(-1)^k s^{(k-1)}c_k\beta_1\right)\right) \\
&= \tilde{\rho}_k^{-1}\tilde{\theta}_k\tilde{c}_{k-1}\dot{\beta}_{k-1} + (-1)^{k+1}\tilde{\rho}_k^{-1}\left(\bar{s}^{(k-1)}\bar{c}_k\bar{\beta}_1 - \tilde{\theta}_k\tilde{s}_{k-1}s^{(k-1)}c_k\beta_1\right) \\
&= \tilde{\rho}_k^{-1}(\bar{\rho}_k\tilde{s}_{k-1})\tilde{c}_{k-1}\dot{\beta}_{k-1} + (-1)^{k+1}\tilde{\rho}_k^{-1}s^{(k-1)}\beta_1\left(\dot{\rho}_k\tilde{c}_{k-1}c_k - \bar{\theta}_{k+1}s_kc_{k+1}\right) \\
&= \tilde{c}_k\tilde{s}_{k-1}\dot{\beta}_{k-1} + (-1)^{k+1}s^{(k-1)}\beta_1\left(\tilde{c}_k\tilde{c}_{k-1}c_k - \tilde{s}_ks_kc_{k+1}\right) \\
&= \tilde{c}_k\left(-\tilde{s}_{k-1}\dot{\beta}_{k-1} + \tilde{c}_{k-1}(-1)^{k+1}s^{(k-1)}c_k\beta_1\right) + \tilde{s}_k(-1)^{k+1}s^{(k)}c_{k+1}\beta_1 \\
&= \tilde{c}_k\dot{\beta}_k + \tilde{s}_k(-1)^{k+1}s^{(k)}c_{k+1}\beta_1 = \tilde{\beta}_k
\end{aligned}
$$

with the second equality obtained by the induction hypothesis, and the fourth from

$$
\begin{aligned}
&\bar{s}^{(k-1)}\bar{c}_k\bar{\beta}_1 - \tilde{\theta}_k\tilde{s}_{k-1}s^{(k-1)}c_k\beta_1 \\
&= s^{(k-1)}c_k\bar{\rho}_k^{-1}\bar{c}_{k-1}^2\rho_k^2\beta_1 - (\tilde{s}_{k-1}\bar{\rho}_k)\tilde{s}_{k-1}s^{(k-1)}c_k\beta_1 \\
&= s^{(k-1)}\beta_1\frac{c_k}{\bar{\rho}_k}\left(\bar{c}_{k-1}^2\rho_k^2 - \tilde{s}_{k-1}^2\bar{\rho}_k^2\right) \\
&= s^{(k-1)}\beta_1\left(\dot{\rho}_k\tilde{c}_{k-1}c_k - \bar{\theta}_{k+1}s_kc_{k+1}\right),
\end{aligned}
$$

where the first equality follows from (3.31) and the last from

$$
\begin{aligned}
\bar{c}_{k-1}^2\rho_k^2 - \tilde{s}_{k-1}^2\bar{\rho}_k^2 &= \left(\bar{\rho}_k^2 - \theta_{k+1}^2\right) - \tilde{s}_{k-1}^2\bar{\rho}_k^2 \\
&= \bar{\rho}_k^2(1-\tilde{s}_{k-1}^2) - \theta_{k+1}^2 = \bar{\rho}_k^2\tilde{c}_{k-1}^2 - \theta_{k+1}^2, \\
\frac{c_k}{\bar{\rho}_k}\bar{\rho}_k^2\tilde{c}_{k-1}^2 &= \bar{\rho}_k\tilde{c}_{k-1}^2c_k = \dot{\rho}_k\tilde{c}_{k-1}c_k, \\
\frac{c_k}{\bar{\rho}_k}\theta_{k+1}^2 &= \frac{\theta_{k+1}}{\bar{\rho}_k}\theta_{k+1}c_k = \frac{\theta_{k+1}\rho_{k+1}}{\bar{\rho}_k}s_k\alpha_{k+1}\frac{c_k}{\rho_{k+1}} \\
&= \bar{\theta}_{k+1}s_kc_{k+1}.
\end{aligned}
$$

By induction, we know that $\tilde{\tau}_i = \tilde{\beta}_i$ for $i = 1, 2, \ldots$. From (3.18), we see that at iteration k, the first $k-1$ elements of $\tilde{b}_k$ and $\tilde{t}_k$ are equal. $\square$

4

LSMR EXPERIMENTS

In this chapter, we perform numerous experiments comparing the convergence of LSQR and LSMR. We discuss overdetermined systems first, then some square examples, followed by underdetermined systems. We also explore ways to speed up convergence using extra memory by reorthogonalization.

4.1 LEAST-SQUARES PROBLEMS

4.1.1 BACKWARD ERROR FOR LEAST-SQUARES

For inconsistent problems with uncertainty in A (but not b), let x be any approximate solution. The *normwise backward error* for x measures the perturbation to A that would make x an exact LS solution:

$$\mu(x) \equiv \min_E \|E\| \quad \text{s.t.} \quad (A+E)^T(A+E)x = (A+E)^T b. \tag{4.1}$$

It is known to be the smallest singular value of a certain $m \times (n+m)$ matrix C; see Waldén et al. [83] and Higham [39, pp392–393]:

$$\mu(x) = \sigma_{\min}(C), \qquad C \equiv \begin{bmatrix} A & \frac{\|r\|}{\|x\|}\left(I - \frac{rr^T}{\|r\|^2}\right) \end{bmatrix}. \tag{4.2}$$

Since it is generally too expensive to evaluate $\mu(x)$, we need to find approximations.

APPROXIMATE BACKWARD ERRORS E_1 AND E_2

In 1975, Stewart [69] discussed a particular backward error estimate that we will call E_1. Let x^* and $r^* = b - Ax^*$ be the exact LS solution and residual. Stewart showed that an approximate solution x with residual $r = b - Ax$ is the exact LS solution of the perturbed problem $\min \|b - (A+E_1)x\|$, where E_1 is the rank-one matrix

$$E_1 = \frac{ex^T}{\|x\|^2}, \qquad \|E_1\| = \frac{\|e\|}{\|x\|}, \qquad e \equiv r - r^*, \tag{4.3}$$

with $\|r\|^2 = \|r^*\|^2 + \|e\|^2$. Soon after, Stewart [70] gave a further important result that can be used within any LS solver. The approximate x and a certain vector $\tilde{r} = b - (A + E_2)x$ are the exact solution and residual of the perturbed LS problem $\min \|b - (A + E_2)x\|$, where

$$E_2 = -\frac{rr^TA}{\|r\|^2}, \qquad \|E_2\| = \frac{\|A^Tr\|}{\|r\|}, \qquad r = b - Ax. \tag{4.4}$$

LSQR and LSMR both compute $\|E_2\|$ for each iterate x_k because the current $\|r_k\|$ and $\|A^Tr_k\|$ can be accurately estimated at almost no cost. An added feature is that for both solvers, $\tilde{r} = b - (A + E_2)x_k = r_k$ because $E_2x_k = 0$ (assuming orthogonality of V_k). That is, x_k and r_k are theoretically exact for the perturbed LS problem $(A + E_2)x \approx b$.

Stopping rule S2 (section 3.2.2) requires $\|E_2\| \le \text{ATOL}\|A\|$. Hence the following property gives LSMR an advantage over LSQR for stopping early.

Theorem 4.1.1. $\|E_2^{LSMR}\| \le \|E_2^{LSQR}\|$.

Proof. This follows from $\|A^Tr_k^{\text{LSMR}}\| \le \|A^Tr_k^{\text{LSQR}}\|$ and $\|r_k^{\text{LSMR}}\| \ge \|r_k^{\text{LSQR}}\|$. □

Approximate optimal backward error $\widetilde{\mu}(x)$

Various authors have derived expressions for a quantity $\widetilde{\mu}(x)$ that has proved to be a very accurate approximation to $\mu(x)$ in (4.1) when x is at least moderately close to the exact solution $\widehat{x}$. Grcar, Saunders, and Su [73; 34] show that $\widetilde{\mu}(x)$ can be obtained from a full-rank LS problem as follows:

$$K = \begin{bmatrix} A \\ \frac{\|r\|}{\|x\|}I \end{bmatrix}, \qquad v = \begin{bmatrix} r \\ 0 \end{bmatrix}, \qquad \min_y \|Ky - v\|, \qquad \widetilde{\mu}(x) = \|Ky\|/\|x\|, \tag{4.5}$$

and give MATLAB Code 4.1 for computing the "economy size" sparse QR factorization $K = QR$ and $c \equiv Q^Tv$ (for which $\|c\| = \|Ky\|$) and thence $\widetilde{\mu}(x)$. In our experiments we use this script to compute $\widetilde{\mu}(x_k)$ for each LSQR and LSMR iterate x_k. We refer to this as the *optimal* backward error for x_k because it is provably very close to the true $\mu(x_k)$ [32].

MATLAB Code 4.1 Approximate optimal backward error

```
[m,n]   = size(A);
r       = b - A*x;
normx   = norm(x);
eta     = norm(r)/normx;
p       = colamd(A);
K       = [A(:,p);  eta*speye(n)];
v       = [r;  zeros(n,1)];
[c,R]   = qr(K,v,0);
mutilde = norm(c)/normx;
```

DATA

For test examples, we have drawn from the University of Florida Sparse Matrix Collection (Davis [18]). Matrices from the `LPnetlib` group and the `NYPA` group are used for our numerical experiments.

The `LPnetlib` group provides data for 138 linear programming problems of widely varying origin, structure, and size. The constraint matrix and objective function may be used to define a sparse LS problem $\min \|Ax-b\|$. Each example was downloaded in MATLAB format, and a sparse matrix A and dense vector b were extracted from the data structure via `A = (Problem.A)'` and `b = Problem.c` (where `'` denotes transpose). Five examples had $b = 0$, and a further six gave $A^T b = 0$. The remaining 127 problems had up to 243000 rows, 10000 columns, and 1.4M nonzeros in A. Diagonal scaling was applied to the columns of $\begin{bmatrix} A & b \end{bmatrix}$ to give a scaled problem $\min \|Ax - b\|$ in which the columns of A (and also b) have unit 2-norm. LSQR and LSMR were run on each of the 127 scaled problems with stopping tolerance ATOL $= 10^{-8}$, generating sequences of approximate solutions $\{x_k^{\text{LSQR}}\}$ and $\{x_k^{\text{LSMR}}\}$. The iteration indices k are omitted below. The associated residual vectors are denoted by r without ambiguity, and x^* is the solution to the LS problem, or the minimum-norm solution to the LS problem if the system is singular. This set of artificially created least-squares test problems provides a wide variety of size and structure for evaluation of the two algorithms. They should be indicative of what we could expect when using iterative methods to estimate the dual variables if the linear programs were modified to have a nonlinear objective function (such as the negative entropy function $\sum x_j \log x_j$).

The `NYPA` group provides data for 8 rank-deficient least-squares problems from the New York Power Authority. Each problem provides a

MATLAB Code 4.2 Right diagonal preconditioning

```
% scale the column norms to 1
cnorms = sqrt(sum(A.*A,1));
D = diag(sparse(1./cnorms));
A = A*D;
```

matrix `Problem.A` and a right-hand side vector `Problem.b`. Two of the problems are underdetermined. For the remaining 6 problems we compared the convergence of LSQR and LSMR on $\min \|Ax - b\|$ with stopping tolerance ATOL $= 10^{-8}$. This set of problems contains matrices with condition number ranging from 3.1E+02 to 5.8E+11.

PRECONDITIONING

For this set of test problems, we apply a right diagonal preconditioning that scales the columns of A to unit 2-norm. (For least-squares systems, a left preconditioner will alter the least-squares solution.) The preconditioning is implemented with MATLAB Code 4.2.

4.1.2 NUMERICAL RESULTS

Observations for the `LPnetlib` group:

1. $\|r^{\text{LSQR}}\|$ is monotonic by design. $\|r^{\text{LSMR}}\|$ is also monotonic (as predicted by Theorem 3.3.11) and *nearly* as small as $\|r^{\text{LSQR}}\|$ for all iterations on almost all problems. Figure 4.1 shows a typical example and a rare case.

2. $\|x\|$ is monotonic for LSQR (Theorem 3.3.1) and for LSMR (Theorem 3.3.6). With $\|r\|$ monotonic for LSQR and for LSMR, $\|E_1\|$ in (4.3) is likely to appear monotonic for both solvers. Although $\|E_1\|$ is not normally available for each iteration, it provides a benchmark for $\|E_2\|$.

3. $\|E_2^{\text{LSQR}}\|$ is *not* monotonic, but $\|E_2^{\text{LSMR}}\|$ appears monotonic almost always. Figure 4.2 shows a typical case. The sole exception for this observation is also shown.

4. Note that Benbow [5] has given numerical results comparing a generalized form of LSQR with application of MINRES to the corresponding normal equation. The curves in [5, Figure 3] show the irregular and smooth behavior of LSQR and MINRES respectively

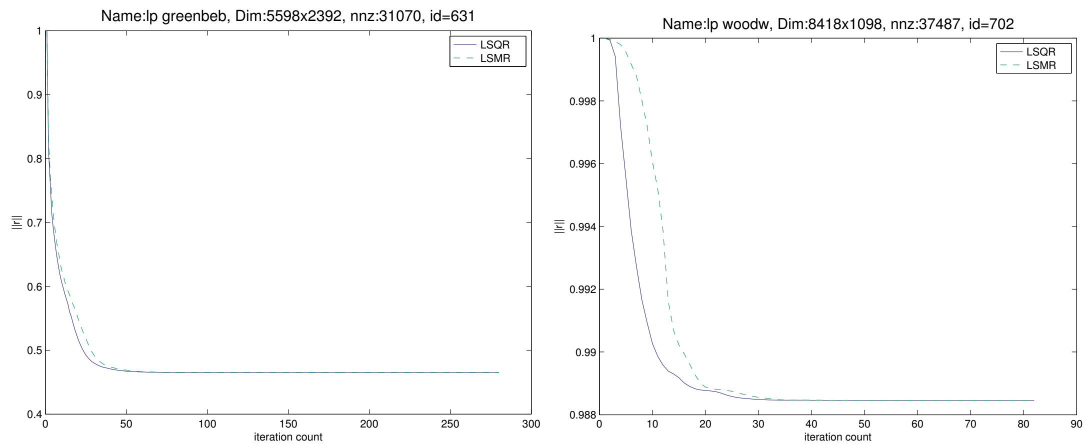


Figure 4.1: For most iterations, $\|r^{\mathrm{LSMR}}\|$ is monotonic and nearly as small as $\|r^{\mathrm{LSQR}}\|$. Left: A typical case (problem `lp_greenbeb`). Right: A rare case (problem `lp_woodw`). LSMR's residual norm is significantly larger than LSQR's during early iterations.

in terms of $\|A^T r_k\|$. Those curves are effectively a preview of the left-hand plots in Figure 4.2 (where LSMR serves as our more reliable implementation of MINRES).

5. $\|E_1^{\mathrm{LSQR}}\| \leq \|E_2^{\mathrm{LSQR}}\|$ often, but not so for LSMR. Some examples are shown on Figure 4.3 along with $\widetilde{\mu}(x_k)$, the accurate estimate (4.5) of the optimal backward error for each point x_k.

6. $\|E_2^{\mathrm{LSMR}}\| \approx \widetilde{\mu}(x^{\mathrm{LSMR}})$ almost always. Figure 4.4 shows a typical example and a rare case. In all such "rare" cases, $\|E_1^{\mathrm{LSMR}}\| \approx \widetilde{\mu}(x^{\mathrm{LSMR}})$ instead!

7. $\widetilde{\mu}(x^{\mathrm{LSQR}})$ is not always monotonic. $\widetilde{\mu}(x^{\mathrm{LSMR}})$ does seem to be monotonic. Figure 4.5 gives examples.

8. $\widetilde{\mu}(x^{\mathrm{LSMR}}) \leq \widetilde{\mu}(x^{\mathrm{LSQR}})$ almost always. Figure 4.6 gives examples.

9. The errors $\|x^* - x^{\mathrm{LSQR}}\|$ and $\|x^* - x^{\mathrm{LSMR}}\|$ decrease monotonically (Theorem 3.3.2 and 3.3.7), with the LSQR error typically smaller than for LSMR. Figure 4.7 gives examples. This is one property for which LSQR seems more desirable (and it has been suggested [57] that for LS problems, LSQR could be terminated when rule S2 would terminate LSMR).

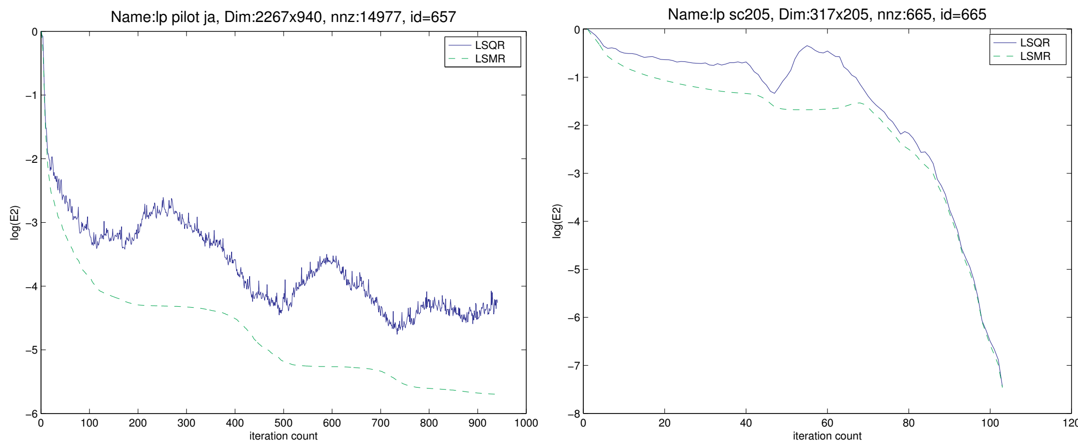


Figure 4.2: For most iterations, $\|E_2^{\mathrm{LSMR}}\|$ appears to be monotonic (but $\|E_2^{\mathrm{LSQR}}\|$ is not). Left: A typical case (problem `lp_pilot_ja`). LSMR is likely to terminate much sooner than LSMR (see Theorem 4.1.1). Right: Sole exception (problem `lp_sc205`) at iterations 54–67. The exception remains even if U_k and/or V_k are reorthogonalized.

For every problem in the `NYPA` group, both solvers satisfied the stopping condition in fewer than $2n$ iterations. Much greater fluctuations are observed in $\|E_2^{\mathrm{LSQR}}\|$ than $\|E_2^{\mathrm{LSMR}}\|$. Figure 4.8 shows the convergence of $\|E_2\|$ for two problems. `Maragal_5` has the largest condition number in the group, while `Maragal_7` has the largest dimensions. $\|E_2^{\mathrm{LSMR}}\|$ converges with small fluctuations, while $\|E_2^{\mathrm{LSQR}}\|$ fluctuates by as much as 5 orders of magnitude.

We should note that when $\mathrm{cond}(A) \geq 10^8$, we cannnot expect any solver to compute a solution with more than about 1 digit of accuracy. The results for problem `Maragal_5` are therefore a little difficult to interpret, but they illustrate the fortunate fact that LSQR and LSMR's *estimates* of $\|A^T r_k\|/\|r_k\|$ do converge toward zero (really $\|A\|\epsilon$), even if the computed vectors $A^T r_k$ are unlikely to become so small.

4.1.3 Effects of preconditioning

The numerical results in the `LPnetlib` test set are generated with every matrix A diagonally preconditioned (i.e., the column norms are scaled to be 1). Before preconditioning, the condition numbers range from 2.9E+00 to 7.2E+12. With preconditioning, they range from 2.7E+00 to 3.4E+08. The condition numbers before and after preconditioning are shown in Figure 4.9.

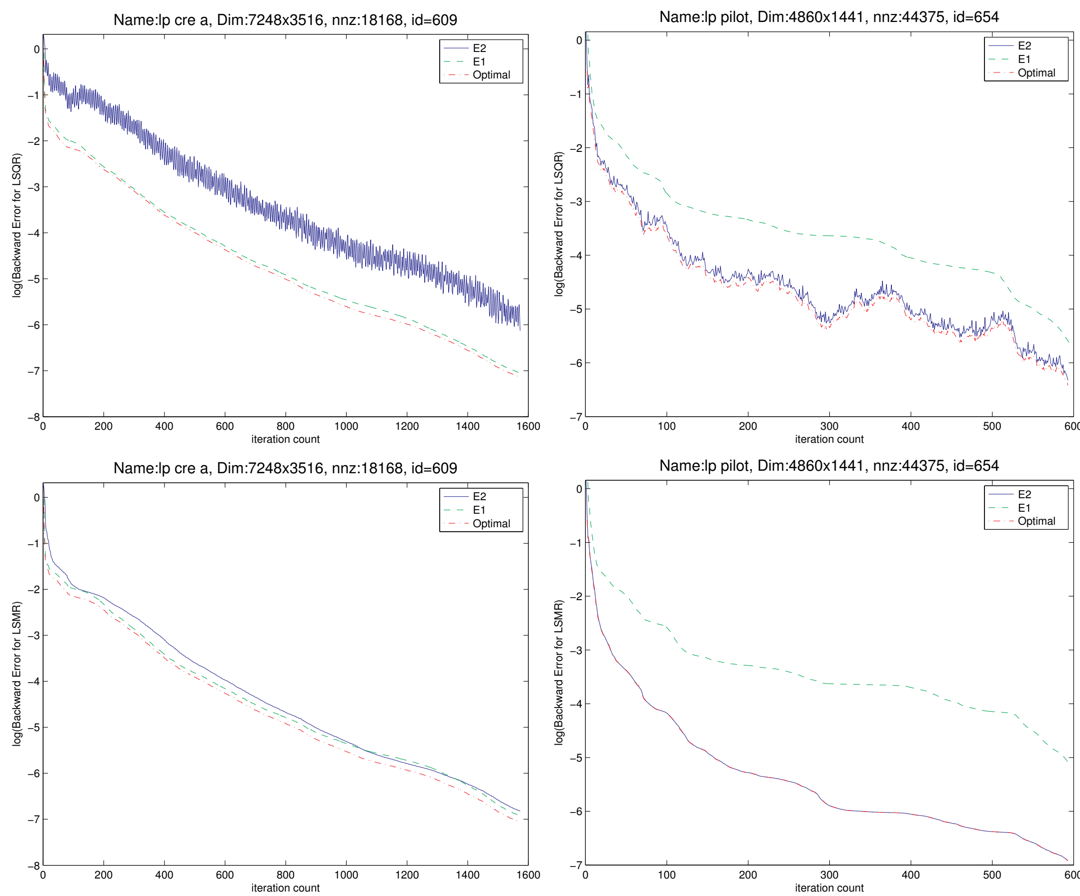


Figure 4.3: $\|E_1\|$, $\|E_2\|$, and $\widetilde{\mu}(x_k)$ for LSQR (top figures) and LSMR (bottom figures). Top left: A typical case. $\|E_1^{\text{LSQR}}\|$ is close to the optimal backward error, but the computable $\|E_2^{\text{LSQR}}\|$ is not. Top right: A rare case in which $\|E_2^{\text{LSQR}}\|$ is close to optimal. Bottom left: $\|E_1^{\text{LSMR}}\|$ and $\|E_2^{\text{LSMR}}\|$ are often both close to the optimal backward error. Bottom right: $\|E_1^{\text{LSMR}}\|$ is far from optimal, but the computable $\|E_2^{\text{LSMR}}\|$ is almost always close (too close to distinguish in the plot!). Problems `lp_cre_a` (left) and `lp_pilot` (right).

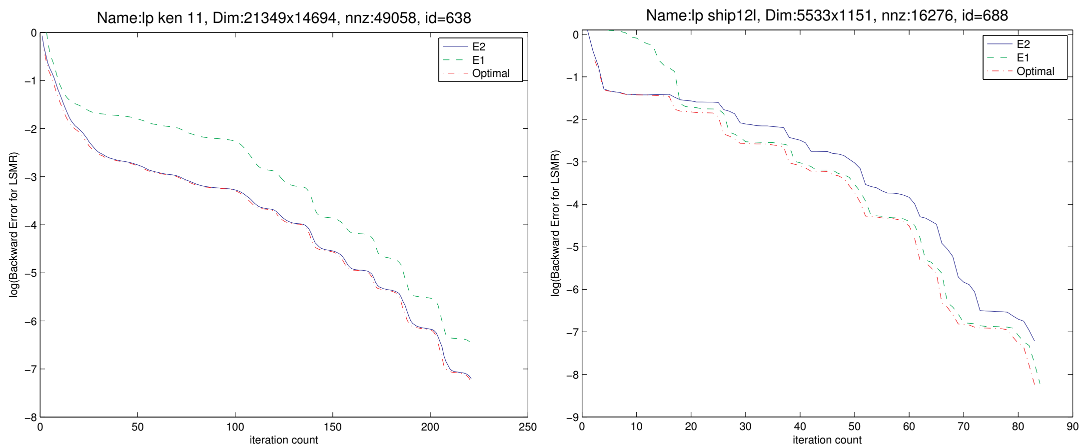


Figure 4.4: Again, $\|E_2^{\text{LSMR}}\| \approx \widetilde{\mu}(x^{\text{LSMR}})$ almost always (the computable backward error estimate is essentially optimal). Left: A typical case (problem `lp_ken_11`). Right: A rare case (problem `lp_ship12l`). Here, $\|E_1^{\text{LSMR}}\| \approx \widetilde{\mu}(x^{\text{LSMR}})$!

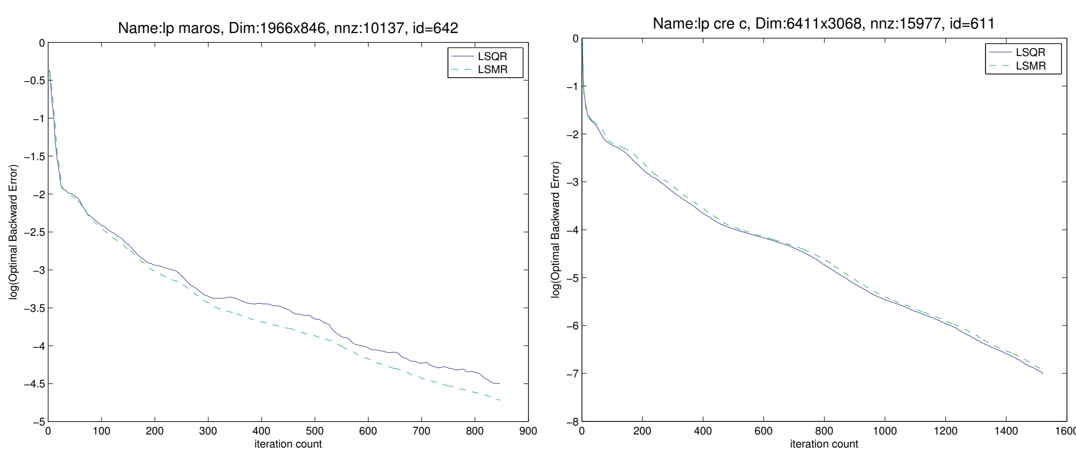


Figure 4.5: $\widetilde{\mu}(x^{\text{LSMR}})$ seems to be always monotonic, but $\widetilde{\mu}(x^{\text{LSQR}})$ is usually not. Left: A typical case for both LSQR and LSMR (problem `lp_maros`). Right: A rare case for LSQR, typical for LSMR (problem `lp_cre_c`).

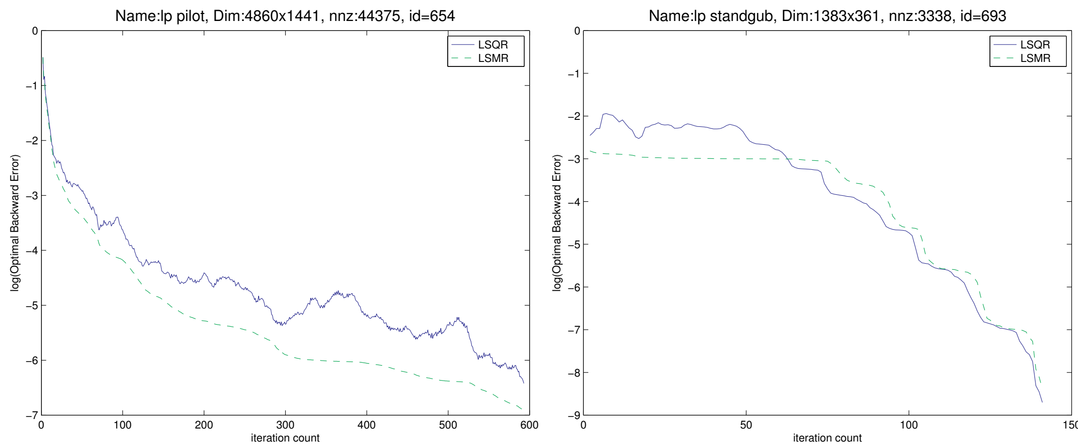


Figure 4.6: $\widetilde{\mu}(x^{\text{LSMR}}) \leq \widetilde{\mu}(x^{\text{LSQR}})$ almost always. Left: A typical case (problem `lp_pilot`). Right: A rare case (problem `lp_standgub`).

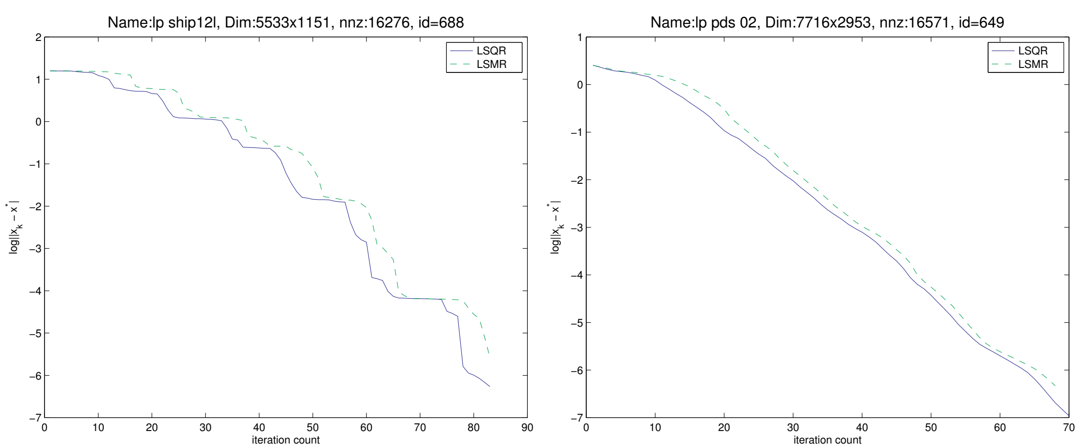


Figure 4.7: The errors $\|x^* - x^{\text{LSQR}}\|$ and $\|x^* - x^{\text{LSMR}}\|$ seem to decrease monotonically, with LSQR's errors smaller than for LSMR. Left: A nonsingular LS system (problem `lp_ship12l`). Right: A singular system (problem `lp_pds_02`). LSQR and LSMR both converge to the minimum-norm LS solution.

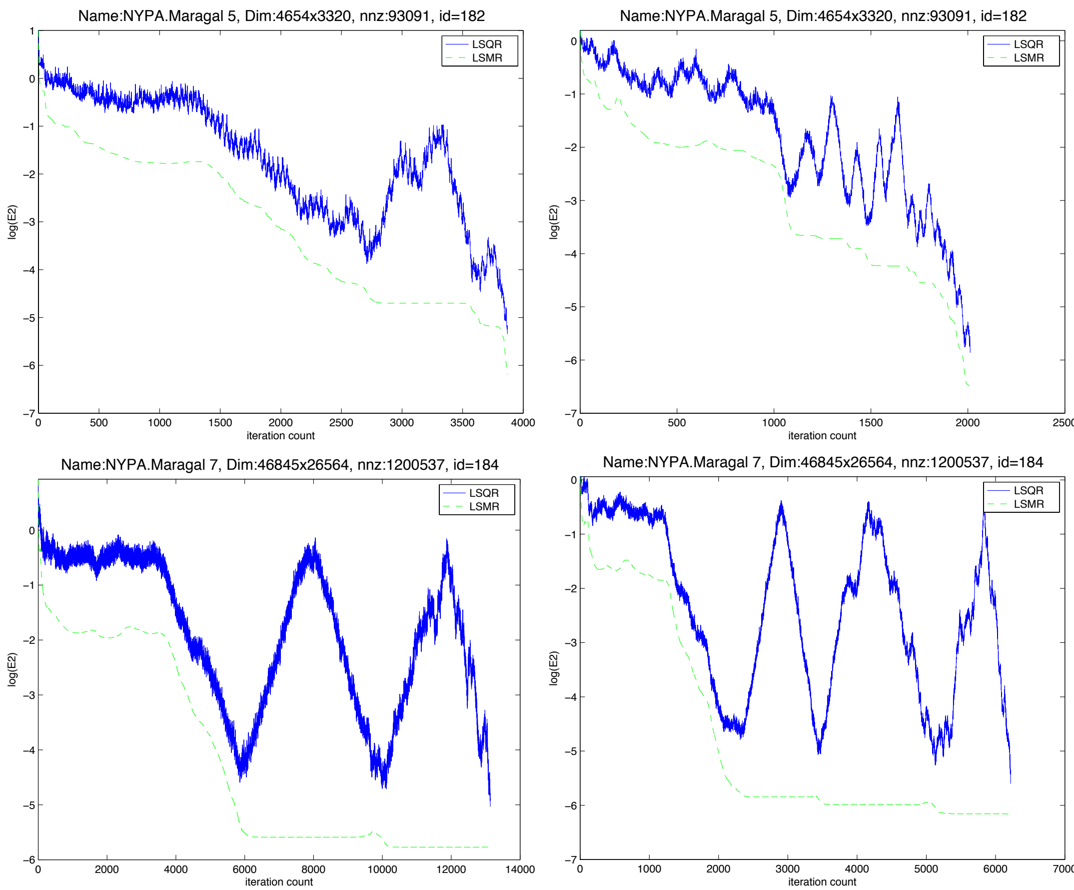


Figure 4.8: Convergence of $\|E_2\|$ for two problems in NYPA group using LSQR and LSMR.
Upper: Problem `Maragal_5`.

Left: No preconditioning applied. cond(A)=5.8E+11. If the iteration limit had been n iterations, the final LSQR point would be very poor.

Right: Right diagonal preconditioning applied. cond(A)=2.6E+12.

Lower: Problem `Maragal_7`.

Left: No preconditioning applied. cond(A)=1.4E+03.

Right: Right diagonal preconditioning applied. cond(A)=4.2E+02.

The peaks for LSQR (where it would be undesirable for LSQR to terminate) correspond to plateaus for LSMR where $\|E_2\|$ remains the smallest value so far, except for slight increases near the end of the LSQR peaks.

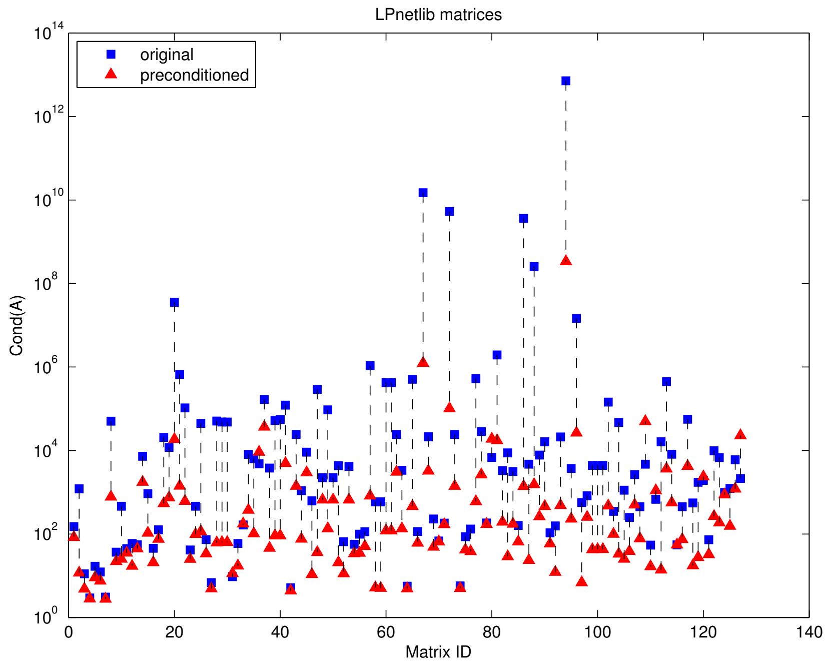


Figure 4.9: Distribution of condition number for `LPnetlib` matrices. Diagonal preconditioning reduces the condition number in 117 out of 127 cases.

To illustrate the effect of this preconditioning on the convergence speed of LSQR and LSMR, we solve each problem $\min \|Ax - b\|$ from the `LPnetlib` set using the two algorithms and summarize the results in Table 4.1. Both algorithms use stopping rule S2 in Section 3.2.2 with ATOL=1E-8, or a limit of $10n$ iterations.

In Table 4.1, we see that for systems that converge quickly, the advantage gained by using LSMR compared with LSQR is relatively small. For example, `lp_osa_60` (Row 127) is a 243246×10280 matrix. LSQR converges in 124 iterations while LSMR converges in 122 iterations. In contrast, for systems that take many iterations to converge, LSMR usually converges much faster than LSQR. For example, `lp_cre_d` (Row 122) is a 73948×8926 matrix. LSQR takes 19496 iterations, while LSMR takes 9259 iterations (53% fewer).

Table 4.1: Effects of diagonal preconditioning on `LPnetlib` matrices†† and convergence of LSQR and LSMR on $\min \|Ax - b\|$

							Original A			Diagonally preconditioned A		
	ID	name	m	n	nnz	$\sigma_i = 0^*$	Cond(A)	$k^{\text{LSQR}\dagger}$	$k^{\text{LSMR}\dagger}$	Cond(A)	$k^{\text{LSQR}\dagger}$	$k^{\text{LSMR}\dagger}$
1	720	lpi_itest6	17	11	29	0	1.5E+02	5	5	8.4E+01	7	7
2	706	lpi_bgprtr	40	20	70	0	1.2E+03	21	21	1.2E+01	19	19
3	597	lp_afiro	51	27	102	0	1.1E+01	22	22	4.9E+00	21	21
4	731	lpi_woodinfe	89	35	140	0	2.9E+00	17	17	2.8E+00	17	17
5	667	lp_sc50b	78	50	148	0	1.7E+01	41	41	9.1E+00	36	36
6	666	lp_sc50a	78	50	160	0	1.2E+01	38	38	7.6E+00	34	34
7	714	lpi_forest6	131	66	246	0	3.1E+00	21	21	2.8E+00	19	19
8	636	lp_kb2	68	43	313	0	5.1E+04	150	147	7.8E+02	128	128
9	664	lp_sc105	163	105	340	0	3.7E+01	68	68	2.2E+01	58	58
10	596	lp_adlittle	138	56	424	0	4.6E+02	61	61	2.5E+01	39	39
11	713	lpi_ex73a	211	193	457	5	4.5E+01	99	96	3.5E+01	85	84
12	669	lp_scagr7	185	129	465	0	6.0E+01	80	80	1.7E+01	60	59
13	712	lpi_ex72a	215	197	467	5	5.6E+01	101	98	4.5E+01	89	88
14	695	lp_stocfor1	165	117	501	0	7.3E+03	107	105	1.8E+03	263	238
15	603	lp_blend	114	74	522	0	9.2E+02	186	186	1.1E+02	118	118
16	707	lpi_box1	261	231	651	17	4.6E+01	28	28	2.1E+01	24	24
17	665	lp_sc205	317	205	665	0	1.3E+02	122	122	7.6E+01	102	101
18	663	lp_recipe	204	91	687	0	2.1E+04	4	4	5.4E+02	4	4
19	682	lp_share2b	162	96	777	0	1.2E+04	516	510	7.5E+02	331	328
20	700	lp_vtp_base	346	198	1051	0	3.6E+07	557	556	1.9E+04	1370	1312
21	641	lp_lotfi	366	153	1136	0	6.6E+05	149	146	1.4E+03	386	386
22	681	lp_share1b ‡	253	117	1179	0	1.0E+05	1170	1170	6.2E+02	482	427
23	724	lpi_mondou2	604	312	1208	1	4.2E+01	151	147	2.5E+01	119	116
24	711	lpi_cplex2	378	224	1215	1	4.6E+02	105	102	9.9E+01	81	80
25	606	lp_bore3d	334	233	1448	2	4.5E+04	782	681	1.2E+02	265	263
26	673	lp_scorpion	466	388	1534	30	7.4E+01	159	152	3.4E+01	116	115
27	709	lpi_chemcom	744	288	1590	0	6.9E+00	40	39	5.0E+00	35	35
28	729	lpi_refinery	464	323	1626	0	5.1E+04	2684	1811	6.2E+01	113	113
29	727	lpi_qual	464	323	1646	0	4.8E+04	2689	1828	6.3E+01	121	120
30	730	lpi_vol1	464	323	1646	0	4.8E+04	2689	1828	6.3E+01	121	120
31	703	lpi_bgdbg1	629	348	1662	0	9.5E+00	44	43	1.1E+01	53	52
32	668	lp_scagr25	671	471	1725	0	5.9E+01	155	147	1.7E+01	97	95
33	678	lp_sctap1	660	300	1872	0	1.7E+02	364	338	1.8E+02	334	327
34	608	lp_capri	482	271	1896	0	8.1E+03	1194	1051	3.8E+02	453	451
35	607	lp_brandy	303	220	2202	27	6.4E+03	665	539	1.0E+02	208	208
36	635	lp_israel	316	174	2443	0	4.8E+03	351	325	9.2E+03	782	720
37	629	lp_gfrd_pnc	1160	616	2445	0	1.6E+05	84	73	3.7E+04	270	210
38	601	lp_bandm	472	305	2494	0	3.8E+03	2155	1854	4.6E+01	196	187
39	621	lp_etamacro	816	400	2537	0	6.0E+04	568	186	9.2E+01	171	162
40	704	lpi_bgetam	816	400	2537	0	5.3E+04	536	186	9.2E+01	171	162
41	728	lpi_reactor	808	318	2591	0	1.4E+05	85	85	5.0E+03	151	149
42	634	lp_grow7	301	140	2612	0	5.2E+00	31	30	4.4E+00	28	28
43	670	lp_scfxm1	600	330	2732	0	2.4E+04	1368	1231	1.4E+03	547	470
44	623	lp_finnis	1064	497	2760	0	1.1E+03	328	327	7.7E+01	279	275
45	620	lp_e226	472	223	2768	0	9.1E+03	591	555	3.0E+03	504	437
46	598	lp_agg	615	488	2862	0	6.2E+02	159	154	1.1E+01	35	35
47	725	lpi_pang	741	361	2933	0	2.9E+05	350	247	3.7E+01	125	111
48	692	lp_standata	1274	359	3230	0	2.2E+03	144	141	6.6E+02	140	139
49	674	lp_scrs8	1275	490	3288	0	9.4E+04	1911	1803	1.4E+02	356	338
50	693	lp_standgub	1383	361	3338	1	2.2E+03	144	141	6.6E+02	140	139
51	602	lp_beaconfd	295	173	3408	0	4.4E+03	254	254	2.1E+01	64	63
52	683	lp_shell	1777	536	3558	1	4.2E+01	88	85	1.1E+01	43	42
53	694	lp_standmps	1274	467	3878	0	4.2E+03	286	255	6.6E+02	201	201
54	691	lp_stair	614	356	4003	0	5.7E+01	122	115	3.4E+01	95	94
55	617	lp_degen2	757	444	4201	2	9.9E+01	264	250	3.5E+01	151	146
56	685	lp_ship04s	1506	402	4400	42	1.1E+02	103	100	5.1E+01	75	75
57	699	lp_tuff	628	333	4561	31	1.1E+06	1021	1013	8.2E+02	648	642
58	599	lp_agg2	758	516	4740	0	5.9E+02	184	175	5.2E+00	31	31
59	600	lp_agg3	758	516	4756	0	5.9E+02	219	208	5.1E+00	32	31
60	655	lp_pilot4	1123	410	5264	0	4.2E+05	1945	1379	1.2E+02	195	190
61	726	lpi_pilot4i	1123	410	5264	0	4.2E+05	2081	1357	1.2E+02	195	191
62	671	lp_scfxm2	1200	660	5469	0	2.4E+04	2154	1575	3.1E+03	975	834
63	604	lp_bnl1	1586	643	5532	1	3.4E+03	1394	1253	1.4E+02	285	278
64	632	lp_grow15	645	300	5620	0	5.6E+00	35	35	4.9E+00	33	32
65	653	lp_perold	1506	625	6148	0	5.1E+05	5922	3173	4.6E+02	706	619
66	684	lp_ship04l	2166	402	6380	42	1.1E+02	77	76	6.1E+01	67	67
67	622	lp_fffff800	1028	524	6401	0	1.5E+10	2064	1161	1.2E+06	5240	5240

Continued on next page…

							Original A			Diagonally preconditioned A		
	ID	name	m	n	nnz	$\sigma_i = 0^*$	Cond(A)	$k^{\text{LSQR}\dagger}$	$k^{\text{LSMR}\dagger}$	Cond(A)	$k^{\text{LSQR}\dagger}$	$k^{\text{LSMR}\dagger}$
68	628	`lp_ganges`	1706	1309	6937	0	2.1E+04	219	216	3.3E+03	161	160
69	687	`lp_ship08s`	2467	778	7194	66	1.6E+02	169	169	4.9E+01	116	116
70	662	`lp_qap8`	1632	912	7296	170	1.9E+01	8	8	6.6E+01	8	8
71	679	`lp_sctap2`	2500	1090	7334	0	1.8E+02	639	585	1.7E+02	450	415
72	690	`lp_sierra`	2735	1227	8001	10	5.0E+09	87	74	1.0E+05	146	146
73	672	`lp_scfxm3`	1800	990	8206	0	2.4E+04	2085	1644	1.4E+03	1121	994
74	633	`lp_grow22`	946	440	8252	0	5.7E+00	39	38	5.0E+00	35	35
75	689	`lp_ship12s`	2869	1151	8284	109	8.7E+01	135	134	4.2E+01	89	88
76	637	`lp_ken_07`	3602	2426	8404	49	1.3E+02	168	168	3.8E+01	98	98
77	658	`lp_pilot_we`	2928	722	9265	0	5.3E+05	5900	3503	6.1E+02	442	246
78	696	`lp_stocfor2`	3045	2157	9357	0	2.8E+04	430	407	2.6E+03	1546	1421
79	680	`lp_sctap3`	3340	1480	9734	0	1.8E+02	683	618	1.7E+02	503	465
80	625	`lp_fit1p`	1677	627	9868	0	6.8E+03	81	81	1.9E+04	500	427
81	642	`lp_maros` ‡	1966	846	10137	0	1.9E+06	8460	7934	1.8E+04	6074	3886
82	594	`lp_25fv47`	1876	821	10705	1	3.3E+03	5443	4403	2.0E+02	702	571
83	614	`lp_czprob`	3562	929	10708	0	8.8E+03	114	110	2.9E+01	29	29
84	710	`lpi_cplex1`	5224	3005	10947	0	1.7E+04	89	79	1.7E+02	53	53
85	686	`lp_ship08l`	4363	778	12882	66	1.6E+02	123	123	6.5E+01	103	103
86	659	`lp_pilotnov` ¶	2446	975	13331	0	3.6E+09	164	343	1.4E+03	1622	1180
87	624	`lp_fit1d`	1049	24	13427	0	4.7E+03	61	61	2.4E+01	28	28
88	657	`lp_pilot_ja`	2267	940	14977	0	2.5E+08	7424	950	1.5E+03	1653	1272
89	605	`lp_bnl2`	4486	2324	14996	0	7.8E+03	1906	1333	2.6E+02	452	390
90	611	`lp_cre_c`	6411	3068	15977	87	1.6E+04	20109	12067	4.6E+02	1553	1333
91	688	`lp_ship12l`	5533	1151	16276	109	1.1E+02	106	104	5.9E+01	82	81
92	649	`lp_pds_02`	7716	2953	16571	11	4.0E+02	129	124	1.2E+01	69	67
93	609	`lp_cre_a`	7248	3516	18168	93	2.1E+04	20196	11219	4.9E+02	1591	1375
94	717	`lpi_gran` ‡	2525	2658	20111	586	7.2E+12	26580	20159	3.4E+08	22413	11257
95	708	`lpi_ceria3d`	4400	3576	21178	0	7.3E+02	57	56	2.3E+02	224	213
96	613	`lp_cycle` ‡	3371	1903	21234	28	1.5E+07	19030	19030	2.7E+04	2911	2349
97	595	`lp_80bau3b`	12061	2262	23264	0	5.7E+02	119	111	6.9E+00	43	42
98	618	`lp_degen3`	2604	1503	25432	2	8.3E+02	1019	969	2.5E+02	448	414
99	630	`lp_greenbea`	5598	2392	31070	3	4.4E+03	2342	2062	4.3E+01	277	251
100	631	`lp_greenbeb`	5598	2392	31070	3	4.4E+03	2342	2062	4.3E+01	277	251
101	718	`lpi_greenbea`	5596	2393	31074	3	4.4E+03	2148	1860	4.3E+01	239	221
102	615	`lp_d2q06c` ‡	5831	2171	33081	0	1.4E+05	21710	15553	4.8E+02	1825	1548
103	619	`lp_dfl001`	12230	6071	35632	13	3.5E+02	937	848	1.0E+02	363	353
104	702	`lp_woodw`	8418	1098	37487	0	4.7E+04	557	553	3.3E+01	81	81
105	616	`lp_d6cube`	6184	415	37704	11	1.1E+03	174	169	2.5E+01	52	52
106	660	`lp_qap12`	8856	3192	38304	398	3.9E+01	8	8	3.9E+01	8	8
107	654	`lp_pilot`	4860	1441	44375	0	2.7E+03	1392	1094	5.0E+02	592	484
108	638	`lp_ken_11`	21349	14694	49058	121	4.6E+02	498	491	7.8E+01	220	207
109	627	`lp_fit2p`	13525	3000	50284	0	4.7E+03	73	73	5.0E+04	3276	1796
110	650	`lp_pds_06`	29351	9881	63220	11	5.4E+01	197	183	1.7E+01	100	97
111	705	`lpi_bgindy`	10880	2671	66266	0	6.7E+02	366	358	1.1E+03	377	356
112	701	`lp_wood1p`	2595	244	70216	1	1.6E+04	53	53	1.4E+01	25	25
113	697	`lp_stocfor3`	23541	16675	72721	0	4.5E+05	832	801	3.6E+03	3442	3096
114	656	`lp_pilot87`	6680	2030	74949	0	8.1E+03	896	751	5.7E+02	297	170
115	661	`lp_qap15`	22275	6330	94950	632	5.5E+01	8	8	5.6E+01	8	8
116	639	`lp_ken_13`	42659	28632	97246	169	4.5E+02	471	462	7.4E+01	205	204
117	716	`lpi_gosh`	13455	3792	99953	2	5.6E+04	3236	1138	4.2E+03	3629	1379
118	651	`lp_pds_10`	49932	16558	107605	11	5.6E+02	223	208	1.8E+01	120	115
119	626	`lp_fit2d`	10524	25	129042	0	1.7E+03	55	55	2.8E+01	29	29
120	645	`lp_osa_07`	25067	1118	144812	0	1.9E+03	105	105	2.4E+03	72	72
121	652	`lp_pds_20`	108175	33874	232647	87	7.3E+01	323	283	3.3E+01	177	165
122	612	`lp_cre_d`	73948	8926	246614	2458	9.9E+03	19496	9259	2.7E+02	1218	1069
123	610	`lp_cre_b`	77137	9648	260785	2416	6.8E+03	14761	7720	1.9E+02	1112	966
124	646	`lp_osa_14`	54797	2337	317097	0	9.9E+02	120	120	8.8E+02	73	73
125	640	`lp_ken_18`	154699	105127	358171	324	1.2E+03	999	957	1.6E+02	422	398
126	647	`lp_osa_30`	104374	4350	604488	0	6.0E+03	116	115	1.2E+03	77	77
127	648	`lp_osa_60`	243246	10280	1408073	0	2.1E+03	124	122	2.3E+04	82	82

* Number of columns in A that are not independent.

†† We are using `A = problem.A'; b = problem.c;` to construct the least-squares problem $\min \|Ax - b\|$.

† Denotes the number of iterations that LSQR or LSMR takes to converge with a tolerance of 10^{-8}.

‡ For problem `lp_maros`, `lpi_gran`, `lp_d2q06c`, LSQR hits the $10n$ iteration limit without preconditioning. For problem `lp_share1b`, `lp_cycle`, both LSQR and LSMR hit the $10n$ iteration limit without preconditioning. Thus the number of iteration that these five problems take to converge doesn't represent the relative improvement provided by LSMR.

¶ Problem `lp_pilotnov` is compatible ($\|r_k\| \to 0$). Therefore LSQR exhibits faster convergence than LSMR. More examples for compatible systems are given in Section 4.2 and 4.3.

MATLAB Code 4.3 Generating preconditioners by perturbation of QR

```
% delta is the chosen standard deviation of Gaussian noise
randn('state',1);
R = qr(A);
[I J S] = find(R);
Sp = S.*(1+delta*randn(length(S), 1));
M = sparse(I,J,Sp);
```

Diagonal preconditioning almost always reduces the condition number of A. For most of the examples, it also reduces the number of iterations for LSQR and LSMR to converge. With diagonal preconditioning, the condition number of `lp_cre_d` reduces from 9.9E+03 to 2.7E+02. The number of iterations for LSQR to converge is reduced to 1218 and that for LSMR is reduced to 1069 (12% less than that of LSQR). Since the preconditioned system needs fewer iterations, there is less advantage in using LSMR in this case. (This phenomenon can be explained by (4.7) in the next section.)

To further illustrate the effect of preconditioning, we construct a sequence of increasingly better preconditioners and investigate their effect on the convergence of LSQR and LSMR. The preconditioners are constructed by first performing a sparse QR factorization $A = QR$, and then adding Gaussian random noise to the nonzeros of R. For a given noise level δ, we use MATLAB Code 4.3 to generate the preconditioner.

Figure 4.10 illustrates the convergence of LSQR and LSMR on problem `lp_d2q06c` (cond(A) = 1.4E+05) with a number of preconditioners. We have a total of 5 options:

- No preconditioner
- Diagonal preconditioner from MATLAB Code 4.2
- Preconditioner from MATLAB Code 4.3 with $\delta = 0.1$
- Preconditioner from MATLAB Code 4.3 with $\delta = 0.01$
- Preconditioner from MATLAB Code 4.3 with $\delta = 0.001$

From the plots in Figure 4.10, we see that when no preconditioner is applied, both algorithms exhibit very slow convergence and LSQR hits the $10n$ iteration limit. The backward error for LSQR lags behind LSMR by at least 1 order of magnitude at the beginning, and the gaps widen to 2 orders of magnitude toward $10n$ iterations. The backward error for LSQR fluctuates significantly across all iterations.

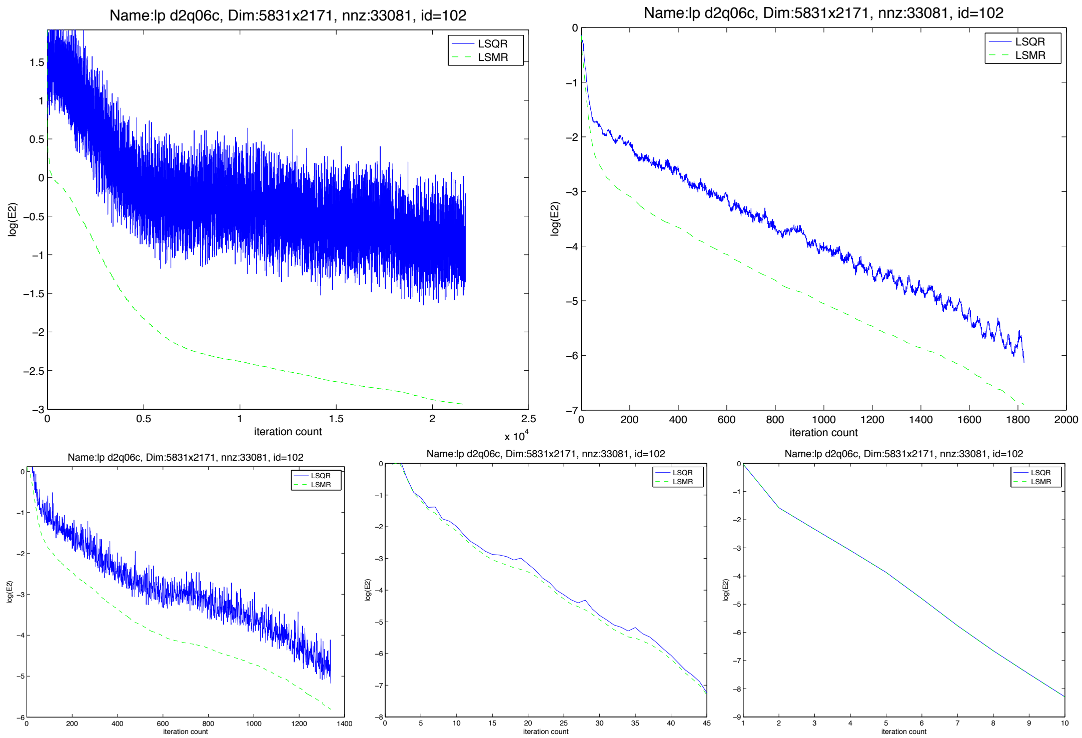


Figure 4.10: Convergence of LSQR and LSMR with increasingly good preconditioners. $\log \|E_2\|$ is plotted against iteration number. LSMR shows an advantage until the preconditioners is almost exact. Top: No preconditioners and diagonal preconditioner. Bottom: Exact preconditioner with noise levels of 10%, 1% and 0.1%.

When diagonal preconditioning is applied, both algorithms take less than n iterations to converge. The backward errors for LSQR lag behind LSMR by 1 order of magnitude. There is also much less fluctuation in the LSQR backward error compared to the unpreconditioned case.

For the increasingly better preconditioners constructed with $\delta = 0.1$, 0.01 and 0.001, we see that the number of iterations to convergence decreases rapidly. With better preconditioners, we also see that the gap between the backward errors for LSQR and LSMR becomes smaller. With an almost perfect preconditioner ($\delta = 0.001$), the backward error for LSQR becomes almost the same as that for LSMR at each iteration. This phenomenon can be explained by (4.7) in the next section.

4.1.4 Why does $\|E_2\|$ for LSQR lag behind LSMR?

David Titley-Peloquin, in joint work with Serge Gratton and Pavel Jiranek [76], has performed extensive analysis of the convergence behavior of LSQR and LSMR for least-square problems. These results are unpublished at the time of writing. We summarize two key insights from their work to provide a more complete picture on how these two algorithms perform.

The residuals $\|r_k^{\text{LSQR}}\|$, $\|r_k^{\text{LSMR}}\|$ and residuals for the normal equation $\|A^T r_k^{\text{LSQR}}\|$, $\|A^T r_k^{\text{LSMR}}\|$ for LSQR and LSMR satisfy the following relations [76], [35, Lemma 5.4.1]:

$$\|r_k^{\text{LSMR}}\|^2 = \|r_k^{\text{LSQR}}\|^2 + \sum_{j=0}^{k-1} \frac{\|A^T r_k^{\text{LSMR}}\|^4}{\|A^T r_j^{\text{LSMR}}\|^4} (\|r_j^{\text{LSQR}}\|^2 - \|r_{j+1}^{\text{LSQR}}\|^2) \tag{4.6}$$

$$\|A^T r_k^{\text{LSQR}}\| = \frac{\|A^T r_k^{\text{LSMR}}\|}{\sqrt{1 - \|A^T r_k^{\text{LSMR}}\|^2 / \|A^T r_{k-1}^{\text{LSMR}}\|^2}}. \tag{4.7}$$

From (4.6), one can infer that $\|r_k^{\text{LSMR}}\|$ is much larger than $\|r_k^{\text{LSQR}}\|$ only if both

- $\|r_j^{\text{LSQR}}\|^2 \ll \|r_{j+1}^{\text{LSQR}}\|^2$ for some $j < k$
- $\|A^T r_j^{\text{LSMR}}\|^4 \approx \|A^T r_{j+1}^{\text{LSMR}}\|^4 \approx \cdots \approx \|A^T r_k^{\text{LSMR}}\|^4$

happened, which is very unlikely in view of the fourth power [76]. This explains our observation in Figure 4.1 that $\|r_k^{\text{LSMR}}\|$ rarely lags behind $\|r_k^{\text{LSQR}}\|$. In cases where $\|r_k^{\text{LSMR}}\|$ lags behind in the early iterations, it catches up very quickly.

From (4.7), one can infer that if $\|A^T r_k^{\text{LSMR}}\|$ decreases a lot between iterations $k-1$ and k, then $\|A^T r_k^{\text{LSQR}}\|$ would be roughly the same as $\|A^T r_k^{\text{LSMR}}\|$. The converse also holds, in that $\|A^T r_k^{\text{LSQR}}\|$ will be much larger than $\|A^T r_k^{\text{LSMR}}\|$ if LSMR is almost stalling at iteration k (i.e., $\|A^T r_k^{\text{LSMR}}\|$ did not decrease much relative to the previous iteration) [76]. This explains the peaks and plateaus observed in Figure 4.8.

We have a further insight about the difference between LSQR and LSMR on least-squares problem that take many iterations. Both solvers stop when

$$\|A^T r_k\| \le \text{ATOL}\|A\|\|r_k\|.$$

MATLAB Code 4.4 Criteria for selecting square systems

```
ids = find(index.nrows > 100000       & ...
           index.nrows < 200000       & ...
           index.nrows == index.ncols & ...
           index.isReal == 1          & ...
           index.posdef == 0          & ...
           index.numerical_symmetry < 1);
```

Since $\|r^*\|$ is often $O(\|r_0\|)$ for least-squares, and it is also safe to assume $\|A^T r_0\|/(\|A\|\|r_0\|) = O(1)$, we know that they will stop at iteration l, where

$$\prod_{k=1}^{l} \frac{\|A^T r_k\|}{\|A^T r_{k-1}\|} = \frac{\|A^T r_l\|}{\|A^T r_0\|} \approx O(\text{ATOL}).$$

Thus on average, $\|A^T r_k^{\text{LSMR}}\|/\|A^T r_{k-1}^{\text{LSMR}}\|$ will be closer to 1 if l is large. This means that the larger l is (in absolute terms), the more LSQR will lag behind LSMR (a bigger gap between $\|A^T r_k^{\text{LSQR}}\|$ and $\|A^T r_k^{\text{LSMR}}\|$).

4.2 Square systems

Since LSQR and LSMR are applicable to consistent systems, it is of interest to compare them on an unbiased test set. We used the search facility of Davis [18] to select a set of square real linear systems $Ax = b$. With `index = UFget`, the criteria in MATLAB Code 4.4 returned a list of 42 examples. Testing `isfield(UFget(id),'b')` left 26 cases for which b was supplied.

Preconditioning

For each linear system, diagonal scaling was first applied to the *rows* of $\begin{bmatrix} A & b \end{bmatrix}$ and then to its columns using MATLAB Code 4.5 to give a scaled problem $Ax = b$ in which the columns of $\begin{bmatrix} A & b \end{bmatrix}$ have unit 2-norm.

In spite of the scaling, most examples required more than n iterations of LSQR or LSMR to reduce $\|r_k\|$ satisfactorily (rule S1 in section 3.2.2 with ATOL $=$ BTOL $= 10^{-8}$). To simulate better preconditioning, we chose two cases that required about $n/5$ and $n/10$ iterations. Figure 4.11 (left) shows both solvers reducing $\|r_k\|$ monotonically but with plateaus that are prolonged for LSMR. With loose stopping tolerances, LSQR could terminate somewhat sooner. Figure 4.11

MATLAB Code 4.5 Diagonal preconditioning

```
% scale the row norms to 1
rnorms = sqrt(sum(A.*A,2));
D = diag(sparse(1./rnorms));
A = D*A;
b = D*b;
% scale the column norms to 1
cnorms = sqrt(sum(A.*A,1));
D = diag(sparse(1./cnorms));
A = A*D;
% scale the 2 norm of b to 1
bnorm = norm(b);
if bnorm ~= 0
  b = b./bnorm;
end
```

(right) shows $\|A^T r_k\|$ for each solver. The plateaus for LSMR correspond to LSQR gaining ground with $\|r_k\|$, but falling significantly backward by the $\|A^T r_k\|$ measure.

Comparison with IDR(s) on square systems

Again we mention that on certain square parameterized systems, the solvers IDR(s) and LSQR or LSMR complement each other [81; 82] (see Section 1.3.5).

4.3 Underdetermined systems

In this section, we study the convergence of LSQR and LSMR when applied to an underdetermined system $Ax = b$. As shown in Section 3.3.2, LSQR and LSMR converge to the minimum-norm solution for a singular system (rank(A) $< n$). The solution solves $\min_{Ax=b} \|x\|_2$.

As a comparison, we also apply MINRES directly to the equation $AA^T y = b$ and take $x = A^T y$ as the solution. This avoids multiplication by $A^T A$ in the Lanczos process, where $A^T A$ is a highly singular operator because A has more columns than rows. It is also useful to note that this application of MINRES is mathematically equivalent to applying LSQR to $Ax = b$.

Theorem 4.3.1. *In exact arithmetic, applying MINRES to $AA^T y = b$ and setting $x_k = A^T y_k$ generates the same iterates as applying LSQR to $Ax = b$.*

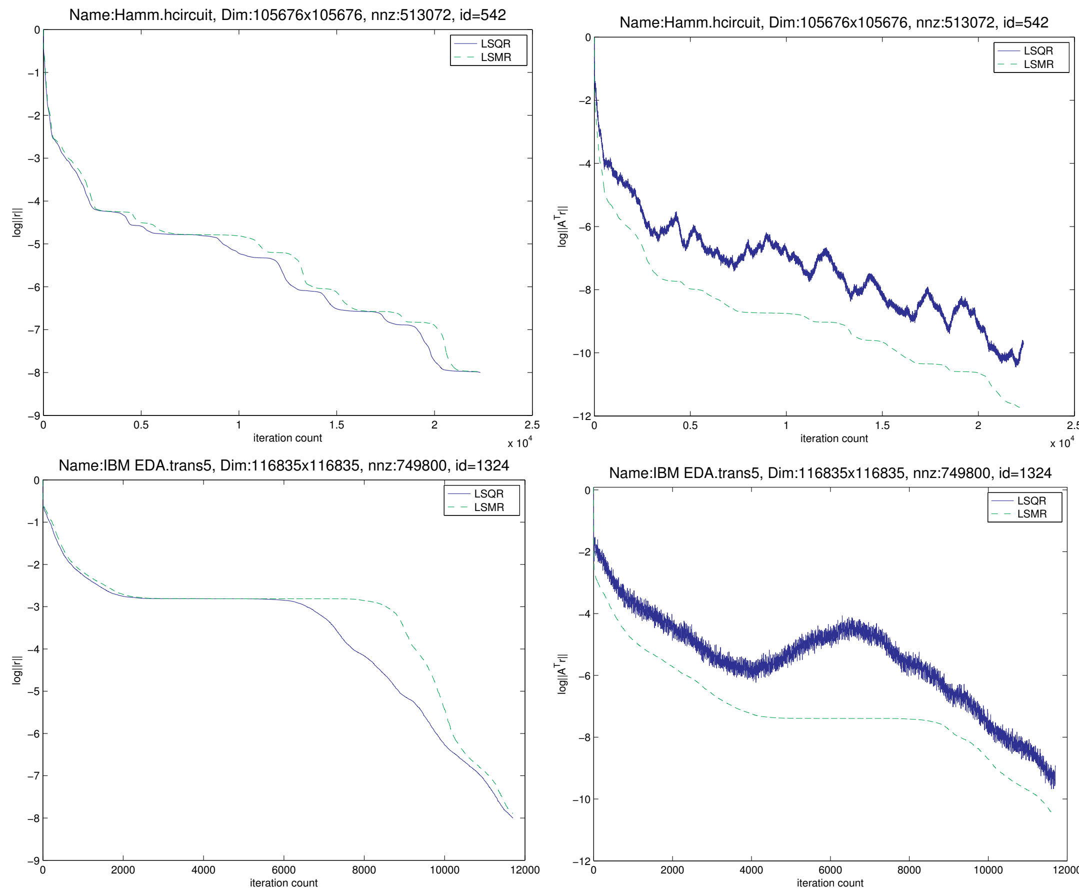


Figure 4.11: LSQR and LSMR solving two square nonsingular systems $Ax = b$: problems Hamm/hcircuit (top) and IBM_EDA/trans5 (bottom). Left: $\log_{10} \|r_k\|$ for both solvers, with prolonged plateaus for LSMR. Right: $\log_{10} \|A^T r_k\|$ (preferable for LSMR).

Table 4.2 Relationship between CG, MINRES, CRAIG, LSQR and LSMR

CRAIG	$\equiv$	CG on $AA^Ty = b$, $x = A^Ty$
LSQR	$\equiv$	CG on $A^TAx = A^Tb$
	$\equiv$	MINRES on $AA^Ty = b$, $x = A^Ty$
LSMR	$\equiv$	MINRES on $A^TAx = A^Tb$

Proof. It suffices to show that the two methods are solving the same subproblem at every iteration. Let x_k^{MINRES} and x_k^{LSQR} be the iterates generated by MINRES and LSQR respectively. Then

$$\begin{aligned} x_k^{\text{MINRES}} &= A^T \text{argmin}_{y \in \mathcal{K}_k(AA^T, b)} \left\|b - AA^Ty\right\| \\ &= \text{argmin}_{x \in \mathcal{K}_k(A^TA, A^Tb)} \|b - Ax\| \\ &= x_k^{\text{LSQR}}. \end{aligned}$$

The first and third equality comes from the subproblems that MINRES and LSQR solve. The second equality follows from the following mapping from points in $\mathcal{K}_k(AA^T, b)$ to $\mathcal{K}_k(A^TA, A^Tb)$:

$$f : \mathcal{K}_k(AA^T, b) \to \mathcal{K}_k(A^TA, A^Tb), \qquad f(y) = A^Ty,$$

and from the fact that for any point $x \in \mathcal{K}_k(A^TA, A^Tb)$, we can write

$$x = \gamma_0 A^Tb + \sum_{i=1}^{k} \gamma_i (A^TA)^i (A^Tb)$$

for some scalars $\{\gamma_i\}_{i=0}^k$. Then the point

$$y = \gamma_0 b + \sum_{i=1}^{k} \gamma_i (AA^T)^i b$$

would be a preimage of x under f. $\square$

This relationship, as well as some other well known ones between CG, MINRES, CRAIG, LSQR and LSMR, are summarized in Table 4.2.

Backward error for underdetermined systems

A linear system $Ax = b$ is ill-posed if A is m-by-n and $m < n$, because the system has an infinite number of solutions (or none). One way to define a unique solution for such a system is to choose the solution

MATLAB Code 4.6 Left diagonal preconditioning

```
% scale the row norms to 1
rnorms = sqrt(full(sum(A.*A,2)));
rnorms = rnorms + (rnorms == 0); % avoid division by 0
D = diag(sparse(1./rnorms));
A = D*A;
b = D*b;
```

x with minimum 2-norm. That is, we want to solve the optimization problem

$$\min_{Ax=b} \|x\|_2.$$

For any approximate solution x to above problem, the *normwise backward error* is defined as the norm of the minimum perturbation to A such that x is a solution of the perturbed optimization problem:

$$\eta(x) = \min_E \|E\| \qquad \text{s.t.} \qquad x = \operatorname{argmin}_{(A+E)x=b} \|x\|.$$

Sun and Sun [39] have shown that this value is given by

$$\eta(x) = \sqrt{\frac{\|r\|_2^2}{\|x\|_2^2} + \sigma_{\min}^2(B)}, \qquad B = A\left(I - \frac{xx^T}{\|x\|_2^2}\right).$$

Since it is computationally prohibitive to compute the minimum singular value at every iteration for the backward error, we will use $\|r\|/\|x\|$ as an approximate backward error in the following analysis.[1]

[1] Note that this estimate is a lower bound on the true backward error. In contrast, the estimates E_1 and E_2 for backward error in least-squares problems are upper bounds.

PRECONDITIONING

For underdetermined systems, a right preconditioner on A will alter the minimum-norm solution. Therefore, only left preconditioners are applicable. In the following experiments, we do a left diagonal preconditioning on A by scaling the rows of A to unit 2-norm; see MATLAB Code 4.6.

DATA

For testing underdetermined systems, we use sparse matrices from the `LPnetlib` group (the same set of data as in Section 4.1).

Each example was downloaded in MATLAB format, and a sparse matrix A and dense vector b were extracted from the data structure via `A = Problem.A` and `b = Problem.b` Then we solve an underdetermined linear system $\min_{Ax=b} \|x\|_2$ with both LSQR and LSMR. MINRES is also used with a change of variable to a form equivalent to LSQR as described above.

Numerical results

The experimental results showed that LSMR converges almost as quickly as LSQR for underdetermined systems. The approximate backward errors for four different problems are shown in Figure 4.12. In only a few cases, LSMR lags behind LSQR for a number of iterations. Thus we conclude that LSMR and LSQR are equally good for finding minimum 2-norm solutions for underdetermined systems.

The experimental results also confirmed our earlier derivation that MINRES on $AA^Ty = b$ and $x = A^Ty$ is equivalent to LSQR on $Ax = b$. MINRES exhibits the same convergence behavior as LSQR, except in cases where they both take more than m iterations to converge. In these cases, the effect of increased condition number of AA^T kicks in and slows down MINRES in the later iterations.

4.4 Reorthogonalization

It is well known that Krylov-subspace methods can take arbitrarily many iterations because of loss of orthogonality. For the Golub-Kahan bidiagonalization, we have two sets of vectors U_k and V_k. As an experiment, we implemented the following options in LSMR:

1. No reorthogonalization.
2. Reorthogonalize V_k (i.e. reorthogonalize v_k with respect to V_{k-1}).
3. Reorthogonalize U_k (i.e. reorthogonalize u_k with respect to U_{k-1}).
4. Both 2 and 3.

Each option was tested on all of the over-determined test problems with fewer than 16K nonzeros. Figure 4.13 shows an "easy" case in which all options converge equally well (convergence before significant loss of orthogonality), and an extreme case in which reorthogonalization makes a large difference.

Unexpectedly, options 2, 3, and 4 proved to be indistinguishable in all cases. To look closer, we forced LSMR to take n iterations. Option 2 (with V_k orthonormal to machine precision ϵ) was found to be keeping

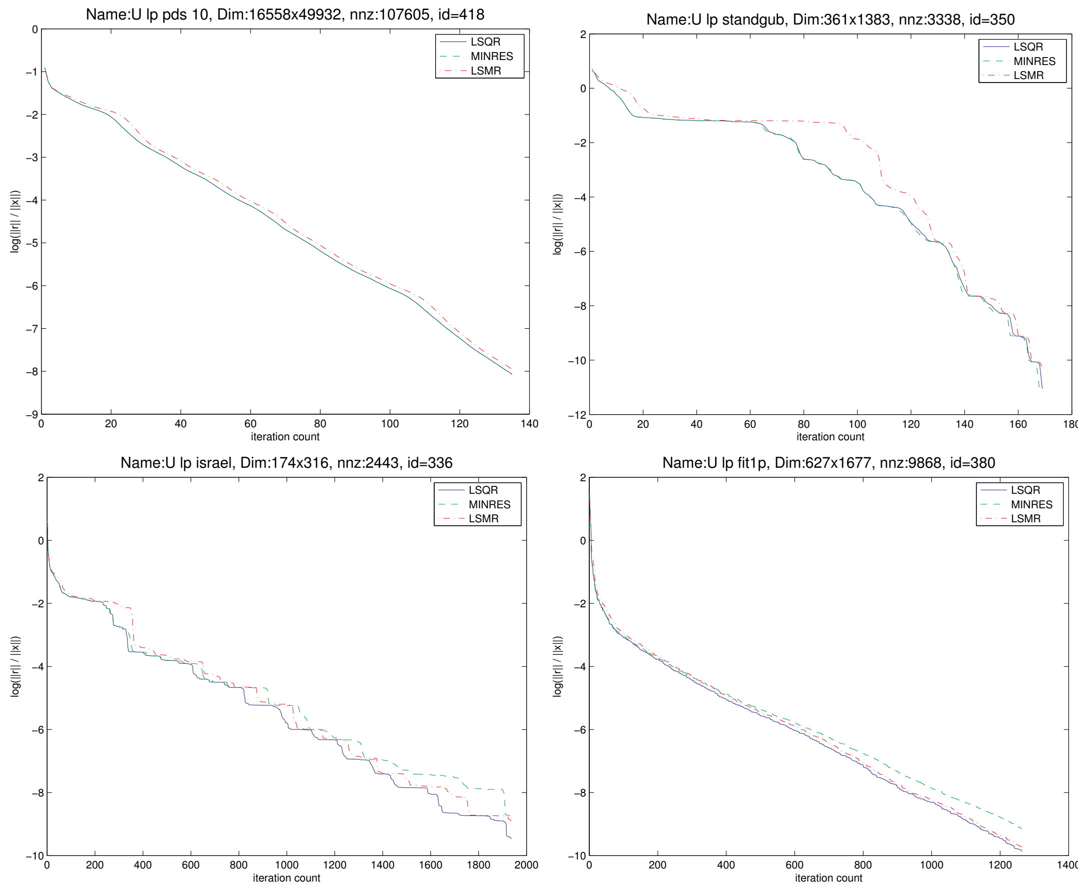


Figure 4.12: The backward errors $\|r_k\|/\|x_k\|$ for LSQR, LSMR and MINRES on four different underdetermined linear systems to find the minimum 2-norm solution. **Upper left**: The backward errors for all three methods converge at a similar rate. Most of our test cases exhibit similar convergence behavior. This shows that LSMR and LSQR perform equally well for underdetermined systems. **Upper right**: A rare case where LSMR lags behind LSQR significantly for some iterations. This plot also confirms our earlier derivation that this special version of MINRES is theoretically equivalent to LSQR, as shown by the almost identical convergence behavior. **Lower left**: An example where all three algorithms take more than m iterations. Since MINRES works with the operator AA^T, the effect of numerical error is greater and MINRES converges slower than LSQR towards the end of computation. **Lower right**: Another example showing that MINRES lags behind LSQR because of greater numerical error.

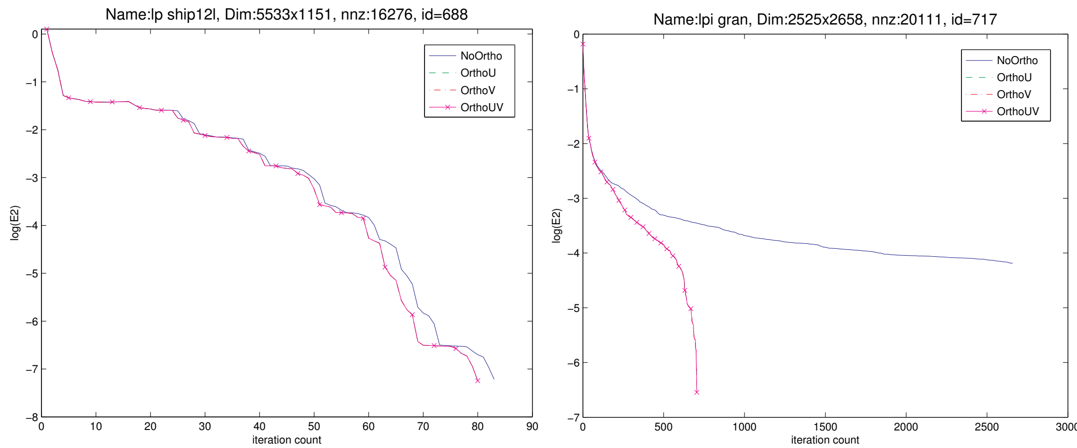


Figure 4.13: LSMR with and without reorthogonalization of V_k and/or U_k. Left: An easy case where all options perform similarly (problem `lp_ship12l`). Right: A helpful case (problem `lp_gran`).

U_k orthonormal to at least $O(\sqrt{\epsilon})$. Option 3 (with U_k orthonormal) was not quite as effective but it kept V_k orthonormal to at least $O(\sqrt{\epsilon})$ up to the point where LSMR would terminate when ATOL $= \sqrt{\epsilon}$.

This effect of one-sided reorthogonalization has also been pointed out in [65].

Note that for square or rectangular A with exact arithmetic, LSMR is equivalent to MINRES on the normal equation (and hence to CR [44] and GMRES [63] on the same equation). Reorthogonalization makes the equivalence essentially true in practice. We now focus on reorthogonalizing V_k but not U_k.

Other authors have presented numerical results involving reorthogonalization. For example, on some randomly generated LS problems of increasing condition number, Hayami *et al.* [37] compare their BA-GMRES method with an implementation of CGLS (equivalent to LSQR [53]) in which V_k is reorthogonalized, and find that the methods require essentially the same number of iterations. The preconditioner chosen for BA-GMRES made that method equivalent to GMRES on $A^TAx = A^Tb$. Thus, GMRES without reorthogonalization was seen to converge essentially as well as CGLS or LSQR with reorthogonalization of V_k (option 2 above). This coincides with the analysis by Paige *et al.* [55], who conclude that MGS-GMRES does not need reorthogonalization of the Arnoldi vectors V_k.

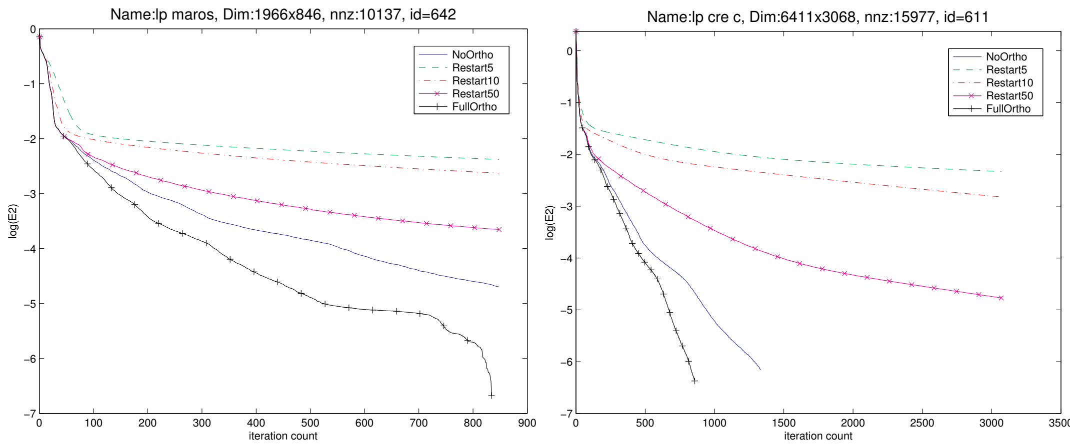


Figure 4.14: LSMR with reorthogonalized V_k and restarting. Restart(l) with $l = 5, 10, 50$ is slower than standard LSMR with or without reorthogonalization. NoOrtho represents LSMR without reorthogonalization. Restart5, Restart10, and Restart50 represents LSMR with V_k reorthogonalized and with restarting every 5, 10 or 50 iterations. FullOrtho represents LSMR with V_k reorthogonalized without restarting. Problems `lp_maros` and `lp_cre_c`.

RESTARTING

To conserve storage, a simple approach is to restart the algorithm every l steps, as with GMRES(l) [63]. To be precise, we set

$$r_l = b - Ax_l, \qquad \min \|A\Delta x - r_l\|, \qquad x_l \leftarrow x_l + \Delta x$$

and repeat the same process until convergence. Our numerical test in Figure 4.14 shows that restarting LSMR even with full reorthogonalization (of V_k) may lead to stagnation. In general, convergence with restarting is much slower than LSMR without reorthogonalization. Restarting does not seem useful in general.

LOCAL REORTHOGONALIZATION

Here we reorthogonalize each new v_k with respect to the previous l vectors, where l is a specified parameter. Figure 4.15 shows that $l = 5$ has little effect, but partial speedup was achieved with $l = 10$ and 50 in the two chosen cases. There is evidence of a useful storage-time tradeoff. It should be emphasized that the potential speedup depends strongly on the computational cost of Av and $A^T u$. If these are cheap, local reorthogonalization may not be worthwhile.

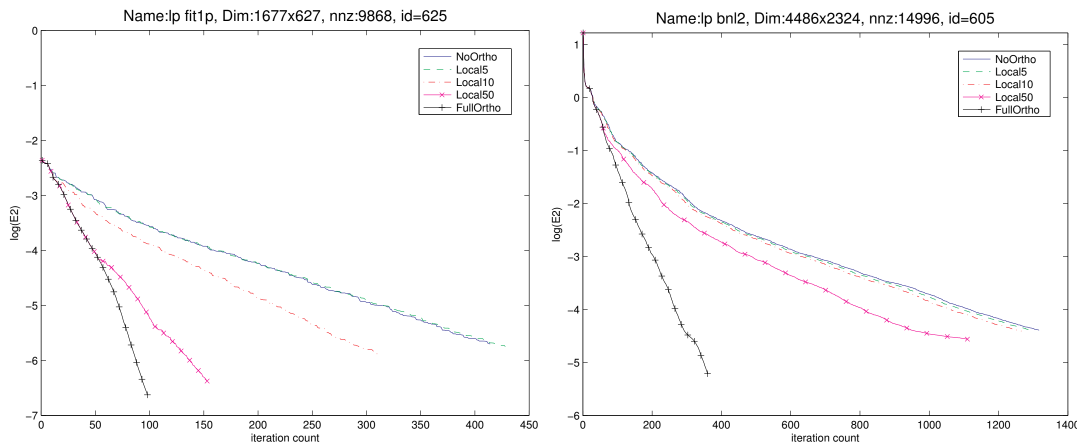


Figure 4.15: LSMR with local reorthogonalization of V_k. NoOrtho represents LSMR without reorthogonalization. Local5, Local10, and Local50 represent LSMR with local reorthogonalization of each v_k with respect to the previous 5, 10, or 50 vectors. FullOrtho represents LSMR with reorthogonalized V_k without restarting. Local(l) with $l = 5, 10, 50$ illustrates reduced iterations as l increases. Problems `lp_fit1p` and `lp_bnl2`.

5 AMRES

In this chapter we describe an efficient and stable iterative algorithm for computing the vector x in the augmented system

$$\begin{pmatrix} \gamma I & A \\ A^T & \delta I \end{pmatrix} \begin{pmatrix} s \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}, \tag{5.1}$$

where A is a rectangular matrix, γ and δ are any scalars, and we define

$$\hat{A} = \begin{pmatrix} \gamma I & A \\ A^T & \delta I \end{pmatrix}, \qquad \widehat{x} = \begin{pmatrix} s \\ x \end{pmatrix}, \qquad \hat{b} = \begin{pmatrix} b \\ 0 \end{pmatrix}. \tag{5.2}$$

Our algorithm is called AMRES, for Augmented-system Minimum RESidual method. It is derived by formally applying MINRES [52] to the augmented system (5.1), but is more economical because it is based on the Golub-Kahan bidiagonalization process [29] and it computes estimates of just x (excluding s).

Note that $\hat{A}$ includes two scaled identity matrices γI and δI in the (2,2)-block. When γ and δ have opposite sign (e.g., $\gamma = \sigma, \delta = -\sigma$), (5.1) is equivalent to a damped least-squares problem

$$\min \left\| \begin{pmatrix} A \\ \sigma I \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|_2 \quad \equiv \quad (A^T A + \sigma^2 I)x = A^T b$$

(also known as a Tikhonov regularization problem). CGLS, LSQR or LSMR may then be applied. They require less work and storage per iteration than AMRES, but the number of iterations and the numerical reliability of all three algorithms would be similar (especially for LSMR).

AMRES is intended for the case where γ and δ have the *same* sign (e.g., $\gamma = \delta = \sigma$). An important application is for the solution of "negatively damped normal equations" of the form

$$(A^T A - \sigma^2 I)x = A^T b, \tag{5.3}$$

which arise from both ordinary and regularized total least-squares (TLS) problems, Rayleigh quotient iteration (RQI) and Curtis-Reid scaling. These equations do not lend themselves to an equivalent least-squares formulation. Hence, there is a need for algorithms specially tailored to the problem. We note in passing that there is a *formal* least-squares formulation for negative shifts:

$$\min \left\| \begin{pmatrix} A \\ i\sigma I \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|_2, \qquad i = \sqrt{-1},$$

but this doesn't lead to a convenient numerical algorithm for large-scale problems.

For small enough values of σ^2, the matrix $A^TA - \sigma^2 I$ is positive definite (if A has full column rank), and this is indeed the case with ordinary TLS problems. However, for regularized TLS problems (where σ^2 plays the role of a regularization parameter that is not known *a priori* and must be found by solving system (5.3) for a range of σ-values), there is no guarantee that $A^TA - \sigma^2 I$ will be positive definite. With a minor extension, CGLS may be applied with any shift [6] (and this could often be the most efficient algorithm), but when the shifted matrix is indefinite, stability cannot be guaranteed.

AMRES is reliable for any shift, even if $A^TA - \sigma^2 I$ is indefinite. Also, if σ happens to be a singular value of A (extreme or interior), AMRES may be used to compute a corresponding singular vector in the manner of inverse iteration,[1] as described in detail in Section 6.3.

[1] Actually, by solving a singular least-squares problem, as in Choi [13].

5.1 Derivation of AMRES

If the Lanczos process Tridiag$(\hat{A}, \hat{b})$ were applied to the matrix and rhs in (5.1), we would have

$$\hat{T}_{2k} = \begin{pmatrix} \gamma & \alpha_1 & & & & \\ \alpha_1 & \delta & \beta_2 & & & \\ & \beta_2 & \gamma & \alpha_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \beta_k & \gamma & \alpha_k \\ & & & & \alpha_k & \delta \end{pmatrix},$$
$$\hat{V}_{2k} = \begin{pmatrix} u_1 & & u_2 & & \dots & u_k & \\ & v_1 & & v_2 & \dots & & v_k \end{pmatrix}, \tag{5.4}$$

and then $\hat{T}_{2k+1}$, $\hat{V}_{2k+1}$ in the obvious way. Because of the structure of $\hat{A}$ and $\hat{b}$, the scalars α_k, β_k and vectors u_k, v_k are independent of γ and δ, and it is more efficient to generate the same quantities by the Golub-Kahan process Bidiag(A, b). To solve (5.1), MINRES would solve the subproblems

$$\min \left\| \hat{H}_k y_k - \beta_1 e_1 \right\|, \qquad \hat{H}_k = \begin{cases} \begin{pmatrix} \hat{T}_k \\ \alpha_{\frac{k+1}{2}} e_k^T \end{pmatrix} & (k \text{ odd}) \\ \begin{pmatrix} \hat{T}_k \\ \beta_{\frac{k}{2}+1} e_k^T \end{pmatrix} & (k \text{ even}) \end{cases}$$

5.1.1 Least-squares subsystem

If $\gamma = \delta = \sigma$, a singular value of A, the matrix $\hat{A}$ is singular. In general we wish to solve $\min_{\widehat{x}} \|\hat{A}\widehat{x} - \hat{b}\|$, where $\widehat{x}$ may not be unique. We define the k-th estimate of $\widehat{x}$ to be $\widehat{x}_k = \hat{V}_k \hat{y}_k$, and then

$$\widehat{r}_k \equiv \hat{b} - \hat{A}\widehat{x}_k = \hat{b} - \hat{A}\hat{V}_k\hat{y}_k = \hat{V}_{k+1}\left(\beta_1 e_1 - \hat{H}_k \hat{y}_k\right). \tag{5.5}$$

To minimize the residual $\widehat{r}_k$, we perform a QR factorization on $\hat{H}_k$:

$$\begin{aligned} \min \|\widehat{r}_k\| &= \min \|\beta_1 e_1 - \hat{H}_k \hat{y}_k\| \\ &= \min \left\| Q_{k+1}^T \left(\begin{pmatrix} z_k \\ \bar{\zeta}_{k+1} \end{pmatrix} - \begin{pmatrix} R_k \\ 0 \end{pmatrix} \hat{y}_k \right) \right\| & (5.6) \\ &= \min \left\| \begin{pmatrix} z_k \\ \bar{\zeta}_{k+1} \end{pmatrix} - \begin{pmatrix} R_k \\ 0 \end{pmatrix} \hat{y}_k \right\|, & (5.7) \end{aligned}$$

where

$$z_k^T = (\zeta_1, \zeta_2, \ldots, \zeta_k), \qquad Q_{k+1} = P_k \cdots P_2 P_1, \tag{5.8}$$

with Q_{k+1} being a product of plane rotations. At iteration k, P_l denotes a plane rotation on rows l and $l+1$ (with $k \geq l$):

$$P_l = \begin{pmatrix} I_{l-1} & & & \\ & c_l & s_l & \\ & -s_l & c_l & \\ & & & I_{k-l} \end{pmatrix}.$$

The solution $\hat{y}_k$ to (5.7) satisfies $R_k \hat{y}_k = z_k$. As in MINRES, to allow a cheap iterative update to $\widehat{x}_k$ we define $\hat{W}_k$ to be the solution of

$$R_k^T \hat{W}_k^T = \hat{V}_k^T, \tag{5.9}$$

which gives us

$$\widehat{x}_k = \hat{V}_k \hat{y}_k = \hat{W}_k R_k \hat{y}_k = \hat{W}_k z_k. \tag{5.10}$$

Since we are interested in the lower half of $\widehat{x}_k = \left(\begin{smallmatrix} s_k \\ x_k \end{smallmatrix}\right)$, we write $\hat{W}_k$ as

$$\hat{W}_k = \begin{pmatrix} W_k^u \\ W_k^v \end{pmatrix}, \quad W_k^u = \begin{pmatrix} w_1^u & w_2^u & \cdots & w_k^u \end{pmatrix}, \quad W_k^v = \begin{pmatrix} w_1^v & w_2^v & \cdots & w_k^v \end{pmatrix}.$$

Then (5.10) can be simplified as $x_k = W_k^v z_k = x_{k-1} + \zeta_k w_k^v$.

5.1.2 QR FACTORIZATION

The effects of the first two rotations in (5.8) are shown here:

$$\begin{pmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{pmatrix} \begin{pmatrix} \gamma & \alpha_1 & \\ \alpha_1 & \delta & \beta_2 \end{pmatrix} = \begin{pmatrix} \rho_1 & \theta_1 & \eta_1 \\ & \bar{\rho}_2 & \bar{\theta}_2 \end{pmatrix}, \tag{5.11}$$

$$\begin{pmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{pmatrix} \begin{pmatrix} \bar{\rho}_2 & \bar{\theta}_2 & \\ \beta_2 & \gamma & \alpha_2 \end{pmatrix} = \begin{pmatrix} \rho_2 & \theta_2 & \eta_2 \\ & \bar{\rho}_3 & \bar{\theta}_3 \end{pmatrix}. \tag{5.12}$$

Later rotations are shown in Algorithm AMRES.

5.1.3 UPDATING W_k^v

Since we are only interested in W_k^v, we can extract it from (5.9) to get

$$R_k^T (W_k^v)^T = \begin{cases} \begin{pmatrix} 0 & v_1 & 0 & v_2 & \cdots & v_{\frac{k-1}{2}} & 0 \end{pmatrix}^T & (k \text{ odd}) \\ \begin{pmatrix} 0 & v_1 & 0 & v_2 & \cdots & 0 & v_{\frac{k}{2}} \end{pmatrix}^T & (k \text{ even}) \end{cases}$$

where the structure of the rhs comes from (5.4). Since R_k^T is lower tridiagonal, the system can be solved by the following recurrences:

$$w_1^v = 0, \qquad w_2^v = v_1/\rho_2$$

$$w_k^v = \begin{cases} (-\eta_{k-2} w_{k-2}^v - \theta_{k-1} w_{k-1}^v)/\rho_k. & (k \text{ odd}) \\ (v_{\frac{k}{2}} - \eta_{k-2} w_{k-2}^v - \theta_{k-1} w_{k-1}^v)/\rho_k. & (k \text{ even}) \end{cases}$$

If we define $h_k = \rho_k w_k^v$, then we arrive at the update rules in step 7 of Algorithm 5.1, which saves $2n$ floating point multiplication for each step of Golub-Kahan bidiagonalization compared with updating w_k^v.

5.1.4 ALGORITHM AMRES

Algorithm 5.1 summarizes the main steps of AMRES, excluding the norm estimates and stopping rules developed later. As usual, $\beta_1 u_1 = b$ is shorthand for $\beta_1 = \|b\|$, $u_1 = b/\beta_1$ (and similarly for all α_k, β_k).

5.2 STOPPING RULES

Stopping rules analogous to those in LSQR [53] are used for AMRES. Three dimensionless quantities are needed: ATOL, BTOL, CONLIM. The first stopping rule applies to compatible systems, the second rule applies to incompatible systems, and the third rule applies to both.

S1: Stop if $\|r_k\| \leq \text{BTOL}\|b\| + \text{ATOL}\|A\|\|x_k\|$

S2: Stop if $\|\hat{A}^T\widehat{r}_k\| \leq \|\hat{A}\|\|\widehat{r}_k\|\text{ATOL}$

S3: Stop if $\text{cond}(A) \geq \text{CONLIM}$

5.3 ESTIMATE OF NORMS

5.3.1 COMPUTING $\|\widehat{r}_k\|$

From (5.7), it's obvious that $\|\widehat{r}_k\| = |\bar{\zeta}_{k+1}|$.

5.3.2 COMPUTING $\|\hat{A}\widehat{r}_k\|$

Starting from (5.5) and using (5.6), we have

$$\begin{aligned}
\hat{A}\widehat{r}_k &= \hat{A}\hat{V}_{k+1}\left(\beta_1 e_1 - \hat{H}_k\hat{y}_k\right) = \hat{V}_{k+2}H_{k+1}\left(\beta_1 e_1 - \hat{H}_k\hat{y}_k\right)\\
&= \hat{V}_{k+2}H_{k+1}Q_{k+1}^T\left(\begin{pmatrix} z_k \\ \hat{\zeta}_{k+1}\end{pmatrix} - \begin{pmatrix} R_k \\ 0\end{pmatrix}\hat{y}_k\right)\\
&= \hat{V}_{k+2}H_{k+1}Q_{k+1}^T\begin{pmatrix} 0 \\ \hat{\zeta}_{k+1}\end{pmatrix} = \hat{V}_{k+2}\begin{pmatrix} R_k^T & 0 \\ \theta_k e_k^T & \bar{\rho}_{k+1} \\ \eta_k e_k^T & \bar{\theta}_{k+1}\end{pmatrix}\begin{pmatrix} 0 \\ \hat{\zeta}_{k+1}\end{pmatrix}\\
&= \hat{\zeta}_{k+1}\hat{V}_{k+2}\begin{pmatrix} 0 \\ \bar{\rho}_{k+1} \\ \bar{\theta}_{k+1}\end{pmatrix} = \hat{\zeta}_{k+1}(\bar{\rho}_{k+1}\hat{v}_{k+1} + \bar{\theta}_{k+1}\hat{v}_{k+2}).
\end{aligned}$$

Therefore, $\|\hat{A}\widehat{r}_k\| = |\hat{\zeta}_{k+1}|\left\|\begin{pmatrix}\bar{\rho}_{k+1} & \bar{\theta}_{k+1}\end{pmatrix}\right\|$.

Algorithm 5.1 Algorithm AMRES

1: (Initialize)

$$\begin{array}{llll} \beta_1 u_1 = b & \alpha_1 v_1 = A^T u_1 & \bar{\rho}_1 = \gamma & \bar{\theta}_1 = \alpha_1 \\ \bar{\zeta}_1 = \beta_1 & w^v_{-1} = \vec{0} & w^v_0 = \vec{0} & x_0 = \vec{0} \\ \eta_{-1} = 0 & \eta_0 = 0 & \theta_0 = 0 & \end{array}$$

2: **for** $l = 1, 2, 3 \ldots$ **do**

3: (Continue the bidiagonalization)

$$\beta_{l+1} u_{l+1} = A v_l - \alpha_l u_l \qquad \alpha_{l+1} v_{l+1} = A^T u_{l+1} - \beta_{l+1} v_l$$

4: **for** $k = 2l-1, 2l$ **do**

5: (Setup temporary variables)

$$\begin{cases} \lambda = \delta & \alpha = \alpha_l \qquad \beta = \beta_{l+1} \quad (k \text{ odd}) \\ \lambda = \gamma & \alpha = \beta_{l+1} \qquad \beta = \alpha_{l+1} \quad (k \text{ even}) \end{cases}$$

6: (Construct and apply rotation P_k)

$$\begin{array}{ll} \rho_k = \left(\bar{\rho}_k^2 + \alpha^2\right)^{\frac{1}{2}} & \\ c_k = \bar{\rho}_k / \rho_k & s_k = \alpha / \rho_k \\ \theta_k = c_k \bar{\theta}_k + s_k \lambda & \bar{\rho}_{k+1} = -s_k \bar{\theta}_k + c_k \lambda \\ \eta_k = s_k \beta & \bar{\theta}_{k+1} = c_k \beta \\ \zeta_k = c_k \bar{\zeta}_k & \bar{\zeta}_{k+1} = -s_k \bar{\zeta}_k \end{array}$$

7: (Update estimates of x)

$$h_k = \begin{cases} -(\eta_{k-2}/\rho_{k-2}) h_{k-2} - (\theta_{k-1}/\rho_{k-1}) h_{k-1} & (k \text{ odd}) \\ v_l - (\eta_{k-2}/\rho_{k-2}) h_{k-2} - (\theta_{k-1}/\rho_{k-1}) h_{k-1} & (k \text{ even}) \end{cases}$$

$$x_k = x_{k-1} + (\zeta_k / \rho_k) h_k$$

8: **end for**

9: **end for**

5.3.3 COMPUTING $\|\widehat{x}\|$

From (5.7) and 5.10, we know that $\widehat{x}_k = \hat{V}_k \hat{y}_k$ and $z_k = R_k \hat{y}_k$. If we do a QR factorization $\tilde{Q}_k R_k^T = \tilde{R}_k$ and let $t_k = \tilde{Q}_k \hat{y}_k$, we can write

$$z_k = R_k \hat{y}_k = R_k \tilde{Q}_k^T \tilde{Q}_k \hat{y}_k = \tilde{R}_k^T t_k, \tag{5.13}$$

giving $\widehat{x}_k = \hat{V}_k \tilde{Q}_k^T t_k$ and hence $\|\widehat{x}_k\| = \|t_k\|$, where $\|t_k\|$ is derived next.

For the QR factorizations (5.8) and (5.13), we write R_k and $\tilde{R}_k$ as

$$R_k = \begin{pmatrix} \rho_1 & \theta_1 & \eta_1 & & & \\ & \rho_2 & \theta_2 & \eta_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \rho_{k-2} & \theta_{k-2} & \eta_{k-2} \\ & & & & \rho_{k-1} & \theta_{k-1} \\ & & & & & \rho_k \end{pmatrix},$$

$$\tilde{R}_k = \begin{pmatrix} \tilde{\rho}_1^{(4)} & \tilde{\theta}_1^{(2)} & \tilde{\eta}_1^{(1)} & & & \\ & \tilde{\rho}_1^{(4)} & \tilde{\theta}_1^{(2)} & \tilde{\eta}_1^{(1)} & & \\ & & \ddots & \ddots & \ddots & \\ & & & \tilde{\rho}_{k-2}^{(4)} & \tilde{\theta}_{k-2}^{(2)} & \tilde{\eta}_{k-2}^{(1)} \\ & & & & \tilde{\rho}_{k-1}^{(3)} & \tilde{\theta}_{k-1}^{(1)} \\ & & & & & \tilde{\rho}_k^{(2)} \end{pmatrix},$$

where the upper index denotes the number of times that element has changed during the decompositions. Also, $\tilde{Q}_k = (\tilde{P}_k \hat{P}_k) \cdots (\tilde{P}_3 \hat{P}_3) \tilde{P}_2$, where $\hat{P}_k$ and $\tilde{P}_k$ are constructed by changing the $(k-2:k) \times (k-2:k)$ submatrix of I_k to

$$\begin{pmatrix} \tilde{c}_k^{(1)} & & \tilde{s}_k^{(1)} \\ & 1 & \\ -\tilde{s}_k^{(1)} & & \tilde{c}_k^{(1)} \end{pmatrix} \text{ and } \begin{pmatrix} 1 & & \\ & \tilde{c}_k^{(2)} & \tilde{s}_k^{(2)} \\ & -\tilde{s}_k^{(2)} & \tilde{c}_k^{(2)} \end{pmatrix}$$

respectively. The effects of these two rotations can be summarized as

$$\begin{pmatrix} \tilde{c}_k^{(1)} & \tilde{s}_k^{(1)} \\ -\tilde{s}_k^{(1)} & \tilde{c}_k^{(1)} \end{pmatrix} \begin{pmatrix} \tilde{\rho}_{k-2}^{(3)} & \tilde{\theta}_{k-2}^{(1)} & \\ \eta_{k-2} & \theta_{k-1} & \rho_k \end{pmatrix} = \begin{pmatrix} \tilde{\rho}_{k-2}^{(4)} & \tilde{\theta}_{k-2}^{(2)} & \tilde{\eta}_{k-2}^{(1)} \\ 0 & \dot{\theta}_{k-1} & \tilde{\rho}_k^{(1)} \end{pmatrix}$$

$$\begin{pmatrix} \tilde{c}_k^{(2)} & \tilde{s}_k^{(2)} \\ -\tilde{s}_k^{(2)} & \tilde{c}_k^{(2)} \end{pmatrix} \begin{pmatrix} \tilde{\rho}_{k-1}^{(2)} & \\ \dot{\theta}_{k-1} & \tilde{\rho}_k^{(1)} \end{pmatrix} = \begin{pmatrix} \tilde{\rho}_{k-1}^{(3)} & \tilde{\theta}_{k-1}^{(1)} \\ 0 & \tilde{\rho}_k^{(2)} \end{pmatrix}.$$

Let $t_k = \begin{pmatrix} \tau_1^{(3)} & \dots & \tau_{k-2}^{(3)} & \tau_{k-1}^{(2)} & \tau_k^{(1)} \end{pmatrix}^T$. We solve for t_k by

$$\begin{aligned}
\tau_{k-2}^{(3)} &= (\zeta_{k-2} - \tilde{\eta}_{k-4}^{(1)} \tau_{k-4}^{(3)} - \tilde{\theta}_{k-3}^{(2)} \tau_{k-3}^{(3)}) / \tilde{\rho}_{k-2}^{(4)} \\
\tau_{k-1}^{(2)} &= (\zeta_{k-1} - \tilde{\eta}_{k-3}^{(1)} \tau_{k-3}^{(3)} - \tilde{\theta}_{k-2}^{(2)} \tau_{k-2}^{(3)}) / \tilde{\rho}_{k-1}^{(3)} \\
\tau_k^{(1)} &= (\zeta_k - \tilde{\eta}_{k-2}^{(1)} \tau_{k-2}^{(3)} - \tilde{\theta}_{k-1}^{(1)} \tau_{k-1}^{(2)}) / \tilde{\rho}_k^{(2)}
\end{aligned}$$

with our estimate of $\|\widehat{x}\|$ obtained from

$$\|\widehat{x}_k\| = \|t_k\| = \left\| \begin{pmatrix} \tau_1^{(3)} & \dots & \tau_{k-2}^{(3)} & \tau_{k-1}^{(2)} & \tau_k^{(1)} \end{pmatrix} \right\|.$$

5.3.4 Estimates of $\|\hat{A}\|$ and cond($\hat{A}$)

Using Lemma 2.32 from Choi [13], we have the following estimates of $\|\hat{A}\|$ at the l-th iteration:

$$\|\hat{A}\| \geq \max_{1 \leq i \leq l} (\alpha_i^2 + \delta^2 + \beta_{i+1}^2)^{\frac{1}{2}}, \tag{5.14}$$

$$\|\hat{A}\| \geq \max_{2 \leq i \leq l} (\beta_i^2 + \gamma^2 + \alpha_i^2)^{\frac{1}{2}}. \tag{5.15}$$

Stewart [71] showed that the minimum and maximum singular values of $\hat{H}_k$ bound the diagonal elements of $\tilde{R}_k$. Since the extreme singular values of $\hat{H}_k$ estimate those of $\hat{A}$, we have an approximation (which is also a lower bound) for the condition number of $\hat{A}$:

$$\text{cond}(\hat{A}) \approx \frac{\max(\tilde{\rho}_1^{(4)}, \dots, \tilde{\rho}_{k-2}^{(4)}, \tilde{\rho}_{k-1}^{(3)}, \tilde{\rho}_k^{(2)})}{\min(\tilde{\rho}_1^{(4)}, \dots, \tilde{\rho}_{k-2}^{(4)}, \tilde{\rho}_{k-1}^{(3)}, \tilde{\rho}_k^{(2)})}. \tag{5.16}$$

5.4 Complexity

The storage and computational of cost at each iteration of AMRES are shown in Table 5.1.

Table 5.1 Storage and computational cost for AMRES

	Storage		Work	
	m	n	m	n
AMRES	Av, u	x, v, h_{k-1}, h_k	3	9
and the cost to compute products Av and $A^T u$				

6

AMRES APPLICATIONS

In this chapter we discuss a number of applications for AMRES. Section 6.1 describes its use for Curtis-Reid scaling, a commonly used method for scaling sparse rectangular matrices such as those arising in linear programming problems. Section 6.2 applies AMRES to Rayleigh quotient iteration for computing singular vectors, and describes a modified version of RQI to allow reuse of the Golub-Kahan vectors. Section 6.3 describes a modified version of AMRES that allows singular vectors to be computed more quickly and reliably when the corresponding singular value is known.

6.1 CURTIS-REID SCALING

In linear programming problems, the constraint matrix is often preprocessed by a scaling procedure before the application of a solver. The goal of such scaling is to reduce the range of magnitudes of the nonzero matrix elements. This may improve the numerical behavior of the solver and reduce the number of iterations to convergence [21].

Scaling procedures multiply a given matrix A by a diagonal matrix on each side:

$$\bar{A} = RAC, \qquad R = \mathrm{diag}(\beta^{-r_i}), \qquad C = \mathrm{diag}(\beta^{-c_j}),$$

where A is m-by-n, $\beta > 1$ is an arbitrary base, and the vectors $r = \begin{pmatrix} r_1 & r_2 & \dots & r_m \end{pmatrix}^T$ and $c = \begin{pmatrix} c_1 & c_2 & \dots & c_n \end{pmatrix}^T$ represent the scaling factors in log-scale. One popular scaling objective is to make each nonzero element of the scaled matrix $\bar{A}$ approximately 1 in absolute value, which translates to the following system of equations for r and c:

$$\begin{aligned} |\bar{a}_{ij}| \approx 1 \qquad &\Rightarrow \qquad \beta^{-r_i}|a_{ij}|\beta^{-c_j} \approx 1 \\ &\Rightarrow \quad -r_i + \log_\beta |a_{ij}| - c_j \approx 0 \end{aligned}$$

Following the above objective, two methods of matrix-scaling have

MATLAB Code 6.1 Least-squares problem for Curtis-Reid scaling

```
[m n] = size(A);
[I J S] = find(A);
z = length(I);
M = sparse([1:z 1:z]', [I; J+m], ones(2*z,1));
d = log(abs(S));
```

been proposed in 1962 and 1971:

$$\min_{r_i, c_j} \max_{a_{ij} \neq 0} |\log_\beta |a_{ij}| - r_i - c_j| \qquad \text{Fulkerson \& Wolfe 1962 [28]} \tag{6.1}$$

$$\min_{r_i, c_j} \sum_{a_{ij} \neq 0} (\log_\beta |a_{ij}| - r_i - c_j)^2 \qquad \text{Hamming 1971 [36]} \tag{6.2}$$

6.1.1 CURTIS-REID SCALING USING CGA

The closed-form solution proposed by Hamming is restricted to a dense matrix A. Curtis and Reid extended Hamming's method to general A and designed a specialized version of CG (which we will call CGA) that allows the scaling to work efficiently for sparse matrices [17]. Experiments by Tomlin [77] indicate that Curtis-Reid scaling is superior to that of Fulkerson and Wolfe. We may describe the key ideas in Curtis-Reid scaling for matrices.

Let z be the number of nonzero elements in A. Equation (6.2) can be written in matrix notation as

$$\min_{r,c} \left\| M \begin{pmatrix} r \\ c \end{pmatrix} - d \right\|, \tag{6.3}$$

where M is a z-by-$(m+n)$ matrix and d is a z vector. Each row of M corresponds to a nonzero A_{ij}. The entire row is 0 except the i-th and $(j+m)$-th column is 1. The corresponding element in d is $\log |A_{ij}|$. M and d are defined by MATLAB Code 6.1.[1]

[1] In practice, matrix M is never constructed because it is more efficient to work with the normal equation directly, as described below.

The normal equation for the least-squares problem (6.3),

$$M^T M \begin{pmatrix} r \\ c \end{pmatrix} = M^T d,$$

can be written as

$$\begin{pmatrix} D_1 & F \\ F^T & D_2 \end{pmatrix} \begin{pmatrix} r \\ c \end{pmatrix} = \begin{pmatrix} s \\ t \end{pmatrix}, \tag{6.4}$$

MATLAB Code 6.2 Normal equation for Curtis-Reid scaling

```
F  = (abs(A)>0);
D1 = diag(sparse(sum(F,2)));    % sparse diagonal matrix
D2 = diag(sparse(sum(F,1)));    % sparse diagonal matrix

abslog = @(x) log(abs(x));
B      = spfun(abslog,A);
s      = sum(B,2);
t      = sum(B,1)';
```

where F has the same dimension and sparsity pattern as A, with every nonzero entry changed to 1, while D_1 and D_2 are diagonal matrices whose diagonal elements represent the number of nonzeros in each row and column of A respectively. s and t are defined by

$$B_{ij} = \begin{cases} \log|A_{ij}| & (A_{ij} \neq 0) \\ 0 & (A_{ij} = 0) \end{cases}, \qquad s_i = \sum_{j=1}^{n} B_{ij}, \qquad t_j = \sum_{i=1}^{m} B_{ij}.$$

The contruction of these submatrices is shown in MATLAB code 6.2.

Equation (6.4) is a consistent positive semi-definite system. Any solution to this system would suffice for scaling perpose as they all produce the same scaling.[2]

Reid [60] developed a specialized version of CG on matrices with Property A.[3] It takes advantage of the sparsity pattern that appear when CG is applied to matrices with Property A to reduce both storage and work. Curtis and Reid [17] applied CGA to (6.4) to find the scaling for matrix A. To compare the performance of CGA on (6.4) against AMRES, we implemented the CGA algorithm as described in [17]. The implementation shown in MATLAB Code 6.3.

[2] It is obvious that $\begin{pmatrix} \mathbb{1} \\ -\mathbb{1} \end{pmatrix}$ is a null vector, which follows directly from the definition of F, D_1 and D_2. If r and c solve (6.4), then $r + \alpha\mathbb{1}$ and $c - \alpha\mathbb{1}$ are also solutions for any α. This corresponds to the fact that RAC and $(\beta^\alpha R)A(\beta^{-\alpha}C)$ give the same scaled $\bar{A}$.

[3] A matrix A is said to have property A if there exist permutations P_1, P_2 such that

$$P_1^T A P_2 = \begin{pmatrix} D_1 & E \\ F & D_2 \end{pmatrix},$$

where D_1 and D_2 are diagonal matrices [87].

6.1.2 CURTIS-REID SCALING USING AMRES

In this section, we discuss how to transform equation (6.4) so that it can be solved by AMRES.

First, we apply a symmetric diagonal preconditioning with $\begin{pmatrix} D_1^{\frac{1}{2}} & \\ & D_2^{\frac{1}{2}} \end{pmatrix}$ as the preconditioner. Then, with the definitions

$$\bar{r} = D_1^{\frac{1}{2}} r, \quad \bar{c} = D_2^{\frac{1}{2}} c, \quad \bar{s} = D_1^{-\frac{1}{2}} s, \quad \bar{t} = D_2^{-\frac{1}{2}} t, \quad C = D_1^{-\frac{1}{2}} F D_2^{-\frac{1}{2}},$$

MATLAB Code 6.3 CGA for Curtis-Reid scaling. This is an implementation of the algorithm described in [17].

```
function [cscale, rscale] = crscaleCGA(A, tol)

% CRSCALE implements the Curtis-Reid scaling algorithm.
%  [cscale, rscale, normrv, normArv, normAv, normxv, condAv] = crscale(A, tol);
%
% Use of the scales:
%
% If C    = diag(sparse(cscale)),    R    = diag(sparse(rscale)),
%    Cinv = diag(sparse(1./cscale)), Rinv = diag(sparse(1./rscale)),
% then Cinv*A*Rinv should have nonzeros that are closer to 1 in absolute value.
%
% To apply the scales to a linear program,
%   min c'x  st  Ax = b,  l <= x <= u,
% we need to define "barred" quantities by the following relations:
%   A = R Abar C,  b = R bbar,  C cbar = c,
%   C l = lbar,    C u = ubar,  C x = xbar.
% This gives the scaled problem
%   min cbar'xbar  st  Abar xbar = bbar,  lbar <= xbar <= ubar.
%
%
% 03 Jun 2011: First version.
%              David Fong and Michael Saunders, ICME, Stanford University.

  E  = (abs(A)>0);                          % sparse if A is sparse
  rowSumE = sum(E,2);
  colSumE = sum(E,1);
  Minv = diag(sparse(1./(rowSumE + (rowSumE == 0))));
  Ninv = diag(sparse(1./(colSumE + (colSumE == 0))));
  [m n] = size(A);
  abslog = @(x) log(abs(x));
  Abar   = spfun(abslog,A);
  % make sure sigma and tau are of correct size even if some entries of A are 1
  sigma  = zeros(m,1) + sum(Abar,2);
  tau    = zeros(n,1) + sum(Abar,1)';

  itnlimit = 100;
  r1 = zeros(length(sigma),1);
  r2 = tau - E'*(Minv*sigma);
  c2 = zeros(length(tau),1);
  c = c2; e1 = 0; e = 0; q2 = 1;
  s2 = r2'*(Ninv*r2);

  for t = 1:itnlimit
    rm1 = r1; r = r2; em2 = e; em1 = e1;
    q = q2; s = s2; cm2 = c; c = c2;
    r1 = -(E*(Ninv*r) + em1 * rm1)/q;
    s1 = r1'*(Minv*r1);
    e = q * s1/s; q1 = 1 - e;
    c2 = c + (Ninv*r + em1*em2*(c-cm2))/(q*q1);
    cv(:,t+1) = c2;
    if s1 < 1e-10; break; end

    r2 = -(E'*(Minv*r1) + e * r)/q1;
    s2 = r2'*(Ninv*r2);
    e1 = q1*s2/s1; q2 = 1 - e1;
  end

  c   = c2;
  rho = Minv*(sigma - E*c);
  gamma    = c;
  rscale   = exp(rho);        cscale   = exp(gamma);
  rmax     = max(rscale);     cmax     = max(cscale);
  s        = sqrt(rmax/cmax);
  cscale   = cscale*s;        rscale   = rscale/s;
end % function crscale
```

(6.4) becomes

$$\begin{pmatrix} I & C \\ C^T & I \end{pmatrix} \begin{pmatrix} \bar{r} \\ \bar{c} \end{pmatrix} = \begin{pmatrix} \bar{s} \\ \bar{t} \end{pmatrix},$$

and with $\hat{c} = \bar{c} - \bar{t}$, we get the form required by AMRES:

$$\begin{pmatrix} I & C \\ C^T & I \end{pmatrix} \begin{pmatrix} \bar{r} \\ \hat{c} \end{pmatrix} = \begin{pmatrix} \bar{s} - C\bar{t} \\ 0 \end{pmatrix}.$$

6.1.3 Comparison of CGA and AMRES

We now apply CGA and AMRES as described in the previous two sections to find the scaling vectors for Curtis-Reid scaling for some matrices from the University of Florida Sparse Matrix Collection (Davis [18]). At the k-th iteration, we compute the estimate $r^{(k)}$, $c^{(k)}$ from the two algorithms, and use them to evaluate the objective function $f(r, c)$ from the least-squares problem in (6.2):

$$f_k = f(r^{(k)}, c^{(k)}) = \sum_{a_{ij} \neq 0} (\log_e |a_{ij}| - r_i^{(k)} - c_j^{(k)})^2. \tag{6.5}$$

We take the same set of problems as in Section 4.1. The `LPnetlib` group includes data for 138 linear programming problems. Each example was downloaded in MATLAB format, and a sparse matrix A was extracted from the data structure via `A = Problem.A`.

In Figure 6.1, we plot $\log_{10}(f_k - f^*)$ against iteration number k for each algorithm. f^*, the optimal objective value, is taken as the minimum objective value attained by running CGA or AMRES after some K iterations. From the results, we see that CGA and AMRES exhibit similar convergence behavior for many matrices. There are also several cases where AMRES converges rapidly at the early iterations, which is important as scaling vectors don't need to be computed to high precision.

6.2 Rayleigh quotient iteration

In this section, we focus on algorithms that improve an approximate singular value and singular vector. The algorithms are based on Rayleigh quotient iteration. In the next section, we focus on finding singular vectors when the corresponding singular value is known.

Rayleigh quotient iteration (RQI) is an iterative procedure developed by John William Strutt, third Baron Rayleigh, for his study on

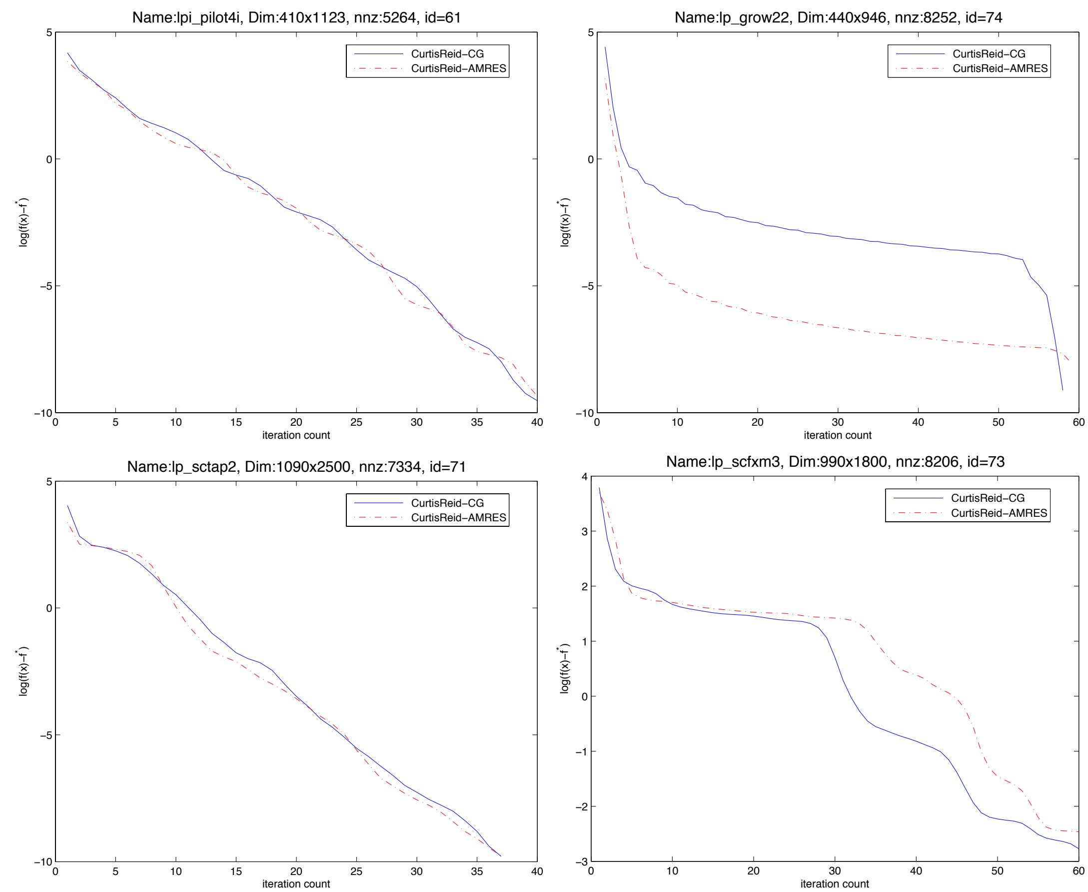


Figure 6.1: Convergence of CGA and AMRES when they are applied to the Curtis-Reid scaling problem. At each iteration, we plot the different between the current value and the optimal value for the objective function in (6.5). **Upper left and lower left**: Typical cases where CGA and AMRES exhibit similar convergence behavior. There is no obvious advantage of choosing one algorithm over the other in these cases. **Upper Right**: A less common case where AMRES converges much faster than CGA at the earlier iterations. This is important because scaling parameters doesn't need to be computed to high accuracy for most applications. AMRES could terminate early in this case. Similar convergence behavior was observed for several other problems. **Lower Right**: In a few cases such as this, CGA converges faster than AMRES during the later iterations.

Algorithm 6.1 Rayleigh quotient iteration (RQI) for square A

1: Given initial eigenvalue and eigenvector estimates ρ_0, x_0
2: **for** $q = 1, 2, \ldots$ **do**
3: Solve $(A - \rho_{q-1} I)t = x_{q-1}$
4: $x_q = t/\|t\|$
5: $\rho_q = x_q^T A x_q$
6: **end for**

Algorithm 6.2 RQI for singular vectors for square or rectangular A

1: Given initial singular value and right singular vector estimates ρ_0, x_0
2: **for** $q = 1, 2, \ldots$ **do**
3: Solve for t in

$$(A^T A - \rho_{q-1}^2 I)t = x_{q-1} \tag{6.6}$$

4: $x_q = t/\|t\|$
5: $\rho_q = \|Ax_q\|$
6: **end for**

the theory of sound [59]. The procedure is summarized in Algorithm 6.1. It improves an approximate eigenvector for a square matrix.

It has been proved by Parlett and Kahan [56] that RQI converges for almost all starting eigenvalue and eigenvector guesses, although in general it is unpredictable to *which* eigenpair RQI will converge. Ostrowski [49] showed that for a symmetric matrix A, RQI exhibits local cubic convergence. It has been shown that MINRES is preferable to SYMMLQ as the solver within RQI [20; 86]. Thus, it is natural to use AMRES, a MINRES-based solver, for extending RQI to compute singular vectors.

In this section, we focus on improving singular vector estimates for a general rectangular matrix A. RQI can be adapted to this task as in Algorithm 6.2. We will refer to the iterations in line 2 as outer iterations, and the iterations inside the solver for line 3 as inner iterations. The system in line 3 becomes increasingly ill-conditioned as the singular value estimate ρ_q converges to a singular value.

We continue our investigation by improving both the stability and the speed of Algorithm 6.2. Section 6.2.1 focuses on improving stability by solving an augmented system using AMRES. Section 6.2.2 improves the speed by a modification to RQI that allows AMRES to reuse precomputed Golub-Kahan vectors in subsequent iterations.

Algorithm 6.3 Stable RQI for singular vectors

1: Given initial singular value and right singular vector estimates ρ_0, x_0
2: **for** $q = 1, 2, \ldots$ **do**
3: Solve for t in
$$\begin{pmatrix} -\rho_{q-1}I & A \\ A^T & -\rho_{q-1}I \end{pmatrix} \begin{pmatrix} s \\ \bar{t} \end{pmatrix} = \begin{pmatrix} Ax_{q-1} \\ 0 \end{pmatrix}$$
4: $t = \bar{t} - x_{q-1}$
5: $x_q = t/\|t\|$
6: $\rho_q = \|Ax_q\|$
7: **end for**

6.2.1 Stable inner iterations for RQI

Compared with Algorithm 6.2, a more stable alternative would be to solve a larger augmented system. Note that line 3 in 6.2 is equivalent to
$$\begin{pmatrix} -\rho_{q-1}I & A \\ A^T & -\rho_{q-1}I \end{pmatrix} \begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} 0 \\ x_{q-1} \end{pmatrix}.$$
If we applied MINRES, we would get a more stable algorithm with twice the computational cost. A better alternative would be to convert it to a form that could be readily solved by AMRES. Since the solution will be normalized as in line 4, we are free to scale the right-hand side:
$$\begin{pmatrix} -\rho_{q-1}I & A \\ A^T & -\rho_{q-1}I \end{pmatrix} \begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} 0 \\ \rho_{q-1}x_{q-1} \end{pmatrix}.$$
We introduce the shift variable $\bar{t} = t + x_{q-1}$ to obtain the system
$$\begin{pmatrix} -\rho_{q-1}I & A \\ A^T & -\rho_{q-1}I \end{pmatrix} \begin{pmatrix} s \\ \bar{t} \end{pmatrix} = \begin{pmatrix} Ax_{q-1} \\ 0 \end{pmatrix}, \tag{6.7}$$
which is suitable for AMRES. This system has a smaller condition number compared with (6.6). Thus, we enjoy both the numerical stability of the larger augmented system and the lower computational cost of the smaller system. We summarize this approach in Algorithm 6.3.

Test data

We describe procedures to construct a linear operator A with known singular value and singular vectors. This method is adapted from [53].

1. For any $m \geq n$ and $\mathcal{C} \geq 1$, pick vectors satisfying

$$y^{(i)} \in \mathbf{R}^m, \quad \|y^{(i)}\| = 1, \quad (1 \leq i \leq \mathcal{C})$$
$$z \in \mathbf{R}^n, \quad \|z\| = 1,$$

   for constructing Householder reflectors. The parameter $\mathcal{C}$ represents the number of Householder reflectors that are used to construct the left singular vectors of A. This allows us to vary the cost of Av and $A^T u$ multiplications for experiments in Section 6.2.2.

2. Pick a vector $d \in \mathbf{R}^n$ whose elements are the singular values of A. We have two different options for choosing them. In the non-clustered option, adjacent singular values are separated by a constant gap and form an arithmetic sequence. In the clustered option, the smaller singular values cluster near the smallest one and form an geometric sequence. The two options are as follows:

$$d_j = \sigma_n + (\sigma_1 - \sigma_n)\frac{n-j}{n-1}, \qquad \text{(Non-clustered)}$$
$$d_j = \sigma_n \left(\frac{\sigma_1}{\sigma_n}\right)^{\frac{n-j}{n-1}}, \qquad \text{(Clustered)}$$

   where σ_1 and σ_n are the largest and smallest singular values of A and we have $d_1 = \sigma_1$ and $d_n = \sigma_n$.

3. Define $Y^{(i)} = I - 2y^{(i)}{y^{(i)}}^T$, $Z = Z^T = I - 2zz^T$, where

$$D = \operatorname{diag}(d), \qquad A = \left(\prod_{i=1}^{l} Y^{(i)}\right)\begin{pmatrix} D \\ 0 \end{pmatrix} Z.$$

This procedure is shown in MATLAB code 6.4.

In each of the following experiments, we pick a singular value σ that we would like to converge to, and extract the corresponding column v from Z as the singular vector. Then we pick a random unit vector w and a scalar δ to control the size of the perturbation that we want to introduce into v. The approximate right singular vector for input to RQI-AMRES and RQI-MINRES is then $(v + \delta w)$. Since RQI is known to converge always to some singular vector, our experiments are meaningful only if δ is quite small.

MATLAB Code 6.4 Generate linear operator with known singular values

```
function [Afun, sigma, v] ...
  = getLinearOperator(m,n,sigmaMax, sigmaMin, p, cost, clustered)
  % p: output the p-th largest singular value as sigma
  %   and the corresponding right singular vector as v
  randn('state',1);
  y = randn(m,cost);
  for i = 1:cost
    y(:,i) = y(:,i)/norm(y(:,i));    % normalize every column of y
  end
  z = randn(n,1); z = z/norm(z);
  if clustered
    d = sigmaMin * (sigmaMax/sigmaMin).^(((n-1):-1:0)'/(n-1));
  else
    d = sigmaMin + (sigmaMax-sigmaMin).*(((n-1):-1:0)'/(n-1));
  end
  ep = zeros(n,1); ep(p) = 1; v = ep - 2*(z'*ep)*z;
  sigma = s(p); Afun = @A;

  function w = A(x,trans)
    w = x;
    if trans == 1
      w = w - 2*(z'*w)*z;
      w = [d.*w; zeros(m-n,1)];
      for i = 1:cost; w = w - 2*(y(:,i)'*w)*y(:,i); end
    elseif trans == 2
      for i = cost:-1:1; w = w - 2*(y(:,i)'*w)*y(:,i); end
      w = d.*w(1:n);
      w = w - 2*(z'*w)*z;
    end
  end
end
```

Naming of algorithms

By applying different linear system solvers in line 3 of Algorithms 6.2 and 6.3, we obtain different versions of RQI for singular vector computation. We refer to these versions by the following names:

- RQI-MINRES: Apply MINRES to line 3 of Algorithm 6.2.
- RQI-AMRES: Apply AMRES to line 3 of Algorithm 6.3.
- MRQI-AMRES[4]: Replace line 3 of Algorithm 6.3 by

$$\begin{pmatrix} -\sigma I & A \\ A^T & -\sigma I \end{pmatrix} \begin{pmatrix} s \\ x_{q+1} \end{pmatrix} = \begin{pmatrix} u \\ 0 \end{pmatrix}, \tag{6.8}$$

[4]Details of MRQI-AMRES are given in Section 6.2.2

where u is a left singular vector estimate, or $u = Av$ if v is a right singular vector estimate. AMRES is applied to this linear system. Note that a further difference here from Algorithm 6.3 is that u is the same for all q.

NUMERICAL RESULTS

We ran both RQI-AMRES and RQI-MINRES to improve given approximate right singular vectors of our constructed linear operators. The accuracy[5] of the new singular vector estimate ρ_q as generated by both algorithms is plotted against the cumulative number of inner iterations (i.e. the number of Golub-Kahan bidiagonalization steps).

[5]For the iterates ρ_q from outer iterations that we want to converge to σ, we use the relative error $|\rho_q - \sigma|/\sigma$ as the accuracy measure.

When the singular values of A are not clustered, we found that RQI-AMRES and RQI-MINRES exhibit very similar convergence behavior as shown in Figure 6.2, with RQI-AMRES showing improved accuracy for small singular values.

When the singular values of A are clustered, we found that RQI-AMRES converges faster than RQI-MINRES as shown in Figure 6.3.

We have performed the same experiments on larger matrices and obtained similar results.

6.2.2 SPEEDING UP THE INNER ITERATIONS OF RQI

In the previous section, we applied AMRES to RQI to obtain a stable method for refining an approximate singular vector. We now explore a modification to RQI itself that allows AMRES to achieve significant speedup. To illustrate the rationale behind the modification, we first revisit some properties of the AMRES algorithm.

AMRES solves linear systems of the form

$$\begin{pmatrix} \gamma I & A \\ A^T & \delta I \end{pmatrix} \begin{pmatrix} s \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}$$

using the Golub-Kahan process $\text{Bidiag}(A, b)$, which is independent of γ and δ. Thus, if we solve a sequence of linear systems of the above form with different γ, δ but A, b kept constant, any computational work in $\text{Bidiag}(A, b)$ can be cached and reused. In Algorithm 6.3, $\gamma = \delta = -\rho_q$ and $b = Ax_{q-1}$. We now propose a modified version of RQI in which the right-hand side is kept constant at $b = Ax_0$ and only ρ_q is updated each outer iteration. We summarize this modification in Algorithm 6.4,

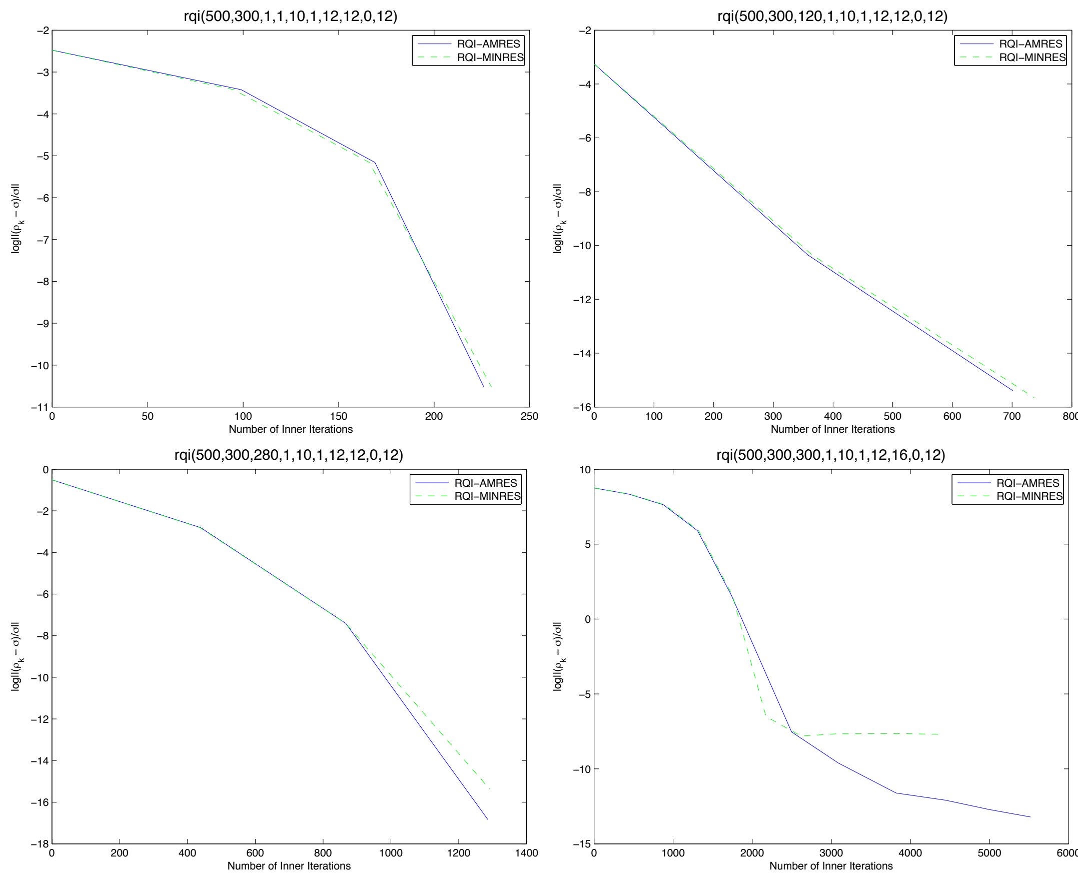


Figure 6.2: Improving an approximate singular value and singular vector for a matrix with non-clustered singular values using RQI-AMRES and RQI-MINRES. The matrix A is constructed by the procedure of Section 6.2.1 with parameters $m = 500$, $n = 300$, $\mathcal{C} = 1$, $\sigma_1 = 1$, $\sigma_n = 10^{-10}$ and non-clustered singular values. The approximate right singular vector $v + \delta w$ is formed with $\delta = 0.1$.
Upper Left: Computing the largest singular value σ_1. **Upper Right**: Computing a singular value σ_{120} near the middle of the spectrum. **Lower Left**: Computing a singular value σ_{280} near the low end of the spectrum. **Lower Right**: Computing the smallest singular value σ_{300}. Here we see that the iterates ρ_q generated by RQI-MINRES fail to converge to precisions higher than 10^{-8}, while RQI-AMRES achieves higher precision with more iterations.

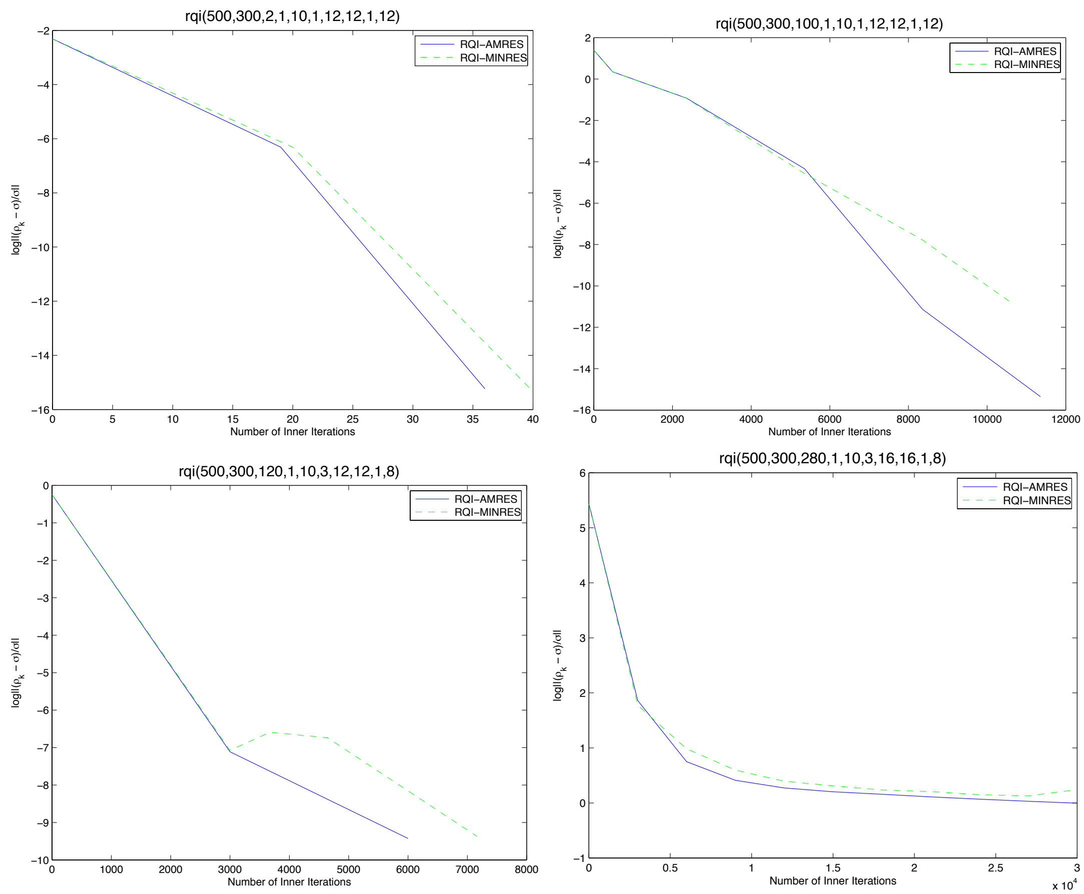


Figure 6.3: Improving an approximate singular value and singular vector for a matrix with clustered singular values using RQI-AMRES and RQI-MINRES. The matrix A is constructed by the procedure of Section 6.2.1 with parameters $m = 500$, $n = 300$, $\mathcal{C} = 1$, $\sigma_1 = 1$, $\sigma_n = 10^{-10}$ and clustered singular values. For the two upper plots, the approximate right singular vector $v + \delta w$ is formed with $\delta = 0.1$. For the two lower plots, $\delta = 0.001$.
Upper Left: Computing the second largest singular value σ_2. We see that RQI-AMRES starts to converge faster after the 1st outer iteration. The plot for σ_1 is not shown here as σ_1 is well separated from the rest of the spectrum, and both algorithms converge in 1 outer iteration, which is a fairly uninteresting plot. **Upper Right**: Computing a singular value σ_{100} near the middle of the spectrum. RQI-AMRES converges faster than RQI-MINRES. **Lower Left**: Computing a singular value σ_{120} near the low end of the spectrum. **Lower Right**: Computing a singular value σ_{280} within the highly clustered region. RQI-AMRES and RQI-MINRES converge but to the wrong σ_j.

Algorithm 6.4 Modified RQI for singular vectors

1: Given initial singular value and right singular vector estimates ρ_0, x_0
2: **for** $q = 1, 2, \ldots$ **do**
3: Solve for t in
$$\begin{pmatrix} -\rho_{q-1} I & A \\ A^T & -\rho_{q-1} I \end{pmatrix} \begin{pmatrix} s \\ \bar{t} \end{pmatrix} = \begin{pmatrix} A x_0 \\ 0 \end{pmatrix}$$
4: $t = \bar{t} - x_0$
5: $x_q = t / \|t\|$
6: $\rho_q = \|A x_q\|$
7: **end for**

which we refer to as MRQI-AMRES when AMRES is used to solve the system on line 3.

Cached Golub-Kahan process

We implement a `GolubKahan` class (see MATLAB Code 6.5) to represent the process Bidiag(A, b). The constructor is invoked by

```
gk = GolubKahan(A,b)
```

with `A` being a matrix or a MATLAB function handle that gives $Ax =$ `A(x,1)` and $A^T y =$ `A(y,2)`. The scalars α_k, β_k and vector v_k can be retrieved by calling

```
[v_k alpha_k beta_k] = gk.getIteration(k)
```

These values will be retrieved from the cache if they have been computed already. Otherwise, `getIteration()` computes them directly and caches the results for later use.

Reorthogonalization

In Section 4.4, reorthogonalization is proposed as a method to speed up LSMR. The major obstacle is the high cost of storing all the v vectors generated by the Golub-Kahan process. For MRQI-AMRES, all such vectors must be stored for reuse in the next MRQI iteration. Therefore the only extra cost to perform reorthogonalization would be the computing cost of modified Gram-Schmidt (MGS). MGS is implemented as in MATLAB Code 6.5. It will be turned on when the constructor is called by `GolubKahan(A,b,1)`.[6] We refer to MRQI-AMRES with reorthogonalization as MRQI-AMRES-Ortho in the following plots.

[6] `Golub-Kahan(A,b)` and `Golub-Kahan(A,b,0)` are the same. They return a Golub-Kahan object that does not preform reorthogonalization.

MATLAB Code 6.5 Cached Golub-Kahan process

```
classdef GolubKahan < handle
  % A cached implementation of Golub-Kahan bidiagonalization
  properties
    V; m; n; A; Afun; u; alphas; betas; reortho; tol = 1e-15;
    kCached = 1;      % number of cached iterations
    kAllocated = 1; % initial size to allocate
  end

  methods
    function o = GolubKahan(A, b, reortho)
      % @param reortho=1 means perform reorthogonalization on o.V
      if nargin < 3 || isempty(reortho), reortho = 0; end
      o.A = A; o.Afun = A; o.reortho = reortho;
      if isa(A, 'numeric'); o.Afun = @o.AfunPrivate; end
      o.betas = zeros(o.kAllocated,1);  o.betas(1) = norm(b);
      o.alphas = zeros(o.kAllocated,1); o.m = length(b);
      if o.betas(1) > o.tol
        o.u = b/o.betas(1);      v = o.Afun(o.u,2);
        o.n = length(v);         o.V = zeros(o.n,o.kAllocated);
        o.alphas(1) = norm(v);
        if o.alphas(1) > o.tol; o.V(:,1) = v/norm(o.alphas(1)); end
      end
    end

    function [v alpha beta] = getIteration(o,k)
      % @return v_k alpha_k beta_k for the k-th iteration of Golub-Kahan
      while o.kCached < k
        % doubling space for cache
        if o.kCached == o.kAllocated
          o.V = [o.V zeros(o.n, o.kAllocated)];
          o.alphas = [o.alphas ;zeros(o.kAllocated,1)];
          o.betas = [o.betas ;zeros(o.kAllocated,1)];
          o.kAllocated = o.kAllocated * 2;
        end

        % bidiagonalization
        o.u = o.Afun(o.V(:,o.kCached),1) - o.alphas(o.kCached)*o.u;
        beta = norm(o.u); alpha = 0;
        if beta > o.tol;
          o.u = o.u/beta;
          v = o.Afun(o.u,2) - beta*o.V(:,o.kCached);
          if o.reortho == 1
            for i = 1:o.kCached; vi = o.V(:,i); v = v - (v'*vi)*vi; end
          end
          alpha = norm(v);
          if alpha > o.tol; v = v/alpha; o.V(:,o.kCached+1) = v; end
        end
        o.kCached = o.kCached + 1;
        o.alphas(o.kCached) = alpha; o.betas(o.kCached) = beta;
      end
      v = o.V(:,k); alpha = o.alphas(k); beta = o.betas(k);
    end

    function out = AfunPrivate(o,x,trans)
      if trans == 1; out = o.A*x; else out = o.A'*x; end
    end
  end
end
```

NUMERICAL RESULTS

We ran RQI-AMRES, MRQI-AMRES and MRQI-AMRES-Ortho to improve given approximate right singular vectors of our constructed linear operators. The accuracy[7] of each new singular value estimate ρ_q generated by the three algorithms at each outer iteration is plotted against the cumulative time (in seconds) used.

[7]For the outer iteration values ρ_q that we want to converge to σ, we use the relative error $|\rho_q - \sigma|/\sigma$ as the accuracy measure.

In Figure 6.4, we see that MRQI-AMRES takes more time than RQI-AMRES for the early outer iterations because of the cost of allocating memory and caching the Golub-Kahan vectors. For subsequent iterations, more cached vectors are used and less time is needed for Golub-Kahan bidiagonalization. MRQI-AMRES overtakes RQI-AMRES starting from the third outer iteration. In the same figure, we see that reorthogonalization significantly improves the convergence rate of MRQI-AMRES. With orthogonality preserved, MRQI-AMRES-Ortho converges faster than RQI-AMRES even at the first outer iteration.

In Figure 6.5, we compare the performance of the three algorithms when the linear operator has different multiplication cost. For operators with very low cost, RQI-AMRES converges faster than the cached methods. The extra time used for caching outweighs the benefits of reusing cached results. As the linear operator gets more expensive, MRQI-AMRES and MRQI-AMRES-Ortho outperforms RQI-AMRES as fewer expensive matrix-vector products are needed in the subsequent RQIs.

In Figure 6.6, we compare the performance of the three algorithms for refining singular vectors corresponding to various singular values. For the large singular values (which are well separated from the rest of the spectrum), RQI converges in very few iterations and there is not much benefit from caching. For the smaller (and more clustered) singular values, caching saves the multiplication time in subsequent RQIs and therefore MRQI-AMRES and MRQI-AMRES-Ortho converge faster than RQI-AMRES. As the singular values become more clustered, the inner solver for the linear system suffers increasingly from loss of orthogonality. In this situation, reorthogonalization greatly reduces the number of Golub-Kahan steps and therefore MRQI-AMRES-Ortho converges much faster than MRQI-AMRES.

Next, we compare RQI-AMRES with the MATLAB `svds` function. `svds` performs singular vector computation by calling `eigs` on the augmented matrix $\begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}$, and `eigs` in turn calls the Fortran library ARPACK to perform symmetric Lanczos iterations. `svds` does not accept a linear

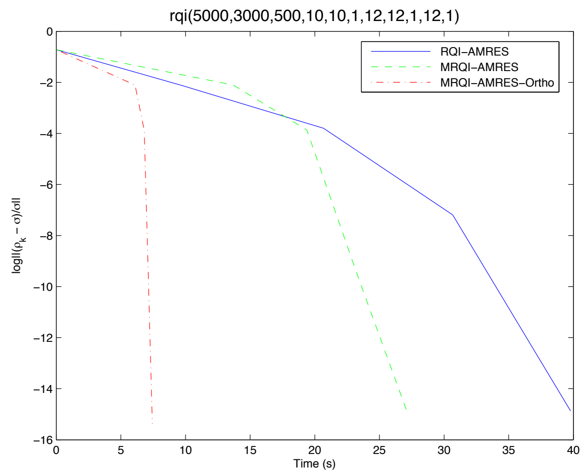
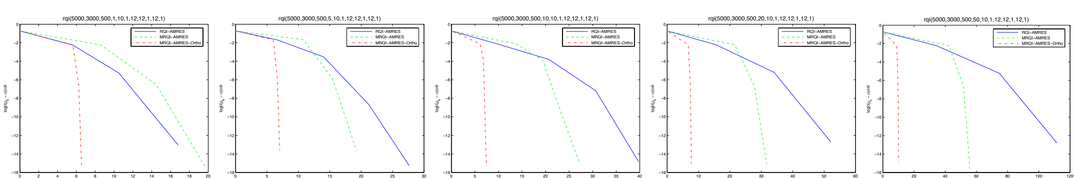
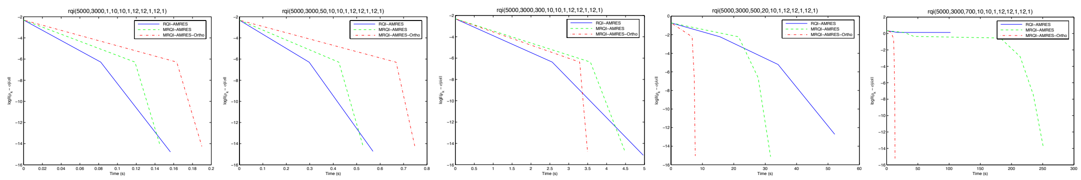


Figure 6.4: Improving an approximate singular value and singular vector for a matrix using RQI-AMRES, MRQI-AMRES and MRQI-AMRES-Ortho. The matrix A is constructed by the procedure of Section 6.2.1 with parameters $m = 5000$, $n = 3000$, $\mathcal{C} = 10$, $\sigma_1 = 1$, $\sigma_n = 10^{-10}$ and clustered singular values. We perturbed the right singular vector v corresponding to the singular value σ_{500} to be the starting approximate singular vector. MRQI-AMRES lags behind RQI-AMRES at the early iterations because of the extra cost in allocating memory to store the Golub-Kahan vectors, but it quickly catches up and converges faster when the cached vectors are reused in the subsequent iterations. MRQI-AMRES with reorthogonalization converges much faster than without reorthogonalization. In this case, reorthogonalization greatly reduces the number of inner iterations for AMRES, and the subsequent outer iterations are basically free.



$\mathcal{C} = 1$ $\mathcal{C} = 5$ $\mathcal{C} = 10$ $\mathcal{C} = 20$ $\mathcal{C} = 50$

Figure 6.5: Convergence of RQI-AMRES, MRQI-AMRES and MRQI-AMRES-Ortho for linear operators of varying cost. The matrix A is constructed as in Figure 6.4 except for the cost $\mathcal{C}$, which is shown below each plot. $\mathcal{C}$ increases from left to right. When $\mathcal{C} = 1$, the time required for caching the Golub-Kahan vectors outweighs the benefits of being able to reuse them later. When the cost $\mathcal{C}$ goes up, the benefit of caching Golub-Kahan vectors outweighs the time for saving them. Thus MRQI-AMRES converges faster than RQI-AMRES.



σ_1 σ_{50} σ_{300} σ_{500} σ_{700}

Figure 6.6: Convergence of RQI-AMRES, MRQI-AMRES and MRQI-AMRES-Ortho for different singular values. The matrix A is constructed as in Figure 6.4.
σ_1, σ_{50}: For the larger singular values, both MRQI-AMRES and MRQI-AMRES-Ortho converge slower than RQI-AMRES as convergence for RQI is very fast and the extra time for caching the Golub-Kahan vectors outweighs the benefits of saving them.
$\sigma_{300}, \sigma_{500}$: For the smaller and more clustered singular values, RQI takes more iterations to converge and the caching effect of MRQI-AMRES and MRQI-AMRES-Ortho makes them converge faster than RQI-AMRES.
σ_{700}: As the singular value gets even smaller and less separated, RQI is less likely to converge to the singular value we want. In this example, RQI-AMRES converged to a different singular value. Also, clustered singular values lead to greater loss of orthogonality, and therefore RQI-AMRES-Ortho converges much faster than RQI-AMRES.

operator (function handle) as input. We slightly modified `svds` to allow an operator to be passed to `eigs`, which finds the eigenvectors corresponding to the largest eigenvalues for a given linear operator.[8]

Figure 6.7 shows the convergence of RQI-AMRES and `svds` computing three different singular values near the large end of the spectrum. When only the largest singular value is needed, the algorithms converge at about the same rate. When any other singular value is needed, `svds` has to compute all singular values (and singular vectors) from the largest to the one required. Thus `svds` takes significantly more time compared to RQI-AMRES.

[8] `eigs` can find eigenvalues of a matrix B close to any given λ, but if B is an operator, the shift-and-invert operator $(B - \lambda I)^{-1}$ must be provided.

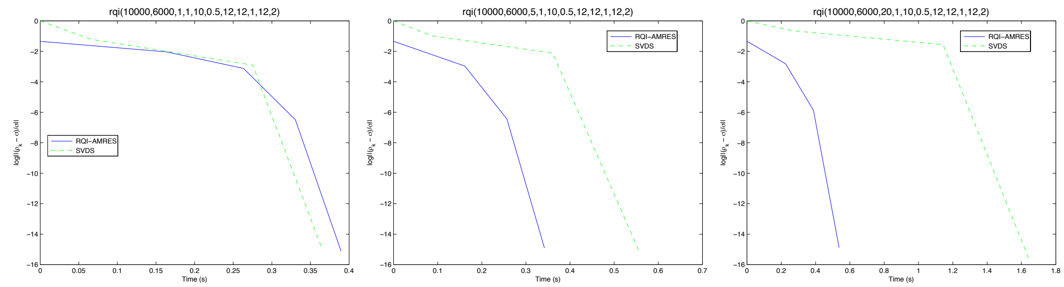
6.3 Singular vector computation

In the previous section, we explored algorithms to refine an approximate singular vector. We now focus on finding a singular vector corresponding to a known singular value.

Suppose σ is a singular value of A. If a nonzero $\begin{pmatrix} u \\ v \end{pmatrix}$ satisfies

$$\begin{pmatrix} -\sigma I & A \\ A^T & -\sigma I \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \tag{6.9}$$

then u and v would be the corresponding singular vectors. The inverse



σ_1 σ_5 σ_{20}

Figure 6.7: Convergence of RQI-AMRES and svds for the largest singular values. The matrix A is constructed by the procedure of Section 6.2.1 with parameters $m = 10000$, $n = 6000$, $\mathcal{C} = 1$, $\sigma_1 = 1$, $\sigma_n = 10^{-10}$ and clustered singular values. The approximate right singular vector $v + \delta w$ is formed with $\delta = 10^{-1/2} \approx 0.316$.
σ_1: For the largest singular value and corresponding singular vector, RQI-AMRES and svds take similar times.
σ_5: For the fifth largest singular value and corresponding singular vector, svds (and hence ARPACK) has to compute all five singular values (from the largest to fifth largest), while RQI-AMRES computes the fifth largest singular value directly. Therefore, RQI-AMRES takes less time than svds.
σ_{20}: As svds needs to compute the largest 20 singular values, it takes significantly more time than RQI-AMRES.

iteration approach to finding the null vector is to take a random vector b and solve the system

$$\begin{pmatrix} -\sigma I & A \\ A^T & -\sigma I \end{pmatrix} \begin{pmatrix} s \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}. \tag{6.10}$$

Whether a direct or iterative solver is used, we expect the solution $\binom{s}{x}$ to be very large, but the normalized vectors $u = s/\|s\|$, $v = x/\|x\|$ will satisfy (6.9) accurately and hence be good approximations to the left and right singular vectors of A.

As noted in Choi [13], it is not necessary to run an iterative solver until the solution norm becomes very large. If one could stop the solver appropriately at a least-squares solution, then the residual vector of that solution would be a null vector of the matrix. Here we apply this idea to singular vector computation.

For the problem $\min \|Ax - b\|$, if x is a solution, we know that the residual vector $r = b - Ax$ satisfies $A^T r = 0$. Therefore r is a null vector for A^T.

Algorithm 6.5 Singular vector computation via residual vector

1: Given matrix A, and singular value σ
2: Find least-squares solution for the singular system:
3: $\quad \min \left\| \begin{pmatrix} b \\ 0 \end{pmatrix} - \begin{pmatrix} -\sigma I & A \\ A^T & -\sigma I \end{pmatrix} \begin{pmatrix} s \\ x \end{pmatrix} \right\|$
4: Compute $\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix} - \begin{pmatrix} -\sigma I & A \\ A^T & -\sigma I \end{pmatrix} \begin{pmatrix} s \\ x \end{pmatrix}$
5: Output u, v as the left and right singular vectors

Thus, if s and x solve the singular least-squares problem

$$\min \left\| \begin{pmatrix} b \\ 0 \end{pmatrix} - \begin{pmatrix} -\sigma I & A \\ A^T & -\sigma I \end{pmatrix} \begin{pmatrix} s \\ x \end{pmatrix} \right\|, \tag{6.11}$$

then the corresponding residual vector

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix} - \begin{pmatrix} -\sigma I & A \\ A^T & -\sigma I \end{pmatrix} \begin{pmatrix} s \\ x \end{pmatrix} \tag{6.12}$$

would satisfy (6.9). This gives us $Av = \sigma u$ and $A^T u = \sigma v$ as required. Therefore u, v are left and right singular vectors corresponding to σ. We summarize this procedure as Algorithm 6.5.

The key to this algorithm lies in finding the residual to the least-squares system accurately and efficiently. An obvious choice would be applying MINRES to (6.11) and computing the residual from the solution returned by MINRES.

AMRES solves the same problem (6.11) with half the computational cost by computing x only. However, this doesn't allow us to construct the residual from x. We therefore developed a specialized version of AMRES, called AMRESR, that directly computes the v part of the residual vector (6.12) without computing x or s. The u part can be recovered from v using $\sigma u = Av$.

If σ is a repeated singular value, the same procedure can be followed using a new random b that is first orthogonalized with respect to v.

6.3.1 AMRESR

To derive an iterative update for $\widehat{r}_k$, we begin with (5.5):

$$\begin{aligned}
\widehat{r}_k &= \hat{V}_{k+1}\left(\beta_1 e_1 - \hat{H}_k \hat{y}_k\right) \\
&= \hat{V}_{k+1} Q_{k+1}^T \left(\begin{pmatrix} z_k \\ \bar{\zeta}_{k+1} \end{pmatrix} - \begin{pmatrix} R_k \\ 0 \end{pmatrix} \hat{y}_k\right) \\
&= \hat{V}_{k+1} Q_{k+1}^T (\bar{\zeta}_{k+1} e_{k+1}).
\end{aligned}$$

By defining $p_k = \hat{V}_{k+1} Q_{k+1}^T e_{k+1}$, we continue with

$$\begin{aligned}
p_k &= \hat{V}_{k+1} Q_{k+1}^T e_{k+1}. \\
&= \hat{V}_{k+1} \begin{pmatrix} Q_k^T & \\ & 1 \end{pmatrix} \begin{pmatrix} I_{k-1} & & \\ & c_k & -s_k \\ & s_k & c_k \end{pmatrix} e_{k+1} \\
&= \begin{pmatrix} \hat{V}_k Q_k^T & \hat{v}_{k+1} \end{pmatrix} \begin{pmatrix} 0 \\ -s_k \\ c_k \end{pmatrix} \\
&= -s_k \hat{V}_k Q_k^T e_k + c_k \hat{v}_{k+1} \\
&= -s_k p_{k-1} + c_k \hat{v}_{k+1}.
\end{aligned}$$

With $\widehat{r}_k \equiv \begin{pmatrix} r_k^u \\ r_k^v \end{pmatrix}$ and $p_k \equiv \begin{pmatrix} p_k^u \\ p_k^v \end{pmatrix}$, we can write an update rule for p_k^v:

$$p_0^v = 0, \qquad p_k^v = \begin{cases} -s_k p_{k-1}^v + c_k v_{\frac{k+1}{2}} & (k \text{ odd}) \\ -s_k p_{k-1}^v & (k \text{ even}) \end{cases}$$

where the separation of two cases follows from (5.4) and

$$r_k^v = \bar{\zeta}_{k+1} p_k^v \tag{6.13}$$

can be used to output r_k^v when AMRESR terminates. With the above recurrence relations for p_k^v, we summarize AMRESR in Algorithm 6.6.

6.3.2 AMRESR EXPERIMENTS

We compare the convergence of AMRESR versus MINRES to find singular vectors corresponding to a known singular value.

Algorithm 6.6 Algorithm AMRESR

(Initialize)

$$\beta_1 u_1 = b \qquad \alpha_1 v_1 = A^T u_1 \qquad \bar{\rho}_1 = \gamma \qquad \bar{\theta}_1 = \alpha_1$$
$$\bar{\zeta}_1 = \beta_1 \qquad p_0^v = 0$$

for $l = 1, 2, 3 \dots$ **do**

(Continue the bidiagonalization)

$$\beta_{l+1} u_{l+1} = A v_l - \alpha_l u_l \qquad \alpha_{l+1} v_{l+1} = A^T u_{l+1} - \beta_{l+1} v_l$$

for $k = 2l-1, 2l$ **do**

(Setup temporary variables)

$$\begin{cases} \lambda = \delta, & \alpha = \alpha_l, \qquad \beta = \beta_{l+1} \quad (k \text{ odd}) \\ \lambda = \gamma, & \alpha = \beta_{l+1}, \qquad \beta = \alpha_{l+1} \quad (k \text{ even}) \end{cases}$$

(Construct and apply rotation $Q_{k+1,k}$)

$$\begin{aligned} \rho_k &= \left(\bar{\rho}_k^2 + \alpha^2\right)^{\frac{1}{2}} & & \\ c_k &= \bar{\rho}_k / \rho_k & s_k &= \alpha / \rho_k \\ \theta_k &= c_k \bar{\theta}_k + s_k \lambda & \bar{\rho}_{k+1} &= -s_k \bar{\theta}_k + c_k \lambda \\ \eta_k &= s_k \beta & \bar{\theta}_{k+1} &= c_k \beta \\ \zeta_k &= c_k \bar{\zeta}_k & \bar{\zeta}_{k+1} &= -s_k \bar{\zeta}_k \end{aligned}$$

(Update estimate of p)

$$p_k^v = \begin{cases} -s_k p_{k-1} + c_k v_{\frac{k+1}{2}} & (k \text{ odd}) \\ -s_k p_{k-1} & (k \text{ even}) \end{cases}$$

end for

end for

(Compute r for output)

$$r_k^v = \bar{\zeta}_{k+1} p_k^v$$

MATLAB Code 6.6 Error measure for singular vector

```
function err = singularVectorError(v)
  v   = v/norm(v);
  Av  = Afun(v,1);                    % Afun(v,1) = A*v;
  u   = Av/norm(Av);
  err = norm(Afun(u,2) - sigma*v);    % Afun(v,2) = A'*u;
end
```

TEST DATA

We use the method of Section 6.2.1 to construct a linear operator.

METHODS

We compare three methods for computing a singular vector corresponding to known singular value σ, where σ is taken as one of the d_i above. We first generate a random m vector b using `randn('state',1)` and `b = randn(m,1)`. Then we apply one of the following methods:

1. Run AMRESR on (6.10). For each k in Algorithm 6.6, Compute r_k^v by (6.13) at each iteration and take $v_k = r_k^v$ as the estimate of the right singular vector corresponding to σ.
2. Inverse iteration: run MINRES on $(A^TA - \sigma^2 I)x = A^Tb$. For the lth MINRES iteration ($l = 1, 2, 3, \ldots$), set $k = 2l$ take $v_k = x_l$ as the estimate of the right singular vector. (This makes k the number of matrix-vector products for each method.)

MEASURE OF CONVERGENCE

For the v_k from each method, we measure convergence by $\|A^Tu_k - \sigma v_k\|$ (with $u_k = Av_k/\|Av_k\|$) as shown in MATLAB Code 6.6. This measure requires two matrix-vector products, which is the same complexity as each Golub-Kahan step. Thus it is too expensive to be used as a practical stopping rule. It is used solely for the plots to analyze the convergence behavior. In practice, the stopping rule would be based on the condition estimate (5.16).

NUMERICAL RESULTS

We ran the experiments described above on rectangular matrices of size 24000×18000. Our observations are as follows:

1. AMRESR always converge faster than MINRES. AMRESR reaches a minimum error of about 10^{-8} for all test examples, and then starts to diverge. In contrast, MINRES is able to converge to singular vectors of higher precision than AMRESR. Thus, AMRESR is more suitable for applications where high precision of the singular vectors is not required.
2. The gain in convergence speed depends on the position of the singular value within the spectrum. From Figure 6.8, there is more improvement for singular vectors near either end of the spectrum, and less improvement for those in the middle of the spectrum.

6.4 Almost singular systems

Björck [6] has derived an algorithm based on Rayleigh-quotient iteration for computing (ordinary) TLS solutions. The algorithm involves repeated solution of positive definite systems $(A^TA - \sigma^2 I)x = A^Tb$ by means of a modified/extended version of CGLS (called CGTLS) that is able to incorporate the shift. A key step in CGTLS is computation of

$$\delta_k = \|p_k\|_2^2 - \sigma^2 \|q_k\|_2^2.$$

Clearly we must have $\sigma < \sigma_{\min}(A)$ for the system to be positive definite. However, this condition cannot be guaranteed and a heuristic is adopted to repeat the computation with a smaller value of σ [6, (4.5)]. Another drawback of CGTLS is that it depends on a complete Cholesky factor of A^TA. This is computed by a sparse QR of A, which is sometimes practical when A is large. Since AMRES handles indefinite systems and does not depend on a sparse factorization, it is applicable in more situations.

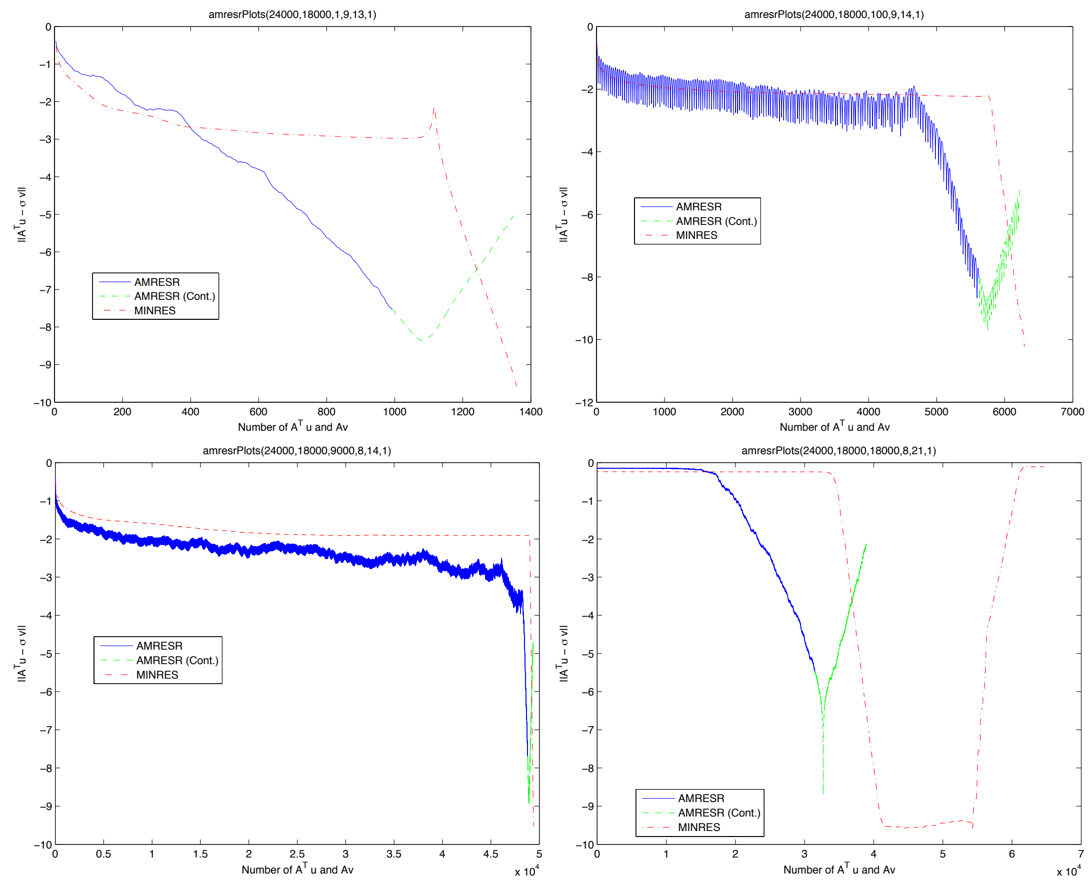


Figure 6.8: Convergence of AMRESR and MINRES for computing singular vectors corresponding to a known singular value as described in Section 6.3.2 for a 24000×18000 matrix. The blue (solid) curve represents where AMRESR would have stopped if there were a suitable limit on the condition number. The green (dash) curve shows how AMRESR will diverge if it continues beyond this stopping criterion. The red (dash-dot) represents the convergence behavior of MINRES. The linear operator is constructed using the method of Section 6.2.1. **Upper Left**: Computing the singular vector corresponding to the largest singular value σ_1. **Upper Right**: Computing the singular vector corresponding to the 100th largest singular value σ_{100}. **Lower Left**: Computing the singular vector corresponding to a singular value in the middle of the spectrum σ_{9000}. **Lower Right**: Computing the singular vector corresponding to the smallest singular value σ_{18000}.

7 Conclusions and future directions

7.1 Contributions

The main contributions of this thesis involve three areas: providing a better understanding of the popular MINRES algorithm; development and analysis of LSMR for least-squares problems; and development and applications of AMRES for the negatively-damped least-squares problem. We summarize our findings for each of these areas in the next three sections.

Chapters 1 to 4 discussed a number of iterative solvers for linear equations. The flowchart in Figure 7.1 helps decide which methods to use for a particular problem.

7.1.1 MINRES

In Chapter 2, we proved a number of properties for MINRES applied to a positive definite system. Table 7.1 compares these properties with known ones for CG. MINRES has a number of monotonic properties that can make it more favorable, especially when the iterative algorithm needs to be terminated early.

In addition, our experimental results show that MINRES converges faster than CG in terms of backward error, often by as much as 2 orders of magnitude (Figure 2.2). On the other hand, CG converges somewhat faster than MINRES in terms of both $\|x_k - x^*\|_A$ and $\|x_k - x^*\|$ (same figure).

Table 7.1 Comparison of CG and MINRES properties on an spd system

	CG	MINRES
$\|x_k\|$	$\nearrow$ [68, Thm 2.1]	$\nearrow$ (Thm 2.1.6)
$\|x^* - x_k\|$	$\searrow$ [38, Thm 4:3]	$\searrow$ (Thm 2.1.7) [38, Thm 7:5]
$\|x^* - x_k\|_A$	$\searrow$ [38, Thm 6:3]	$\searrow$ (Thm 2.1.8) [38, Thm 7:4]
$\|r_k\|$	not-monotonic	$\searrow$ [52] [38, Thm 7:2]
$\|r_k\|/\|x_k\|$	not-monotonic	$\searrow$ (Thm 2.2.1)

$\nearrow$ monotonically increasing
$\searrow$ monotonically decreasing

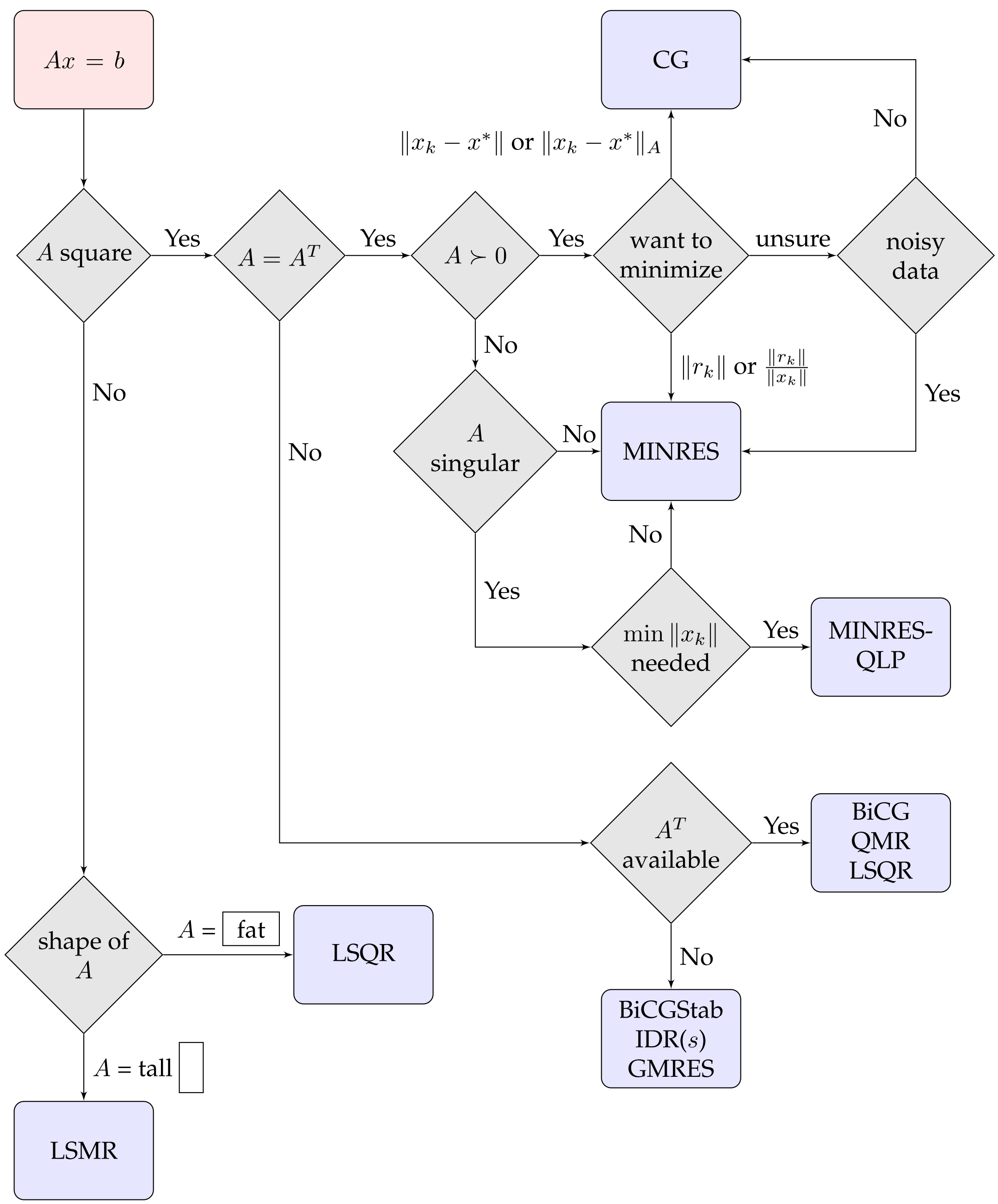


Figure 7.1: Flowchart on choosing iterative solvers

Table 7.2 Comparison of LSQR and LSMR properties

	LSQR	LSMR
$\|x_k\|$	$\nearrow$ (Thm 3.3.1)	$\nearrow$ (Thm 3.3.6)
$\|x_k - x^*\|$	$\searrow$ (Thm 3.3.2)	$\searrow$ (Thm 3.3.7)
$\|A^T r_k\|$	not-monotonic	$\searrow$ (Thm 3.3.5)
$\|r_k\|$	$\searrow$ (Thm 3.3.4)	$\searrow$ (Thm 3.3.11)
x_k converges to minimum norm x^* for singular systems		
$\|E_2^{\text{LSQR}}\| \geq \|E_2^{\text{LSMR}}\|$		

$\nearrow$ monotonically increasing
$\searrow$ monotonically decreasing

7.1.2 LSMR

We presented LSMR, an iterative algorithm for square or rectangular systems, along with details of its implementation, theoretical properties, and experimental results to suggest that it has advantages over the widely adopted LSQR algorithm.

As in LSQR, theoretical and practical stopping criteria are provided for solving $Ax = b$ and $\min \|Ax - b\|$ with optional Tikhonov regularization. Formulae for computing $\|r_k\|$, $\|A^T r_k\|$, $\|x_k\|$ and estimating $\|A\|$ and $\text{cond}(A)$ in $O(1)$ work per iteration are available.

For least-squares problems, the Stewart backward error estimate $\|E_2\|$ (4.4) is computable in $O(1)$ work and is proved to be at least as small as that of LSQR. In practice, $\|E_2\|$ for LSMR is smaller than that of LSQR by 1 to 2 orders of magnitude, depending on the condition of the given system. This often allows LSMR to terminate significantly sooner than LSQR.

In addition, $\|E_2\|$ seems experimentally to be very close to the *optimal* backward error $\mu(x_k)$ at each LSMR iterate x_k (Section 4.1.1). This allows LSMR to be stopped reliably using rules based on $\|E_2\|$.

MATLAB, Python, and Fortran 90 implementations of LSMR are available from [42]. They all allow local reorthogonalization of V_k.

7.1.3 AMRES

We developed AMRES, an iterative algorithm for solving the augmented system

$$\begin{pmatrix} \gamma I & A \\ A^T & \delta I \end{pmatrix} \begin{pmatrix} s \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix},$$

where $\gamma\delta > 0$. It is equivalent to MINRES on the same system and is reliable even when $\left(\begin{smallmatrix} \delta I & A \\ A^T & \delta I \end{smallmatrix}\right)$ is indefinite or singular. It is based on the Golub-Kahan bidiagonalization of A and requires half the computational cost compared to MINRES. AMRES is applicable to Curtis-Reid scaling; improving approximate singular vectors; and computing singular vectors from known singular values.

For Curtis-Reid scaling, AMRES sometimes exhibits faster convergence compared to the specialized version of CG proposed by Curtis and Reid. However, both algorithms exhibit similar performance on many test matrices.

Using AMRES as the solver for inner iterations, we have developed Rayleigh quotient-based algorithms (RQI-AMRES, MRQI-AMRES, MRQI-AMRES-Ortho) that refine a given approximate singular vector to an accurate one. They converge faster than the MATLAB `svds` function for singular vectors corresponding to interior singular values. When the singular values are clustered, or the linear operator A is expensive, MRQI-AMRES and MRQI-AMRES-Ortho are more stable and converge much faster than RQI-AMRES or direct use of MINRES within RQI.

For a given singular value, we developed AMRESR, a modified version of AMRES, to compute the corresponding singular vectors to a precision of $O(\sqrt{\epsilon})$. Our algorithm converges faster than inverse iteration. If singular vectors of higher precision are needed, inverse iteration is preferred.

7.2 Future directions

Several ideas are highly related to the research in this thesis, but their potential has not been fully explored yet. We summarize these directions below as a pointer for future research.

7.2.1 Conjecture

From our experiments, we conjecture that the optimal backward error $\mu(x_k)$ (4.1) and its approximate $\tilde{\mu}(x_k)$ (4.5) decrease monotonically for LSMR.

7.2.2 Partial reorthogonalization

Larsen [58] uses partial reorthogonalization of both V_k and U_k within his PROPACK software for computing a set of singular values and vectors for a sparse rectangular matrix A. This involves a smaller computational cost compared to full reorthogonalization, as each v_k or u_k

does not need to perform an inner product with each of the previous v_i's or u_i's. Similar techniques might prove helpful within LSMR and AMRES.

7.2.3 EFFICIENT OPTIMAL BACKWARD ERROR ESTIMATES

Both LSQR and LSMR stop when $\|E_2\|$, an upper bound for the optimal backward error that's computable in $O(1)$ work per iteration, is sufficiently small. Ideally, an iterative algorithm for least-squares should use the optimal backward error $\mu(x_k)$ itself in the stopping rule. However, direct computation using (4.2) is more expensive than solving the least-squares problem itself. An accurate approximation is given in (4.5). This cheaper estimate involves solving a least-squares problem at each iteration of any iterative solver, and thus is still not practical for use as a stopping rule. It would be of great interest if a provably accurate estimate of the optimal backward error could be found in $O(n)$ work at each iteration, as it would allow iterative solvers to stop precisely at the iteration where the desired accuracy has been attained.

More precise stopping rules have been derived recently by Arioli and Gratton [2] and Titley-Peloquin et al. [11; 50; 75]. The rules allow for uncertainty in both A and b, and may prove to be useful for LSQR, LSMR, and least-squares methods in general.

7.2.4 SYMMLQ-BASED LEAST-SQUARES SOLVER

LSMR is derived to be a method mathematically equivalent to MINRES on the normal equation, and thus LSMR minimizes $\|A^T r_k\|$ for x_k in the Krylov subspace $\mathcal{K}_k(A^TA, A^Tb)$.

For a symmetric system $Ax = b$, SYMMLQ [52] solves

$$\min_{x_k \in A\mathcal{K}_k(A,b)} \|x_k - x^*\|$$

at the k-th iteration [47], [27, p65]. Thus if we derive an algorithm for the least-squares problem that is mathematically equivalent to SYMMLQ on the normal equation but uses Golub-Kahan bidiagonalization, this new algorithm will minimize $\|x_k - x^*\|$ for x_k in the Krylov subspace $A^TA\mathcal{K}_k(A^TA, A^Tb)$. This may produce smaller errors compared to LSQR or LSMR, and may be desirable in applications where the algorithm has to be terminated early with a smaller error (rather than a smaller backward error).

Bibliography

[1] M. Arioli. A stopping criterion for the conjugate gradient algorithm in a finite element method framework. *Numerische Mathematik*, 97(1):1–24, 2004. doi:10.1007/s00211-003-0500-y.

[2] M. Arioli and S. Gratton. Least-squares problems, normal equations, and stopping criteria for the conjugate gradient method. Technical Report RAL-TR-2008-008, Rutherford Appleton Laboratory, Oxfordshire, UK, 2008.

[3] M. Arioli, I. Duff, and D. Ruiz. Stopping criteria for iterative solvers. *SIAM J. Matrix Anal. Appl.*, 13:138, 1992. doi:10.1137/0613012.

[4] M. Arioli, E. Noulard, and A. Russo. Stopping criteria for iterative methods: applications to PDE's. *Calcolo*, 38(2): 97–112, 2001. doi:10.1007/s100920170006.

[5] S. J. Benbow. Solving generalized least-squares problems with LSQR. *SIAM J. Matrix Anal. Appl.*, 21(1):166–177, 1999. doi:10.1137/S0895479897321830.

[6] Å. Björck, P. Heggernes, and P. Matstoms. Methods for large scale total least squares problems. *SIAM J. Matrix Anal. Appl.*, 22(2), 2000. doi:10.1137/S0895479899355414.

[7] F. Cajori. *A History of Mathematics*. The MacMillan Company, New York, second edition, 1919.

[8] D. Calvetti, B. Lewis, and L. Reichel. An L-curve for the MINRES method. In Franklin T. Luk, editor, *Advanced Signal Processing Algorithms, Architectures, and Implementations X*, volume 4116, pages 385–395. SPIE, 2000. doi:10.1117/12.406517.

[9] D. Calvetti, S. Morigi, L. Reichel, and F. Sgallari. Computable error bounds and estimates for the conjugate gradient method. *Numerical Algorithms*, 25:75–88, 2000. doi:10.1023/A:1016661024093.

[10] F. Chaitin-Chatelin and V. Frayssé. *Lectures on Finite Precision Computations*. SIAM, Philadelphia, 1996. doi:10.1137/1.9780898719673.

[11] X.-W. Chang, C. C. Paige, and D. Titley-Péloquin. Stopping criteria for the iterative solution of linear least squares problems. *SIAM J. Matrix Anal. Appl.*, 31(2):831–852, 2009. doi:10.1137/080724071.

[12] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. *ACM Trans. Math. Softw.*, 35:22:1–22:14, October 2008. doi:10.1145/1391989.1391995.

[13] S.-C. Choi. *Iterative Methods for Singular Linear Equations and Least-Squares Problems*. PhD thesis, Stanford University, Stanford, CA, 2006.

[14] S.-C. Choi, C. C. Paige, and M. A. Saunders. MINRES-QLP: a Krylov subspace method for indefinite or singular symmetric systems. *SIAM J. Sci. Comput.*, 33(4):00–00, 2011. doi:10.1137/100787921.

[15] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-region Methods*, volume 1. SIAM, Philadelphia, 2000. doi:10.1137/1.9780898719857.

[16] G. Cramer. *Introduction à l'Analyse des Lignes Courbes Algébriques*. 1750.

[17] A. R. Curtis and J. K. Reid. On the automatic scaling of matrices for Gaussian elimination. *J. Inst. Maths. Applics.*, 10:118–124, 1972. doi:10.1093/imamat/10.1.118.

[18] T. A. Davis. University of Florida Sparse Matrix Collection. `http://www.cise.ufl.edu/research/sparse/matrices`.

[19] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK Users' Guide*. SIAM, Philadelphia, 1979. doi:10.1137/1.9781611971811.

[20] F. A. Dul. MINRES and MINERR are better than SYMMLQ in eigenpair computations. *SIAM J. Sci. Comput.*, 19(6): 1767–1782, 1998. doi:10.1137/S106482759528226X.

[21] J. M. Elble and N. V. Sahinidis. Scaling linear optimization problems prior to application of the simplex method. *Computational Optimization and Applications*, pages 1–27, 2011. doi:10.1007/s10589-011-9420-4.

[22] D. K. Faddeev and V. N. Faddeeva. *Computational Methods of Linear Algebra*. Freeman, London, 1963. doi:10.1007/BF01086544.

[23] R. Fletcher. Conjugate gradient methods for indefinite systems. In G. Watson, editor, *Numerical Analysis*, volume 506 of *Lecture Notes in Mathematics*, pages 73–89. Springer, Berlin / Heidelberg, 1976. doi:10.1007/BFb0080116.

[24] D. C.-L. Fong and M. A. Saunders. LSMR: An iterative algorithm for sparse least-squares problems. *SIAM J. Sci. Comput.*, 33:2950–2971, 2011. doi:10.1137/10079687X.

[25] V. Frayssé. The power of backward error analysis. Technical Report TH/PA/00/65, CERFACS, 2000.

[26] R. W. Freund and N. M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numerische Mathematik*, 60:315–339, 1991. doi:10.1007/BF01385726.

[27] R. W. Freund, G. H. Golub, and N. M. Nachtigal. Iterative solution of linear systems. *Acta Numerica*, 1:57–100, 1992. doi:10.1017/S0962492900002245.

[28] D. R. Fulkerson and P. Wolfe. An algorithm for scaling matrices. *SIAM Review*, 4(2):142–146, 1962. doi:10.1137/1004032.

[29] G. H. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *J. of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224, 1965. doi:10.1137/0702016.

[30] G. H. Golub and G. A. Meurant. Matrices, moments and quadrature II; How to compute the norm of the error in iterative methods. *BIT Numerical Mathematics*, 37:687–705, 1997. doi:10.1007/BF02510247.

[31] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, third edition, 1996.

[32] S. Gratton, P. Jiránek, and D. Titley-Péloquin. On the accuracy of the Karlson-Waldén estimate of the backward error for linear least squares problems. Submitted for publication (SIMAX), 2011.

[33] J. F. Grcar. Mathematicians of Gaussian elimination. *Notices of the AMS*, 58(6):782–792, 2011.

[34] J. F. Grcar, M. A. Saunders, and Z. Su. Estimates of optimal backward perturbations for linear least squares problems. Report SOL 2007-1, Department of Management Science and Engineering, Stanford University, Stanford, CA, 2007. 21 pp.

[35] A. Greenbaum. *Iterative Methods for Solving Linear Systems*. SIAM, Philadelphia, 1997. doi:10.1137/1.9781611970937.

[36] R. W. Hamming. *Introduction to Applied Numerical Analysis*. McGraw-Hill computer science series. McGraw-Hill, 1971.

[37] K. Hayami, J.-F. Yin, and T. Ito. GMRES methods for least squares problems. *SIAM J. Matrix Anal. Appl.*, 31(5): 2400–2430, 2010. doi:10.1137/070696313.

[38] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952.

[39] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, second edition, 2002. doi:10.1137/1.9780898718027.

[40] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Bur. Standards*, 45:255–282, 1950.

[41] L. Ljung. *System Identification: Theory for the User*. Prentice Hall, second edition, January 1999.

[42] LSMR. Software for linear systems and least squares. `http://www.stanford.edu/group/SOL/software.html`.

[43] D. G. Luenberger. Hyperbolic pairs in the method of conjugate gradients. *SIAM J. Appl. Math.*, 17:1263–1267, 1969. doi:10.1137/0117118.

[44] D. G. Luenberger. The conjugate residual method for constrained minimization problems. *SIAM J. Numer. Anal.*, 7 (3):390–398, 1970. doi:10.1137/0707032.

[45] W. Menke. *Geophysical Data Analysis: Discrete Inverse Theory*, volume 45. Academic Press, San Diego, 1989.

[46] G. A. Meurant. *The Lanczos and Conjugate Gradient Algorithms: From Theory to Finite Precision Computations*, volume 19 of *Software, Environments, and Tools*. SIAM, Philadelphia, 2006.

[47] J. Modersitzki, H. A. Van der Vorst, and G. L. G. Sleijpen. Differences in the effects of rounding errors in Krylov solvers for symmetric indefinite linear systems. *SIAM J. Matrix Anal. Appl.*, 22(3):726–751, 2001. doi:10.1137/S0895479897323087.

[48] W. Oettli and W. Prager. Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides. *Numerische Mathematik*, 6:405–409, 1964. doi:10.1007/BF01386090.

[49] A. M. Ostrowski. On the convergence of the Rayleigh quotient iteration for the computation of the characteristic roots and vectors. I. *Archive for Rational Mechanics and Analysis*, 1(1):233–241, 1957. doi:10.1007/BF00298007.

[50] P. Jiránek and D. Titley-Péloquin. Estimating the backward error in LSQR. *SIAM J. Matrix Anal. Appl.*, 31(4):2055–2074, 2010. doi:10.1137/090770655.

[51] C. C. Paige. Bidiagonalization of matrices and solution of linear equations. *SIAM J. Numer. Anal.*, 11(1):197–209, 1974. doi:10.1137/0711019.

[52] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12 (4):617–629, 1975. doi:10.1137/0712047.

[53] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Softw.*, 8(1):43–71, 1982. doi:10.1145/355984.355989.

[54] C. C. Paige and M. A. Saunders. Algorithm 583; LSQR: Sparse linear equations and least-squares problems. *ACM Trans. Math. Softw.*, 8(2):195–209, 1982. doi:10.1145/355993.356000.

[55] C. C. Paige, M. Rozloznik, and Z. Strakos. Modified Gram-Schmidt (MGS), least squares, and backward stability of MGS-GMRES. *SIAM J. Matrix Anal. Appl.*, 28(1):264–284, 2006. doi:10.1137/050630416.

[56] B. N. Parlett and W. Kahan. On the convergence of a practical QR algorithm. *Information Processing*, 68:114–118, 1969.

[57] V. Pereyra, 2010. Private communication.

[58] PROPACK. Software for SVD of sparse matrices. `http://soi.stanford.edu/~rmunk/PROPACK/`.

[59] B. J. W. S. Rayleigh. *The Theory of Sound*, volume 2. Macmillan, 1896.

[60] J. K. Reid. The use of conjugate gradients for systems of linear equations possessing "Property A". *SIAM J. Numer. Anal.*, 9(2):325–332, 1972. doi:10.1137/0709032.

[61] J. L. Rigal and J. Gaches. On the compatibility of a given solution with the data of a linear system. *J. ACM*, 14(3): 543–548, 1967. doi:10.1145/321406.321416.

[62] Y. Saad. Krylov subspace methods for solving large unsymmetric linear systems. *Mathematics of Computation*, 37 (155):pp. 105–126, 1981. doi:10.2307/2007504.

[63] Y. Saad and M. H. Schultz. GMRES: a generalized minimum residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. and Statist. Comput.*, 7(3):856–869, 1986. doi:10.1137/0907058.

[64] K. Shen, J. N. Crossley, A. W. C. Lun, and H. Liu. *The Nine Chapters on the Mathematical Art: Companion and Commentary*. Oxford University Press, New York, 1999.

[65] H. D. Simon and H. Zha. Low-rank matrix approximation using the Lanczos bidiagonalization process with applications. *SIAM J. Sci. Comput.*, 21(6):2257–2274, 2000. doi:10.1137/S1064827597327309.

[66] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 10:36–52, 1989. doi:10.1137/0910004.

[67] P. Sonneveld and M. B. van Gijzen. IDR(s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations. *SIAM J. Sci. Comput.*, 31:1035–1062, 2008. doi:10.1137/070685804.

[68] T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM J. Numer. Anal.*, 20 (3):626–637, 1983. doi:10.1137/0720042.

[69] G. W. Stewart. An inverse perturbation theorem for the linear least squares problem. *SIGNUM Newsletter*, 10:39–40, 1975. doi:10.1145/1053197.1053199.

[70] G. W. Stewart. Research, development and LINPACK. In J. R. Rice, editor, *Mathematical Software III*, pages 1–14. Academic Press, New York, 1977.

[71] G. W. Stewart. The QLP approximation to the singular value decomposition. *SIAM J. Sci. Comput.*, 20(4):1336–1348, 1999. ISSN 1064-8275. doi:10.1137/S1064827597319519.

[72] E. Stiefel. Relaxationsmethoden bester strategie zur lösung linearer gleichungssysteme. *Comm. Math. Helv.*, 29: 157–179, 1955. doi:10.1007/BF02564277.

[73] Z. Su. *Computational Methods for Least Squares Problems and Clinical Trials*. PhD thesis, SCCM, Stanford University, Stanford, CA, 2005.

[74] Y. Sun. *The Filter Algorithm for Solving Large-Scale Eigenproblems from Accelerator Simulations*. PhD thesis, Stanford University, Stanford, CA, 2003.

[75] D. Titley-Péloquin. *Backward Perturbation Analysis of Least Squares Problems*. PhD thesis, School of Computer Science, McGill University, Montreal, PQ, 2010.

[76] D. Titley-Peloquin. Convergence of backward error bounds in LSQR and LSMR. Private communication, 2011.

[77] J. A. Tomlin. On scaling linear programming problems. In *Computational Practice in Mathematical Programming*, volume 4 of *Mathematical Programming Studies*, pages 146–166. Springer, Berlin / Heidelberg, 1975. doi:10.1007/BFb0120718.

[78] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 13(2):631–644, 1992. doi:10.1137/0913035.

[79] H. A. van der Vorst. *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press, Cambridge, first edition, 2003. doi:10.1017/CBO9780511615115.

[80] S. J. van der Walt. *Super-resolution Imaging*. PhD thesis, Stellenbosch University, Cape Town, South Africa, 2010.

[81] M. B. van Gijzen. IDR(s) website. http://ta.twi.tudelft.nl/nw/users/gijzen/IDR.html.

[82] M. B. van Gijzen. Private communication, Dec 2010.

[83] B. Waldén, R. Karlson, and J.-G. Sun. Optimal backward perturbation bounds for the linear least squares problem. *Numerical Linear Algebra with Applications*, 2(3):271–286, 1995. doi:10.1002/nla.1680020308.

[84] D. S. Watkins. *Fundamentals of Matrix Computations*. Pure and Applied Mathematics. Wiley, Hoboken, NJ, third edition, 2010.

[85] P. Wesseling and P. Sonneveld. Numerical experiments with a multiple grid and a preconditioned Lanczos type method. In Reimund Rautmann, editor, *Approximation Methods for Navier-Stokes Problems*, volume 771 of *Lecture Notes in Mathematics*, pages 543–562. Springer, Berlin / Heidelberg, 1980. doi:10.1007/BFb0086930.

[86] F. Xue and H. C. Elman. Convergence analysis of iterative solvers in inexact Rayleigh quotient iteration. *SIAM J. Matrix Anal. Appl.*, 31(3):877–899, 2009. doi:10.1137/080712908.

[87] D. Young. Iterative methods for solving partial difference equations of elliptic type. *Trans. Amer. Math. Soc*, 76(92): 111, 1954. doi:10.2307/1990745.

[88] D. M. Young. *Iterative Methods for Solving Partial Difference Equations of Elliptic Type*. PhD thesis, Harvard University, Cambridge, MA, 1950.