IMPLICITLY RESTARTED KRYLOV SUBSPACE METHODS FOR LARGE-SCALE LEAST-SQUARES PROBLEMS

Daniel J. Richmond
University of Rhode Island, djrichmond1@gmail.com

IMPLICITLY RESTARTED KRYLOV SUBSPACE METHODS FOR LARGE-SCALE LEAST-SQUARES PROBLEMS

BY

DANIEL J. RICHMOND

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

MATHEMATICS

UNIVERSITY OF RHODE ISLAND

2014

DOCTOR OF PHILOSOPHY DISSERTATION

OF

DANIEL J. RICHMOND

APPROVED:

Dissertation Committee:

Major Professor	James Baglama
	Tom Bella
	Richard Vaccaro
	Nasser H. Zawia DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2014

ABSTRACT

The LSQR algorithm is a popular Krylov subspace method for obtaining solutions to large-scale least-squares problems. For some matrices, however, LSQR may require a prohibitively large number of iterations to determine an approximate solution within a desired accuracy. This is often the case when the solution vector has large components in the direction of the singular vectors associated with the smallest singular values of the matrix. This dissertation describes how the Krylov subspaces generated from LSQR can be conveniently updated to contain good approximations to the singular vectors corresponding to the smallest singular values of the matrix. The updates can be carried out by using harmonic Ritz vectors to augment the Krylov subspaces, or by applying harmonic Ritz values as implicit shifts. Computed examples show each proposed method to be competitive with existing methods. Furthermore, theoretical results show the connection between the proposed methods, and MATLAB functions and demos are provided showing their implementation and correct use.

ACKNOWLEDGMENTS

I would first like to express my gratitude to Dr. James Baglama for designing an accessible and interesting research project for me. I consider myself fortunate to have met James and even more fortunate to have him as my academic advisor. Without his guidance, support, and patience, many professional relationships, a career opportunity, and of course this dissertation, would not have been possible. James devoted a lot of time and patience in guiding me through all facets of the research project. This includes helping me learn and become proficient with the programming language MATLAB, understanding important background material, creating publishable work, and correcting my mistakes. I believe James Baglama is one of the top advisors a mathematics student could hope to work with.

I would like to thank Dr. Lothar Reichel. His previous work with Dr. Baglama, having discussions with him, and his comments proved invaluable for completing my first research paper.

I would like to acknowledge the generosity of the mathematics department for funding me with a teaching assistantship for four years, providing me with opportunities to teach over the summers, lending me a computer, and providing generous office space. The faculty and professors here have all been friendly, encouraging, provided a great environment for learning, and formed an excellent network of support. I am especially grateful to Tom Bella, Li Wu, and the electrical engineering department's Richard Vaccaro and Steven Kay, all for agreeing to be part of my dissertation committee, and offering valuable comments which improved this dissertation.

Last but certainly not least, I would like to thank my family for their unending support and encouragement throughout all my scholarly activities.

PREFACE

This dissertation has been prepared using manuscript format and contains as the main body of work the two research papers: "An augmented LSQR method" by James Baglama, Lothar Reichel, and Daniel Richmond, published in *Numerical Algorithms*, volume 64, issue 2 (2013) pp. 263-293, and "Implicitly restarting the LSQR algorithm" by James Baglama and Daniel Richmond, which has been accepted for publication in *Electronic Transactions on Numerical Analysis* on February 7, 2014, but has yet to appear as of this writing.

Dr. James Baglama was involved with the ideas, design, and writing of both research papers and Dr. Lothar Reichel assisted in providing ideas, comments, and writing for the first research paper.

A list of references used for the respective manuscripts are provided at the end of each, and a bibliography containing all references used is given at the end of this dissertation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introductory remarks

1.1 Statement of the problem

The focus of this dissertation is the development and implementation of new and improved algorithms to provide fast and efficient solutions, $x \in \mathbb{R}^n$, to large-scale least-squares (LS) problems of the form

$$\min_{x\in\mathbb{R}^n} \|b - Ax\|_2, \qquad A \in \mathbb{R}^{\ell\times n}, \qquad b \in \mathbb{R}^{\ell}, \qquad \ell \geq n. \tag{1.1}$$

The matrix A is considered to be sparse, and too large to effectively apply the use of direct methods (e.g., QR factorization, singular value decomposition), therefore, iterative methods (e.g., stationary, Krylov subsapce) must be used. There are many iterative methods that provide solutions to (1.1), and the most popular is the Krylov subspace method LSQR of Paige and Saunders [1]. The LSQR method can, however, exhibit extremely slow, or even no convergence in circumstances where the matrix A and/or the solution x have certain undesired properties. The most profound conditions for causing slow/no convergence are if the matrix A is ill-conditioned, or the solution x has strong components in the direction of the right singular vectors associated with the smallest singular values. Finding fast and efficient solutions to this class of LS problems is the primary focus of this dissertation.

1.2 Motivation

The need to develop fast and efficient solvers for large-scale LS problems is apparent in many applications across many scientific disciplines. The increase in computational power, memory, and instrumentation has provided the ability to quickly compute and store larger sets of data for various problems, and as a result the matrices in corresponding LS problems have grown larger. In turn, these larger problems tend to lead to ill-conditioned matrices. The need for fast and efficient LS solvers is so apparent that *The Matrix Market Collection* [2] and *The University of Florida Sparse Matrix Collection* [3] provide forums for researchers in scientific fields to upload matrices from LS (and other) problems that arise in their disciplines. These problems are ones which researchers are interested in solving, however, existing solvers generally do not perform well on, or in some cases, cannot even obtain a viable solution.

For example, the need to solve large-scale LS problems is vital in the discipline of surveying. The original matrices tested by Paige and Saunders [1] in determining the effectiveness of the LSQR algorithm were problems from surveying of size 1850×712. These problems are still currently listed on *The Matrix Market* and *The University of Florida Sparse Matrix Collection.* By today's standards, problems of this size are not considered large-scale, however, modern geodetic survey problems contain many large-scale LS problems, see e.g., [4]. Furthermore, the techniques of X-ray tomography and microtomography also require the solution of large-scale LS problems for image reconstruction that can be up to the order of millions, see e.g., [5, 6] and references therein. As recent as 2005, researchers in this field have expressed the desire and need for innovative iterative methods to solve their LS problems. There are, of course, many other applications where the solution of large-scale LS problems are necessary, and the desire for fast and efficient methods to solve large-scale LS problems by researchers in such fields serves as the motivation for improving and developing better LS solver algorithms.

1.3 Dissertation structure

The rest of the dissertation is structured as follows: Chapter 2 details an augmented LSQR method, in which the Krylov subspaces are augmented with the harmonic Ritz vectors associated with the smallest harmonic Ritz values. The algorithm which provides the basis for LSQR, the Golub-Kahan bidiagonalization, is given and explained. It is shown how to efficiently compute the harmonic Ritz vectors using the information given from the Golub-Kahan bidiagonalization, and how LSQR is effectively restarted on the augmented spaces. Theoretical results are given as to how using augmented Krylov subspaces improves the convergence rate of LSQR, and why the harmonic Ritz vectors are selected as augmenting vectors. Numerical examples are given at the end of the chapter.

Chapter 3 details an implicitly restarted LSQR method, in which the Krylov subspaces are improved by applying the largest harmonic Ritz values as implicit shifts and restarting LSQR. Results are given as to why harmonic Ritz values are chosen as shifts, and how they are effectively applied to successfully restart the LSQR algorithm. A gap strategy is proposed to provide a near optimal method for determining how many shifts should be applied. Theoretical results are given showing how the implicitly restarted LSQR method is connected with the augmented LSQR of Chapter 2, and numerical examples showing the competitiveness of the method are given at the end of the chapter.

Appendix A provides an overview of using a right preconditioner in conjunction with the LSQR algorithm. The creation of a specific preconditioner that is comparable to the methods proposed in Chapter 2 and Chapter 3 is proposed and a brief comparison is made between the Krylov subspaces that result from applying the proposed preconditioner and the Krylov subspaces from Chapter 2.

Finally, Appendix B provides all MATLAB [7] functions and demos necessary for the use of the augmented LSQR method of Chapter 2 and the implicitly restarted LSQR method of Chapter 3.

List of References

[1] C. C. Paige and M. A. Saunders, “LSQR: An algorithm for sparse linear equations and sparse least squares,” *ACM Transactions on Mathematical Software*, vol. 8, no. 1, pp. 43–71, 1982.

[2] R. F. Boisvert, R. Pozo, K. A. Remington, R. F. Barrett, and J. Dongarra, “Matrix Market: a web resource for test matrix collections,,” in *Quality of Numerical Software*, 1996, pp. 125–137.

[3] T. A. Davis and Y. Hu, “The University of Florida sparse matrix collection,” *ACM Transactions on Mathematical Software*, vol. 38, no. 1, p. 1, 2011.

[4] D. B. Zilkoski, J. H. Richards, and G. M. Young, “Special report: Results of the general adjustment of the North American vertical datum of 1988,” *Surveying and Land Information Systems*, vol. 52, no. 3, pp. 133–149, 1992.

[5] J. Baruchel, J.-Y. Buffiere, and E. Maire, *X-ray tomography in material science.* Hermes Science, 2000.

[6] E. Maire, A. Fazekas, L. Salvo, R. Dendievel, S. Youssef, P. Cloetens, and J. M. Letang, “X-ray tomography applied to the characterization of cellular materials. Related finite element modeling problems,” *Composites Science and Technology*, vol. 63, no. 16, pp. 2431–2443, 2003.

[7] MATLAB, *version R2011a.* Natick, Massachusetts: The MathWorks Inc., 2011.

CHAPTER 2

An augmented LSQR method

James Baglama[1], Lothar Reichel[2], and Daniel Richmond[3]

Published in *Numerical Algorithms*, volume 64, issue 2 (2013), pp. 263-293.

[1]Professor, Department of Mathematics, University of Rhode Island, Kingston, RI 02881.

e-mail: jbaglama@math.uri.edu

URL: http://math.uri.edu/∼jbaglama

[2]Professor, Department of Mathematical Sciences, Kent State University, Kent, OH 44242.

e-mail: reichel@math.kent.edu

URL: http://www.math.kent.edu/∼reichel

[3]PhD Candidate, Department of Mathematics, University of Rhode Island, Kingston, RI 02881.

e-mail: dan@math.uri.edu

URL: http://math.uri.edu/∼dan

Abstract. The LSQR iterative method when applied to the solution of least-squares problems may require many iterations to determine an approximate solution with desired accuracy. This often depends on the fact that singular vector components of the solution associated with small singular values of the matrix require many iterations to be determined. Augmentation of Krylov subspaces with harmonic Ritz vectors often makes it possible to determine the singular vectors associated with small singular values with fewer iterations than without augmentation. This paper describes how Krylov subspaces generated by the LSQR iterative method can be conveniently augmented with harmonic Ritz vectors. The augmentation may be viewed as adaptive preconditioning of LSQR. Computed examples illustrate the competitiveness of the preconditioned LSQR method proposed.

Keywords. Partial singular value decomposition, iterative method, large-scale computation, least-squares approximation, LSQR, precondition, Krylov subspace, augmentation.

AMS Subject Classification. 65F15, 15A18

2.1 Introduction

We consider the solution of least-squares (LS) problems

$$\min_{x\in\mathbb{R}^n}\|b-Ax\|, \tag{2.1}$$

where $A\in\mathbb{R}^{\ell\times n}$ is a large sparse matrix with $\ell\geq n$ and $b\in\mathbb{R}^{\ell}$. Throughout, $\|\cdot\|$ denotes the Euclidean vector norm or the associated induced matrix norm. The matrix A is assumed to be too large to be factored. We therefore seek to solve (2.1) by an iterative method. Unless stated otherwise, A is assumed to have full column rank. Problem (2.1) then has a unique solution, which we denote by x^+. The associated residual vector $r^+=b-Ax^+$ vanishes if and only if b lies in the range of A, denoted by $\mathcal{R}(A)$.

Many iterative methods have been proposed for the solution of (2.1); see, e.g., [1, 2, 3, 4, 5, 6, 7] and references therein. A popular method is LSQR by Paige and Saunders [6]. This method does not require the matrix A to be stored; instead each iteration requires that one matrix-vector product with A and one matrix-vector product with A^T be evaluated. A mathematically, but not numerically, equivalent method is CGLS proposed by Björck; see, e.g., [1] for a discussion of CGLS.

LSQR [6] is based on the Golub-Kahan (GK) bidiagonalization of A. Let x_0 be an initial approximate solution of (2.1) and define $r_0 = b - Ax_0$. Generically, $m \ll \min\{\ell, n\}$ steps of the GK bidiagonalization determine orthonormal bases $\{q_1, q_2, \dots, q_m\}$ and $\{p_1, p_2, \dots, p_m\}$ for the Krylov subspaces

$$\begin{aligned} \mathcal{K}_m\left(AA^T, q_1\right) &= \operatorname{span}\left\{q_1, AA^Tq_1, (AA^T)^2q_1, \dots, (AA^T)^{m-1}q_1\right\} \\ \mathcal{K}_m\left(A^TA, p_1\right) &= \operatorname{span}\left\{p_1, A^TAp_1, (A^TA)^2p_1, \dots, (A^TA)^{m-1}p_1\right\} \end{aligned} \tag{2.2}$$

respectively, with initial vectors $q_1 = r_0/\|r_0\|$ and $p_1 = A^Tq_1/\|A^Tq_1\|$. LSQR computes an approximate solution x_m of (2.1) by minimizing $\|b - Ax\|$ over the set $x_0 + \mathcal{K}_m\left(A^TA, p_1\right)$. The associated residual vector $r_m = b - Ax_m$ lies in $\mathcal{K}_m\left(AA^T, q_1\right)$; see [6] or Section 2.4 for details.

The GK bidiagonalization, and therefore LSQR, will in exact arithmetic terminate before m steps have been carried out if the Krylov subspace $\mathcal{K}_m(A^TA, p_1)$ is of dimension less than m. LSQR delivers, in this situation, the solution of (2.1). However, early termination is rare and it is common for LSQR to require many iterations before an approximation of the solution x^+ of (2.1) of desired accuracy has been determined. The rate of convergence of LSQR depends on the condition number of A and on the distribution of the singular values of the matrix; convergence may be slow when A has a large condition number; see [1] or Section 2.2 for details.

The rate of convergence of LSQR can be improved by using a preconditioner.

Instead of solving (2.1), one may solve the right preconditioned LS problem

$$\min_{y\in\mathbb{R}^n} \|AMy - b\|. \tag{2.3}$$

The preconditioner $M \in \mathbb{R}^{n\times n}$ should be nonsingular and such that i) the condition number of AM is smaller than the condition number of A, or AM has improved clustering of its singular values, and ii) matrix-vector products with the matrices M and M^T can be evaluated fairly quickly; see, e.g., [8, 1, 9, 4, 10, 11] and references therein for several approaches to constructing preconditioners. Many such preconditioners are constructed prior to computing a solution to the LS problem, and their determination may require significant computational effort and storage. Preconditioners affect the Krylov subspaces in which approximate solutions are determined. We describe another approach for modifying Krylov subspaces in which approximate solutions are computed. Specifically, we determine approximations of the singular vectors of A associated with the smallest singular values and augment the Krylov subspaces (2.2) by these vectors. This augmentation is carried out while improved approximate solutions of (2.1) are computed, and changes the Krylov subspaces to improve convergence. Our method can be used in conjunction with a preconditioner.

The idea of augmenting a Krylov subspace with vectors to improve convergence was first discussed by Morgan [12], who considered the solution of linear systems of equations with a square nonsingular matrix by GMRES. Morgan proposed to augment the Krylov subspaces used by GMRES with harmonic Ritz vectors associated with the harmonic Ritz values of smallest magnitude to increase the rate of convergence. Subsequently, Morgan showed in [13, 14] that the residual vectors associated with the harmonic Ritz vectors are multiples of the residual vector at every restart of the (standard) GMRES method and that, therefore, the augmented Krylov subspace is a Krylov subspace generated by a different starting vector. This result suggested that the augmenting vectors should be chosen to be harmonic Ritz vectors.

The initial iterations of our augmentation method for LSQR is analogous to Morgan's augmented method for GMRES [14] in that we augment the Krylov subspaces (2.2) with harmonic Ritz vectors for AA^T and associated vectors for A^TA. During the initial iterations with LSQR, we compute both improved approximations of the solution of (2.1) and improved approximations to harmonic Ritz vectors. When the latter approximations are deemed accurate enough, we stop updating these vectors and carry out LSQR iterations using augmented Krylov subspaces until a solution of (2.1) with desired accuracy has been found; the solution subspaces are augmented with fixed harmonic Ritz vectors.

Section 2.2 discusses convergence of LSQR when the Krylov subspaces (2.2) are augmented with singular vectors of A associated with the smallest singular values. These singular vectors generally are not explicitly known. We therefore describe in Section 2.3 how approximations of these vectors can be computed by a restarted GK bidiagonalization method, which is augmented by harmonic Ritz vectors of AA^T associated with the smallest harmonic Ritz values and with related vectors for A^TA. The method is related to a scheme described in [15], but differs in certain design aspects to fit better with the restarted LSQR method described in Section 2.4. In Section 2.5 we show that all residual vectors of the harmonic Ritz vectors are multiples of the residual vector of the restarted LSQR method. This result is important for the design of our augmented LSQR method. It implies that the augmented Krylov subspaces also are Krylov subspaces. Moreover, Section 2.5 describes our augmented LSQR method. Application of this algorithm to LS problems (2.1) with a rank-deficient matrix A is discussed in Section 2.6. A few numerical examples are presented in Section 2.7 and concluding remarks can be found in Section 2.8.

We would like to emphasize that the proposed iterative method is not a restarted LSQR method. Restarting may lead to stagnation; see [3, Section 7.3.1] for remarks on restarting the related LSMR method. Our method consists of two stages: i) the augmenting stage, which uses restarted LSQR to approximate the singular vectors associated with the smallest singular values of A and simultaneously improve an available approximation of the solution of (2.1), and ii) the LSQR stage, in which LSQR is applied using the augmented Krylov subspaces with fixed harmonic Ritz vectors to solve the LS problem (2.1).

2.2 Convergence of LSQR using augmented Krylov subspaces

Let u_i and v_i denote the left and right singular vectors of A associated with the singular value σ_i. Define $U_n = [u_1, u_2, \ldots, u_n] \in \mathbb{R}^{\ell \times n}$ and $V_n = [v_1, v_2, \ldots, v_n] \in \mathbb{R}^{n \times n}$ with orthonormal columns, as well as $\Sigma_n = \operatorname{diag}[\sigma_1, \sigma_2, \ldots, \sigma_n] \in \mathbb{R}^{n \times n}$. Then

$$AV_n = U_n\Sigma_n \qquad \text{and} \qquad A^TU_n = V_n\Sigma_n \tag{2.4}$$

are singular value decompositions (SVDs) of A and A^T, respectively. We assume the singular values to be ordered from the smallest to the largest one, i.e.,

$$0 < \sigma_1 \leq \sigma_2 \leq \ldots \leq \sigma_n.$$

While this ordering is nonstandard, it simplifies the notation in the subsequent sections. The condition number of A is given by $\kappa(A) = \sigma_n/\sigma_1$. The residual $r_m = b - Ax_m$ associated with the mth iterate, x_m, determined by LSQR with initial approximate solution x_0 satisfies

$$\|r_m - r^+\| \leq 2\left(\frac{\sigma_n - \sigma_1}{\sigma_n + \sigma_1}\right)^m \|r_0 - r^+\| = 2\left(\frac{\kappa(A) - 1}{\kappa(A) + 1}\right)^m \|r_0 - r^+\|, \tag{2.5}$$

where x^+ denotes the solution of (2.1) and r^+ is the corresponding residual; see [1]. Furthermore, if $b \in \mathcal{R}(A)$, then

$$\|r_m\| \leq 2\left(\frac{\sigma_n - \sigma_1}{\sigma_n + \sigma_1}\right)^m \|r_0\|.$$

From equation (2.5), it can be seen that for well-conditioned LS problems, LSQR will converge quickly, however, ill-conditioned problems may require a prohibitively large number of iterations. The use of a preconditioner M with $\kappa(AM) \ll \kappa(A)$ may alleviate this difficulty.

We first describe how augmentation of the Krylov subspaces (2.2) by singular vectors of A associated with the smallest singular values reduces the bound (2.5) and therefore can be expected to speed up convergence. Thus, consider the augmented Krylov subspaces

$$\begin{aligned} \mathcal{K}_m(AA^T, u_1, \ldots, u_k, q_1) &= \text{span}\left\{u_1, \ldots, u_k, q_1, AA^T q_1, \ldots, (AA^T)^{m-k-1} q_1\right\} \\ \mathcal{K}_m(A^T A, v_1, \ldots, v_k, p_1) &= \text{span}\left\{v_1, \ldots, v_k, p_1, A^T A p_1, \ldots, (A^T A)^{m-k-1} p_1\right\} \end{aligned} \tag{2.6}$$

obtained by augmenting the Krylov subspace $\mathcal{K}_{m-k}\left(AA^T, q_1\right)$ by the left singular vectors $u_1, \ldots, u_k$ associated with the k smallest singular values, and by augmenting $\mathcal{K}_{m-k}\left(A^T A, p_1\right)$ by the corresponding right singular vectors $v_1, \ldots, v_k$. At iteration m, the augmented method determines an approximate solution in a subspace of at most dimension m. The following result shows that the upper bound for the residual error (2.5) may be reduced considerably by augmentation.

Theorem 2.1. *Let $A \in \mathbb{R}^{\ell \times n}$ have the SVD (2.4) and let x_m minimize $\|b - Ax\|$ over the augmented and shifted Krylov subspace $x_0 + \mathcal{K}_m\left(A^T A, v_1, \ldots, v_k, p_1\right)$. Then with $r_m = b - Ax_m$,*

$$\|r_m - r^+\| \leq 2\left(\frac{\sigma_n - \sigma_{k+1}}{\sigma_n + \sigma_{k+1}}\right)^{m-k} \|r_0 - r^+\|.$$

Proof. Let x_m be *any* vector from $x_0 + \mathcal{K}_m\left(A^TA, v_1, \ldots, v_k, p_1\right)$ and define $r_m = b - Ax_m$. Then

$$x_m = x_0 + \sum_{i=1}^{k} \tau_i v_i + \phi(A^TA)A^T r_0, \tag{2.7}$$

where ϕ is a polynomial of degree at most $m-k-1$ and $\tau_i \in \mathbb{R}$. Let $P_{\mathcal{R}(A)}$ and $P_{\mathcal{N}(A^T)}$ denote the orthogonal projectors onto the range of A and the null space of A^T, respectively. Split the vector b according to

$$b = P_{\mathcal{R}(A)}b + P_{\mathcal{N}(A^T)}b = \sum_{i=1}^{n} \omega_i u_i + P_{\mathcal{N}(A^T)}b,$$

where the u_i are the left singular vectors of A; cf. (2.4). Then

$$A^T r_0 = A^T b - A^T A x_0 = \sum_{i=1}^{n} \omega_i A^T u_i - A^T A x_0 = \sum_{i=1}^{n} \tilde{\omega}_i v_i \tag{2.8}$$

since $\{v_1, \ldots, v_n\}$ is an orthonormal basis for $\mathbb{R}^n$. Using (2.7) and (2.8) we obtain

$$A^T r_m = \psi(A^TA)A^T r_0 - \sum_{i=1}^{k} \tau_i \sigma_i^2 v_i = \sum_{i=1}^{n} \tilde{\omega}_i \psi(\sigma_i^2) v_i - \sum_{i=1}^{k} \tau_i \sigma_i^2 v_i, \tag{2.9}$$

where $\psi(x) = 1 - x\phi(x)$. Let $\gamma_i = -\tau_i \sigma_i^2 + \tilde{\omega}_i \psi(\sigma_i^2)$. Then

$$A^T r_m = \sum_{i=1}^{k} \gamma_i v_i + \sum_{i=k+1}^{n} \tilde{\omega}_i \psi(\sigma_i^2) v_i.$$

We may now choose $\tau_i = \frac{\tilde{\omega}_i \psi(\sigma_i^2)}{\sigma_i^2}$ to define x_m in (2.7). This yields $\gamma_i = 0$ and, therefore,

$$A^T r_m = \sum_{i=k+1}^{n} \tilde{\omega}_i \psi(\sigma_i^2) v_i. \tag{2.10}$$

Now let ψ be the shifted Chebyshev polynomial of degree $m-k-1$ for the interval $[\sigma_{k+1}^2, \sigma_n^2]$, scaled so that $\psi(0) = 1$, and take the $(A^TA)^{-1}$ norm of both sides of (2.10). Using properties of the scaled and shifted Chebyshev polynomial, we obtain

$$\|A^T r_m\|_{(A^TA)^{-1}} \leq 2\left(\frac{\sigma_n - \sigma_{k+1}}{\sigma_n + \sigma_{k+1}}\right)^{m-k} \|A^T r_0\|_{(A^TA)^{-1}}.$$

The desired result follows from the observations that

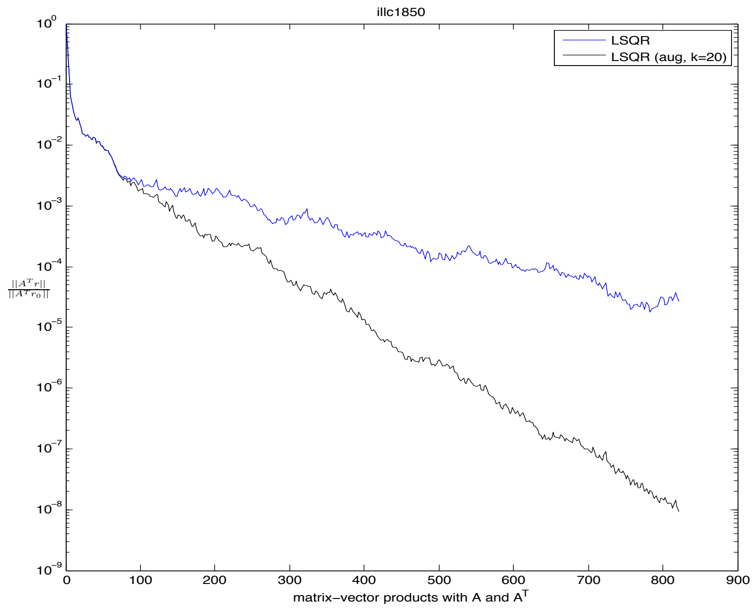
$$\|A^T r_m\|_{(A^TA)^{-1}} = \|r_m - r^+\| \tag{2.11}$$

and that the norm of the residual vector $r_m = b - Ax_m$ associated with the vector x_m in the statement of the theorem is at least as small as the norm obtained for our choices of τ and ψ. $\square$

Morgan [12] discussed the use of augmented Krylov subspaces of the form $\text{span}\{b, Ab, \ldots, A^{m-1}b, z_1, \ldots, z_k\}$, where $z_1, \ldots, z_k$ are eigenvectors of A, to increase the rate of convergence of restarted GMRES, and showed a result analogous to Theorem 2.1 for this situation.

Example 2.1. Let $A \in \mathbb{R}^{1850\times 712}$ be the matrix ILLC1850 and let b be the vector ILLC1850_RHS1 from the LSQ set of the Matrix Market Collection [16, 17]. Figure 2.1 compares the augmented LSQR method using the Krylov subspaces (2.6) with $k = 20$ and the standard LSQR method, with $x_0 = 0$ for both methods. Figure 2.1 displays the convergence of the quotients $\|A^T r\|/\|A^T r_0\|$ as a function of the number of matrix-vector products with A and A^T. Here $r_0 = b$ is the residual associated with the initial iterate x_0, and r is the residual associated with the currently available iterate. The top graph shows implementations of the methods with full reorthogonalization, while the bottom graph displays the performance of the methods without reorthogonalization. In this case, we see that reorthogonalization does not change the convergence behavior much, but that augmentation as described in Theorem 2.1 increases the rate of convergence significantly.

The initial vector q_1 for the Krylov subspace in the augmented Krylov subspace $\mathcal{K}_m\left(AA^T, u_1, \ldots, u_{20}, q_1\right)$ is orthogonalized against the $k = 20$ left singular vectors $\{u_1, \ldots, u_{20}\}$. This makes the vector $p_1 = A^T q_1/\|A^T q_1\|$ in the augmented Krylov subspace $\mathcal{K}_m\left(A^TA, v_1, \ldots, v_k, p_1\right)$ orthogonal to the right singular vectors $\{v_1, \ldots, v_{20}\}$.



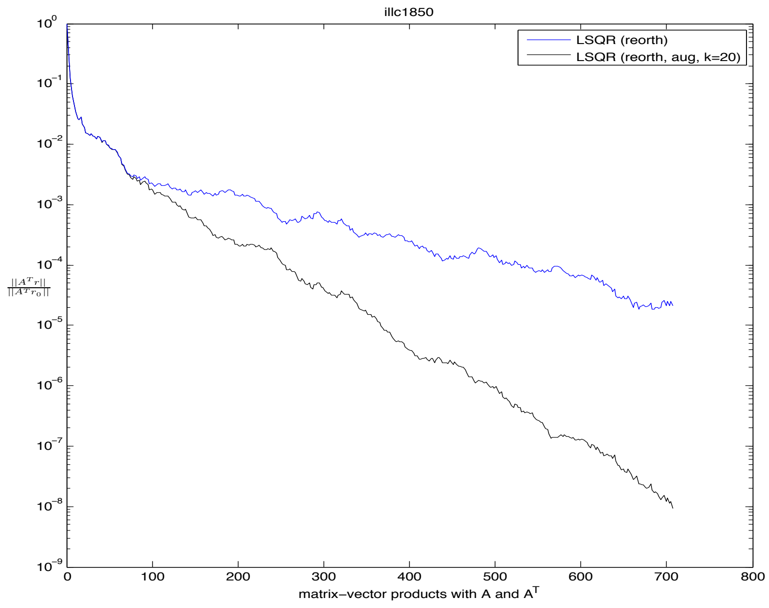


Figure 2.1. Example 2.1: A comparison of augmented and standard LSQR.

The singular vectors $\{u_1, \ldots, u_k\}$ and $\{v_1, \ldots, v_k\}$ associated with the k smallest singular values of A are generally not explicitly known. We therefore seek to determine approximations of these vectors while simultaneously computing improved approximations of the solution of (2.1). This is achieved with a restarted LSQR method. Typically augmenting vectors do not have to be accurate approximations of the singular vectors of A to yield beneficial results. This is illustrated by the following theorem as well as by numerical examples in Section 2.7. The theorem is an analog of a result by Morgan [12], concerned with augmenting a Krylov subspace by approximate eigenvectors to increase the rate of convergence of restarted GMRES.

Theorem 2.2. *Let $A \in \mathbb{R}^{\ell \times n}$ have the SVD (2.4) and let x_m minimize $\|b - Ax\|$ over the augmented and shifted Krylov subspace $x_0 + \mathcal{K}_m\left(A^TA, y_1, p_1\right)$, where the unit-length vector $y_1 \in \mathbb{R}^n$ is an approximation of the right singular vector v_1. Let ζ be the angle between y_1 and v_1, and let $\tilde{\omega}_1$ be defined in (2.8) from Theorem 2.1. Then with $r_m = b - Ax_m$,*

$$\|r_m - r^+\| \leq 2\left(\frac{\sigma_n - \sigma_2}{\sigma_n + \sigma_2}\right)^{m-1} \|r_0 - r^+\| + \frac{\|A^TA\|}{\sigma_1^2}\tan(\zeta)|\tilde{\omega}_1|. \tag{2.12}$$

Proof. Similarly to (2.7) and (2.9) we have

$$\begin{aligned} x_m &= x_0 + \tau_1 y_1 + \phi(A^TA)A^Tr_0, \\ A^Tr_m &= \sum_{i=1}^{n}\tilde{\omega}_i\psi(\sigma_i^2)v_i - \tau_1 A^TAy_1, \end{aligned} \tag{2.13}$$

where $\phi(x)$ is a polynomial of degree at most $m-2$ and $\psi(x) = 1 - x\phi(x)$ is a polynomial of degree at most $m-1$. Let

$$y_1 = \cos(\zeta)v_1 + \sin(\zeta)z, \tag{2.14}$$

where $z \in \text{span}\{v_2, \ldots, v_n\}$ is a unit-length vector. Using (2.14) and the SVD of A, equation (2.13) becomes

$$A^Tr_m = \sum_{i=1}^{n}\tilde{\omega}_i\psi(\sigma_i^2)v_i - \tau_1\sigma_1^2 v_1\cos(\zeta) - \tau_1 A^TAz\sin(\zeta).$$

With $\tau_1 = \frac{\tilde{\omega}_1 \psi(\sigma_1^2)}{\sigma_1^2 \cos(\zeta)}$, we obtain

$$A^T r_m = \sum_{i=2}^{n} \tilde{\omega}_i \psi(\sigma_i^2) v_i - \frac{\tilde{\omega}_1 \psi(\sigma_1^2) A^T A z \tan(\zeta)}{\sigma_1^2}. \tag{2.15}$$

Let ψ be the shifted Chebyshev polynomial for the interval $[\sigma_2^2, \sigma_n^2]$, scaled so that $\psi(0) = 1$, and take the $(A^T A)^{-1}$ norm of both sides of (2.15). Using properties of the shifted and scaled Chebyshev polynomials, we get

$$\|A^T r_m\|_{(A^T A)^{-1}} \leq \left(\frac{\sigma_n - \sigma_2}{\sigma_n + \sigma_2} \right)^{m-1} \|A^T r_0\|_{(A^T A)^{-1}} + \frac{\|A^T A\|}{\sigma_1^2} \tan(\zeta) |\tilde{\omega}_1|.$$

The theorem now follows from (2.11). □

We remark that the right-hand side of (2.12) shows that if the smallest singular value σ_1 is very close to zero or to σ_2, then y_1 has to be a fairly accurate approximation of the singular vector v_1 in order to be effective.

2.3 A restarted augmented GK bidiagonalization method

This section describes a restarted GK bidiagonalization method for approximating the singular triplets $\{\sigma_i, u_i, v_i\}_{i=1}^k$ associated with the k smallest singular values of A. We refer to these singular triplets as the k smallest singular triplets. Let the matrices $U_k \in \mathbb{R}^{\ell \times k}$ and $V_k \in \mathbb{R}^{n \times k}$ consist of the first k columns of the matrices U_n and V_n in the SVD (2.4) of A, and introduce $\Sigma_k = \text{diag}\,[\sigma_1, \ldots, \sigma_k] \in \mathbb{R}^{k \times k}$. Then, analogously to (2.4), we have the partial SVDs

$$AV_k = U_k \Sigma_k \qquad \text{and} \qquad A^T U_k = V_k \Sigma_k.$$

There are numerous methods available for computing approximations of the singular triplets $\{\sigma_i, u_i, v_i\}_{i=1}^k$; see, e.g., [15, 18, 19, 5, 20, 21, 22, 23, 24] and references therein. We are interested in using a method that is related to LSQR, so that while computing these approximations, we also can determine improved approximate solutions of (2.1). Therefore, we will use a restarted augmented harmonic

GK bidiagonalization method to determine approximations of the desired singular triplets. We show in Section 2.4 why this approach is attractive. The restarted augmented harmonic GK bidiagonalization method of this paper is closely related to the method presented in [15]; it differs in that here we use a lower bidiagonal matrix. This makes it easier to connect our method to LSQR. The following algorithm describes the computations required for a GK bidiagonalization. We comment on the algorithm below.

Algorithm 2.1. GK BIDIAGONALIZATION METHOD

```
Input: $A \in \mathbb{R}^{\ell \times n}$ or functions for evaluating products with $A$ and $A^T$,
       $q_1 \in \mathbb{R}^{\ell}$ : initial vector,
       $m$ : number of bidiagonalization steps.

Output: $P_m = [p_1, \ldots, p_m] \in \mathbb{R}^{n \times m}$ : matrix with orthonormal columns,
        $Q_{m+1} = [q_1, \ldots, q_{m+1}] \in \mathbb{R}^{\ell \times (m+1)}$ : matrix with orthonormal columns,
        $B_{m+1,m} \in \mathbb{R}^{(m+1) \times m}$ : lower bidiagonal matrix (2.17),
        $p_{m+1} \in \mathbb{R}^n$ : residual vector,
        $\alpha_{m+1} \in \mathbb{R}$.

1. Compute $\beta_1 := \|q_1\|$;   $q_1 := q_1/\beta_1$;   $Q_1 := q_1$
2. Compute $p_1 := A^T q_1$;   $\alpha_1 := \|p_1\|$;   $p_1 := p_1/\alpha_1$;   $P_1 := p_1$
3. For $j = 1 : m$
    4. Compute $q_{j+1} := Ap_j - q_j \alpha_j$
    5. Reorthogonalize: $q_{j+1} := q_{j+1} - Q_{(1:j)}(Q_{(1:j)}^T q_{j+1})$
    6. Compute $\beta_{j+1} := \|q_{j+1}\|$;   $q_{j+1} := q_{j+1}/\beta_{j+1}$;   $Q_{j+1} := [Q_j, q_{j+1}]$
    7. Compute $p_{j+1} := A^T q_{j+1} - p_j \beta_{j+1}$
    8. Reorthogonalize: $p_{j+1} := p_{j+1} - P_{(1:j)}(P_{(1:j)}^T p_{j+1})$
    9. Compute $\alpha_{j+1} := \|p_{j+1}\|$;   $p_{j+1} := p_{j+1}/\alpha_{j+1}$
    10. if $j < m$
        11. $P_{j+1} := [P_j, p_{j+1}]$
    12. End
13. End
```

To avoid loss of orthogonality due to finite precision arithmetic, we reorthogonalize in lines 5 and 8 of the algorithm; see Section 2.5 for a few remarks on reorthogonalization in the context of the GK bidiagonalization.

A matrix interpretation of the computations of Algorithm 2.1 shows that the algorithm determines the decompositions

$$\begin{aligned} A^T Q_{m+1} &= P_m B_{m+1,m}^T + \alpha_{m+1} p_{m+1} e_{m+1}^T \\ A P_m &= Q_{m+1} B_{m+1,m}, \end{aligned} \tag{2.16}$$

where the matrices $Q_{m+1} = [q_1, \dots, q_{m+1}] \in \mathbb{R}^{\ell \times (m+1)}$ and $P_m = [p_1, \dots, p_m] \in \mathbb{R}^{n \times m}$ have orthonormal columns, the residual vector $p_{m+1} \in \mathbb{R}^n$ satisfies $P_m^T p_{m+1} = 0$, and e_{m+1} is the $(m+1)$st axis vector of appropriate dimension. The matrix

$$B_{m+1,m} = \begin{bmatrix} \alpha_1 & & & 0 \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \ddots & \\ & & \ddots & \alpha_m \\ 0 & & & \beta_{m+1} \end{bmatrix} \in \mathbb{R}^{(m+1)\times m} \tag{2.17}$$

is lower bidiagonal. We refer to (2.16) as a partial GK bidiagonalization of A. The number of bidiagonalization steps $m \ll \min\{\ell, n\}$ is assumed to be small enough so that the partial bidiagonalization (2.16) with the stated properties exists. We assume in the following that Algorithm 2.1 does not terminate early, i.e., that all

$\alpha_j > 0$ and $\beta_j > 0$ for $1 \leq j \leq m+1$. Early termination will be commented on in Section 2.5.

The decompositions (2.16) are closely related to a partial Lanczos tridiagonalization of A^TA and AA^T. For instance, multiplying the second equation in (2.16) by A^T yields the partial Lanczos tridiagonalization of A^TA,

$$A^TAP_m = P_m B_{m+1,m}^T B_{m+1,m} + (\alpha_{m+1}\beta_{m+1})\, p_{m+1} e_m^T. \tag{2.18}$$

Analogously, multiplying the first equation in (2.16) by A gives

$$AA^TQ_{m+1} = Q_{m+1} B_{m+1,m} B_{m+1,m}^T + \alpha_{m+1} A p_{m+1} e_{m+1}^T,$$

and then equating the first m columns yields the partial Lanczos tridiagonalization of AA^T,

$$AA^TQ_m = Q_m B_m B_m^T + \alpha_m \beta_{m+1} q_{m+1} e_m^T, \tag{2.19}$$

where B_m is the leading $m \times m$ principal submatrix of $B_{m+1,m}$, $Q_m \in \mathbb{R}^{\ell \times m}$ consists of the first m columns of the matrix Q_{m+1}, and q_{m+1} is the last column of Q_{m+1}.

The LSQR method is started or restarted with Krylov subspaces of the form (2.2). We therefore consider the decomposition (2.19) for determining harmonic Ritz vectors. The harmonic Ritz values $\hat{\theta}_j$ of AA^T determined by (2.19) are the eigenvalues $\hat{\theta}_j$ of the generalized eigenvalue problem

$$\left(\left(B_m B_m^T\right) + \alpha_m^2 \beta_{m+1}^2 \left(B_m B_m^T\right)^{-1} e_m e_m^T \right) \tilde{g}_j = \hat{\theta}_j \tilde{g}_j, \quad 1 \leq j \leq m, \tag{2.20}$$

where $\tilde{g}_j \in \mathbb{R}^m \backslash \{0\}$ is an eigenvector; see, e.g., [25, 26] for properties of and discussions on harmonic Ritz values.

The eigenpairs $\left\{\hat{\theta}_j, \tilde{g}_j\right\}_{j=1}^m$ of (2.20) can be computed without forming the matrix $B_m B_m^T$. Instead, determine the SVD of $B_{m+1,m}$, which satisfies

$$\begin{aligned} B_{m+1,m}\tilde{V}_m &= \left[\tilde{U}_{m+1,m}\ \tilde{u}_{m+1}\right] \begin{bmatrix} \tilde{\Sigma}_m \\ 0 \end{bmatrix}, \\ B_{m+1,m}^T \left[\tilde{U}_{m+1,m}\ \tilde{u}_{m+1}\right] &= \tilde{V}_m \left[\tilde{\Sigma}_m \quad 0\right], \end{aligned} \tag{2.21}$$

where the matrices $\tilde{V}_m = [\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_m] \in \mathbb{R}^{m\times m}$ and $\tilde{U}_{m+1,m} = [\tilde{u}_1, \tilde{u}_2, \ldots, \tilde{u}_m] \in \mathbb{R}^{(m+1)\times m}$ have orthonormal columns, $\tilde{u}_{m+1} \in \mathbb{R}^{m+1}$ is a unit-length vector such that $\tilde{u}_{m+1}^T \tilde{U}_{m+1,m} = 0$, and $\tilde{\Sigma}_m = \operatorname{diag}[\tilde{\sigma}_1, \tilde{\sigma}_2, \ldots, \tilde{\sigma}_m] \in \mathbb{R}^{m\times m}$. We order the m singular values according to

$$0 < \tilde{\sigma}_1 \leq \tilde{\sigma}_2 \leq \ldots \leq \tilde{\sigma}_m.$$

The vector $\tilde{u}_{m+1}$ lies in $\mathcal{N}\left(B_{m+1,m}^T\right)$ and we will refer to it as the null space vector of $B_{m+1,m}^T$.

Consider the $(m+1)\times(m+1)$ symmetric tridiagonal matrix

$$B_{m+1,m}B_{m+1,m}^T = \left[\begin{array}{c|c} B_m B_m^T & \alpha_m \beta_{m+1} e_m \\ \hline \alpha_m \beta_{m+1} e_m^T & \beta_{m+1}^2 \end{array}\right].$$

The m nonvanishing eigenvalues of this matrix are harmonic Ritz values, i.e., they are the eigenvalues of (2.20). We have $\hat{\theta}_j = \tilde{\sigma}_j^2$; see [26]. The harmonic Ritz vectors of AA^T can be computed by using the matrix

$$S = \begin{bmatrix} I_m & \alpha_m\beta_{m+1}\left(B_m B_m^T\right)^{-1} e_m \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} I_m & \beta_{m+1} B_m^{-T} e_m \\ 0 & 1 \end{bmatrix}$$

and noticing that

$$SB_{m+1,m}B_{m+1,m}^T S^{-1} = \left[\begin{array}{c|c} B_m B_m^T + \alpha_m^2 \beta_{m+1}^2 (B_m B_m^T)^{-1} e_m e_m^T & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \hline \alpha_m \beta_{m+1} e_m^T & \end{array}\right].$$

Thus, the first m rows of $S\tilde{U}_{m+1,m}$ are the eigenvectors in (2.20), i.e.,

$$[\tilde{g}_1, \tilde{g}_2, \ldots, \tilde{g}_m] = \begin{bmatrix} I_m & \beta_{m+1} B_m^{-T} e_m \end{bmatrix} \tilde{U}_{m+1,m}.$$

It follows that the harmonic Ritz vector of AA^T associated with the harmonic Ritz value $\hat{\theta}_j$ is given by

$$\hat{u}_j := Q_m \tilde{g}_j. \tag{2.22}$$

Morgan [13] pointed out that the residual vectors associated with different harmonic Ritz pairs $\left\{\hat{\theta}_j, \hat{u}_j\right\}$ are parallel in the context of the Arnoldi process and GMRES. We show this result for the problem at hand, because this property is central for our augmentation method. Using (2.19), (2.20), and (2.22), we obtain

$$\begin{aligned}
AA^T\hat{u}_j - \hat{\theta}_j\hat{u}_j &= AA^T Q_m\tilde{g}_j - \hat{\theta}_j Q_m\tilde{g}_j \\
&= \left(Q_m B_m B_m^T + \alpha_m\beta_{m+1}q_{m+1}e_{m+1}^T\right)\tilde{g}_j - \hat{\theta}_j Q_m\tilde{g}_j \\
&= Q_m\left(B_m B_m^T - \hat{\theta}_j I_m\right)\tilde{g}_j + \alpha_m\beta_{m+1}q_{m+1}e_m^T\tilde{g}_j \\
&= Q_m\left(-\left(\alpha_m\beta_{m+1}\right)^2\left(B_m B_m^T\right)^{-1}e_m e_m^T\right)\tilde{g}_j + \alpha_m\beta_{m+1}q_{m+1}e_m^T\tilde{g}_j \\
&= \left(\alpha_m\beta_{m+1}e_m^T\tilde{g}_j\right)Q_{m+1}\begin{bmatrix}-\alpha_m\beta_{m+1}\left(B_m B_m^T\right)^{-1}e_m \\ 1\end{bmatrix} \\
&= \left(\alpha_m\beta_{m+1}e_m^T\tilde{g}_j\right)Q_{m+1}\begin{bmatrix}-\beta_{m+1}B_m^{-T}e_m \\ 1\end{bmatrix}.
\end{aligned}$$

This shows that all the residuals for the harmonic Ritz pairs for AA^T are multiples of the same vector.

Define the residual vector for the harmonic Ritz pairs,

$$r_m^{\text{harm}} = Q_{m+1}\begin{bmatrix}-\beta_{m+1}B_m^{-T}e_m \\ 1\end{bmatrix} \tag{2.23}$$

and assume that we are interested in the k smallest singular triplets. Our augmentation process can now be described by considering the starting matrix

$$\left[\hat{u}_1, \ldots, \hat{u}_k, r_m^{\text{harm}}\right] = Q_{m+1}\begin{bmatrix}\begin{bmatrix}I_m & \beta_{m+1}B_m^{-T}e_m\end{bmatrix}\tilde{U}_{m+1,k} & -\beta_{m+1}B_m^{-T}e_m \\ 0 & 1\end{bmatrix}. \tag{2.24}$$

The columns of the matrix in (2.24) are not orthogonal. We therefore compute its QR decomposition

$$\begin{bmatrix}\begin{bmatrix}I_m & \beta_{m+1}B_m^{-T}e_m\end{bmatrix}\tilde{U}_{m+1,k} & -\beta_{m+1}B_m^{-T}e_m \\ 0 & 1\end{bmatrix} = \tilde{Q}\tilde{R}, \tag{2.25}$$

where $\tilde{Q} \in \mathbb{R}^{(m+1)\times(k+1)}$ has orthonormal columns and $\tilde{R} \in \mathbb{R}^{(k+1)\times(k+1)}$ is upper triangular, and use

$$\hat{Q}_{k+1} = Q_{m+1}\tilde{Q} \tag{2.26}$$

as the starting matrix. Application of (2.16), (2.21), (2.23), and (2.25) yields

$$A^T\hat{Q}_{k+1} = A^T Q_{m+1}\tilde{Q} = \left[P_m\tilde{V}_k\tilde{\Sigma}_k \quad A^T r_m^{\text{harm}}\right]\tilde{R}^{-1}, \tag{2.27}$$

where $\tilde{V}_k = [\tilde{v}_1, \ldots, \tilde{v}_k]$ and $\tilde{\Sigma}_k = \text{diag}\,[\tilde{\sigma}_1, \tilde{\sigma}_2, \ldots, \tilde{\sigma}_k]$.

The relation

$$A^T r_m^{\text{harm}} = \alpha_{m+1}p_{m+1} \tag{2.28}$$

can be shown by using

$$A^T\hat{Q}_{k+1} = \left(P_m B_{m+1,m}^T + \alpha_{m+1}p_{m+1}e_{m+1}^T\right)\tilde{Q} \tag{2.29}$$

and by equating the right-hand sides of (2.27) and (2.29) and applying (2.25). Therefore, we have

$$\begin{aligned} A^T\hat{Q}_{k+1} &= \left[P_m\tilde{V}_k \quad p_{m+1}\right]\begin{bmatrix} \tilde{\sigma}_1 & & & & 0 \\ & \tilde{\sigma}_2 & & & \\ & & \ddots & & \\ & & & \tilde{\sigma}_k & \\ 0 & & & & \alpha_{m+1} \end{bmatrix}\tilde{R}^{-1} \\ &= \hat{P}_k\left(\tilde{\Sigma}_k\tilde{R}_{k,k+1}^{-1}\right) + \frac{\alpha_{m+1}}{\tilde{r}_{k+1,k+1}}p_{m+1}e_{k+1}^T, \end{aligned} \tag{2.30}$$

where

$$\hat{P}_k = P_m\tilde{V}_k, \tag{2.31}$$

the matrix $\tilde{R}_{k,k+1}^{-1}$ is the leading $k \times (k+1)$ submatrix of $\tilde{R}^{-1}$, and $\tilde{r}_{k+1,k+1}$ is the $(k+1)$st diagonal entry of $\tilde{R}$. It follows from the structure of the matrix on the left-hand side of (2.25) that $1/\tilde{r}_{k+1,k+1} = \tilde{q}_{m+1,k+1}$, the $(m+1, k+1)$-element of the matrix $\tilde{Q}$. It follows from $\hat{P}_k^T p_{m+1} = 0$ that

$$\hat{P}_k^T A^T\hat{Q}_{k+1} = \tilde{\Sigma}_k\tilde{R}_{k,k+1}^{-1}. \tag{2.32}$$

The decomposition (2.30) is important for the derivation of our iterative method; it is analogous to the first decomposition in (2.16).

We now derive a decomposition for $A\hat{P}_k$ that is analogous to the second decomposition in (2.16). Using (2.16), (2.21), and (2.31), we obtain

$$A\hat{P}_k = Q_{m+1}\tilde{U}_{m+1,k}\tilde{\Sigma}_k. \tag{2.33}$$

This gives

$$B_{m+1,m}^T = B_m^T \begin{bmatrix} I_m & \beta_{m+1}B_m^{-T}e_m \end{bmatrix},$$

and from (2.21) it follows that

$$\begin{bmatrix} I_m & \beta_{m+1}B_m^{-T}e_m \end{bmatrix}\tilde{U}_{m+1,k} = B_m^{-T}\tilde{V}_k\tilde{\Sigma}_k \tag{2.34}$$

and therefore

$$\tilde{U}_{m+1,k} = \begin{bmatrix} B_m^{-T}\tilde{V}_k\tilde{\Sigma}_k & -\beta_{m+1}B_m^{-T}e_m \\ 0 & 1 \end{bmatrix}\begin{bmatrix} I_k \\ e_{m+1}^T\tilde{U}_{m+1,k} \end{bmatrix}. \tag{2.35}$$

We obtain from (2.25), (2.34), and (2.35) that

$$\tilde{U}_{m+1,k} = \tilde{Q}\tilde{Q}^T\tilde{U}_{m+1,k}, \tag{2.36}$$

and inserting (2.36) into (2.33) yields

$$A\hat{P}_k = Q_{m+1}\tilde{Q}\tilde{Q}^T\tilde{U}_{m+1,k}\tilde{\Sigma}_k = \hat{Q}_{k+1}\tilde{Q}^T\tilde{U}_{m+1,k}\tilde{\Sigma}_k. \tag{2.37}$$

Now using (2.32) and (2.37), we get

$$\hat{Q}_{k+1}^T A\hat{P}_k = \tilde{Q}^T\tilde{U}_{m+1,k}\tilde{\Sigma}_k = \left(\tilde{\Sigma}_k\tilde{R}_{k,k+1}^{-1}\right)^T. \tag{2.38}$$

Let

$$\hat{B}_{k+1,k} = \tilde{Q}^T\tilde{U}_{m+1,k}\tilde{\Sigma}_k, \tag{2.39}$$

$$\hat{\alpha}_{k+1} = \alpha_{m+1}\tilde{q}_{m+1,k+1}. \tag{2.40}$$

Then from (2.30) and (2.37)–(2.40), we obtain

$$\begin{aligned} A^T\hat{Q}_{k+1} &= \hat{P}_k\hat{B}^T_{k+1,k} + \hat{\alpha}_{k+1}\hat{p}_{k+1}e^T_{k+1} \\ A\hat{P}_k &= \hat{Q}_{k+1}\hat{B}_{k+1,k}, \end{aligned} \tag{2.41}$$

where $\hat{p}_{k+1} = p_{m+1}$ and $\hat{B}_{k+1,k} \in \mathbb{R}^{(k+1)\times k}$ is lower triangular. This is the desired analogue of (2.16).

Starting with (2.41), computations with the GK bidiagonalization can be continued using Algorithm 2.1 with $\hat{q}_{k+1}$, the $(k+1)st$ column of $\hat{Q}_{k+1}$. Application of $m-k$ steps of GK bidiagonalization yields the new decompositions

$$\begin{aligned} A^T\begin{bmatrix}\hat{Q}_{k+1} & \hat{Q}_{m-k}\end{bmatrix} &= \begin{bmatrix}\hat{P}_k & \hat{P}_{m-k}\end{bmatrix}\hat{B}^T_{m+1,m} + \hat{\alpha}_{m+1}\hat{p}_{m+1}e^T_{m+1}, \\ A\begin{bmatrix}\hat{P}_k & \hat{P}_{m-k}\end{bmatrix} &= \begin{bmatrix}\hat{Q}_{k+1} & \hat{Q}_{m-k}\end{bmatrix}\hat{B}_{m+1,m}, \end{aligned} \tag{2.42}$$

where the first column of $\hat{P}_{m-k}$ is $\hat{p}_{k+1}$,

$$\hat{B}_{m+1,m} = \begin{bmatrix} \hat{B}_{k+1,k} & \hat{\alpha}_{k+1} & & 0 \\ & \hat{\beta}_{k+2} & \ddots & \\ & & \ddots & \hat{\alpha}_m \\ 0 & & & \hat{\beta}_{m+1} \end{bmatrix} \in \mathbb{R}^{(m+1)\times m}, \tag{2.43}$$

and the matrices $\begin{bmatrix}\hat{Q}_{k+1} & \hat{Q}_{m-k}\end{bmatrix} \in \mathbb{R}^{\ell\times(m+1)}$ and $\begin{bmatrix}\hat{P}_k & \hat{P}_{m-k}\end{bmatrix} \in \mathbb{R}^{n\times m}$ have orthonormal columns. We now proceed by computing the SVD of $\hat{B}_{m+1,m}$, harmonic Ritz vectors of AA^T, cf. (2.22), and then new decompositions analogous to (2.41) and (2.42). The k smallest singular triplets

$$\{\tilde{\sigma}_j, \hat{q}_j, \hat{p}_j\}_{j=1}^k, \tag{2.44}$$

where $\hat{q}_j$, $j = 1, \ldots, k$, are the first k columns of $\hat{Q}_{k+1}$ and the $\hat{p}_j$, $j = 1, \ldots, k$, are the first k columns of $\hat{P}_k$, furnish approximations of the k smallest singular triplets $\{\sigma_j, u_j, v_j\}_{j=1}^k$ of A.

A singular triplet $\{\tilde{\sigma}_j, \hat{q}_j, \hat{p}_j\}$ defined by (2.44) is accepted as an approximate singular triplet of A if

$$\begin{aligned}\sqrt{\|A\hat{p}_j - \tilde{\sigma}_j\hat{q}_j\|^2 + \|A^T\hat{q}_j - \tilde{\sigma}_j\hat{p}_i\|^2} \\ &= \sqrt{\tilde{\sigma}_j^2\|\tilde{u}_j - \tilde{q}_j\|^2 + \|B_{m+1,m}^T\tilde{q}_j - \tilde{\sigma}_j\tilde{v}_j\|^2 + |\alpha_{m+1}e_{m+1}^T\tilde{q}_j|^2} \\ &\leq \delta^{\text{harm}}\|A\|,\end{aligned} \tag{2.45}$$

where $\tilde{q}_j$ is the jth column of $\tilde{Q}$ from (2.25), $\tilde{u}_j$ and $\tilde{v}_j$ are the jth columns of $\tilde{U}_{m+1,m}$ and $\tilde{V}_m$ respectively in the SVD (2.21) of $\hat{B}_{m,m+1}$, and $\delta^{\text{harm}} > 0$ is a user-specified tolerance. In (2.45) $\|A\|$ can be approximated by $\tilde{\sigma}_m$, the largest singular value of $\hat{B}_{m+1,m}$. Typically, several matrices $\hat{B}_{m+1,m}$ are generated during the iterations and therefore an acceptable approximation of $\|A\|$ can be obtained from the largest singular value of all the matrices $\hat{B}_{m+1,m}$ generated.

We remark that accurate computation of the vector $B_m^{-T}e_m$, used in (2.25), might be difficult when B_m has a large condition number. This computation can be avoided by noticing that the vector

$$\begin{bmatrix} -\beta_{m+1}B_m^{-T}e_m \\ 1 \end{bmatrix} \tag{2.46}$$

is in the null space of $\begin{bmatrix} I_m & \beta_{m+1}B_m^{-T}e_m \end{bmatrix} \in \mathbb{R}^{m\times(m+1)}$, and

$$B_{m+1,m}^T = B_m^T \begin{bmatrix} I_m & \beta_{m+1}B_m^{-T}e_m \end{bmatrix}.$$

Therefore, the vector (2.46) is a multiple of the null space vector $\tilde{u}_{m+1}$ of $B_{m+1,m}^T$, cf. (2.21). We have

$$\begin{bmatrix} -\beta_{m+1}B_m^{-T}e_m \\ 1 \end{bmatrix} = (1/\tilde{u}_{m+1,m+1})\,\tilde{u}_{m+1}, \tag{2.47}$$

where $\tilde{u}_{m+1,m+1}$ is the last element of the vector $\tilde{u}_{m+1}$. It follows that any multiple of the matrix

$$\left[\begin{array}{c|c} [\tilde{u}_{m+1,m+1}I_m \quad -\tilde{u}_{m+1,1:m}]\,\tilde{U}_{m+1,k} & \tilde{u}_{m+1} \\ \hline 0 & \end{array}\right] \tag{2.48}$$

can be used in place of the left-hand side of (2.25). Here $\tilde{u}_{m+1,1:m}$ denotes the vector consisting of the first m elements of $\tilde{u}_{m+1}$.

The restarted GK bidiagonalization method described above will be combined with the restarted LSQR method reviewed in the following section.

2.4 A restarted LSQR method

We describe a restarted LSQR method for solving the LS problem (2.1). The method will be used in conjunction with the restarted GK bidiagonalization method for computing harmonic Ritz vectors presented in the previous section. The description of our restarted LSQR method parallels as much as possible that of the standard LSQR method [6].

Application of k steps of Algorithm 2.1 with starting vector $q_1 \in \mathbb{R}^\ell$ yields the decompositions

$$\begin{aligned} A^T Q_{k+1} &= P_k B_{k+1,k}^T + \alpha_{k+1} p_{k+1} e_{k+1}^T \\ A P_k &= Q_{k+1} B_{k+1,k}. \end{aligned} \tag{2.49}$$

Let $r_k = b - Ax_k$ for some vector $x_k \in \mathbb{R}^n$ such that $r_k = Q_{k+1} f_{k+1}$ for some $f_{k+1} \in \mathbb{R}^{k+1}$; if $k = 0$, then we let $r_0 = q_1 f_1$ where $f_1 = \|r_0\|$.

Extend the k-step decompositions (2.49) by carrying out $m - k$ additional GK bidiagonalization steps to obtain the m-step decompositions (2.16). Let $x_m = x_k + P_m y_m$ and notice that

$$\begin{aligned} r_m = b - Ax_m &= b - A\left(x_k + P_m y_m\right) \\ &= r_k - AP_m y_m \\ &= r_k - Q_{m+1} B_{m+1,m} y_m \\ &= Q_{m+1}\left(\begin{bmatrix} f_{k+1} \\ 0 \end{bmatrix} - B_{m+1,m} y_m\right). \end{aligned}$$

It follows that

$$\min_{x_m \in x_k + \mathcal{K}_m(A^TA, p_1)} \|b - Ax_m\| = \min_{y \in \mathbb{R}^m} \left\| \begin{bmatrix} f_{k+1} \\ 0 \end{bmatrix} - B_{m+1,m} y \right\|. \tag{2.50}$$

We solve (2.50) with the aid of the QR decomposition

$$B_{m+1,m} = \tilde{Q}^{(B)}_{m+1} \tilde{R}^{(B)}_{m+1,m}, \tag{2.51}$$

where $\tilde{Q}^{(B)}_{m+1} \in \mathbb{R}^{(m+1)\times(m+1)}$ is orthogonal and $\tilde{R}^{(B)}_{m+1,m} \in \mathbb{R}^{(m+1)\times m}$ is upper triangular. Substituting (2.51) into (2.50) yields the equivalent minimization problem

$$\min_{y\in\mathbb{R}^m} \left\| \left(\tilde{Q}^{(B)}_{m+1}\right)^T \begin{bmatrix} f_{k+1} \\ 0 \end{bmatrix} - \tilde{R}^{(B)}_{m+1,m} y \right\|. \tag{2.52}$$

Since the last row of $\tilde{R}^{(B)}_{m+1,m}$ vanishes, the LS solution y_m of (2.52) satisfies the first m rows exactly. The residual norm for (2.52) is given by

$$\bar{\phi}_{m+1} = e^T_{m+1} \left(\tilde{Q}^{(B)}_{m+1}\right)^T \begin{bmatrix} f_{k+1} \\ 0 \end{bmatrix}.$$

This yields the residual vector for the LSQR method

$$\begin{aligned} r^{\mathrm{lsqr}}_m &= b - Ax_m \\ &= Q_{m+1}\left(\begin{bmatrix} f_{k+1} \\ 0 \end{bmatrix} - B_{m+1,m} y_m\right) \\ &= Q_{m+1}\tilde{Q}^{(B)}_{m+1}\left(\left(\tilde{Q}^{(B)}_{m+1}\right)^T \begin{bmatrix} f_{k+1} \\ 0 \end{bmatrix} - \tilde{R}^{(B)}_{m+1,m} y_m\right) \\ &= Q_{m+1}\bar{\phi}_{m+1}\tilde{Q}^{(B)}_{m+1} e_{m+1}. \end{aligned} \tag{2.53}$$

The process can be restarted with the vectors $x_k = x_m$ and $r_k = r^{\mathrm{lsqr}}_m$, where we again assume that r_k is a linear combination of the columns of the matrix Q_{k+1} in (2.49). Section 2.5 shows how this condition can be guaranteed.

There are several ways to compute the QR decomposition in (2.51). In the context of the restarted GK bidiagonalization method of Section 2.3, the first $k+1$ rows and k columns of $\hat{B}_{m+1,m}$ in (2.43) is the matrix $\hat{B}_{k+1,k}$ in (2.39), which is lower triangular and typically not lower bidiagonal. We compute a QR decomposition of $\hat{B}_{k+1,k}$ by an arbitrary method and then switch to using Givens rotations when carrying out $m-k$ GK bidiagonalization steps to produce the bottom part of the matrix $\hat{B}_{m+1,m}$. This approach allows our algorithm to incorporate all of the formulas, e.g., for computing residual norms, of the standard LSQR algorithm [6] from step $k+1$ and onwards.

The following algorithm describes our restarted LSQR method, where we assume that the starting residual vector r_k is in $\mathcal{R}(Q_{k+1})$. The algorithm uses the elegant formulas of the LSQR method by Paige and Saunders [6] whenever possible to reduce the computational cost and storage requirements. We comment further on the algorithm below.

Algorithm 2.2. A RESTARTED LSQR METHOD

Input: $A \in \mathbb{R}^{\ell \times n}$ *or functions for evaluating products with* A *and* A^T,

k-step GK bidiagonalization decomposition (2.49),

$x_k \in \mathbb{R}^n$: *initial approximate solution of (2.1),*

$f_{k+1} \in \mathbb{R}^{k+1}$: *where* $r_k = b - Ax_k = Q_{k+1}f_{k+1}$, Q_{k+1} *is given in (2.49),*

$m \geq k+2$: *maximum number of iterations,*

m_{reorth} : *maximum number of vectors for reorthogonalization in steps 25 and 28,*

δ^{lsqr} : *tolerance for accepting an approximate solution to (2.1).*

Output: *Approximate solution* x_m *to (2.1),*

(optional) $\bar{\phi}_{m+1}$, c_m, *and m-step GK bidiagonalization (2.16).*

```
1. If $k = 0$
    2. Compute $q_1 := r_0/f_1$;    $Q_1 := q_1$
    3. Compute $p_1 := A^T q_1$;    $\alpha_1 := \|p_1\|$;    $p_1 := p_1/\alpha_1$;    $P_1 := p_1$
    4. Set $B_{1,0} := [\ ]$
5. End
6. Compute $q_{k+2} := Ap_{k+1} - q_{k+1}\alpha_{k+1}$
7. Reorthogonalize: $q_{k+2} := q_{k+2} - Q_{(1:k+1)}\left(Q_{(1:k+1)}^T q_{k+2}\right)$
8. Compute $\beta_{k+2} := \|q_{k+2}\|$;   $q_{k+2} := q_{k+2}/\beta_{k+2}$;   $Q_{k+2} := [Q_{k+1}, q_{k+2}]$
9. Compute $p_{k+2} := A^T q_{k+2} - p_{k+1}\beta_{k+2}$
10. Reorthogonalize: $p_{k+2} := p_{k+2} - P_{(1:k+1)}\left(P_{(1:k+1)}^T p_{k+2}\right)$
11. Compute $\alpha_{k+2} := \|p_{k+2}\|$;   $p_{k+2} := p_{k+2}/\alpha_{k+2}$;   $P_{k+2} := [P_{k+1}, p_{k+2}]$
12. Compute QR decomposition $B_{k+2,k+1} = \widetilde{Q}\widetilde{R}$ of
        $$B_{k+2,k+1} := \begin{bmatrix} B_{k+1,k} & \alpha_{k+1} \\ 0 & \beta_{k+2} \end{bmatrix} \in \mathbb{R}^{(k+2)\times(k+1)},$$
    where $\widetilde{Q} \in \mathbb{R}^{(k+2)\times(k+2)}$ and $\widetilde{R} \in \mathbb{R}^{(k+2)\times(k+1)}$
13. Compute $\tilde{f}_{k+2} := \widetilde{Q}^T \begin{bmatrix} f_{k+1} \\ 0 \end{bmatrix}$
14. Compute $\bar{\rho}_{k+2} := \alpha_{k+2}\left(e_{k+2}^T \widetilde{Q} e_{k+2}\right)$
15. Compute $\bar{\phi}_{k+2} := e_{k+2}^T \tilde{f}_{k+2}$
16. Solve $\widetilde{R}_{k+1,k+1} y = \tilde{f}_{1:k+1}$, where $\widetilde{R}_{k+1,k+1} \in \mathbb{R}^{(k+1)\times(k+1)}$ is the
    leading submatrix of $\widetilde{R}$
17. Update solution vector $x_{k+1} := x_k + P_{(1:k+1)} y$
18. Compute $\|r_{k+1}\| := |\bar{\phi}_{k+2}|$
19. Compute $\|A^T r_{k+1}\| := \alpha_{k+2}\beta_{k+2}|e_{k+1}^T y|$
20. Check convergence: if (2.54) is satisfied, then exit.
21. Compute $\theta_{k+2} := e_{k+1}^T \widetilde{Q}^T \begin{bmatrix} B_{k+2,k+1} & \begin{matrix} 0 \\ \alpha_{k+2} \end{matrix} \end{bmatrix} e_{k+2}$
22. Compute $w := p_{k+2} - P_{(1:k+1)} y\,(\theta_{k+2}/f_{k+1,k+1})$
23. For $j = k+2 : m$
    24. Compute $q_{j+1} := Ap_j - q_j\alpha_j$
    25. Reorthogonalize:
        Compute $i := \max\{1, j - m_{\text{reorth}} + 1\}$
        Compute $q_{j+1} := q_{j+1} - Q_{(i:j)} \left( Q_{(i:j)}^T q_{j+1} \right)$
    26. Compute $\beta_{j+1} := \|q_{j+1}\|$;   $q_{j+1} := q_{j+1}/\beta_{j+1}$;   $Q_{j+1} := [Q_j, q_{j+1}]$;
    27. Compute $p_{j+1} := A^T q_{j+1} - p_j \beta_{j+1}$
    28. Reorthogonalize:
        Compute $i := \max\{1, j - m_{\text{reorth}} + 1\}$
        Compute $p_{j+1} := p_{j+1} - P_{(i:j)} \left( P_{(i:j)}^T p_{j+1} \right)$
    29. Compute $\alpha_{j+1} := \|p_{j+1}\|$;   $p_{j+1} := p_{j+1}/\alpha_{j+1}$
    30. if $j < m$
        31. $P_{j+1} := [P_j, p_{j+1}]$
    32. End
    33. Compute $\rho_j := \sqrt{\beta_{j+1}^2 + \bar{\rho}_j^2}$; $c_j := \bar{\rho}_j/\rho_j$; $s_j := \beta_{j+1}/\rho_j$
    34. Compute $\theta_j := s_j \alpha_{j+1}$
    35. Compute $\bar{\rho}_{j+1} := -c_j \alpha_{j+1}$
    36. Compute $\phi_j := c_j \bar{\phi}_j$;   $\bar{\phi}_{j+1} := s_j \bar{\phi}_j$
    37. Compute $x_j := x_{j-1} + (\phi_j/\rho_j)\, w$;   $w := p_{j+1} - (\theta_{j+1}/\rho_j) w$
    38. Compute $\|r_j\| := |\bar{\phi}_{j+1}|$
    39. Compute $\|A^T r_j\| := |\bar{\phi}_{j+1} \bar{\rho}_{j+1}|$
    40. Check convergence: if (2.54) is satisfied, then exit.
41. End
```

When $k = 0$ on input to Algorithm 2.2 and no reorthogonalization and accumulation of the matrices $B_{m+1,m}$, P_m, and Q_{m+1} is carried out, m steps of the algorithm are equivalent to m steps of the LSQR method of Paige and Saunders [6]. In particular, Algorithm 2.2 can be used as a restarted or nonrestarted LSQR method.

The stopping criteria outlined in [3, 6] can be used in the convergence tests (lines 20 and 40) of Algorithm 2.2. This is recommend for public domain implementations of the algorithm. For ease of comparison with other methods, we terminate the computations in the examples reported in Section 2.7 when in lines 20 or 40 the inequality

$$\|A^T r_j\| \leq \delta^{\text{lsqr}} \|A^T r_0\| \tag{2.54}$$

holds, where $\delta^{\text{lsqr}} > 0$ is a user-specified tolerance.

The formula for $\|r_{k+1}\|$ in line 18 follows from (2.53), and the expression for $\|A^T r_{k+1}\|$ in line 19 is taken from Jia [27]. The formulas for $\|r_j\|$ and $\|A^T r_j\|$ in lines 35 and 36, respectively, are obtained from [6]. If $\alpha_{j+1} = 0$ or $\beta_{j+1} = 0$ for some j, then $\|A^T r_j\| = 0$; see [28] and more recently [27, Theorem 2].

We reorthogonalize in lines 25 and 28 of Algorithm 2.2 to avoid loss of orthogonality due to finite precision arithmetic. Reorthogonalization requires the accumulation of the matrices $Q_{(i:j)}$ in line 25 and $P_{(i:j)}$ in line 28. Both these matrices have a fixed maximum number of columns, denoted by m_{reorth}. Several reorthogonalization strategies are discussed in [15, 23, 29]. When $\ell \gg n$, reorthogonalization of the columns of $P_{(i:j)}$ only, reduces the computational effort required to compute the decompositions (2.16) considerably, compared with reorthogonalization of the columns of both the matrices $P_{(i:j)}$ and $Q_{(i:j)}$. We refer to reorthogonalization of the columns of $P_{(i:j)}$ only as one-sided reorthogonalization. Algorithm 2.2 can easily be modified to implement one-sided reorthogonalization; see [15, 29] for discussions on this reorthogonalization approach.

We are interested in combining Algorithm 2.2 with the augmented harmonic GK bidiagonalization method of Section 2.3. In this context, we assume that $m \ll \min\{\ell, n\}$ and apply one-sided reorthogonalization as described in [15] and applied in the MATLAB code `irlba` accompanying [18]. When, instead, Algorithm 2.2 is used as a nonrestarted LSQR algorithm, either no reorthogonalization is carried out or only the last generated m_{reorth} columns of $P_{(i:j)}$ are reorthogonalized. The latter reorthogonalization approach also is implemented by Fong and Saunders [3] in their MATLAB code `lsmr`. Reorthogonalization in lines 7 and 10 of Algorithm 2.2 is always carried out when $k > 0$. Moreover, when $k > 0$ we use a k-step GK bidiagonalization (2.49) as input. To be able to apply the formulas of the LSQR algorithm [6], we carry out the $(k+1)$st step of the GK bidiagonalization separately, i.e., we perform the computations of lines 6–11 of Algorithm 2.2, and subsequently determine the quantities $\bar{\rho}_{k+2}$ in line 14, $\bar{\phi}_{k+2}$ in line 15, θ_{k+2} in line 21, and w in line 22 by formulas analogous to [6, equations (4.6)–(4.12)].

Line 12 of Algorithm 2.2 computes the QR decomposition of the matrix $B_{k+2,k+1}$. This can be done with MATLAB's internal `qr` function. The input restriction $m \geq k+2$ ensures that the For-loop (lines 23–38) is executed at least once. Typically, k is quite small; in the computed examples of Section 2.7, we let $k \leq 20$.

2.5 An augmented LSQR algorithm

In order to be able to conveniently combine the restarted LSQR method of Section 2.4 with the restarted augmented GK bidiagonalization method of Section 2.3, the residual vector from restarted LSQR, r_m^{lsqr} in (2.53), should be in the range of the matrix $\hat{Q}_{k+1}$ defined in (2.26). We now show that the residual vector r_m^{harm} of the harmonic Ritz vectors, defined by (2.23), and r_m^{lsqr} are parallel. It then follows from (2.23)–(2.26) that $r_m^{\text{lsqr}} \in \mathcal{R}(\hat{Q}_{k+1})$.

Theorem 2.3. *The residual vector of the harmonic Ritz vectors r_m^{harm}, defined by (2.23), and the residual vector of the restarted LSQR method r_m^{lsqr}, given by (2.53), are parallel provided that the lower bidiagonal matrix $B_{m+1,m}$ (2.17) from GK bidiagonalization (2.16) is unreduced. Moreover, r_m^{harm} and r_m^{lsqr} are multiples* *of* $Q_{m+1}\tilde{u}_{m+1}$*, where* $\tilde{u}_{m+1} \in \mathcal{N}(B_{m+1,m}^T)$*, cf. (2.21).*

Proof. Consider the $(m+1)$-vector

$$\tilde{Q}_{m+1}^{(B)} \bar{\phi}_{m+1} e_{m+1} \tag{2.55}$$

of r_m^{lsqr} and note that this vector is in $\mathcal{N}(B_{m+1,m}^T)$, i.e.,

$$\begin{aligned} B_{m+1,m}^T \tilde{Q}_{m+1}^{(B)} \bar{\phi}_{m+1} e_{m+1} &= \bar{\phi}_{m+1} \left(e_{m+1}^T \left(\tilde{Q}_{m+1}^{(B)} \right)^T B_{m+1,m} \right)^T \\ &= \bar{\phi}_{m+1} \left(e_{m+1}^T \tilde{R}_{m+1,m}^{(B)} \right)^T \\ &= 0. \end{aligned} \tag{2.56}$$

It is easy to see that the $(m+1)$-vector

$$\begin{bmatrix} -\beta_{m+1} B_m^{-T} e_m \\ 1 \end{bmatrix} \tag{2.57}$$

in the definition (2.23) of r_m^{harm} lies in $\mathcal{N}\left(B_{m+1,m}^T\right)$:

$$\begin{bmatrix} B_m^T & \beta_{m+1} e_m \end{bmatrix} \begin{bmatrix} -\beta_{m+1} B_m^{-T} e_m \\ 1 \end{bmatrix} = 0. \tag{2.58}$$

The matrix $B_{m+1,m}$ is unreduced by assumption. Therefore, it has rank m and so does its transpose $B_{m+1,m}^T$. Equations (2.56) and (2.58) show that the vectors

$$\tilde{Q}_{m+1}^{(B)} \bar{\phi}_{m+1} e_{m+1} \quad \text{and} \quad \begin{bmatrix} -\beta_{m+1} B_m^{-T} e_m \\ 1 \end{bmatrix}$$

are in $\mathcal{N}\left(B_{m+1,m}^T\right)$. It follows that they are multiples of each other and of the vector $\tilde{u}_{m+1}$ defined in (2.21). □

We can easily determine the scalar multiplier between r_m^{harm} (2.23) and r_m^{lsqr} (2.53) by examining the For-loop (lines 23–38) in Algorithm 2.2. LSQR eliminates the subdiagonal element of the lower bidiagonal matrix via Givens rotations, but does not explicitly form the orthogonal matrix made up by the products of these rotations. If this matrix were generated, then in the last iteration (lines 23–41) of Algorithm 2.2, we would obtain

$$\tilde{Q}_{m+1}^{(B)} := \begin{bmatrix} I_{m-1} & 0 \\ 0 & \begin{bmatrix} c_m & s_m \\ s_m & -c_m \end{bmatrix} \end{bmatrix} \begin{bmatrix} \tilde{Q}_m^{(B)} & 0 \\ 0 & 1 \end{bmatrix}, \tag{2.59}$$

where $\tilde{Q}_m^{(B)} \in \mathbb{R}^{m\times m}$ is the orthogonal matrix from the QR factorization of $B_{m,m-1}$. It follows from (2.59) that the last element of the vector (2.55) is $-c_m\bar{\phi}_{m+1}$. Moreover, the last element of the vector (2.57) is one. Therefore,

$$r_m^{\text{lsqr}} = -c_m\bar{\phi}_{m+1} r_m^{\text{harm}}.$$

Using (2.47), we also have that

$$\begin{aligned} \tilde{Q}_{m+1}^{(B)}\bar{\phi}_{m+1}e_{m+1} &= -c_m\bar{\phi}_{m+1}\begin{bmatrix} -\beta_{m+1}B_m^{-T}e_m \\ 1 \end{bmatrix} \\ &= \left(-c_m\bar{\phi}_{m+1}/\tilde{u}_{m+1,m+1}\right)\tilde{u}_{m+1}. \end{aligned}$$

If $\tilde{Q}$ is the matrix with orthonormal columns in the QR decomposition of (2.48), then

$$r_m^{\text{lsqr}} = \hat{Q}_{k+1}f_{k+1},$$

where $f_{k+1} = \left(-c_m\bar{\phi}_{m+1}/\tilde{u}_{m+1,m+1}\right)\tilde{Q}^T\tilde{u}_{m+1}$.

We are now in a position to describe our augmented LSQR algorithm that combines the methods of Sections 2.3 and 2.4. We assume that augmentation is carried out with vectors that approximate the singular vectors associated with the smallest singular values.

Algorithm 2.3. An augmented LSQR method

Input: $A \in \mathbb{R}^{\ell\times n}$ *or functions for evaluating products with* A *and* A^T*,*

$x_0 \in \mathbb{R}^n$: *initial approximate solution of (2.1),*

```
    $r_0 := b - Ax_0 \in \mathbb{R}^\ell$ : initial residual vector,
    $k$ : number of augmenting vectors,
    $m \geq k+2$ : maximum length GK bidiagonalization,
    $max_{\rm aug}$: maximum number of iteration for augmenting stage,
    $max_{\rm lsqr}$: maximum number of iteration for the non-restarted LSQR method,
    $\delta^{\rm lsqr}$ : tolerance for accepting an approximate solution to (2.1),
    $\delta^{\rm harm}$ : tolerance for accepting approximate singular triplet, cf. (2.45),

Output: Approximate solution $x$ to (2.1).

1. Call Algorithm 2.2
     Input: $A$, $k := 0$, $x_0$, $f_1 := \|r_0\|$, $q_1 := r_0/f_1$, $m_{\rm reorth} := m$, $m$ and $\delta^{\rm lsqr}$
     Output: $x_m$, $\bar{\phi}_{m+1}$, $c_m$, and an $m$-step GK bidiagonalization (2.16)
2. For $i = 1 : max_{\rm aug}$
    3. Compute the singular value decomposition (2.21) of $B_{m+1,m}$
    4. Compute the augmenting vectors:
         Compute the QR factorization of (2.48).
         Determine the matrices $\hat{Q}_{k+1}$, $\hat{P}_k$, and $\hat{B}_{k+1,k}$ by (2.26), (2.31) and (2.39),
         respectively and $\hat{\alpha}_{k+1}$ by (2.40) to get (2.41).
    5. Check convergence: if all $k$ singular triplets satisfy (2.45), then goto 9.
    6. Call Algorithm 2.2
         Input: $A$, $x_k := x_m$, $f_{k+1} := \left(-c_m\bar{\phi}_{m+1}/\tilde{u}_{m+1,m+1}\right)\tilde{Q}^T\tilde{u}_{m+1}$, $m_{\rm reorth} := m$,
                $m$, $\delta^{\rm lsqr}$, and a $k$-step GK bidiagonalization (2.41)
         Output: $x_m$, $\bar{\phi}_{m+1}$, $c_m$, and an $m$-step GK bidiagonalization (2.42)
    7. Set
            $B_{m+1} := \hat{B}_{m+1,m}$
            $Q_{m+1} := \begin{bmatrix} \hat{Q}_{k+1} & \hat{Q}_{m-k} \end{bmatrix}$
            $P_m := \begin{bmatrix} \hat{P}_k & \hat{P}_{m-k} \end{bmatrix}$
            $p_{m+1} := \hat{p}_{m+1}$
            $\alpha_{m+1} := \hat{\alpha}_{m+1}$
8. End
9. Call Algorithm 2.2
    Input: $A$, $x_k := x_m$, $f_{k+1} := \left(-c_m \bar{\phi}_{m+1}/\tilde{u}_{m+1,m+1}\right) \tilde{Q}^T \tilde{u}_{m+1}$, $m_{\text{reorth}} := m$,
        $m := max_{\text{lsqr}}$, $\delta^{\text{lsqr}}$ and a k-step GK bidiagonalization (2.41)
    Output: $x_m$
```

The above algorithm describes a simplification of the actual computations carried out. For instance, the number of augmenting vectors used at each restart is typically chosen to be larger than the number of desired singular triplets. This often yields faster convergence without increasing the memory requirement; see [15, 18] for a discussion. The number of vectors to be reorthogonalized, m_{reorth}, is set to the maximum number of columns of the computed GK bidiagonalization. This is to ensure that accurate approximations of the singular vectors are computed.

In the nonrestarted LSQR stage of Algorithm 2.3, i.e., in line 9, the reorthogonalization applied is that of the nonrestarted LSQR method described by Algorithm 2.2. We set $m_{\text{reorth}} = m$. Letting $0 \leq m_{\text{reorth}} < m$ instead would reduce the computational work for each iteration, but could require more iterations to satisfy the convergence criterion and, therefore, may require more computational effort in total. The choice $m_{\text{reorth}} > m$ increases the storage requirement and therefore is avoided.

2.6 Rank-deficient LS problems

An LS problem (2.1) is said to be rank-deficient if A has linearly dependent columns. We are interested in determining the unique solution, x^+, of minimal Euclidean norm. This solution is orthogonal to $\mathcal{N}(A)$ and therefore lies in $\mathcal{R}(A^T)$; see, e.g., [1] for a discussion on rank-deficient LS problems.

The standard LSQR algorithm [6] produces a sequence of iterates that lie in $\mathcal{R}(A^T)$ provided the initial iterate x_0 does. To ensure the latter one may choose $x_0 = 0$. Note that the iterates determined in lines 17 and 34 of Algorithm 2.2 are in $\mathcal{R}(A^T)$ if the initial approximation x_k of x^+ used in Algorithm 2.2 is in $\mathcal{R}(A^T)$. In order to show that the approximate solutions determined by Algorithm 2.3 are in $\mathcal{R}(A^T)$ when this holds for the first iterate x_0, it remains to establish that the harmonic Ritz vectors used to augment the Krylov subspace in Algorithm 2.3 also lie in $\mathcal{R}(A^T)$. Observe that the restarted augmented harmonic method of Section 2.3 does not determine approximations of eigenvectors associated with the eigenvalue zero. The reason for this is that the harmonic Ritz values are the square of the nonvanishing singular values of $B_{m+1,m}$ (2.17). The singular values are nonvanishing, since by assumption all α_j and β_j are nonzero. The situation when some α_j or β_j vanish is discussed in Section 2.4. The iterations with the augmented Krylov subspaces of Algorithm 2.3 determine approximate solutions x_m of (2.1) in subspaces of the form

$$\mathcal{K}_m(A^TA, \hat{p}_1, \ldots, \hat{p}_k, \hat{p}_{k+1}) = \text{span}\left\{\hat{p}_1, \ldots, \hat{p}_k, \hat{p}_{k+1}, A^TA\hat{p}_{k+1}, \ldots, (A^TA)^{m-k-1}\hat{p}_{k+1}\right\}$$

where $\hat{p}_1, \ldots, \hat{p}_k$ are approximate right singular vectors of A associated with non-vanishing singular values, and $\hat{p}_{k+1} = p_{m+1}$ is the residual vector of the GK bidi-agonalization (2.16); see also Algorithm 2.1. Using (2.18) and (2.28), we have for

$j \leq k$,

$$\begin{aligned}\hat{p}_j &= \frac{1}{\tilde{\sigma}_j^2}\left(A^TA\hat{p}_j - \left(\beta_{m+1}e_m^T\tilde{v}_k\right)\alpha_{m+1}p_{m+1}\right)\\ &= \frac{1}{\tilde{\sigma}_j^2}A^T\left(A\hat{p}_j - \left(\beta_{m+1}e_m^T\tilde{v}_k\right)r_m^{\text{harm}}\right).\end{aligned}$$

It follows that $\mathcal{K}_m\left(A^TA, \hat{p}_1, \ldots, \hat{p}_k, \hat{p}_{k+1}\right) \subset \mathcal{R}(A^T)$. Example 2.7 in Section 2.7 illustrates the performance of Algorithm 2.3 when applied to a rank-deficient LS problem.

2.7 Numerical examples

We describe a few numerical experiments that illustrate the performance of Algorithm 2.3 as implemented by the MATLAB code `alsqr`[4]. This code uses the following user-specified parameters:

[4]Code is available in Appendix B.

adjust	Additional vectors used together with k augmenting vectors to speed up convergence; see [15] for comments on the inclusion of additional vectors.
k	Number of augmenting vectors.
maxitp	Maximum number of iterations in the augmenting stage.
maxitl	Maximum number of iterations with the nonrestarted LSQR method when the augmented vectors are kept fixed.
m	Maximum number of GK vectors.
*reorth*012	String deciding whether no, one, or two-sided reorthogonalization is used in either stage.
mreorth	Number of vectors to be reorthogonalized during the nonrestarted LSQR stage, when the augmented vectors are kept fixed. If $mreorth > 0$, then one-sided reorthogonalization is applied to the "short" vectors.
tollsqr	Tolerance δ^{lsqr} in (2.54) for accepting a computed approximate solution as the solution of (2.1).
tolharm	Tolerance δ^{harm} in (2.45) for accepting an approximate singular triplet as a singular triplet of A and use it for augmentation.

We compare `alsqr` to the MATLAB code `lsqr`[5] for the standard LSQR method by Paige and Saunders [6] and to the MATLAB code `lsmr`[6] by Fong and

[5]The `lsqr` MATLAB code is not the code that comes with MATLAB. The used code was adapted to output the norm of the residual error in each iteration and to carry out reorthogonalization as described in Section 2.4.

[6]http://www.stanford.edu/group/SOL/software/lsmr.html. The code was adapted to output the norm of the residual error in each iteration.

Saunders [3]. We remark that the performance of the methods in our comparisons depends on the machine architecture, coding style, and stopping criteria. These may significantly affect the performance, regardless of the theoretical properties of the methods. We therefore do not report CPU times, but instead measure performance in terms of the required number of matrix-vector product evaluations with the matrices A and A^T. We set all common parameters for different methods to the same values for each example, and reorthogonalize only against the last m vectors in each method. We use the initial approximate solution $x_0 = 0$ for all methods and examples.

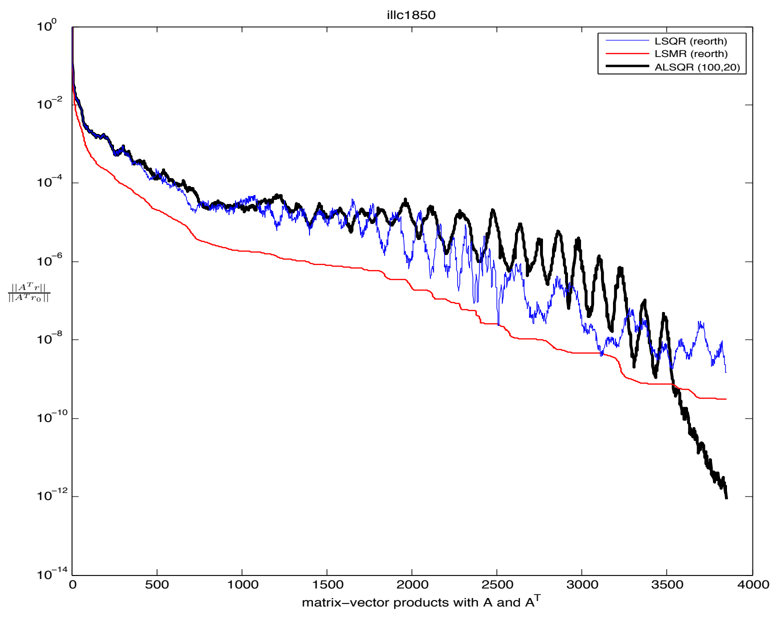
There are many preconditioned iterative methods available for the solution of (2.1). It is difficult to make a fair comparison, because the construction of many preconditioners is determined by several parameters, including drop tolerance and available storage. Here we only note that our method is unique in that an approximate solution to the LS problem is computed already during the construction of the augmented Krylov subspaces.

We present six numerical examples with matrices from the Matrix Market collection [16, 17]. The matrices A, their properties, as well as the definition of the vector b, are described in Table 2.1. All matrices are of full column rank except for the matrix of Example 2.7. In Table 2.1 "ℓ" denotes the number of rows, "n" the number of columns, and "nnz" the number of nonzero entries of the matrices. The column labeled "Cond. #" shows the condition number estimate computed by the MATLAB function `condest` when A is square. For the rectangular matrix ILLC1850, we determined the condition number with the MATLAB function `cond`. The vectors b were also chosen from the Matrix Market collection when available, otherwise we computed the vector b with the MATLAB function `b=rand(size(A,1),1)`. This yields a vector b with uniformly distributed entries in the interval $(0, 1)$. All computations were carried out using MATLAB version 7.12.0.0635 R2011a [30] on a Dell XPS workstation with an Intel Core2 Quad processor and 4 GB of memory running under the Windows Vista operating system. Machine precision is $2.2 \cdot 10^{-16}$. One-sided reorthogonalization is used in both stages for all examples except for Example 2.4 where two-sided reorthogonalization is used in the augmenting stage and one-sided reorthogonalization is used in the LSQR stage. The matrix A in Example 2.4 is very ill-conditioned, see Table 2.1; hence two-sided reorthogonalization is required during the iteration process to approximate singular vectors. See [15, 29] for remarks on requiring two-sided reorthogonalization during the GK process for singular triplet approximation.

Table 2.1. Matrix Market collection of matrices A, properties, and vectors b used in the numerical examples. The rank-deficient matrix ILLC1850* was obtained from ILLC1850 by replacing the second column by twice the first column.

Example	Matrix	ℓ	n	nnz	Cond. #	b
Example 2.2	ILLC1850	1850	712	8758	$1.4 \cdot 10^3$	ILLC1850_RHS1
Example 2.3	E05R0000	236	236	5856	$5.9 \cdot 10^4$	E05R0000_RHS1
Example 2.4	E20R0100	4241	4241	131566	$2.2 \cdot 10^{10}$	E20R0100_RHS1
Example 2.5	NOS5	468	468	2820	$2.9 \cdot 10^4$	`rand(468,1)`
Example 2.6	CK656	656	656	3884	$1.2 \cdot 10^7$	`rand(656,1)`
Example 2.7	ILLC1850*	1850	712	8645	–	ILLC1850_RHS1

Example 2.2. This example uses the same matrix A and vector b as Example 2.1 of Section 2.2. The vector b is not in $\mathcal{R}(A)$. The top graph of Figure 2.2 is determined with the code `alsqr` using the parameter values $k = 20$, $adjust = 40$, and $m = 100$. The bottom graph of Figure 2.2 is obtained with `alsqr` using the parameters $k = 20$, $adjust = 70$, and $m = 140$. We used $tolharm = 5 \cdot 10^{-2}$ to determine when to accept approximate singular vectors. The iterations were continued until the residual vectors r generated by `alsqr` for the first time satisfied $\|A^T r\|/\|A^T r_0\| \leq 10^{-12}$. The graphs of Figure 2.2 show the quotient $\|A^T r\|/\|A^T r_0\|$



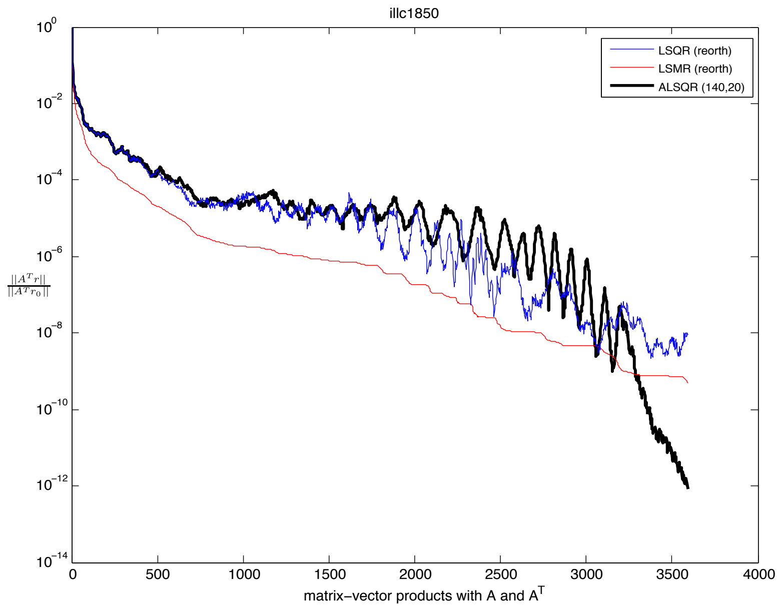
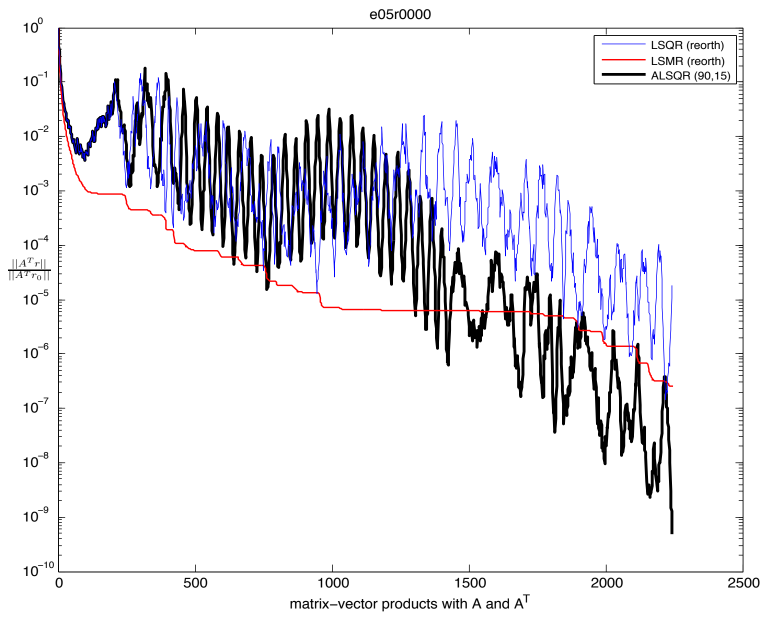


Figure 2.2. Example 2.2: LSQR(reorth) and LSMR(reorth) denote that reorthogonalization was applied to the last m vectors. ALSQR(100,20) denotes `alsqr` with parameters $m = 100$ and $k = 20$, and ALSQR(140,20) shows the performance of `alsqr` with $m = 140$ and $k = 20$. `alsqr` switched to nonrestarted LSQR at $2,840$ matrix-vector products in the *top graph* and at $2,680$ matrix-vector products for the *bottom graph*.

versus the number of matrix-vector products with A and A^T for each iteration of each method. The graphs marked `lsqr(reorth)` and `lsmr(reorth)` are for iterations with reorthogonalization. All methods reorthogonalized the last 100 vectors for the top graph and the last 140 vectors for the bottom graph of Figure 2.2. The `alsqr` algorithm exited the augmenting stage with all $k = 20$ approximate singular vectors converged after $2,840$ matrix-vector product evaluations for the top graph, and after $2,680$ matrix-vector product evaluations for the bottom graph. Having computed these approximate singular vectors, `alsqr` continued the iterations as a nonrestarted augmented LSQR method. The graphs show that augmentation by approximate singular vectors led to faster convergence and that `alsqr` converged before `lsqr` and `lsmr`.

Example 2.3. We let the matrix A and vector b be E05R0000 and E05R0000_RHS1, respectively, from the DRIVCAV set of the Matrix Market collection. The intended use of the linear systems in this collection is for testing iterative Krylov solvers, because it is difficult to find suitable preconditioners for the matrices. Since the linear system of equations is consistent, we can show convergence of both the quotients $\|A^T r\|/\|A^T r_0\|$ and $\|r\|/\|r_0\|$, where as usual r denotes the generated residual vector and r_0 the initial residual vector. We use the parameters $k = 15$, $adjust = 40$, $m = 90$ for `alsqr`. The value $tolharm = 3.5 \cdot 10^{-3}$ was used when deciding when to accept computed approximate singular vectors as converged. `alsqr` exited the augmening stage with all $k = 15$ approximate singular vectors converged when the matrix-vector product count was $1,230$. The iterations were continued with the fixed augmenting vectors until a residual vector satisfied $\|A^T r\|/\|A^T r_0\| \leq 10^{-9}$.

The top graph of Figure 2.3 displays $\|A^T r\|/\|A^T r_0\|$ versus the number of matrix-vector products with the matrices A and A^T for each iteration and for



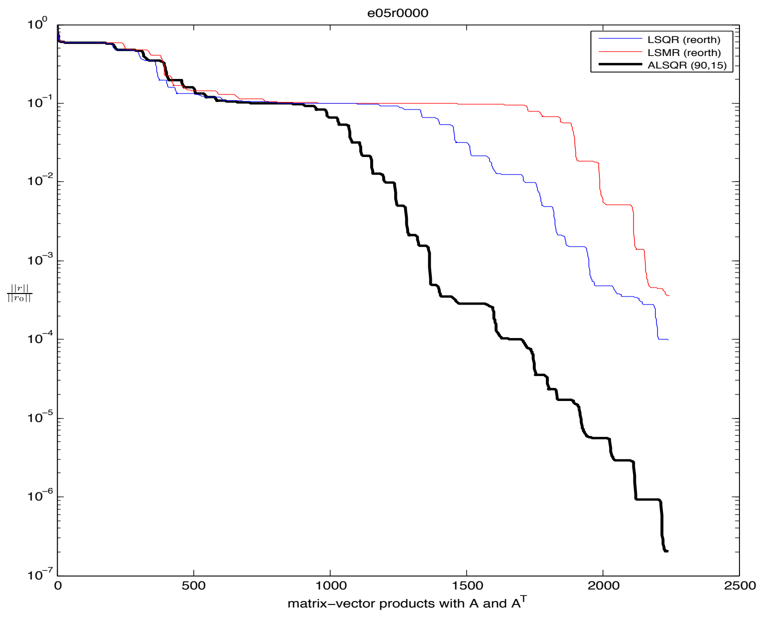
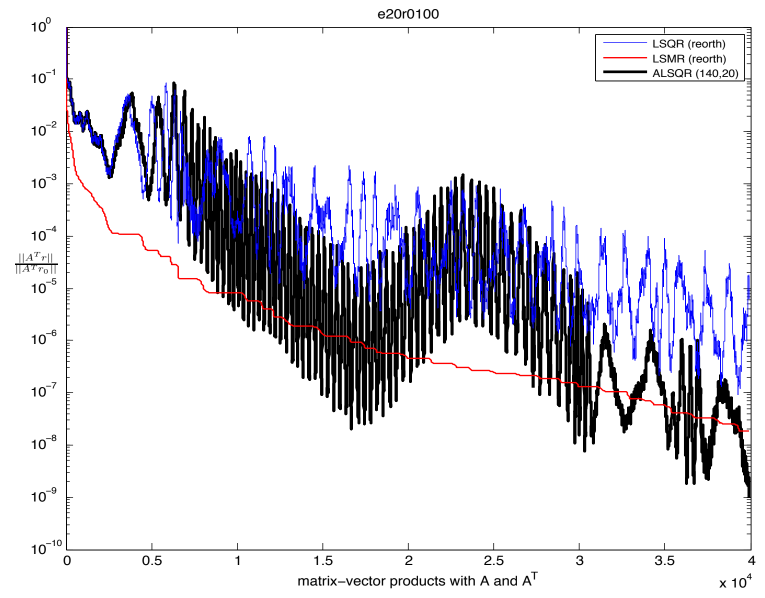


Figure 2.3. Example 2.3: LSQR(reorth) and LSMR(reorth) indicates that reorthogonalization of the last m vectors was carried out. ALSQR(90,15) denotes `alsqr` with parameters $m = 90$ and $k = 15$. `alsqr` switched to nonrestarted LSQR at $1,230$ matrix-vector product evaluations. The *top graph* shows $\|A^T r\|/\|A^T r_0\|$ for each iteration and the *bottom graph* displays $\|r\|/\|r_0\|$ for each iteration.

each method in our comparison. The bottom graph is analogous; it displays the quotients $\|r\|/\|r_0\|$ instead of $\|A^T r\|/\|A^T r_0\|$. This graph shows a fast steady decrease of the residual norm when `alsqr` carries out LSQR iterations with the fixed augmenting vectors.

Example 2.4. Let the matrix A and vector b be E20R0100 and E20R0100_RHS1, respectively, from the DRIVCAV set of the Matrix Market collection; see Example 2.3 for comments on this set of linear systems of equations. The code `alsqr` used the parameter values $k = 20$, $adjust = 90$, and $m = 140$. The matrix has a large condition number, $2.2 \cdot 10^{10}$, which leads to large oscillations in the quotients $\|A^T r\|/\|A^T r_0\|$ and very slow convergence. We used the same stopping criterion as in Example 2.3. Figure 2.4 is analogous to Figure 2.3. We used the parameter value $tolharm = 1.22 \cdot 10^{-4}$ to decide when approximate singular vectors could be considered converged. The code `alsqr` exited the augmenting stage with $k = 20$ converged approximate singular vectors when $30,280$ matrix-vector products with A and A^T had been computed. Notice that the residual curve in the bottom graph starts to decrease steadily long before the augmenting stage ends. This illustrates the positive effect of augmentation already while the augmenting vectors are computed.

Example 2.5. The matrix A is NOS5 from the LANPRO set in the Matrix Market collection. The matrices in this set stem from linear equations in structural engineering. This matrix set does not contain vectors b that can be used in (2.1). We therefore let b be a random vector with uniformly distributed entries in the interval $(0, 1)$. We use the parameter values $k = 20$, $adjust = 60$, $m = 120$, and $tolharm = 10^{-2}$ for the code `alsqr`. The augmenting stage, which lasted until $k = 20$ approximate singular vectors had converged, required $4,000$ matrix-vector product evaluations with A and A^T. Iterations were then continued with



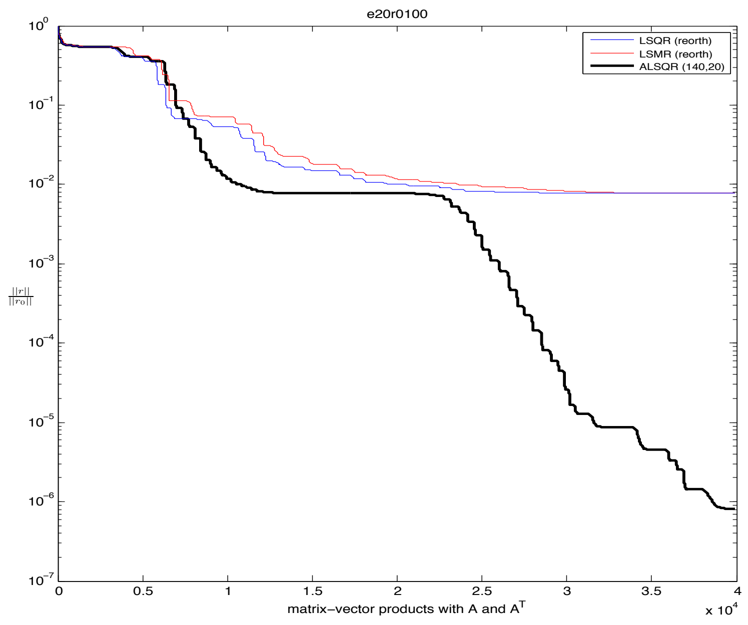
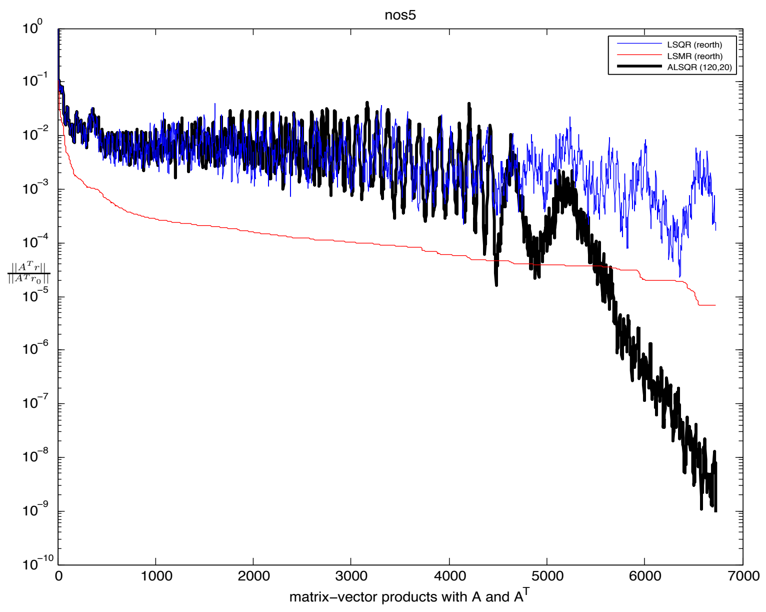


Figure 2.4. Example 2.4: LSQR(reorth) and LSMR(reorth) indicate that reorthogonalization of the m last vectors was carried out. The method ALSQR(m,k) for $m = 140$ and $k = 20$ is compared with LSQR and LSMR. `alsqr` switched to non-restarted LSQR after $30,280$ matrix-vector product evaluations. The *top graph* depicts $\|A^T r\|/\|A^T r_0\|$ for each iteration, while the *bottom graph* shows $\|r\|/\|r_0\|$ for each iteration.



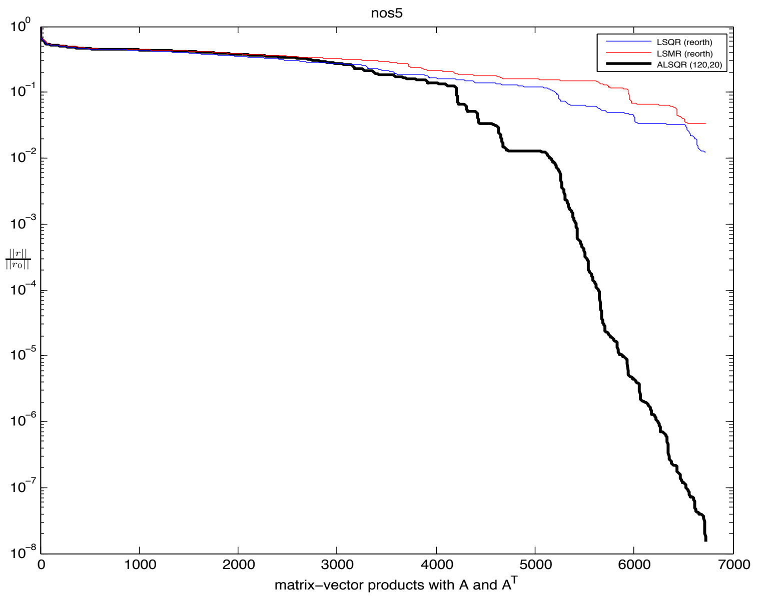
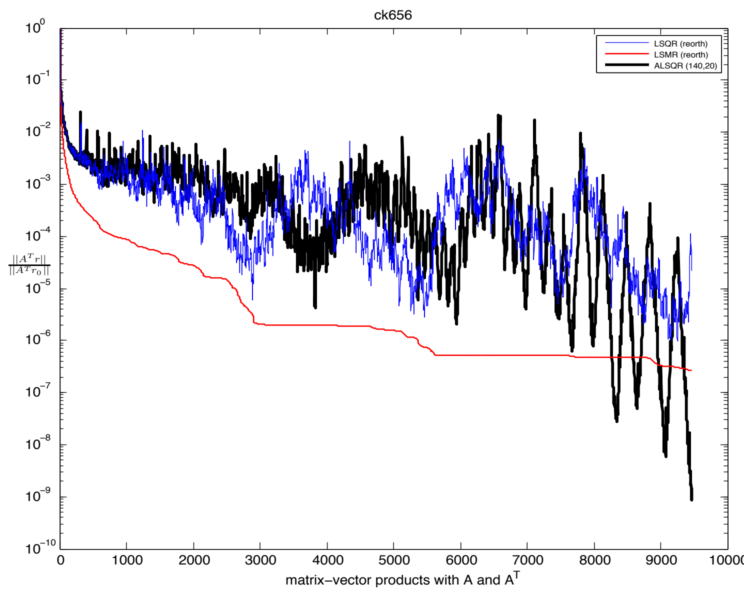


Figure 2.5. Example 2.5: LSQR(reorth) and LSMR(reorth) denote that reorthogonalization of the last m vectors was performed. The method ALSQR(m,k) is for $m = 120$ and $k = 20$ compared to LSQR and LSMR. `alsqr` switched to non-restarted LSQR after $4,000$ matrix-vector product evaluations. The *top graph* shows $\|A^T r\|/\|A^T r_0\|$ for each iteration and the *bottom graph* displays $\|r\|/\|r_0\|$ for each iteration.

the augmented LSQR method until $\|A^T r\|/\|A^T r_0\| \leq 10^{-9}$. Figure 2.5 is analogous to Figure 2.4. The bottom graph displays fast and steady decrease of $\|r\|/\|r_0\|$ of the nonrestarted LSQR method with fixed augmented vectors.

Example 2.6. The matrix A is chosen to be CK656, which is the largest matrix in the CHUCK set of the Matrix Market collection. This matrix has many clustered and multiple eigenvalues. The matrices in this collection arise from linear systems of equations in structural engineering. This collection does not contain right-hand side vectors. Therefore, we let b be a vector with random entries as in Example 2.5. We use the parameters $k = 20$, $adjust = 80$, $m = 140$, and $tolharm = 10^{-4}$ for `alsqr`. Iterations were terminated when $\|A^T r\|/\|A^T r_0\| \leq 10^{-9}$. The top graph of Figure 2.5 depicts $\|A^T r\|/\|A^T r_0\|$ versus the number of matrix-vector products with A and A^T. Figure 2.6 is analogous to Figure 2.5. In this example, `alsqr` did not exit the augmenting stage before the stopping criterion was satisfied, i.e., the stopping condition was satisfied before $k = 20$ approximate singular vectors had converged.

Example 2.7. The matrix A used in this example is obtained from the matrix ILLC1850 of Example 2.2 by letting the second column be twice the first column. We refer to the rank-deficient matrix so obtained as ILLC1850*. The vector b is the same as in Example 2.1. The LS problem (2.1) is inconsistent. We chose the parameters $k = 20$, $adjust = 40$, and $m = 100$ for `alsqr`, and used $tolharm = 4 \cdot 10^{-2}$ to decide when to accept approximate singular vectors as converged. All methods reorthogonalized the 100 last vectors. The required $k = 20$ approximate singular vectors had converge after $3,080$ matrix-vector product evaluations with A and A^T. At this point the code switched to run as an augmented nonrestarted LSQR method. The iterations were terminated as soon as $\|A^T r\|/\|A^T r_0\| \leq 10^{-11}$.



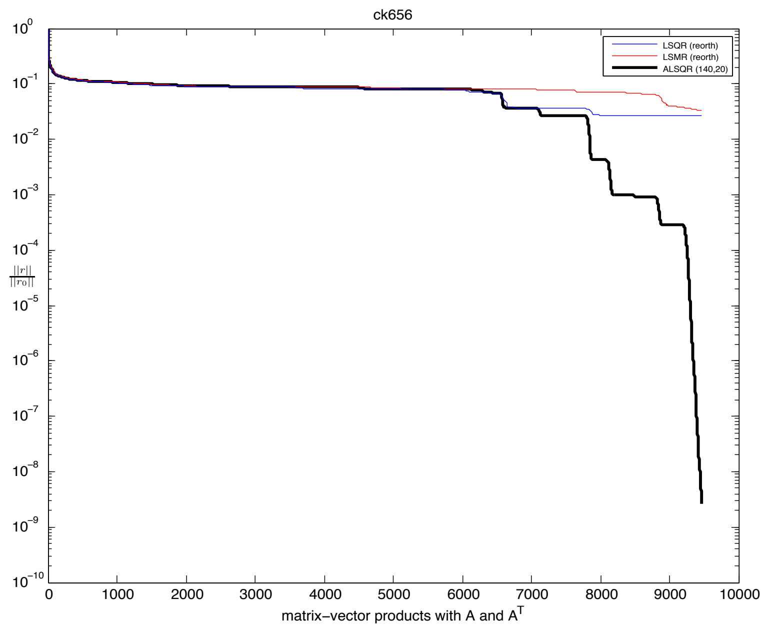


Figure 2.6. Example 2.6: LSQR(reorth) and LSMR(reorth) denotes that reorthogonalization of the last m vectors was carried out. ALSQR(140,20) indicates that `alsqr` is applied with $m = 140$ and $k = 20$. The code `alsqr` did not switch to nonrestarted LSQR before the convergence criterion was satisfied. The *top graph* displays $\|A^T r\|/\|A^T r_0\|$ for each iteration, and the *bottom graph* shows $\|r\|/\|r_0\|$ for each iteration.

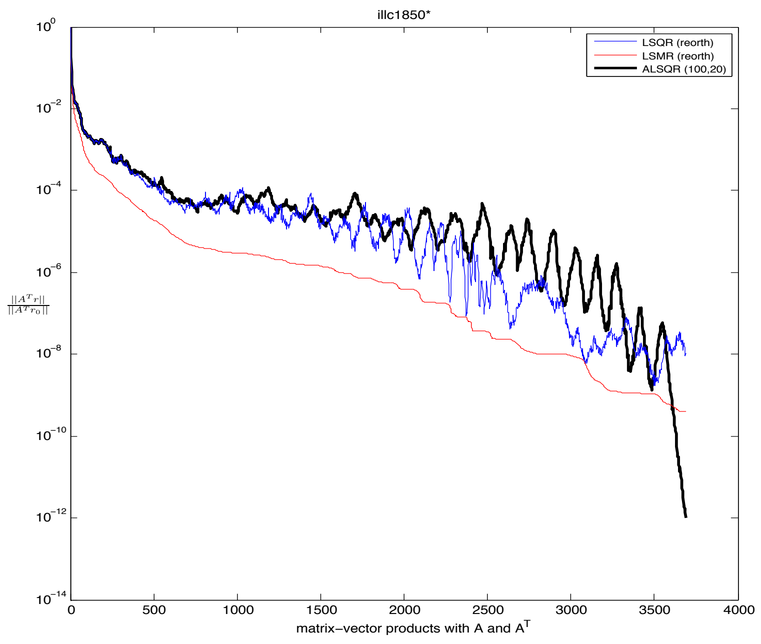


Figure 2.7. Example 2.7: The matrix A in this example is rank-deficient and the right-hand size b is not in the column space of A. Therefore, we show only the graph $\|A^T r\|/\|A^T r_0\|$ versus the number of matrix-vector products with A and A^T. The graphs LSQR(reorth) and LSMR(reorth) display results obtained when reorthogonalization of the last m vectors was carried out. ALSQR(100,20) denotes that `alsqr` is applied with the parameters $m = 100$ and $k = 20$. `alsqr` switched over to nonrestarted LSQR after $3,080$ matrix-vector product evaluation.

Figure 2.7 shows $\|A^T r\|/\|A^T r_0\|$ versus the number of matrix-vector product evaluations with A and A^T. This example illustrates that `alsqr` can be competitive also when applied to a rank-deficient LS problem.

2.8 Conclusion

We have described a new augmented LSQR method for large-scale linear LS problems or linear systems of equations. During the initial iterations, the method computes approximations of harmonic Ritz vectors that are used for augmenting the solution subspaces. Simultaneously, the method computes improved approximate solutions of the LS problem (2.1). Subsequently, the augmented vectors are kept fixed and used to form nonstandard Krylov subspaces used by a nonrestarted

LSQR method. Numerical examples show the proposed method to be competitive.

Acknowledgment

We would like to thank the referees for carefully reading the paper and for many comments that improved the presentation.

List of References

[1] Å. Björck, *Numerical methods for least squares problems.* Siam, 1996.

[2] S.-C. T. Choi, "Iterative methods for singular linear equations and least-squares problems," Ph.D. dissertation, Stanford University, 2006.

[3] D. C.-L. Fong and M. Saunders, "LSMR: An iterative algorithm for sparse least-squares problems," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2950–2971, 2011.

[4] K. Hayami, J.-F. Yin, and T. Ito, "GMRES methods for least squares problems," *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 5, pp. 2400–2430, 2010.

[5] M. E. Hochstenbach, "Harmonic and refined extraction methods for the singular value problem, with applications in least squares problems," *BIT Numerical Mathematics*, vol. 44, no. 4, pp. 721–754, 2004.

[6] C. C. Paige and M. A. Saunders, "LSQR: An algorithm for sparse linear equations and sparse least squares," *ACM Transactions on Mathematical Software*, vol. 8, no. 1, pp. 43–71, 1982.

[7] L. Reichel and Q. Ye, "A generalized LSQR algorithm," *Numerical Linear Algebra with Applications*, vol. 15, no. 7, pp. 643–660, 2008.

[8] M. Benzi and M. Tuma, "A robust preconditioner with low memory requirements for large sparse least squares problems," *SIAM Journal on Scientific Computing*, vol. 25, no. 2, pp. 499–512, 2003.

[9] Å. Björck and J. Yuan, "Preconditioners for least squares problems by LU factorization," *Electronic Transactions on Numerical Analysis*, vol. 8, pp. 26–35, 1999.

[10] S. Karimi, D. K. Salkuyeh, and F. Toutounian, "A preconditioner for the LSQR algorithm," *Journal of Applied Mathematics and Informatics*, vol. 26, no. 1-2, pp. 213–222, 2008.

[11] Y. Saad, *Iterative methods for sparse linear systems.* SIAM, 2003.

[12] R. B. Morgan, “A restarted GMRES method augmented with eigenvectors,” *SIAM Journal on Matrix Analysis and Applications*, vol. 16, no. 4, pp. 1154–1171, 1995.

[13] R. B. Morgan, “Implicitly restarted GMRES and Arnoldi methods for nonsymmetric systems of equations,” *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1112–1135, 2000.

[14] R. B. Morgan, “GMRES with deflated restarting,” *SIAM Journal on Scientific Computing*, vol. 24, no. 1, pp. 20–37, 2002.

[15] J. Baglama and L. Reichel, “Augmented implicitly restarted Lanczos bidiagonalization methods,” *SIAM Journal on Scientific Computing*, vol. 27, no. 1, pp. 19–42, 2005.

[16] R. F. Boisvert, R. Pozo, K. A. Remington, R. F. Barrett, and J. Dongarra, “Matrix Market: a web resource for test matrix collections,,” in *Quality of Numerical Software*, 1996, pp. 125–137.

[17] I. S. Duff, R. G. Grimes, and J. G. Lewis, “Users’ guide for the Harwell-Boeing sparse matrix collection (Release I),” Report RAL-92-086, Atlas Centre, Rutherford Appleton Laboratory, Didcot, Oxon, UK, Tech. Rep., 1992.

[18] J. Baglama and L. Reichel, “Restarted block Lanczos bidiagonalization methods,” *Numerical Algorithms*, vol. 43, no. 3, pp. 251–272, 2006.

[19] J. Baglama and L. Reichel, “An implicitly restarted block Lanczos bidiagonalization method using Leja shifts,” *BIT Numerical Mathematics*, vol. 53, no. 2, pp. 285–310, 2013.

[20] Z. Jia and D. Niu, “An implicitly restarted refined bidiagonalization Lanczos method for computing a partial singular value decomposition,” *SIAM journal on matrix analysis and applications*, vol. 25, no. 1, pp. 246–265, 2003.

[21] Z. Jia and D. Niu, “A refined harmonic Lanczos bidiagonalization method and an implicitly restarted algorithm for computing the smallest singular triplets of large matrices,” *SIAM Journal on Scientific Computing*, vol. 32, no. 2, pp. 714–744, 2010.

[22] E. Kokiopoulou, C. Bekas, and E. Gallopoulos, “Computing smallest singular triplets with implicitly restarted Lanczos bidiagonalization,” *Applied numerical mathematics*, vol. 49, no. 1, pp. 39–61, 2004.

[23] R. M. Larsen, “Lanczos bidiagonalization with partial reorthogonalization,” *DAIMI Report Series*, vol. 27, no. 537, 1998.

[24] R. M. Larsen, "Combining implicit restarts and partial reorthogonalization in Lanczos bidiagonalization," *Program in Scientific Computing and Computational Mathematics, Stanford University*, 2001.

[25] R. B. Morgan, "Computing interior eigenvalues of large matrices," *Linear Algebra and its Applications*, vol. 154, pp. 289–309, 1991.

[26] C. C. Paige, B. N. Parlett, and H. A. van der Vorst, "Approximate solutions and eigenvalue bounds from Krylov subspaces," *Numerical linear algebra with applications*, vol. 2, no. 2, pp. 115–133, 1995.

[27] Z. Jia, "Some properties of LSQR for large sparse linear least squares problems," *Journal of Systems Science and Complexity*, vol. 23, no. 4, pp. 815–821, 2010.

[28] C. C. Paige, "Bidiagonalization of matrices and solution of linear equations," *SIAM Journal on Numerical Analysis*, vol. 11, no. 1, pp. 197–209, 1974.

[29] H. D. Simon and H. Zha, "Low-rank matrix approximation using the Lanczos bidiagonalization process with applications," *SIAM Journal on Scientific Computing*, vol. 21, no. 6, pp. 2257–2274, 2000.

[30] MATLAB, *version R2011a.* Natick, Massachusetts: The MathWorks Inc., 2011.

CHAPTER 3

Implicitly restarting the LSQR algorithm

James Baglama[1] **and Daniel Richmond**[2]

Accepted for publication in *Electronic Transactions on Numerical Analysis* on February 7, 2014.

[1]Professor, Department of Mathematics, University of Rhode Island, Kingston, RI 02881.

e-mail: jbaglama@math.uri.edu

URL: http://math.uri.edu/∼jbaglama

[2]PhD Candidate, Department of Mathematics, University of Rhode Island, Kingston, RI 02881.

e-mail: dan@math.uri.edu

URL: http://math.uri.edu/∼dan

Abstract. The LSQR algorithm is a popular method for solving least-squares problems. For some matrices, LSQR may require a prohibitively large number of iterations to determine an approximate solution within a desired accuracy. This paper develops a strategy that couples the LSQR algorithm with an implicitly restarted Golub-Kahan bidiagonalization method to improve the convergence rate. The restart is carried out by applying the largest harmonic Ritz values as shifts and LSQR is used to compute the solution to the least-squares problem. Theoretical results show how this method is connected to the augmented LSQR method of [1] in which the Krylov subspaces are augmented with the harmonic Ritz vectors corresponding to the smallest harmonic Ritz values. Computed examples show the proposed method to be competitive with other methods.

Keywords. Golub-Kahan bidiagonalization, iterative method, implicit restarting, harmonic Ritz values, large-scale computation, least-squares, LSQR, Krylov subspace.

AMS Subject Classification. 65F15, 15A18

3.1 Introduction

In this paper, we will investigate large-scale least-squares (LS) problems

$$\min_{x\in\mathbb{R}^n}\|b - Ax\|, \qquad A \in \mathbb{R}^{\ell\times n}, \qquad b \in \mathbb{R}^{\ell}, \qquad \ell \geq n \tag{3.1}$$

where $\|\cdot\|$ denotes the Euclidean vector norm. The matrix A is assumed to be sparse and too large to apply the use of direct solvers efficiently, therefore iterative methods, which can also take advantage of the sparse structure of A, are required in order to solve the LS problem. When $\ell \geq n$ the preferred iterative method for solving LS problems is the LSQR Algorithm of Paige and Saunders [2]. LSQR is a Krylov subspace method that is based on the Golub-Kahan (GK) bidiagonalization, in which orthonormal bases for the m-dimensional Krylov subspaces

$$\begin{aligned}\mathcal{K}_m\left(AA^T, w_1\right) &= \operatorname{span}\left\{w_1, AA^T w_1, (AA^T)^2 w_1, \ldots, (AA^T)^{m-1} w_1\right\} \\ \mathcal{K}_m\left(A^T A, p_1\right) &= \operatorname{span}\left\{p_1, A^T A p_1, (A^T A)^2 p_1, \ldots, (A^T A)^{m-1} p_1\right\}\end{aligned} \tag{3.2}$$

are formed using the starting vectors $w_1 = r_0/\|r_0\|$ and $p_1 = A^T w_1/\|A^T w_1\|$, where $r_0 = b - Ax_0$ for an initial guess solution x_0 of the LS problem. Using the orthonormal bases for the spaces in (3.2) the LSQR Algorithm computes an approximate solution $x_m \in x_0 + \mathcal{K}_m\left(A^T A, p_1\right)$ and corresponding residual $r_m = b - Ax_m \in \mathcal{K}_m\left(AA^T, w_1\right)$ such that $\|b - Ax_m\|$ is minimized over all possible choices for x_m. The LSQR algorithm is a non-restarted method where the dimension m is increased until an acceptable solution of the LS problem is found. The theoretical foundation of LSQR yields a process that only requires the storage of a few basis vectors for each Krylov subspace. In exact arithmetic, LSQR terminates with the solution of the LS problem when linear dependence is established in (3.2). For LS problems with a well-conditioned matrix A or a small effective condition number, LSQR converges quickly yielding an approximation of the solution of the LS problem of desired accuracy long before linear dependence is encountered in (3.2), see Björck [3] for remarks. However, for LS problems with an ill-conditioned matrix A and a solution vector x with many components in the direction of the singular vectors associated with the smallest singular values, LSQR may require a prohibitively large number of iterations, see [3]. A contributing reason is that in finite arithmetic, the storage of only a few basis vectors at a time cannot maintain orthogonality among all previously non-stored basis vectors, hence the generated Krylov subspaces have a difficulty obtaining good approximations to the smallest singular triplets. The loss of orthogonality can be overcome by keeping previously computed basis vectors and reorthogonalizing. However, as m becomes large, this can become computationally expensive with an impractical storage requirement.

One solution is to use a restarted Krylov subspace method to solve the LS problem. Restarting Krylov subspace methods after $m << n$ can maintain orthogonality with a modest storage requirement. The restarted GMRES method of Saad and Schultz [4] is one of the most popular Krylov subspace methods for solving the LS problem when $\ell = n$. Using the restarted GMRES method to solve the LS problem introduces another problem, stagnation and/or slow convergence, [5, 6]. To overcome stagnation and/or slow convergence, restarted GMRES is often combined with a preconditioner or generated over an augmented Krylov subspace, see [7, 8, 9, 10, 11, 12] and references within.

If we implement a restarted LSQR method, i.e. restarting LSQR after $m << n$ iterations, we can maintain strong orthogonality among the bases by keeping all the vectors in storage, however, similar to GMRES, the restarted LSQR method can encounter stagnation and even slower convergence than using LSQR without reorthogonalization (cf. [13] for details on restarting the related LSMR algorithm). To overcome stagnation and/or slow convergence of restarting LSQR, we propose to solve the LS problem implicitly over an improved Krylov subspace, a form of preconditioning. We consider implicitly restarting the GK bidiagonalization (and hence LSQR) with a starting vector w_1^+, such that $w_1^+ = \phi\left(AA^T\right) w_1$ for some polynomial ϕ that is strong in the direction of the left singular vectors associated with the smallest singular values. The Krylov subspaces $\mathcal{K}_m\left(AA^T, w_1^+\right)$ and $\mathcal{K}_m\left(A^TA, p_1^+\right)$ will then contain good approximations to the left and right singular vectors corresponding to the smallest singular values, respectively. Also, with judiciously chosen shifts (i.e. zeros of $\phi\left(AA^T\right)$) we can ensure that $\mathcal{K}_m\left(AA^T, w_1^+\right)$ will contain the LSQR residual vector on each iteration of the restarted method. This is essential so that our restarted LSQR method produces a non-increasing residual curve. Since the singular values of A are not known prior to starting the

LSQR method, approximations must be found.

Implicitly restarted GK bidiagonalization methods [14, 15, 1, 16, 17, 18] have been used very successfully in providing good approximations to the smallest and largest singular triplets of a very large matrix A while using a small storage space and not many matrix-vector products. In this paper, we describe an implicitly restarted GK bidiagonalization method which selects a polynomial filter that produces good approximations of the singular vectors associated with the smallest singular values, thus improving the search spaces, while simultaneously computing approximate solutions to the LS problem. There are many methods for preconditioning LSQR to improve convergence [19, 3, 20, 21, 12], however most methods require constructions prior to approximating solutions to the LS problem adding to the storage and/or computational time.

In [1], we solved the LS problem with an LSQR method over a Krylov subspace that was explicitly augmented by approximate singular vectors of A. Augmenting Krylov subspaces in conjunction with solving the LS problem when $\ell = n$ with the restarted GMRES method was first discussed by Morgan in [10]. Later, Morgan showed the mathematical equivalence between applying harmonic Ritz values as implicit shifts and augmenting the Krylov subspaces by harmonic Ritz vectors to solve the LS problem when $\ell = n$ with restarted GMRES cf. [11]. Similarly, in Section 3.5, we show that our proposed method of this paper, applying harmonic Ritz values as implicit shifts to a restarted LSQR method to improve the Krylov subspaces is mathematically equivalent to the routine in [1] that obtains Krylov subspaces by explicitly augmenting them with the harmonic Ritz vectors to improve convergence. Therefore, the theorems from [1] which show improved convergence for LSQR using augmented spaces are applicable to this method. Applying the shifts implicitly is simple, and we introduce a new strategy for choosing and applying the shifts, which, based on our heuristics, further improves the convergence rates.

The paper is organized as follows: Section 3.2 describes, in detail, an implicitly restarted GK bidiagonalization method and the simplifications that can be utilized when using the harmonic Ritz values as shifts. Section 3.3 describes how LSQR can be successfully restarted by using the implicitly restarted GK bidiagonalization algorithm with harmonic Ritz values as shifts. The numerical issues of implicitly shifting via the buglechasing method are discussed in Section 3.4 along with a new method for implicitly applying harmonic Ritz values as a shift. Section 3.5 gives the theoretical results of how the implicitly restarted LSQR algorithm generates the same updated Krylov subspaces as the augmented LSQR algorithm from [1]. Section 3.6 gives numerical experiments to show the competitiveness of the proposed method, and Section 3.7 gives concluding remarks.

Throughout this paper, we will denote $\mathcal{N}(C)$ as the null space and $\mathcal{R}(C)$ as the range of the matrix C.

3.2 Implicitly restarted Golub-Kahan bidiagonalization

The GK bidiagonalization forms the basis for the LSQR algorithm discussed in Section 3.3 and is needed to approximate a set of the smallest singular triplets of A. Define $U_n = [u_1, u_2, \ldots, u_n] \in \mathbb{R}^{\ell \times n}$ and $V_n = [v_1, v_2, \ldots, v_n] \in \mathbb{R}^{n \times n}$ with orthonormal columns, as well as $\Sigma_n = \operatorname{diag}[\sigma_1, \sigma_2, \ldots, \sigma_n] \in \mathbb{R}^{n \times n}$. Then

$$AV_n = U_n\Sigma_n \qquad \text{and} \qquad A^TU_n = V_n\Sigma_n \tag{3.1}$$

are singular value decompositions (SVD) of A and A^T, respectively and

$$AV_k = U_k\Sigma_k \qquad \text{and} \qquad A^TU_k = V_k\Sigma_k \tag{3.2}$$

for $k << n$ are partial singular value decompositions (PSVD) of A and A^T, respectively. We assume the singular values to be ordered from the smallest to the largest one, i.e.,

$$0 < \sigma_1 \leq \sigma_2 \leq \ldots \leq \sigma_n,$$

since we are interested in the smallest singular values of A.

The GK bidiagonalization was originally proposed in [22] as a method for transforming a matrix A into upper bidiagonal form, however, for its connection to the LSQR algorithm in solving (3.1), we consider the variant that transforms A to lower bidiagonal form (cf. [2, bidiag 1]), described in Algorithm 3.1. The lower bidiagonal algorithm was described by Björk [23] as the more stable version of the GK bidiagonalization method and this form fits nicely into our implicitly restarted method.

Algorithm 3.1. GK Bidiagonalization Method

```
Input: $A \in \mathbb{R}^{\ell \times n}$ or functions for evaluating products with $A$ and $A^T$,
       $w_1 \in \mathbb{R}^{\ell}$ : initial starting vector,
       $m$ : number of bidiagonalization steps.

Output: $P_m = [p_1, \ldots, p_m] \in \mathbb{R}^{n \times m}$ : matrix with orthonormal columns,
        $W_{m+1} = [w_1, \ldots, w_{m+1}] \in \mathbb{R}^{\ell \times (m+1)}$ : matrix with orthonormal columns,
        $B_{m+1,m} \in \mathbb{R}^{(m+1) \times m}$ : lower bidiagonal matrix,
        $p_{m+1} \in \mathbb{R}^n$ : residual vector,
        $\alpha_{m+1} \in \mathbb{R}$.

1. Compute $\beta_1 := \|w_1\|$;   $w_1 := w_1/\beta_1$;  $W_1 := w_1$
2. Compute $p_1 := A^T w_1$;   $\alpha_1 := \|p_1\|$;   $p_1 := p_1/\alpha_1$;   $P_1 := p_1$
3. for $j = 1 : m$
```

```
    4. Compute $w_{j+1} := Ap_j - w_j\alpha_j$
    5. Reorthogonalization step: $w_{j+1} := w_{j+1} - W_{(1:j)}\left(W_{(1:j)}^T w_{j+1}\right)$
    6. Compute $\beta_{j+1} := \|w_{j+1}\|$;   $w_{j+1} := w_{j+1}/\beta_{j+1}$
    7. Compute $p_{j+1} := A^T w_{j+1} - p_j\beta_{j+1}$
    8. Reorthogonalization step: $p_{j+1} := p_{j+1} - P_{(1:j)}\left(P_{(1:j)}^T p_{j+1}\right)$
    9. Compute $\alpha_{j+1} := \|p_{j+1}\|$;   $p_{j+1} := p_{j+1}/\alpha_{j+1}$
    10. if $j < m$
        11. $P_{j+1} := [P_j, p_{j+1}]$
    12. endif
13. endfor
```

After $m << n$ steps, Algorithm 3.1 determines matrices W_{m+1} and P_m whose columns form orthonormal bases for the Kyrlov subspaces $\mathcal{K}_{m+1}\left(AA^T, w_1\right)$ and $\mathcal{K}_m\left(A^T A, p_1\right)$, respectively, as well as the decompositions

$$\begin{aligned} A^T W_{m+1} &= P_m B_{m+1,m}^T + \alpha_{m+1} p_{m+1} e_{m+1}^T \\ AP_m &= W_{m+1} B_{m+1,m} \end{aligned} \tag{3.3}$$

where $p_{m+1}^T P_m = 0$, and e_{m+1} is the $(m+1)$st axis vector. The matrix

$$B_{m+1,m} = \begin{bmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & 0 \\ & \beta_3 & \ddots & \\ & & \ddots & \alpha_m \\ 0 & & & \beta_{m+1} \end{bmatrix} \in \mathbb{R}^{(m+1)\times m} \tag{3.4}$$

is lower bidiagonal. We assume that Algorithm 3.1 does not terminate early, that is, $\alpha_j \neq 0$ and $\beta_j \neq 0$ for $1 \leq j \leq m+1$, see [1] for a discussion on how to handle early termination. To avoid loss of orthogonality in finite precision arithmetic in the basis vectors W_{m+1} and P_m, we reorthogonalize in lines 5 and 8 of the algorithm. The reorthogonalization steps do not add significant computational cost when $m << n$. For discussions and schemes on reorthogonalization we refer the reader to [14, 1, 13, 24, 25] and references within. For the numerical examples in Section 3.6 we follow the same scheme used in [1].

It is well known that using a Krylov subspace to obtain acceptable approximations to the smallest singular triplets of A with equations (3.3) can require a prohibitively large value of m. Therefore, a restarting strategy is required. The most effective restarting strategy is to use an implicit restart technique. By implicitly restarting after $m << n$ steps of the GK bidiagonalization, storage requirements can be kept relatively small and provide good approximations to the desired singular vectors from the generated Krylov subspaces. The following section provides a detailed discussion on how to implicitly restart the GK bidiagonalization method.

3.2.1 Implicit restart formulas for the GK bidiagonalization

Implicitly restarting a GK bidiagonalization method was first discussed in [23] and used in [14, 15, 1, 16, 17, 18]. Starting with the m-step GK bidiagonalization decomposition (3.3), the implicit restarting is done by selecting a shift μ and applying the shift via the Golub-Kahan SVD step [26, alg 8.6.1]. The algorithm given in [26] assumes an upper bidiagonal matrix is given, we modify the algorithm for a lower bidiagonal matrix and it is given as the bulgechasing (lower bidiagonal) algorithm (cf. Algorithm 3.2). Algorithm 3.2 uses the shift μ and generates upper Hessenberg orthogonal matrices $Q_L \in \mathbb{R}^{(m+1)\times(m+1)}$ and $Q_R \in \mathbb{R}^{m\times m}$ such that $B^+_{m+1,m} = Q_L^T B_{m+1,m} Q_R$ is lower bidiagonal. Multiplying the first equation of (3.3) by Q_L from the right and the second equation of (3.3) by Q_R also from the right yields

$$\begin{aligned} A^T W_{m+1} Q_L &= P_m B^T_{m+1,m} Q_L + \alpha_{m+1} p_{m+1} e^T_{m+1} Q_L \\ A P_m Q_R &= W_{m+1} B_{m+1,m} Q_R. \end{aligned} \tag{3.5}$$

Let $W_{m+1}^{+} = W_{m+1}Q_L$, $P_m^{+} = P_m Q_R$, and

$$p_m^{+} = \frac{\alpha_m^{+} p_m^{+} + (\alpha_{m+1} q_{L_{m+1,m}}) p_{m+1}}{\|\alpha_m^{+} p_m^{+} + (\alpha_{m+1} q_{L_{m+1,m}}) p_{m+1}\|} \tag{3.6}$$

where α_m^{+} is the (m, m) diagonal entry of $B_{m+1,m}^{+}$ and $q_{L_{m+1,m}}$ is the $(m+1, m)$ entry of Q_L. Now set $\alpha_m^{+} = \|\alpha_m^{+} p_m^{+} + (\alpha_{m+1} q_{L_{m+1,m}}) p_{m+1}\|$. Then we have after removing the last column from both sides of the equations in (3.5) a valid $(m-1)$-step GK bidiagonalization decomposition,

$$\begin{aligned} A^T W_m^{+} &= P_{m-1}^{+} B_{m,m-1}^{+T} + \alpha_m^{+} p_m^{+} e_m^T \\ A P_{m-1}^{+} &= W_m^{+} B_{m,m-1}^{+}. \end{aligned} \tag{3.7}$$

The $(m-1)$-step GK bidiagonalization decomposition (3.7) is the decomposition that we would have obtained by applying $(m-1)$ steps of Algorithm 3.1 with the starting vector $w_1^{+} = \gamma \left(AA^T - \mu I \right) w_1$, i.e a polynomial filter has been applied to w_1. See [15, 23, 18] for detailed discussions on polynomial filters in the context of implicitly restarting a GK bidiagonalization method. Given a suitable choice of shift μ the polynomial filter helps dampen unwanted singular vector components of A from w_1. Multiple shifts ($p = m - k$ shifts $\mu_1, \mu_2, \ldots, \mu_p$) can be applied via this process yielding the following valid k-step GK bidiagonalization decomposition,

$$\begin{aligned} A^T W_{k+1}^{+} &= P_k^{+} B_{k+1,k}^{+T} + \alpha_{k+1}^{+} p_{k+1}^{+} e_{k+1}^T \\ A P_k^{+} &= W_{k+1}^{+} B_{k+1,k}^{+} \end{aligned} \tag{3.8}$$

which would have been obtained by applying k-steps of Algorithm 3.1 with the starting vector $w_1^{+} = \tilde{\gamma} \prod_{i=1}^{p} \left(AA^T - \mu_i I \right) w_1$. Using the vectors p_{k+1}^{+}, w_{k+1}^{+} the $(k+1)$st vector of W_{k+1}^{+}, and the scalar α_{k+1}^{+} the k-step GK bidiagonalization decomposition (3.8) can be extended to an m-step GK bidiagonalization decomposition (3.3) by starting at step 4 of Algorithm 3.1 and continuing for p more iterations.

Algorithm 3.2. Bulgechasing (lower bidiagonal)

```
Input: $B_{m+1,m} \in \mathbb{R}^{(m+1)\times m}$ lower bidiagonal matrix,
       $\mu$ : implicit shift.

Output: $Q_L \in \mathbb{R}^{(m+1)\times(m+1)}$ : upper Hessenberg matrix with orthonormal columns,
        $Q_R \in \mathbb{R}^{m\times m}$ : upper Hessenberg matrix with orthonormal columns,
        $B^{+}_{m+1,m} = Q_L^T B_{m+1,m} Q_R \in \mathbb{R}^{(m+1)\times m}$ : updated lower bidiagonal matrix.

1. Determine the $(m+1)\times(m+1)$ Givens rotation matrix $G(1,2,\theta_1)$ such that
   $\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} b_{1,1}^2 - \mu \\ b_{1,1}\cdot b_{2,1} \end{bmatrix} = \begin{bmatrix} \star \\ 0 \end{bmatrix}$
2. Set $Q_L^T := G(1,2,\theta_1)$;   $Q_R := I_m$;   $B^{+}_{m+1,m} := G(1,2,\theta_1)B_{m+1,m}$
3. for $i = 1 : m-1$
    4. Determine the $m \times m$ Givens rotation matrix $G(i,i+1,\theta_i)$ such that
       $\begin{bmatrix} b^{+}_{i,i} & b^{+}_{i,i+1} \end{bmatrix} \begin{bmatrix} c & -s \\ s & c \end{bmatrix} = \begin{bmatrix} \star & 0 \end{bmatrix}$
    5. Update $Q_R := Q_R G(i,i+1,\theta_i)$;   $B^{+}_{m+1,m} := B^{+}_{m+1,m} G(i,i+1,\theta_i)$
    6. Determine the $(m+1)\times(m+1)$ Givens rotation matrix $G(i+1,i+2,\theta_{i+1})$
       such that
       $\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} b^{+}_{i+1,i} \\ b^{+}_{i+2,i} \end{bmatrix} = \begin{bmatrix} \star \\ 0 \end{bmatrix}$
    7. Update $Q_L^T := G(i+1,i+2,\theta_{i+1})Q_L^T$;   $B^{+}_{m+1,m} := G(i+1,i+2,\theta_{i+1})B^{+}_{m+1,m}$
8. endfor
```

3.2.2 Implicit restart with harmonic Ritz values as shifts

The dampening effect of the polynomial filter, $\prod_{i=1}^{p}\left(AA^T - \mu_i I\right)$, depends on the choice of shifts μ_i. There are several choices for μ_i that have been investigated in the literature in this context; Ritz and harmonic Ritz values [18], refined Ritz values [16], refined harmonic Ritz values [17], and Leja points [15]. We examine the choice of using harmonic Ritz values as shifts for our implicitly restarted method. Harmonic Ritz values not only provide good approximations to the smallest singular values of A they have a much needed connection with the LSQR algorithm described in Section 3.3.

The harmonic Ritz values $\hat{\theta}_j$ of AA^T are defined as the eigenvalues to the generalized eigenvalue problem

$$\left(\left(B_{m,m}B_{m,m}^T\right) + \alpha_m^2\beta_{m+1}^2\left(B_{m,m}B_{m,m}^T\right)^{-1} e_m e_m^T\right) g_j = \hat{\theta}_j g_j, \quad 1 \le j \le m \tag{3.9}$$

where $B_{m,m}$ is the $m \times m$ principal submatrix of $B_{m+1,m}$, and $g_j \in \mathbb{R}^m \backslash \{0\}$ is an eigenvector, see e.g., [27, 28] for properties and discussions of harmonic Ritz values. The eigenpairs $\left\{\hat{\theta}_j, g_j\right\}_{j=1}^m$ can be computed without forming the matrix $B_{m,m}B_{m,m}^T$ from the SVD of $B_{m+1,m}$,

$$\begin{aligned} B_{m+1,m}\tilde{V}_m &= \left[\tilde{U}_m \ \tilde{u}_{m+1}\right]\begin{bmatrix}\tilde{\Sigma}_m \\ 0\end{bmatrix}, \\ B_{m+1,m}^T\left[\tilde{U}_m \ \tilde{u}_{m+1}\right] &= \tilde{V}_m\left[\tilde{\Sigma}_m \quad 0\right], \end{aligned} \tag{3.10}$$

where the matrices $\tilde{V}_m = [\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_m] \in \mathbb{R}^{m\times m}$ and $\tilde{U}_m = [\tilde{u}_1, \tilde{u}_2, \ldots, \tilde{u}_m] \in \mathbb{R}^{(m+1)\times m}$ have orthonormal columns, $\tilde{u}_{m+1} \in \mathbb{R}^{m+1}$ (the null vector) is a unit-length vector such that $\tilde{u}_{m+1}^T\tilde{U}_m = 0$, and $\tilde{\Sigma}_m = \text{diag}\left[\tilde{\sigma}_1, \tilde{\sigma}_2, \ldots, \tilde{\sigma}_m\right] \in \mathbb{R}^{m\times m}$. We order the m singular values according to

$$0 < \tilde{\sigma}_1 < \tilde{\sigma}_2 < \ldots < \tilde{\sigma}_m. \tag{3.11}$$

The strict inequalities come from the assumption that the diagonal and sub-diagonal entries of $B_{m+1,m}$ are all nonzero [29, Lemma 7.7.1].

We have $\hat{\theta}_j = \tilde{\sigma}_j^2$, see [28] for details. The eigenvectors g_j are the columns of $\left[I_m \quad \beta_{m+1}B_{m,m}^{-T}e_m\right]\tilde{U}_m$, see [1] for details. Furthermore, if $\tilde{\sigma}_j^2$ is used as a shift in Algorithm 3.2 then the return matrix $B_{m+1,m}^+$ has entries $\alpha_m^+ = 0$ and $\beta_{m+1}^+ = \pm\tilde{\sigma}_j$. The following theorem shows this result.

Theorem 3.1. *Given a lower bidiagonal matrix $B_{m+1,m}$ (3.4) where $\alpha_j \neq 0$ for $1 \leq j \leq m$ and $\beta_j \neq 0$ for $2 \leq j \leq m+1$. In Algorithm 3.2, $\mu = \tilde{\sigma}_j^2$ (3.11) if and only if the return matrix $B^+_{m+1,m}$ has $\alpha^+_m = 0$ and $\beta^+_{m+1} = \pm\tilde{\sigma}_j$. Furthermore, Algorithm 3.2 returns the matrices Q_L and Q_R such that $Q_L e_{m+1} = \pm\tilde{u}_j$ and $Q_R e_m = \pm\tilde{v}_j$.*

Proof. Compute the QR-factorization of $B_{m+1,m}B^T_{m+1,m} - \mu I_{m+1} = QR$ where $Q \in \mathbb{R}^{(m+1)\times(m+1)}$ is orthogonal and $R \in \mathbb{R}^{(m+1)\times(m+1)}$ is upper triangular. An inspection of steps 1 and 2 in Algorithm 3.2 show that the first columns of Q_L and Q are equal. Therefore, via the implicit Q Theorem [26, Theorem 7.4.2] we have $Q_L = QD$ where $D = \text{diag}[1, \pm1, \ldots, \pm1]$ and

$$B^+_{m+1,m}B^{+T}_{m+1,m} = Q_L^T B_{m+1,m}B^T_{m+1,m}Q_L = DQ^T B_{m+1,m}B^T_{m+1,m}QD. \tag{3.12}$$

The matrix $B^+_{m+1,m}B^{+T}_{m+1,m}$ is a symmetric tridiagonal matrix and if μ is an eigenvalue of $B_{m+1,m}B^T_{m+1,m}$ then $Q_L^T B_{m+1,m}B^T_{m+1,m}Q_L e_{m+1} = DQ^T B_{m+1,m}B^T_{m+1,m}QDe_{m+1} = \mu e_{m+1}$, [26, Section 8.3.3]. Therefore,

$$B^+_{m+1,m}B^{+T}_{m+1,m}e_{m+1} = DQ^T B_{m+1,m}B^T_{m+1,m}QDe_{m+1} = \mu e_{m+1} \tag{3.13}$$

and $\beta^+_{m+1}\alpha^+_m = 0$ and $\beta^{+^2}_{m+1} = \mu$. Since $\tilde{\sigma}_j^2 \neq 0$ are eigenvalues of $B_{m+1,m}B^T_{m+1,m}$, we have $\alpha^+_m = 0$ and $\beta^+_{m+1} = \pm\tilde{\sigma}_j$. The reverse holds by noticing that $B_{m+1,m}B^T_{m+1,m}$ is unreduced, and if μ is not an eigenvalue, then $B^+_{m+1,m}B^{+T}_{m+1,m}$ must also be unreduced [29, Lemma 8.13.1]. Algorithm 3.2 returns the relationships $B_{m+1,m}Q_R = Q_L B^+_{m+1,m}$ and $B^T_{m+1,m}Q_L = Q_R B^{+T}_{m+1,m}$. Using the structure of the last column of $B^+_{m+1,m}$ we have

$$\begin{aligned} B_{m+1,m}Q_R e_m &= Q_L B^+_{m+1,m}e_m &&= \pm\tilde{\sigma}_j Q_L e_{m+1} \\ B^T_{m+1,m}Q_L e_{m+1} &= Q_R B^{+T}_{m+1,m}e_{m+1} &&= \pm\tilde{\sigma}_j Q_R e_m. \end{aligned} \tag{3.14}$$

The result $Q_L e_{m+1} = \pm\tilde{u}_j$ and $Q_R e_m = \pm\tilde{v}_j$ follows from the SVD of $B_{m+1,m}$ (3.10) and that the singular vectors of non-degenerate singular values are unique up to sign difference [3]. □

Calling Algorithm 3.2 with $B_{m+1,m}$ and $\mu = \tilde{\sigma}_m^2$ returns the upper Hessenberg orthogonal matrices $Q_L = [Q_{L_m}, \pm\tilde{u}_m] \in \mathbb{R}^{(m+1)\times(m+1)}$ where $Q_{L_m} = [q_{L_1}, \ldots, q_{L_m}] \in \mathbb{R}^{(m+1)\times m}$, $Q_R = \left[Q_{R_{m-1}}, \pm\tilde{v}_m\right] \in \mathbb{R}^{m\times m}$ where $Q_{R_{m-1}} = \left[q_{R_1}, \ldots, q_{R_{m-1}}\right] \in \mathbb{R}^{m\times(m-1)}$, and the lower bidiagonal matrix,

$$B_{m+1,m}^{+} = \begin{bmatrix} \begin{bmatrix} & B_{m,m-1}^{+} & \\ & & \end{bmatrix} & 0 \\ 0 & \pm\tilde{\sigma}_m \end{bmatrix} \in \mathbb{R}^{(m+1)\times m}. \tag{3.15}$$

The singular values of $B_{m,m-1}^{+}$ are

$$0 < \tilde{\sigma}_1 < \tilde{\sigma}_2 < \ldots < \tilde{\sigma}_{m-1} \tag{3.16}$$

and the SVD of $B_{m,m-1}^{+}$ is

$$\begin{aligned} B_{m,m-1}^{+} Q_{R_{m-1}}^T \tilde{V}_{m-1} &= Q_{L_m}^T \tilde{U}_{m-1} \tilde{\Sigma}_{m-1} \\ B_{m,m-1}^{+T} Q_{L_m}^T \tilde{U}_{m-1} &= Q_{R_{m-1}}^T \tilde{V}_{m-1} \tilde{\Sigma}_{m-1}. \end{aligned} \tag{3.17}$$

Calling Algorithm 3.2 with $B_{m,m-1}^{+}$ and $\mu = \tilde{\sigma}_{m-1}^2$ returns the upper Hessenberg orthogonal matrices $Q_{L_m}^{+} = \left[Q_{L_{m-1}}^{+}, \pm Q_{L_{m-1}}^T \tilde{u}_{m-1}\right] \in \mathbb{R}^{m\times m}$ where $Q_{L_{m-1}}^{+} = \left[q_{L_1}^{+}, \ldots, q_{L_{m-1}}^{+}\right] \in \mathbb{R}^{m\times(m-1)}$, $Q_{R_{m-1}}^{+} = \left[Q_{R_{m-2}}^{+}, \pm Q_{R_{m-1}}^T \tilde{v}_{m-1}\right] \in \mathbb{R}^{(m-1)\times(m-1)}$ where $Q_{R_{m-2}}^{+} = \left[q_{R_1}^{+}, \ldots, q_{R_{m-2}}^{+}\right] \in \mathbb{R}^{(m-1)\times(m-2)}$ and the lower bidiagonal matrix,

$$B_{m,m-1}^{++} = \begin{bmatrix} \begin{bmatrix} & B_{m-1,m-2}^{++} & \\ & & \end{bmatrix} & 0 \\ 0 & \pm\tilde{\sigma}_{m-1} \end{bmatrix} \in \mathbb{R}^{m\times(m-1)}. \tag{3.18}$$

Since columns of $Q_{R_{m-1}}$ are orthonormal and $\tilde{v}_{m-1} \in \mathcal{R}\left(Q_{R_{m-1}}\right)$ we have $Q_{R_{m-1}} Q_{R_{m-1}}^T \tilde{v}_{m-1} = \tilde{v}_{m-1}$. Likewise $Q_{L_{m-1}} Q_{L_{m-1}}^T \tilde{u}_{m-1} = \tilde{u}_{m-1}$. Therefore,

$$\begin{aligned} &\left(Q_L \begin{bmatrix} Q_{L_m}^{+} & 0 \\ 0 & 1 \end{bmatrix}\right)^T B_{m+1,m}\, Q_R \begin{bmatrix} Q_{R_{m-1}}^{+} & 0 \\ 0 & 1 \end{bmatrix} = \\ &\begin{bmatrix} \begin{bmatrix} & B_{m-1,m-2}^{++} & \\ & & \end{bmatrix} & 0 & \\ & \pm\tilde{\sigma}_{m-1} & \\ 0 & & \pm\tilde{\sigma}_m \end{bmatrix} \end{aligned} \tag{3.19}$$

where

$$Q_R \begin{bmatrix} Q^+_{R_{m-1}} & 0 \\ 0 & 1 \end{bmatrix} = \left[q^{++}_{R_1}, \ldots, q^{++}_{R_{m-2}}, \tilde{v}_{m-1}, \tilde{v}_m\right] \tag{3.20}$$

and

$$Q_L \begin{bmatrix} Q^+_{L_m} & 0 \\ 0 & 1 \end{bmatrix} = \left[q^{++}_{L_1}, \ldots, q^{++}_{L_{m-1}}, \tilde{u}_{m-1}, \tilde{u}_m\right]. \tag{3.21}$$

The matrices (3.20) and (3.21) are no longer upper Hessenberg, they have an additional nonzero sub-diagonal below the diagonal, increasing the lower band width to 2.

Repeating the process we can use Algorithm 3.2 to apply multiple shifts. After applying the largest $p = m - k$ harmonic Ritz values $\left(\tilde{\sigma}^2_m, \ldots, \tilde{\sigma}^2_{k+1}\right)$ as shifts we have

$$Q^T_L B_{m+1,m} Q_R = \begin{bmatrix} \begin{bmatrix} & B^+_{k+1,k} & \end{bmatrix} & & \text{\Large 0} \\ & \pm\tilde{\sigma}_{k+1} & \\ & & \ddots & \\ \text{\Large 0} & & & \pm\tilde{\sigma}_m \end{bmatrix} \tag{3.22}$$

where

$$\begin{aligned} Q_R &= \left[Q_{R_k}, \tilde{v}_{k+1}, \ldots, \tilde{v}_m\right] \\ Q_L &= \left[Q_{L_{k+1}}, \tilde{u}_{k+1}, \ldots, \tilde{u}_m\right]. \end{aligned} \tag{3.23}$$

The matrices (3.23) now have a lower band width equal to p. Using the process outlined in Section 3.2.1 with (3.22) and (3.23) we have analogous to (3.8) a k-step GK bidiagonalization,

$$\begin{aligned} A^T W^+_{k+1} &= P^+_k B^{+T}_{k+1,k} + \alpha^+_{k+1} p^+_{k+1} e^T_{k+1} \\ A P^+_k &= W^+_{k+1} B^+_{k+1,k} \end{aligned} \tag{3.24}$$

where $W^+_{k+1} = W_{m+1} Q_{L_{k+1}}$, $P^+_k = P_m Q_{R_k}$, $B^+_{k+1,k} = Q^T_{L_{k+1}} B_{m+1,m} Q_{R_k}$,

$$p^+_{k+1} = \frac{(\alpha_{m+1} q_{L_{m+1,k+1}})}{|\alpha_{m+1} q_{L_{m+1,k+1}}|} p_{m+1} \tag{3.25}$$

and $\alpha^+_{k+1} = |\alpha_{m+1} q_{L_{m+1,k+1}}|$. Using the vectors p^+_{k+1}, w^+_{k+1} the $(k+1)$st vector of W^+_{k+1}, and the scalar α^+_{k+1} the k-step GK bidiagonalization decomposition (3.24) can be extended to an m-step GK bidiagonalization decomposition (3.3) by starting at step 4 of Algorithm 3.1 and continuing for p more iterations.

We remark again that the importance of using harmonic Ritz values as shifts is the connection with the LSQR method described in Section 3.3 where zeroing out the diagonal elements of $B^+_{m+1,m}$ cf. (3.15, 3.18, 3.19, 3.22) is essential for restarting the LSQR method.

3.2.3 Adaptive shift strategy

In order to help speed up convergence to the smallest singular triplets and ultimately speed up our implicitly restarted LSQR algorithm, we developed an adaptive shift strategy. It was first observed in [30] that if a shift μ_{k+1} that is numerically close to σ^2_k is used in the implicitly restarted GK bidiagonalization method, then the component along the k-th left singular vector can be greatly damped in

$$w^+_1 = \prod_{i=k+1}^{m} \left(AA^T - \mu_i I\right) w_1. \tag{3.26}$$

This can cause the resulting spaces W^+_{k+1} and V^+_k (3.24) to contain poor approximations to the left and right singular vector corresponding to σ_k, respectively. When using an implicitly restarted GK bidiagonalization to solve for a PSVD of A, a heuristic was proposed in [30] to require that the relative gap between the approximating value $\tilde{\sigma}^2_k$ and all shifts μ_i, defined by

$$\text{relgap}_{ki} = \frac{(\tilde{\sigma}^2_k - e_k) - \mu_i}{\tilde{\sigma}^2_k} \tag{3.27}$$

where e_k is the error bound on $\tilde{\sigma}^2_k$, be greater than 10^{-3}. In the context of [30], the shifts considered to be too close, i.e the “bad shifts”, were simply replaced by zero shifts. This strategy was adapted and applied in [16, 17] in which harmonic and refined harmonic Ritz values were used as shifts for computing some of the smallest and largest singular triplets. When searching for the smallest singular triplets, the “bad shifts” were replaced with the largest among all shift. In either case, through observation and numerical experiment, this improved the convergence of the smallest singular triplets. When implicitly restarting the GK bidiagonalization in combination with the LSQR algorithm, we cannot replace a “bad shift” by the largest among all shifts,. i.e. our combined routine does not allow repeated shifts. This would destroy the required Hessenberg structure of Q_R and Q_L in the equations given in Section 2. We also cannot use a zero shift, this would remove the null vector $\tilde{u}_{m+1}$ of $B_{m+1,m}$ from the space, see Section 3.3 for details. In our case, we are not just concerned with a finding approximations to the k smallest singular triplets of A, but rather to find a solution to the LS problem. Instead of applying p shifts, we therefore opt to dynamically change the number of shifts to apply in order to have the best approximations to a set of singular triplets in our updated spaces W^+_{k+1} and V^+_k (3.24). That is, we look for the largest gap between certain $\tilde{\sigma}$s and only apply shifts up to the gap.

Our heuristic is based on two properties; that the harmonic Ritz singular value approximation $\tilde{\sigma}_i$ to σ_i is such that $\sigma_i \leq \tilde{\sigma}_i$ [31] and the interlace property of the harmonic Ritz and Ritz values [28]. Using these properties lead us to examine the gaps between consecutive harmonic Ritz singular value approximations $\tilde{\sigma}$. If $\tilde{\sigma}_i$ is very near to $\tilde{\sigma}_{i+1}$ (and hence σ_i is possibly very near to σ_{i+1}) then components of the updated starting vector in the direction of u_i from (3.1) may be greatly damped by applying $\tilde{\sigma}^2_{i+1} = \hat{\theta}_{i+1}$ as a shift, which is undesired. To minimize the possibility of this happening, our heuristic method fixes a small value j and searches the interval $\left[\hat{\theta}_{k+1-j}, \ldots, \hat{\theta}_{k+1}, \ldots, \hat{\theta}_{k+1+j}\right]$ around $\hat{\theta}_{k+1}$ for the largest gap between any two consecutive harmonic Ritz values. That is, an index k_j is chosen such that

$$\max_{k+1-j \leq k_j \leq k+j} \left| \hat{\theta}_{k_{j+1}} - \hat{\theta}_{k_j} \right| \tag{3.28}$$

and k is replaced with k_j where the number of shifts in the implicitly restarted GK bidiagonalization is set to $p = m - k_j$. Through numerical observation, a suitable choice for j is typically between 2 and 6. Choosing j too large can have a dramatic negative effect on the convergence rate. See Table 6.2 in Section 3.6 for numerical results on different values of j, and the improved convergence rates obtained when using this adaptive shifting strategy.

3.3 Implicitly restarted LSQR

In this section we describe our implicitly restarted LSQR method, Algorithm 3.4, which is a combination of a restarted LSQR method with the implicitly restarted GK bidiagonalization method described in Section 3.2. Algorithm 3.3 outlines a single step of a restarted LSQR method that we will need. A first call to Algorithm 3.3 with an initial approximate solution of the LS problem x_0, $r_0 = b - Ax_0$ and $w_1 = r_0$ will produced the same output x_m and r_m as the Paige and Saunders [2] routine. However, in order to call Algorithm 3.3 again after we use the implicitly restarted formulas of Section 3.2 to reduce the m-step GK bidiagonalization (3.3) to a k-step GK bidiagonalization (3.24) we need to have $r_m \in \mathcal{R}\left(W_{k+1}^{+}\right)$. If $r_m \notin \mathcal{R}\left(W_{k+1}^{+}\right)$ then we are using a Krylov subspace that does not contain the residual vector. This would require an approximation of r_m from the Krylov subspace, which can severally slow down or produce no convergence.

However, it was shown in [1] that if we have an m-step GK bidiagonalization (3.3) and compute x_m from LSQR equations, i.e. steps 2 and 3 of Algorithm 3.3 then $r_m = \gamma W_{m+1}\tilde{u}_{m+1}$, where $\tilde{u}_{m+1}$ is the null vector (3.10), of $B_{m+1,m}$ and $\gamma \in \mathbb{R}$. Using the implicitly restarted formulas of Section 3.2 with an application of the p largest harmonic Ritz values as shifts we obtain a k-step GK bidiagonalization decomposition (3.24) with $W^+_{k+1} = W_{m+1}Q_{L_{k+1}}$. Equation (3.23) shows that we must have $\tilde{u}_{m+1} \in \mathcal{R}\left(Q_{L_{k+1}}\right)$ and hence $r_m = W^+_{k+1}f_{k+1}$ for some vector $f_{k+1} \in \mathbb{R}^{k+1}$, i.e. $r_m \in \mathcal{R}\left(W^+_{k+1}\right)$.

Algorithm 3.3. Restarted LSQR step

```
Input: k-step GK bidiagonalization (3.24) or (3.31) or the k-step factorization
        (3.30) where $r_k \in \mathcal{R}\left(W^+_{k+1}\right)$,
    $p = m - k$ : number of additional bidiagonalization steps,
    $x_k \in \mathbb{R}^n$ : approximation to LS problem.

Output: m-step GK bidiagonalization (3.3),
    $x_m \in \mathbb{R}^n$ : approximation to LS problem,
    $r_m \in \mathbb{R}^\ell$ : residual vector.

1. Apply $p = m - k$ additional steps of Algorithm 3.1 to obtain an
   m-step GK bidiagonalization (3.3)
2. Solve $\min_{y_m \in \mathbb{R}^m} \left\| \begin{bmatrix} f_{k+1} \\ 0 \end{bmatrix} - B_{m+1,m}y_m \right\|$ for $y_m$
   where $r_k = W_{m+1} \begin{bmatrix} f_{k+1} \\ 0 \end{bmatrix}$ for some $f_{k+1} \in \mathbb{R}^{k+1}$
3. Set $x_m = x_k + P_m y_m$
4. $r_m = r_k - W_{m+1}B_{m+1,m}y_m$
```

The residual and approximate solution to the LS problem can be updated during step 1 of Algorithm 3.3, i.e. during the GK bidiagonalization Algorithm 3.1. The MATLAB code `irlsqr` used for numerical examples in Section 6 which implements Algorithm 3.4 updates the LSQR approximation and residual during

GK bidiagonalization steps. Below is our algorithm that outlines the main routine of this paper.

Algorithm 3.4. Implicitly Restarted LSQR (IRLSQR)

Input: $A \in \mathbb{R}^{\ell \times n}$ *or functions for evaluating products with* A *and* A^T,

$x_0 \in \mathbb{R}^n$*: Initial approximate solution to LS problem,*

$r_0 = b - Ax_0 \in \mathbb{R}^\ell$ *: initial residual vector,*

m *: maximum size of GK bidiagonalization decomposition,*

p *: number of shifts to apply,*

j *: integer used to adjust number of shifts (3.28),*

δ *: tolerance for accepting an approximate solution.*

Output: x_m *: approximate solution to the LS problem (3.1),*

$r_m = b - Ax_m \in \mathbb{R}^\ell$ *: residual vector.*

1. Set $w_1 = r_0$ *and* $k = 0$.

2. Call Algorithm 3.3 to obtain m*-step GK bidiagonalization*

$$A^T W_{m+1} = P_m B_{m+1,m}^T + \alpha_{m+1} p_{m+1} e_{m+1}^T$$

$$AP_m = W_{m+1} B_{m+1,m}$$

and solution x_m *and residual* r_m.

3. If $\|A^T r_m\| / \|A^T r_0\| < \delta$ *then exit.*

4. Compute the m *harmonic Ritz values, (3.11).*

5. Adjust the number of shifts p *using user input* j *and (3.28).*

6. Apply the largest p *harmonic Ritz values as shifts to obtain the* *k-step GK bidiagonalization (3.24),*

$$A^T W_{k+1}^+ = P_k^+ B_{k+1,k}^{+T} + \alpha_{k+1}^+ p_{k+1}^+ e_{k+1}^T$$
$$AP_k^+ = W_{k+1}^+ B_{k+1,k}^+$$

7. Set $x_k = x_m$ and $r_k = r_m$ and goto 2.

We remark that computation of $\|A^T r_m\|/\|A^T r_0\|$ in line 3 can be done efficiently using the formula in [32]. The applications of the implicit shifts of the harmonic Ritz values in Step 6 of Algorithm 3.4 with the buglehasing algorithm 3.2 does not always yield the required structure of $B_{m+1,m}^+$, i.e. $\alpha_m^+ = 0$. For small values of m we do get $\alpha_m^+ \approx 0$, however for modest values, $m \approx 100$, we get $\alpha_m^+ \neq 0$. Therefore, we developed an alternate method for applying the shifts that is discussed in the next section.

3.4 Harmonic bidiagonal method

The bulgechasing algorithm applies a shift implicitly to the bidiagonal matrix $B_{m+1,m}$ while outputting two orthogonal upper Hessenberg matrices Q_R and Q_L, such that $B_{m+1,m}^+ = Q_L^T B_{m+1,m} Q_R$. For the success of our method we need the output matrices Q_R and Q_L to be upper Hessenberg with the last columns as singular vectors and α_m^+ of $B_{m+1,m}^+$ cf. (3.15, 3.18, 3.19, 3.22) zero. However, in finite precision arithmetic Algorithm 3.2 (and the upper bidiagonal form of the algorithm) is prone to round off errors and the diagonal element α_m^+ of $B_{m+1,m}^+$ is not always zero, cf. Table 4.1 and [33] for a discussion. If the diagonal entry α_m^+ of $B_{m+1,m}^+$ is nonzero, then by Theorem 3.1 we did not shift by a harmonic Ritz value $\tilde{\sigma}_j^2$ and hence, $r_m \notin \mathcal{R}\left(W_{k+1}^+\right)$ cf. the discussion in Section 3.3.

Other implicitly restarted methods [34, 32, 16, 17, 18], that apply shifts implicitly can overcome the issue of a nonzero α_m^+ by incorporating α_m^+ into equation (3.6). This strategy does not work in our method. Alternatively, the bulgechasing algorithm 3.2 can be called repeatedly with the same shift until α_m^+ becomes small. This process destroys the required upper Hessenberg structure of Q_R and Q_L and often requires many calls for a single shift. To overcome this difficulty and force the desired (i.e. required) structure for this paper, we developed a method, Algorithm 3.5, for implicitly applying the harmonic Ritz values as shifts that utilizes the singular values and vectors of $B_{m+1,m}$.

Algorithm 3.5 takes advantage of the known structure of the orthogonal matrices Q_L and Q_R. That is, in exact arithmetic, the application of $p = m - k$ harmonic Ritz values $\left(\tilde{\sigma}_m^2, \ldots, \tilde{\sigma}_{k+1}^2\right)$ with Algorithm 3.2 yields banded upper Hessenberg matrices (3.22)-(3.23),

$$\begin{aligned} Q_L &= \left[Q_{L_{k+1}}, \tilde{u}_{k+1}, \ldots, \tilde{u}_m\right], \\ Q_R &= \left[Q_{R_k}, \tilde{v}_{k+1}, \ldots, \tilde{v}_m\right] \end{aligned} \tag{3.29}$$

with p sub-diagonals below the diagonal. The first vector, $q_{L_1} \in Q_{L_{k+1}}$ has, at most, the first $p+1$ entries nonzero and is orthogonal to the p vectors $\{\tilde{u}_{k+1}, \ldots, \tilde{u}_m\}$. The vector q_{L_1} can be easily constructed by finding a vector of length $p+1$ that is orthogonal to the first $p+1$ entries of each vector in $\{\tilde{u}_{k+1}, \ldots, \tilde{u}_m\}$ and replacing the first $p+1$ entries of q_{L_1} with that vector. The process can be repeated to find the second vector $q_{L_2} \in Q_{L_{k+1}}$, by finding a vector of length $p+2$ that is orthogonal to first $p+2$ entries of each vector in $\{q_{L_1}, \tilde{u}_{k+1}, \ldots, \tilde{u}_m\}$ and replacing the first $p+2$ entries of q_{L_2} with that vector. The matrix Q_R is constructed in the same manner. The matrices Q_L and Q_R are constructed to be orthogonal banded upper Hessenberg matrices, and $B_{m+1,m}^+ = Q_L^T B_{m+1,m} Q_R$ will have each $\alpha_i^+ = 0$ for $i = k+1$ to m. However, the lower bidiagonal structure of $B_{m+1,m}^+$ may be compromised. It may happen that for some (or many) values of i, the first $p+i$ entries of the columns of $[\tilde{u}_{k+1}, \ldots, \tilde{u}_m]$ (and $[\tilde{v}_{k+1}, \ldots, \tilde{v}_m]$) may form a rank deficient matrix, and hence steps 3 and 6 of Algorithm 3.5 may return

Table 3.1. The numerical value of $|\alpha_m^+|$ from $B_{m+1,m}^+$ after computing an m-step GK bidiagonalization (3.5) for the matrix [35] ILLC1850 $\in \mathbb{R}^{1850\times 712}$ and calling Algorithms 3.2 and 3.5 to apply the largest harmonic Ritz value as a shift. The computation time is not reported since it is considered negligible in the overall method.

m	Method of Implicit Shift	$\|\alpha_m^+\|$	$\\|q_{L_{m+1}} - u_m\\|$	$\\|q_{R_m} - v_m\\|$
20	Algorithm 3.2	2.3e-11	2.2e-11	2.6e-11
20	Algorithm 3.5	1.1e-19	0	0
40	Algorithm 3.2	7.7e-10	1.1e-4	6.4e-5
40	Algorithm 3.5	3.6e-17	0	0
80	Algorithm 3.2	1.41	1.52	1.61
80	Algorithm 3.5	2.7e-17	0	0
120	Algorithm 3.2	1.1e-6	1.4	1.4
120	Algorithm 3.5	1.8e-16	0	0

multiple vectors that satisfy the above criteria. The matrices Q_L, Q_R, and $B_{m+1,m}^+$, however, will have the required structure for our method and since Q_L and Q_R are orthogonal transformations, the singular values of the updated $B_{m+1,m}^+$ (not necessarily bidiagonal) matrix obtained from Algorithm 3.5 will be the same as the bidiagonal matrix which would have been obtained from Algorithm 3.2.

After using Algorithm 3.5 to apply the shifts we have the following k-step factorization

$$\begin{aligned} A^T W_{k+1}^+ &= P_k^+ B_{k+1,k}^{+T} + \alpha_{k+1}^+ p_{k+1}^+ e_{k+1}^T \\ A P_k^+ &= W_{k+1}^+ B_{k+1,k}^+ \end{aligned} \tag{3.30}$$

which is similar to (3.24) except that $B_{k+1,k}^+$ may not be lower bidiagonal. Algorithm 3.4 can be successfully used with equation (3.30) by applying the shifts in step 6 of Algorithm 3.4 via Algorithm 3.5.

The k-step GK bidiagonalization decomposition (3.24) can be recaptured by returning $B_{k+1,k}^+$ to lower bidiagonal form via orthogonal transformations with a rowwise Householder method starting with the last row, see e.g. [36, 37]. Using rowwise Householder transformations starting with the last row, creates orthogonal matrices $\breve{Q}_L \in \mathbb{R}^{(k+1)\times(k+1)}$ and $\breve{Q}_R \in \mathbb{R}^{k\times k}$ such that $\breve{B}_{k+1,k} = \breve{Q}_L^T B_{k+1,k}^+ \breve{Q}_R$ is lower bidiagonal where

$$\breve{Q}_L = \begin{bmatrix} \star & 0 \\ 0 & 1 \end{bmatrix}.$$

Letting $\breve{P}_k = P_k^+ \breve{Q}_R$ and $\breve{W}_{k+1} = W_{k+1}^+ \breve{Q}_L$, we can recover a k-step GK bidiagonalization decomposition

$$\begin{aligned} A^T \breve{W}_{k+1} &= \breve{P}_k \breve{B}_{k+1,k}^T + \alpha_{k+1}^+ p_{k+1}^+ e_{k+1}^T \\ A\breve{P}_k &= \breve{W}_{k+1} \breve{B}_{k+1,k} \end{aligned} \tag{3.31}$$

where $\breve{B}_{k+1,k}$ is lower bidiagonal. MATLAB code `irlsqr` used for numerical examples in Section 6 which implements Algorithm 3.4 can be used with either structure (3.30) and (3.31). The authors notice no numerical differences.

Algorithm 3.5. Harmonic bidiagonal method

```
Input: $[\tilde{u}_{k+1}, \tilde{u}_{k+2}, \ldots, \tilde{u}_m] \in \mathbb{R}^{(m+1)\times p}$ : left singular vectors of $B_{m+1,m}$ (3.10),
       $[\tilde{v}_{k+1}, \tilde{v}_{k+2}, \ldots, \tilde{v}_m] \in \mathbb{R}^{m\times p}$ : right singular vectors of $B_{m+1,m}$ (3.10).

Output: $Q_L \in \mathbb{R}^{(m+1)\times(m+1)}$ : banded orthogonal upper Hessenberg,
        $Q_R \in \mathbb{R}^{m\times m}$ : banded orthogonal upper Hessenberg,
        $B_{m+1,m}^+ = Q_L^T B_{m+1,m} Q_R \in \mathbb{R}^{(m+1)\times m}$ : updated matrix.

1. Set $Q_{L_{k+1}} := [\ ]$ and $Q_{R_{k+1}} := [\ ]$
2. for $i = 1 : k+1$
    3. Find a vector $q_{L_i} \in \mathbb{R}^{p+i}$ orthogonal to the first $p+i$ rows of each
       column of $\left[Q_{L_{k+1}}, \tilde{u}_{k+1}, \tilde{u}_{k+2}, \ldots, \tilde{u}_m\right]$.
    4. Set $Q_{L_{k+1}} := \left[ Q_{L_{k+1}}, \begin{bmatrix} q_{L_i} \\ 0 \end{bmatrix} \right]$
    5. if $i \leq k$
            6. Find a vector $q_{R_i} \in \mathbb{R}^{p+i}$ orthogonal to the first $p+i$ rows of each
               column of $\left[ Q_{R_{k+1}}, \tilde{v}_{k+1}, \tilde{v}_{k+2}, \ldots, \tilde{v}_m \right]$.
            7. Set $Q_{R_{k+1}} := \left[ Q_{R_{k+1}}, \begin{bmatrix} q_{R_i} \\ 0 \end{bmatrix} \right]$
    8. endif
9. endfor
10. Set $B^+_{m+1,m} = Q_L^T B_{m+1,m} Q_R$.
```

Steps 3 and 6 of Algorithm 3.5 can be done several ways, e.g. the MATLAB command `null` applied to the transpose of the first $p+i$ rows of $\left[Q_{L_{k+1}}, \tilde{u}_{k+1}, \tilde{u}_{k+2}, \ldots, \tilde{u}_m \right]$.

3.5 Connection to augmented LSQR

This section shows the parallels between the augmented LSQR algorithm described in [1] and the implicitly restarted LSQR algorithm described in this paper. Both algorithms use a restarted GK bidiagonalization in conjunction with LSQR to solve the LS problem. The augmented LSQR algorithm of [1] is carried out by explicitly augmenting the Krylov subspaces (3.2) with the harmonic Ritz vectors associated with the smallest harmonic Ritz values. We briefly describe the spaces that result from the augmenting routine and refer the reader to [1] for the full details.

The harmonic Ritz vector of AA^T associated with the harmonic Ritz value $\hat{\theta}_j$ is defined as

$$\hat{u}_j = W_m g_j \tag{3.32}$$

where g_j is the corresponding eigenvector from equation (3.9). Furthermore, it was shown in [1] that the eigenvector g_j can also be expressed as,

$$g_j = \begin{bmatrix} I_m & \beta_{m+1} B_{m,m}^{-T} e_m \end{bmatrix} \tilde{u}_j \tag{3.33}$$

where $\tilde{u}_j$ is the corresponding left singular vector associated with the singular value $\tilde{\sigma}_j$ from (3.10). Similar to our method for the initial iteration, the augmenting method in [1] sets $w_1 = r_0$ and calls Algorithm 3.2 to obtain the m-step GK bidiagonalization (3.3). The augmenting restarting step of [1] creates a variant of the equations (3.24),

$$\begin{aligned} A^T \hat{W}_{k+1} &= \hat{P}_k \hat{B}_{k+1,k}^T + (\alpha_{m+1} \hat{q}_{m+1,k+1})\, p_{m+1} e_{m+1}^T \\ A \hat{P}_k &= \hat{W}_{k+1} \hat{B}_{k+1,k} \end{aligned} \tag{3.34}$$

where $\hat{W}_{k+1} = W_{m+1}\hat{Q}$, $\hat{P}_k = P_m \tilde{V}_k$, $\hat{B}_{k+1,k} = \hat{Q}^T \tilde{U}_k \tilde{\Sigma}_k$ and $\hat{q}_{m+1,k+1}$ is the $(m+1, k+1)$ element of $\hat{Q}$. The matrices $\tilde{U}_k$ and $\tilde{V}_k$ are the left and right singular vectors of $B_{m+1,m}$ respectively, associated with the k smallest singular values, and $\tilde{\Sigma}_k$ is the diagonal matrix of the k smallest singular values. The matrix $\hat{Q}$ is taken from the QR decomposition of

$$\hat{Q}\hat{R} = \left[\begin{array}{c|c} \begin{bmatrix} \tilde{u}_{m+1_{m+1}} I_m & -\tilde{u}_{m+1_{1:m}} \end{bmatrix} \tilde{U}_k & \tilde{u}_{m+1} \\ \hline 0 & \end{array} \right] \tag{3.35}$$

where $\tilde{u}_{m+1_{m+1}} \in \mathbb{R}$ is the $m+1$ element of the null vector $\tilde{u}_{m+1}$ and $\tilde{u}_{m+1_{1:m}} \in \mathbb{R}^m$ is the first m elements of the null vector $\tilde{u}_{m+1}$. The matrix on the right side of (3.35) is considered to be full rank and hence $\hat{R}$ is invertible. We will show that $\mathcal{R}(W_{k+1}^+) = \mathcal{R}(\hat{W}_{k+1})$ and $\mathcal{R}(P_k^+) = \mathcal{R}(\hat{P}_k)$.

Theorem 3.2. *Let $w_1 = r_0$ and call Algorithm 3.2 to obtain the m-step GK bidiagonalization (3.3). Then the matrices W_{k+1}^+ and P_k^+ of (3.24) that are created by applying the $p = m - k$ largest harmonic Ritz values as shifts, spans respectively, the same spaces as the matrices $\hat{W}_{k+1}$ and $\hat{P}_k$ of (3.34), i.e. $\mathcal{R}(W_{k+1}^+) = \mathcal{R}(\hat{W}_{k+1})$, and $\mathcal{R}(P_k^+) = \mathcal{R}(\hat{P}_k)$.*

Proof. Using the formulas of Section 2 to apply the largest $p = m - k$ harmonic Ritz value as shifts generates the orthogonal matrices $Q_R = [Q_{R_k}, \tilde{v}_{k+1}, \ldots, \tilde{v}_m]$ and $Q_L = \left[Q_{L_{k+1}}, \tilde{u}_{k+1}, \ldots, \tilde{u}_m\right]$, cf. (3.23). Since $\mathcal{R}(Q_{R_k}) = \mathcal{R}(\tilde{V}_k)$, $P_k^+ = P_m Q_{R_k}$ and $\hat{P}_k = P_m \tilde{V}_k$, we have

$$\mathcal{R}(P_k^+) = \mathcal{R}(\hat{P}_k).$$

Define $\tilde{U}_{k+1:m} = [\tilde{u}_{k+1}, \ldots, \tilde{u}_m]$ and notice that $\tilde{U}_{k+1:m}^T Q_{L_{k+1}} = 0$ and

$$\tilde{U}_{k+1:m}^T \left[\begin{array}{c|c} \left[\tilde{u}_{m+1_{m+1}} I_m \quad - \tilde{u}_{m+1_{1:m}}\right] \tilde{U}_k & \tilde{u}_{m+1} \\ \hline 0 & \end{array} \right] = 0.$$

Since the matrices of (3.35) are full rank we have $\mathcal{N}(\hat{Q}^T) = \mathcal{N}(Q_{L_{k+1}}^T)$ and $\mathcal{R}(Q_{L_{k+1}}) = \mathcal{R}(\hat{Q})$. Since $W_{k+1}^+ = W_{m+1} Q_{L_{k+1}}$, and $\hat{W}_{k+1} = W_{m+1}\hat{Q}$, we have $\mathcal{R}(W_{k+1}^+) = \mathcal{R}(\hat{W}_{k+1})$. $\square$

Section 3.3 showed the residual r_m of LSQR is in the restarted space W_{k+1}^+ when implicit restarting is applied with the largest p harmonic Ritz values as shifts, and it is in $\hat{W}_{k+1}$ by construction (cf. [1, equation 3.9]). Furthermore, the restart vector p_{k+1}^+ (3.25) of the implicitly restarted method is a multiple of p_{m+1} and extending the k-step GK bidiagonalization methods (3.24) and (3.34) back to m-step GK bidiagonalization will produce $\mathcal{R}(W_{m+1}^+) = \mathcal{R}(\hat{W}_{m+1})$ and $\mathcal{R}(P_m^+) = \mathcal{R}(\hat{P}_m)$. The process is then repeated.

3.6 Numerical examples

In this section we have some numerical examples to show the performance of Algorithm 3.4 which is implemented by the Matlab code `irlsqr`[3]. The code uses the following user-specified parameters:

[3]Code is available at in Appendix B.

tol	Tolerance for accepting a computed approximate solution x as a solution to the LS problem (3.1), i.e. $\|A^T r\|/\|A^T r_0\| \leq$ *tol*.
m	Maximum number of GK bidiagonal steps.
p	Number of shifts to apply.
j	Size of interval around $\hat{\theta}_{k+1}$ to find largest gap.
maxit	Maximum number of restarts.
reorth12	String deciding whether one or two sided reorthogonalization is used in Algorithm 3.1.

We compare `irlsqr` against the `lsqr` and `lsmr` algorithms. The latter codes were adapted to output the norm of the residuals after each bidiagonal step. Furthermore, LSQR was adapted to perform either one or two sided reorthogonalization against any specified number of previous GK vectors. In order to make a fair comparison between the three methods and to keep storage requirements the same with the GK vectors, LSQR and LSMR will reorthogonalize against only the previous m vectors. No comparisons were made with ALSQR of [1], since these routines are mathematically equivalent, see Section 3.5.

All computations were carried out using MATLAB version 7.12.0.0635 R2011a on a Dell XPS workstation with an Intel Core2 Quad processor and 4 GB of memory running under the Windows Vista operating system. Machine precision is $2.2 \cdot 10^{-16}$. We present numerical examples with matrices taken from the University of Florida Sparse Matrix Collection [38] and from the Matrix Market Collection [39], see Table 6.1. All examples use the initial guess solution as $x_0 = 0$ and $r_0 = b$.

In Table 6.2, we used two matrices ILLC1850 and E30R0000 from the Matrix Market Collection along with the accompanied b vectors ILLC1850_RHS1 and

Table 3.2. List of matrices A, properties, and b vectors used in numerical examples. The first two matrices are taken from the Matrix Market Collection and the last two are from the University of Florida Sparse Matrix Collection.

Example	A	ℓ	n	nnz	b
3.1	ILLC1850	1850	712	8638	ILLC1850_RHS1
3.2	E30R0000	9661	9661	305794	E30R0000_RHS1
3.3	LANDMARK	71952	2704	1146848	`rand(71952,1)`
3.4	BIG_DUAL	30269	30269	89858	A·`rand(30269,1)`

Table 3.3. Number of matrix vector products with A and A^T required to get $\|A^T r\|/\|A^T r_0\| \leq 10^{-12}$ using different values of j in the gap strategy given in Section 3.2. The table shows two different choices of p (number of shifts) and sets $m = 100$ for the examples with the matrix ILLC1850 and $m = 200$ for the examples with the matrix E30R0000. Column $j = 0$ corresponds to no adjustments.

A	p	$j = 0$	$j = 3$	$j = 6$	$j = 9$
ILLC1850	20	3825	3647	3630	3657
ILLC1850	30	3750	3689	3681	3679
E30R0000	30	31753	30953	30153	42223
E30R0000	40	34731	30723	31037	31223

E30R0000_RHS1, respectively to show the results for various values j for the gap strategy outlined in Section 3.2.3. For all of the numerical examples, we set j equal to 5.

Example 3.1. We let the matrix A and vector b be ILLC1850 and ILLC1850_RHS1, respectively. This matrix and b vector are from the LSQ group of the matrix market which consist of LS problems in surveying. The use of the matrix is to test iterative solvers, and was one of the matrices used by Paige and Saunders [2] in the testing of the LSQR algorithm. For this example, $b \notin \mathcal{R}(A)$, and therefore we only show convergence of the quotient $\|A^T r\|/\|A^T r_0\|$. `irlsqr` is used with parameters $m = 100$, $p = 30$, and $reorth12 = 1$. We set $tol = 1 \cdot 10^{-12}$. Figure 6.1 plots $\|A^T r\|/\|A^T r_0\|$ vs. the number of matrix-vector products with A

and A^T.

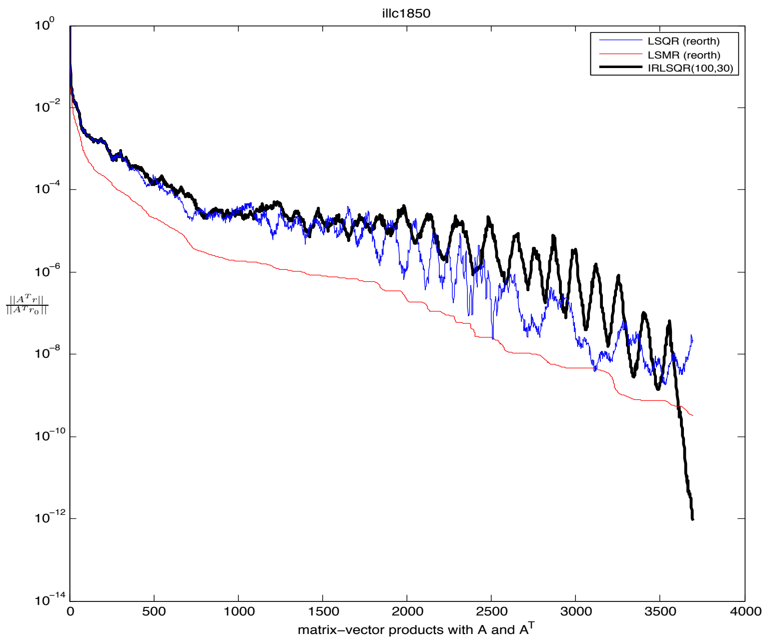
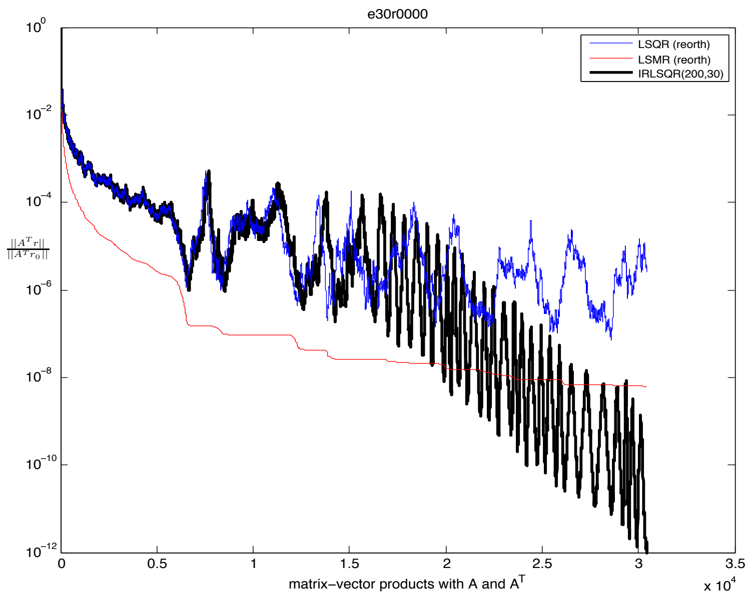


Figure 3.1. Example 3.1: A = ILLC1850, b = ILLC1850_RHS1. IRLSQR(100,30) indicates $m = 100$ and $p = 30$. LSMR (reorth) and LSQR (reorth) indicate reorthogonalization is carried out back m vectors. IRLSQR(100,30) converged at 3,693 matrix-vector products.

Example 3.2. The matrix A and vector b are chosen to be E30R0000 and E30R0000_RHS1, respectively. The matrix and vector b are from the DRICAV group of matrices from the matrix market collection. The group consists of matrices used from modeling 2D fluid flow in a driven cavity, and the main purpose of matrices from this collection is for testing iterative solvers. The matrix is nonsymmetric and indefinite. Since the matrix A is square and full rank, $b \in \mathcal{R}(A)$ and therefore we show convergence of the quotients $\|A^T r\|/\|A^T r_0\|$ and $\|r\|/\|r_0\|$, see Figure 6.2. `irlsqr` is used with parameters $m = 200$, $p = 30$, and $reorth12 = 2$. We used two-sided reorthogonalization since the condition number of this matrix is approximately $3.47 \cdot 10^{11}$. We set $tol = 1 \cdot 10^{-12}$ and accept an iterate x as a solution to the LS problem if $\|A^T r\|/\|A^T r_0\| < 1 \cdot 10^{-12}$.



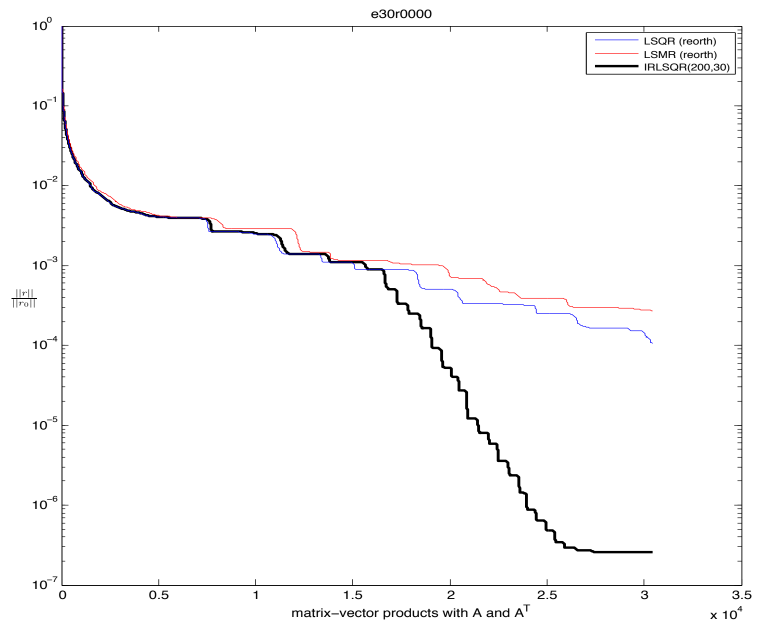


Figure 3.2. Example 3.2: $A =$ E30R0000, $b =$ E30R0000_RHS1. IRLSQR(200,30) indicates $m = 200$ and $p = 30$. LSMR (reorth) and LSQR (reorth) indicate reorthogonalization is carried out back 200 vectors. The *top graph* shows the convergence of $\|A^T r\|/\|A^T r_0\|$ and the *bottom graph* shows the convergence of $\|r\|/\|r_0\|$. IRLSQR(200,30) converged at 30,421 matrix-vector products.

Example 3.3. The matrix A is chosen to be LANDMARK of the Pereyra group from the University of Florida Sparse Matrix Collection. It comes from a LS problem. The matrix LANDMARK does not have a corresponding b vector, hence we chose it to be random with the MATLAB command `rand(71952,1)`. The rank of the matrix A is 2671 and we do not assume $b \in \mathcal{R}(A)$, therefore we only show convergence of the quotient $\|A^T r\|/\|A^T r_0\|$, see Figure 6.3. `irlsqr` is used with parameters $m = 250$, $p = 35$, and $reorth12 = 1$. Setting $tol = 1 \cdot 10^{-10}$, an iterate x is accepted as a solution to the LS problem if $\|A^T r\|/\|A^T r_0\| < 1 \cdot 10^{-10}$.

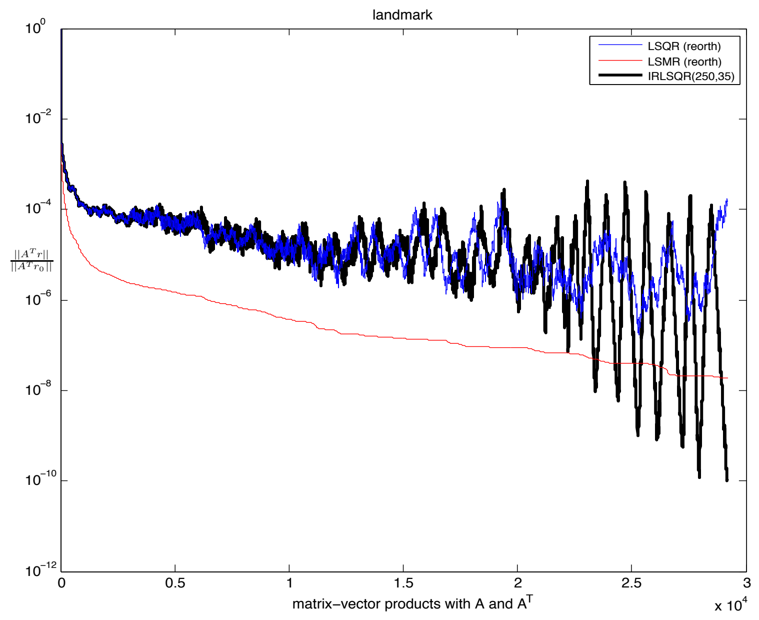


Figure 3.3. Example 3.3: A = LANDMARK and b = `rand(71952,1)`. IRLSQR(250,35) indicates $m = 250$ and $p = 35$. LSMR (reorth) and LSQR (reorth) indicate reorthogonalization is carried out back 250 vectors. The graph shows the convergence of $\|A^T r\|/\|A^T r_0\|$. IRLSQR(250,35) converged at 29,185 matrix-vector products.

Example 3.4. The matrix A is chosen to be BIG_DUAL of the AG-Monien group from the University of Florida Sparse Matrix Collection. The matrix is from a 2D finite element problem. The matrix BIG_DUAL does not have a corresponding b vector. The rank of the matrix is 30,239 (not full rank), we choose the vector b to be $A\cdot$`rand(30269,1)` so that $b \in \mathcal{R}(A)$. We plot the quotients $\|A^T r\|/\|A^T r_0\|$ and $\|r\|/\|r_0\|$, see Figure 6.4. `irlsqr` is used with parameters $m = 300$, $p = 45$, and $reorth12 = 1$. Setting $tol = 1 \cdot 10^{-14}$, an iterate x is accepted as a solution to the LS problem if $\|A^T r\|/\|A^T r_0\| < 1 \cdot 10^{-14}$.

3.7 Conclusion

We have presented a new implicitly restarted LSQR algorithm for the solving the LS problem. Theoretical results show the restarting to be equivalent to the augmented LSQR algorithm of [1], however, much simpler to implement. The gap strategy and ease of implementation of this method make it desirable. Numerical examples show the method is competitive with pre-existing methods.

List of References

[1] J. Baglama, L. Reichel, and D. Richmond, "An augmented LSQR method," *Numerical Algorithms*, vol. 64, no. 2, pp. 263–293, 2013.

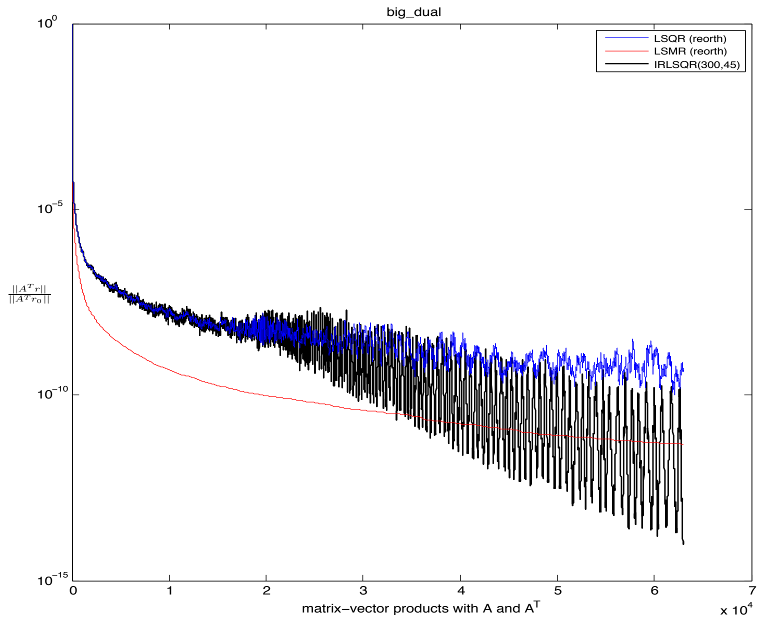
[2] C. C. Paige and M. A. Saunders, "LSQR: An algorithm for sparse linear equations and sparse least squares," *ACM Transactions on Mathematical Software*, vol. 8, no. 1, pp. 43–71, 1982.

[3] Å. Björck, *Numerical methods for least squares problems.* Siam, 1996.

[4] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on scientific and statistical computing*, vol. 7, no. 3, pp. 856–869, 1986.

[5] A. H. Baker, E. R. Jessup, and T. Manteuffel, "A technique for accelerating the convergence of restarted GMRES," *SIAM Journal on Matrix Analysis and Applications*, vol. 26, no. 4, pp. 962–984, 2005.

[6] I. Zavorin, D. P. O'Leary, and H. Elman, "Complete stagnation of GMRES," *Linear Algebra and its Applications*, vol. 367, pp. 165–183, 2003.



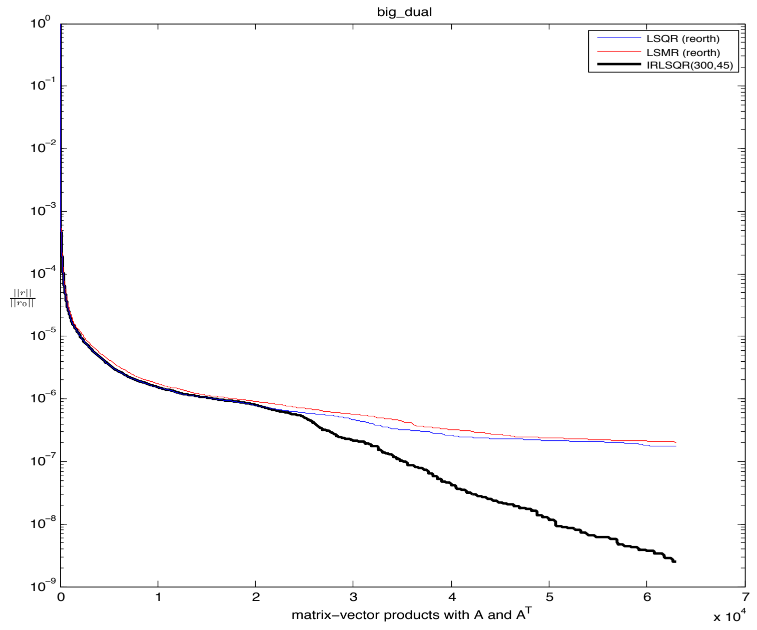


Figure 3.4. Example 3.4: A = BIG_DUAL and $b = A\cdot$ `rand(30269,1)`. IRLSQR(300,45) indicates $m = 300$ and $p = 45$. LSMR (reorth) and LSQR (reorth) indicate reorthogonalization is carried out back 300 vectors. The *top graph* shows the convergence of $\|A^T r\|/\|A^T r_0\|$ and the *bottom graph* shows the convergence of $\|r\|/\|r_0\|$. IRLSQR(300,45) converged at 62,961 matrix-vector products.

[7] J. Baglama, D. Calvetti, G. H. Golub, and L. Reichel, “Adaptively precondi-

tioned GMRES algorithms," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 243–269, 1998.

[8] M. Benzi, "Preconditioning techniques for large linear systems: a survey," *Journal of Computational Physics*, vol. 182, no. 2, pp. 418–477, 2002.

[9] T. Ito and K. Hayami, "Preconditioned GMRES methods for least squares problems," NII Technical Report, NII-2004-006E, National Institute of Informatics, Tokyo, Japan, Tech. Rep., 2004.

[10] R. B. Morgan, "A restarted GMRES method augmented with eigenvectors," *SIAM Journal on Matrix Analysis and Applications*, vol. 16, no. 4, pp. 1154–1171, 1995.

[11] R. B. Morgan, "Implicitly restarted GMRES and Arnoldi methods for nonsymmetric systems of equations," *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1112–1135, 2000.

[12] Y. Saad, *Iterative methods for sparse linear systems.* SIAM, 2003.

[13] D. C.-L. Fong and M. Saunders, "LSMR: An iterative algorithm for sparse least-squares problems," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2950–2971, 2011.

[14] J. Baglama and L. Reichel, "Augmented implicitly restarted Lanczos bidiagonalization methods," *SIAM Journal on Scientific Computing*, vol. 27, no. 1, pp. 19–42, 2005.

[15] J. Baglama and L. Reichel, "Restarted block Lanczos bidiagonalization methods," *Numerical Algorithms*, vol. 43, no. 3, pp. 251–272, 2006.

[16] Z. Jia and D. Niu, "An implicitly restarted refined bidiagonalization Lanczos method for computing a partial singular value decomposition," *SIAM journal on matrix analysis and applications*, vol. 25, no. 1, pp. 246–265, 2003.

[17] Z. Jia and D. Niu, "A refined harmonic Lanczos bidiagonalization method and an implicitly restarted algorithm for computing the smallest singular triplets of large matrices," *SIAM Journal on Scientific Computing*, vol. 32, no. 2, pp. 714–744, 2010.

[18] E. Kokiopoulou, C. Bekas, and E. Gallopoulos, "Computing smallest singular triplets with implicitly restarted Lanczos bidiagonalization," *Applied numerical mathematics*, vol. 49, no. 1, pp. 39–61, 2004.

[19] M. Benzi and M. Tuma, "A robust preconditioner with low memory requirements for large sparse least squares problems," *SIAM Journal on Scientific Computing*, vol. 25, no. 2, pp. 499–512, 2003.

[20] Å. Björck and J. Yuan, "Preconditioners for least squares problems by LU factorization," *Electronic Transactions on Numerical Analysis*, vol. 8, pp. 26–35, 1999.

[21] S. Karimi, D. K. Salkuyeh, and F. Toutounian, "A preconditioner for the LSQR algorithm," *Journal of Applied Mathematics and Informatics*, vol. 26, no. 1-2, pp. 213–222, 2008.

[22] G. Golub and W. Kahan, "Calculating the singular values and pseudo-inverse of a matrix," *Journal of the Society for Industrial & Applied Mathematics, Series B: Numerical Analysis*, vol. 2, no. 2, pp. 205–224, 1965.

[23] Å. Björck, E. Grimme, and P. Van Dooren, "An implicit shift bidiagonalization algorithm for ill-posed systems," *BIT Numerical Mathematics*, vol. 34, no. 4, pp. 510–534, 1994.

[24] R. M. Larsen, "Lanczos bidiagonalization with partial reorthogonalization," *DAIMI Report Series*, vol. 27, no. 537, 1998.

[25] H. D. Simon and H. Zha, "Low-rank matrix approximation using the Lanczos bidiagonalization process with applications," *SIAM Journal on Scientific Computing*, vol. 21, no. 6, pp. 2257–2274, 2000.

[26] G. H. Golub and C. F. Van Loan, *Matrix computations.* Johns Hopkins University Press, 2012, vol. 3.

[27] R. B. Morgan, "Computing interior eigenvalues of large matrices," *Linear Algebra and its Applications*, vol. 154, pp. 289–309, 1991.

[28] C. C. Paige, B. N. Parlett, and H. A. van der Vorst, "Approximate solutions and eigenvalue bounds from Krylov subspaces," *Numerical linear algebra with applications*, vol. 2, no. 2, pp. 115–133, 1995.

[29] B. N. Parlett, *The symmetric eigenvalue problem.* SIAM, 1980, vol. 7.

[30] R. M. Larsen, "Combining implicit restarts and partial reorthogonalization in Lanczos bidiagonalization," *Program in Scientific Computing and Computational Mathematics, Stanford University*, 2001.

[31] M. E. Hochstenbach, "Harmonic and refined extraction methods for the singular value problem, with applications in least squares problems," *BIT Numerical Mathematics*, vol. 44, no. 4, pp. 721–754, 2004.

[32] Z. Jia, "Some properties of LSQR for large sparse linear least squares problems," *Journal of Systems Science and Complexity*, vol. 23, no. 4, pp. 815–821, 2010.

[33] D. S. Watkins, *The matrix eigenvalue problem: GR and Krylov subspace methods*. Siam, 2007.

[34] J. Baglama and L. Reichel, "An implicitly restarted block Lanczos bidiagonalization method using Leja shifts," *BIT Numerical Mathematics*, vol. 53, no. 2, pp. 285–310, 2013.

[35] I. S. Duff, R. G. Grimes, and J. G. Lewis, "Users' guide for the Harwell-Boeing sparse matrix collection (Release I)," Report RAL-92-086, Atlas Centre, Rutherford Appleton Laboratory, Didcot, Oxon, UK, Tech. Rep., 1992.

[36] G. Stewart, "A Krylov–Schur algorithm for large eigenproblems," *SIAM Journal on Matrix Analysis and Applications*, vol. 23, no. 3, pp. 601–614, 2002.

[37] M. Stoll, "A Krylov–Schur approach to the truncated svd," *Linear Algebra and its Applications*, vol. 436, no. 8, pp. 2795–2806, 2012.

[38] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Transactions on Mathematical Software*, vol. 38, no. 1, p. 1, 2011.

[39] R. F. Boisvert, R. Pozo, K. A. Remington, R. F. Barrett, and J. Dongarra, "Matrix Market: a web resource for test matrix collections,," in *Quality of Numerical Software*, 1996, pp. 125–137.

CHAPTER 4

Conclusions

Results of this dissertation have increased the existing body of knowledge of algorithms for obtaining fast and accurate solutions to large-scale least-squares problems. This dissertation puts an emphasis on solvers which provide fast and accurate solutions to ill-conditioned least-squares problems, which are the most difficult class of least-squares problems to solve. Furthermore, the solvers proposed remain effective in providing solutions to well-conditioned least-squares problems.

The augmented LSQR method of Chapter 2 uses initial iterations to compute harmonic Ritz vector approximations to the singular vectors that are associated with the smallest singular values. These vectors are used for augmenting the Krylov subspaces while simultaneously computing improved solutions to the least-squares problem. Once sufficient harmonic Ritz vector approximations are found, the vectors are kept fixed and a nonrestarted LSQR method can be run on the augmented Krylov subspaces.

The implicitly restarted LSQR method of Chapter 3 is theoretically equivalent to the augmented LSQR method of Chapter 2, however, much simpler to implement. Some of the largest harmonic Ritz values are used as implicit shifts in a restarted Golub-Kahan bidiagonalization to improve the Krylov subspaces. The gap strategy proposed further improves the convergence rates, and the ease of implementation of this method make it desirable. Theoretical and computed results show these methods offer improved convergence rates over existing methods over a wide variety of test problems.

APPENDIX A

Speculative discussion

The augmented LSQR method of Chapter 2 and the implicitly restarted LSQR method of Chapter 3 may be viewed as a form of preconditioning. The preconditioning, however, is not done in the usual way where a preconditioning matrix is created and applied to the problem, but instead, as a type of *adaptive* preconditioning, where the use of augmenting vectors or implicit shifts are applied to the problem. It was shown in Theorem 3.2 that the methods of Chapter 2 and Chapter 3 are theoretically equivalent, and this appendix provides preliminary results on the existence of a preconditioner, that when applied in the standard way, produces comparable results to the augmented and implicitly restarted LSQR methods. This appendix is organized as follows: Section A.1 reviews how a right preconditioner can be applied in the context of LSQR, Section A.2 provides the details on the creation of a candidate preconditioner that is comparable to the methods of Chapter 2 and Chapter 3, and Section A.3 shows the comparison between the Krylov subspaces generated by a right preconditioned LSQR algorithm using the preconditioner of Section A.2 and the augmented LSQR method of Section 2.2. Notation from Chapter 2 will be used throughout this appendix.

A.1 Right preconditioned LSQR

Given an LS problem of the form (1.1) and an invertible matrix $M \in \mathbb{R}^{n\times n}$, the right preconditioned LS problem is constructed as

$$\min_{x\in\mathbb{R}^n} \left\| b - AM^{-1}(Mx) \right\| . \tag{A.1}$$

Using the LSQR algorithm to obtain the solution to a right preconditioned LS problem is straightforward. Choosing an initial guess solution, x_0, a starting vector

$$q_1 = b - AM^{-1}Mx_0 = b - Ax_0 = r_0$$

is computed, and applying Algorithm 2.1 with input matrix AM^{-1}, starting vector q_1 and number of bidiagonal steps m, the matrices $B_{m+1,m}, P_m$, and Q_{m+1} are output with corresponding GK bidiagonal equations

$$\begin{aligned} \left(AM^{-1}\right)^T Q_{m+1} &= P_m B_{m+1,m}^T + \alpha_m p_{m+1} e_{m+1}^T \\ \left(AM^{-1}\right) P_m &= Q_{m+1} B_{m+1,m}. \end{aligned} \tag{A.2}$$

The matrices Q_m and P_m form orthonormal bases for the Krylov subspaces

$$\begin{aligned} &\mathcal{K}_m \left(\left(AM^{-1}\right) \left(AM^{-1}\right)^T, q_1 \right) \\ &\mathcal{K}_m \left(\left(AM^{-1}\right)^T \left(AM^{-1}\right), p_1 \right), \end{aligned} \tag{A.3}$$

respectively. Using the LSQR formulas to obtain the solution to the right preconditioned LS problem, it follows that Mx_m is a linear combination of the columns of P_m, that is, $Mx_m = P_m y_m$ for $y_m \in \mathbb{R}^m$. The solution to the original LS problem, x_m, can be easily obtained as $x_m = M^{-1}P_m y_m$. Therefore, the right preconditioned LSQR algorithm is obtained by slightly modifying Algorithm 2.2, in which line 17 is changed to $x_{k+1} := x_k + M^{-1}P_{(1:k+1)}y$ and line 37 is changed to $x_j := x_{j-1} + (\phi_j / p_j)M^{-1}w$.

A.2 Creation of a suitable preconditioner

There are many approaches to creating preconditioners that can be applied to solve LS problems, see Section 2.2 and references therein for examples. The preconditioners that most closely resemble the results from this dissertation, and the ones that we use as a basis for our construction, are the ones proposed in [1] and [2]. In [1], a right preconditioner is created, which when applied to a square matrix A, moves the eigenvalues to improve convergence of the GMRES algorithm for solving linear systems. Similarly, in [2] a left preconditioner is created from spectral information gathered by the Arnoldi process in conjunction with restarting the GMRES algorithm for solving linear systems. The method worked to determine an invariant subspace of a square matrix A associated with eigenvalues close to the origin, and move them so that a higher rate of convergence could be achieved. In generalizing these preconditioners to apply to the LSQR algorithm for solving LS problems, we choose to apply a right preconditioner in order to guarantee a nonincreasing residual norm, [1, Proposition 2.1]. We give preliminary results on generalizing the works of [1] and [2] to create a right preconditoiner for using LSQR to solve LS problems that is comparable to the other methods of this dissertation.

Assume, similar to Section 2.2, that the SVD of the matrix A is given as

$$A = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} \tag{A.4}$$

where the singular values are ordered

$$0 < \sigma_1 \leq \sigma_2 \leq \cdots \leq \sigma_n.$$

Let (U_1, Σ_1, V_1) denote the singular triplets corresponding to the k smallest singular values. Furthermore, let A be scaled so that $\sigma_n = 1$. Define the matrix

$$M = V_1 \Sigma_1 V_1^T + I - V_1 V_1^T,$$

which has inverse

$$M^{-1} = V_1 \Sigma_1^{-1} V_1^T + I - V_1 V_1^T. \tag{A.5}$$

Applying M^{-1} as a right preconditioner to the matrix A, and using the facts that $AM^{-1} = U_1 V_1^T + A\left(I - V_1 V_1^T\right)$ and $V_1 V_1^T + V_2 V_2^T = I$, the SVD of the matrix AM^{-1} is obtained as

$$AM^{-1} = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & \Sigma_2 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix}. \tag{A.6}$$

From (A.6), it can be seen that the matrix AM^{-1} has the same left and right singular vectors as A, however, the k smallest singular values have been moved to 1, i.e.,

$$\sigma\left(AM^{-1}\right) = \{\sigma_{k+1}, \sigma_{k+2}, \ldots, \sigma_n, 1, 1, \ldots, 1\}.$$

The use of the preconditioner (A.5) effectively improves the condition number of the problem, that is,

$$\kappa\left(AM^{-1}\right) = \frac{\sigma_n}{\sigma_{k+1}} < \kappa(A) = \frac{\sigma_n}{\sigma_1}.$$

A bound for the residual norms when using right preconditioned LSQR with the input matrix A and preconditioner M^{-1} (A.5) can be computed from equation (2.5), that is, the residual $r_m = b - Ax_m$ associated with the mth iterate, x_m, determined by right preconditioned LSQR with initial approximate solution x_0 satisfies

$$\|r_m - r^+\| \leq 2\left(\frac{\sigma_n - \sigma_{k+1}}{\sigma_n + \sigma_{k+1}}\right)^m \|r_0 - r^+\|. \tag{A.7}$$

Using the preconditioner (A.5) will significantly reduce the bound on the residual norm. Similarly, when using the augmented Krylov subspaces from (2.6), where the Krylov subspaces are augmented with the k singular vectors corresponding to the smallest singular values, and applying m steps of LSQR, the bound on the residual norm is given by Theorem 2.1, and satisfies

$$\|r_m - r^+\| \leq 2\left(\frac{\sigma_n - \sigma_{k+1}}{\sigma_n + \sigma_{k+1}}\right)^m \|r_0 - r^+\|. \tag{A.8}$$

From equations (A.7) and (A.8), it is seen that applying m steps of right preconditioned LSQR with the preconditioner (A.5), and applying m steps of LSQR using the augmented Krylov subspaces of Section 2.2 offer the same improvement to the upper bound for the residual norm. This, however, does not prove that the methods are theoretically equivalent, as the actual norm of the residuals may be better than the bounds given in either case. Further inspection of the underlying Krylov subspaces are needed to draw any conclusions.

A.3 Comparsion of Krylov subspaces

We compare the solution that is computed by LSQR from the Krylov subspaces in (2.6) and (A.3). For simplicity in our comparison, assume $x_0 = 0$ is the initial guess and $q_1 = b$ is the starting vector in each case. Furthermore, we compare only the solutions that are obtained after a single iteration of LSQR, that is, we compare the Krylov subspaces obtained by setting $m = k - 1$ in (2.6) and $m = 1$ in (A.3).

We first investigate the solution obtained from the augmented Krylov subspaces of (2.6). Initially, the starting vector q_1 is orthogonalized against the k left singular vectors corresponding to the k smallest singular values, cf. Example 2.1, that is, set

$$q_1^{aug} = q_1 - (u_1^T q_1)u_1 - \cdots - (u_k^T q_1)u_k$$

The vector q_1^{aug} is then normalized. The vector p_1^{aug} is then computed,

$$p_1^{aug} = A^T q_1^{aug},$$

and also normalized. Using the SVD relations of A we can write

$$p_1^{aug} = A^T q_1 - (u_1^T q_1)\sigma_1 v_1 - \cdots - (u_k^T q_1)\sigma_k v_k.$$

The solution, x_m^{aug}, chosen by LSQR at this point will be a linear combination of the vectors $v_1, v_2, \ldots, v_k$ and p_1^{aug}. The following theorem will determine the weights given in the linear combination.

Theorem A.1. *Let $A \in \mathbb{R}^{\ell \times n}$ have the SVD (A.4) and let x_m minimize $\|b - Ax\|$ over the augmented Krylov subspace $\mathcal{K}_m\left(A^T A, v_1, \ldots, v_k, p_1^{aug}\right)$, then the weight corresponding to the vector v_i in the solution x_m with initial guess $x_0 = 0$, is $c_i = \frac{u_i^T b}{\sigma_i}$.*

Proof. x_m is chosen to minimize $\|r_m\|$, therefore x_m is chosen as close as possible to the pseudoinverse solution x^+, that is, x_m is chosen to minimize $\|x_m - x^+\|$. The pseudoinverse solution, x^+, to an LS problem is given by

$$x^+ = \sum_{i=1}^{n} \frac{u_i^T b}{\sigma_i} v_i. \tag{A.9}$$

Since p_1^{aug} is made orthogonal to $v_1, \ldots, v_k$, cf. Example 2.1, it is some linear combination of the vectors $v_{k+1}, \ldots, v_n$. The solution x_m can therefore be written as some linear combination $c_1 v_1 + c_2 v_2 + \cdots + c_k v_k + c_{k+1} p_1^{aug}$. Since the best possible solution chosen is the one which is closest to x^+, that is, in minimizing

$$\left\| \sum_{i=1}^{n} \frac{u_i^T b}{\sigma_i} v_i - (c_1 v_1 + c_2 v_2 + \cdots + c_k v_k + c_{k+1} p_1^{aug}) \right\|$$

it can be seen that the weights should be chosen to be $c_i = \frac{u_i b}{\sigma_i}$. $\square$

Using theorem A.1, noting that $q_1 = b$, the solution chosen from the augmented space will be

$$x_m^{aug} = \frac{u_1^T q_1}{\sigma_1} v_1 + \cdots + \frac{u_k^T q_1}{\sigma_k} v_k - c_{k+1} p_1^{aug}$$

where $c_{k+1} \in \mathbb{R}$.

We now investigate the solution chosen from the right preconditioned LSQR. Assuming the same initial solution $x_0 = 0$ is chosen, the starting vector is the same as when using the augment method, that is, $q_1 = b$. There are no previous vectors in the Krylov subspace to orthogonalize against, so set

$$q_1^{pre} = q_1$$

and normalize. The next vector, p_1^{pre}, is created as

$$p_1^{pre} = \left(AM^{-1}\right)^T q_1 = M^{-1} A^T q_1,$$

and normalized. Multiplying, using the SVD relations, and simplifying gives

$$p_1^{pre} = (u_1^T q_1)v_1 + \cdots + (u_k^T q_1)v_k - p_1^{aug}.$$

Since this is the only vector in the space, the solution Mx_m^{pre} is a linear combination of it, that is,

$$Mx_m^{pre} = c_1 \left((u_1^T q_1)v_1 + \cdots + (u_k^T q_1)v_k + p_1^{aug}\right)$$

for $c_1 \in \mathbb{R}$. The solution, x_m^{pre}, is obtained by multiplying both sides of the previous equation by M^{-1}. Using the SVD properties and simplifying we obtain

$$x_m^{pre} = c_1 \left(\frac{u_1^T q_1}{\sigma_1} v_1 + \cdots + \frac{u_k^T q_1}{\sigma_k} v_k + p_1^{aug}\right).$$

In comparing the two solutions, although similar, the augmentation case has a higher degree of freedom than the right preconditioned case. We conclude that augmenting the Krylov subspaces gives way to a more accurate solution and is superior than applying this preconditioner. This appendix is only a preliminary report, further comparisons or tweaking of the proposed preconditioned are required to make any final conclusions between the methods.

List of References

[1] J. Erhel, K. Burrage, and B. Pohl, "Restarted GMRES preconditioned by deflation," *Journal of computational and applied mathematics*, vol. 69, no. 2, pp. 303–318, 1996.

[2] J. Baglama, D. Calvetti, G. H. Golub, and L. Reichel, "Adaptively preconditioned GMRES algorithms," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 243–269, 1998.

APPENDIX B

MATLAB code

B.1 MATLAB function `alsqr.m`

```
function [x,flag,Arrnorm] = alsqr(A,b,x,OPTS)
% ALSQR is an augmented LSQR method. The augmenting vectors are
% harmonic Ritz vectors computed via a Golub-Kahan bidiagonalization
% method. The program then runs LSQR on the augmented spaces.
% The ALSQR method solves the least-squares problem:
%
%                   min || b - A*x ||
%                    x
%
% where A is an (m x n) matrix, b an (m x 1) vector, and x is
% an (n x 1) vector. b is not assumed to be in span(Col(A)).
% ||.|| always denotes the 2 norm.
%
%---------------------------------------------------------------------
%
% INPUT OPTIONS:
% --------------
%
% I.)   [x,FLAG,Arrnorm] = ALSQR(A,b,x)
%            The first input argument into ... = alsqr(A,b,x)
%            can be a numeric matrix A or an M-file 'Afunc'.
%            If the m x n matrix A is a filename then z = Afunc(x,1)
%            computes z = A*x and z = Afunc(x,2) computes z = A'*x.
%            The second input is the vector b and third input is the
%            vector x, an initial approximation to min ||b - A*x||,
%            typically chosen to be 0.
%
```

```
% II.)  [x,FLAG,Arrnorm] = ALSQR(A,b,x,OPTS)
%            OPTS is a structure containing input parameters.
%            The input parameters can be given in any order. The
%            structure OPTS may contain some or all of the following
%            input parameters. The string for the input parameters
%            can contain upper or lower case characters.
%            alsqr(A,b,x) without the structure OPTS will use all of
%            the default values for OPTS.
%
%
%  INPUT PARAMETER            DESCRIPTION
%
%  OPTS.ADJUST       Number of vectors to add to the k augmenting
%                    vectors.
%                    DEFAULT VALUE:  OPTS.ADJUST = 40
%
%  OPTS.HTOL         Tolerance used for convergence of harmonic Ritz
%                    vector approximations to the k smallest singular
%                    triplets in part I. Convergence is determined as
%                    sqrt(|| A'*U - V*S ||^2 + || A*V - U*S ||^2)
%                                                   <= HTOL*||A||
%                    where V is an orthonormal n x k matrix of
%                    "right" singular vectors, U is an orthonormal
%                    m x k matrix of "left" singular vectors, and S
%                    is a k x k diagonal matrix of singular values.
%                    DEFAULT VALUE:  HTOL = 1d-3
%
%  OPTS.K            Number of harmonic Ritz vectors to augment the
%                    Krylov subspace with. Setting K = 0 will call
%                    the nonrestarted LSQR method only.
%                    DEFAULT VALUE:  K = 20
%
```

```
% OPTS.LTOL         Tolerance used for stopping rule
%                   ||A'*r|| <= LTOL*||A'*r_0||
%                   DEFAULT VALUE:  LTOL = 1d-6
%
% OPTS.MAXITP       Maximum number of iterations in part I.
%                   DEFAULT VALUE:  MAXITP = 1000
%
% OPTS.MAXITL       Maximum number of iterations of nonrestarted
%                   LSQR, part II.
%                   DEFAULT VALUE:  MAXITL = min(m,n)
%
% OPTS.M_B          Size of the GK bidiagonal matrix. The
%                   program may increase M_B to ensure certain
%                   requirements are satisfied. A warning message
%                   will be displayed if M_B is increased.
%                   DEFAULT VALUE:  M_B = 120
%
% OPTS.M_REORTH     Number of vectors to reorthogonalize back
%                   against in part II, nonrestarted LSQR stage.
%                   Set M_REORTH to MAXITL performs
%                   reorthogonalization of all vectors computed.
%                   Does not apply in part I. M_REORTH must be set
%                   to be >= M_B.
%                   DEFAULT VALUE:  M_REORTH = M_B
%
% OPTS.REORTH012    Part I restarted LSQR stage: Full
%                   reorthogonalization (one or two sided) is
%                   always performed on the M_B vectors during
%                   the restarted stage. Part II
%                   nonrestarted LSQR stage: Either no, one-sided
%                   or two-sided reorthogonalization is performed.
%                   REORTH012 = 0 One-sided reorthogonalization
%                                    restarted stage (Part I)
%                                    and no reorthogonalization for
%                                    nonrestarted LSQR stage (Part II).
%                       REORTH012 = 1 One—sided reorthogonalization
%                                    restarted stage (Part I)
%                                    and one—sided reorthogonalization
%                                    for nonrestarted LSQR
%                                    stage (Part II).
%                       REORTH012 = 2 Two—sided reorthogonalization
%                                    restarted stage (Part I)
%                                    and two—sided reorthogonalization
%                                    for nonrestarted LSQR stage
%                                    (Part II).
%                       REORTH012 = 3 Two—sided reorthogonalization
%                                    restarted stage (Part I)
%                                    and one—sided reorthogonalization
%                                    for nonrestarted LSQR stage
%                                    (Part II).
%                       DEFAULT VALUE:  REORTH012 = 0
%
%——————————————————————————————————————————————————————————————————
%
% OUTPUT OPTIONS:
% ———————————————
%
% I.)    x = ALSQR(A,b,x,OPTS)
%           x  —  An approximate solution of min || b — A*x ||.
%
% II.)  [x,FLAG,Arrnorm] = ALSQR(A,b,x,OPTS)
%           x  —  An approximate solution of min || b — A*x ||.
%
%       FLAG  —  Four dimensional array
%                   (r = b-A*x and r_0 = b-A*x_0 where x is the current
%                   approximation to min || b - A*x || and x_0 is the
%                   initial approximation on input):
%                   FLAG(1) = 1  Stopping rule is satisfied:
%                                ||A'*r|| <= LTOL*||A'*r_0||
%                           = 0  Stopping rule is not satisfied.
%
%                   FLAG(2) = The approximate condition number, cond(A),
%                             of the matrix A.
%
%                   FLAG(3) = Number of iterations of part I restarted
%                             LSQR.
%
%                   FLAG(4) = Number of iterations of part II
%                             nonrestarted LSQR.
%
%    Arrnorm  -  Three dimensional array such that
%                   Arrnorm(j,1) = ||A'*r||/||A'r_0||
%
%                   Arrnorm(j,2) = ||r||/||r_0||
%
%                   Arrnorm(j,3) = total number of matrix-vector
%                                  products with A and A' required to
%                                  compute the solution x(j).
%
%---------------------------------------------------------------------
%
% DATE:  4/20/12
% VER:  1.0
%
% AUTHORS:
% James Baglama
% University of Rhode Island
% EMAIL: jbaglama@math.uri.edu
%
% Lothar Reichel
% Kent State University
% EMAIL: reichel@math.kent.edu
%
% Daniel Richmond
% University of Rhode Island
% EMAIL: dan@math.uri.edu
%
% REFERENCE:
%     J. Baglama, L.Reichel, and D. Richmond,
%         "An augmented LSQR method". Numer. Algorithms, Vol. 64,
%          Issue 2 (2013), pp. 263-293.
%
%-----------------------------------------------------------------------------------

%----------------------------%
% BEGIN: PARSE INPUT VALUES %
%----------------------------%

% Wrong number of input or output values.
if (nargin < 3 || nargin > 4 || nargout < 1 ...
        || nargout == 2 || nargout > 3 )
    error('alsqr: Incorrect number of input or output arguments.');
end

% Matrix A can be input as a numeric matrix or as a function handle.
if isa(A,'numeric')
    [m,n] = size(A); Anumeric = true;
elseif isa(A,'function_handle')
```

```
    m = size(b,1); n = size(x,1); Anumeric = false;
else
    error('alsqr: Unknown type for matrix A.');
end

% Set all input options to default values.
adjust = 40; k = 20; htol = 1d-3; ltol = 1d-6; maxitp = 1000;
maxitl = min(m,n); m_b = 120; m_reorth = m_b; reorth012 = 0;

% Get user input options from the structure OPTS.
if nargin == 4
    if ~isa(OPTS,'struct')
            error('alsqr:OptionsNotStructure',...
               'Options argument must be a structure.')
    end
    names = fieldnames(OPTS);
    I = strcmpi('ADJUST',names); I=find(I>0);
    if ~isempty(I), adjust = OPTS.(names{I}); end
    I = strcmpi('K',names);I=find(I>0);
    if  ~isempty(I), k = OPTS.(names{I}); end
    I = strcmpi('HTOL',names); I=find(I>0);
    if ~isempty(I), htol = OPTS.(names{I}); end
    I = strcmpi('LTOL',names); I=find(I>0);
    if ~isempty(I), ltol = OPTS.(names{I}); end
    I = strcmpi('MAXITP',names); I=find(I>0);
    if ~isempty(I), maxitp = OPTS.(names{I}); end
    I = strcmpi('MAXITL',names); I=find(I>0);
    if ~isempty(I), maxitl = OPTS.(names{I}); end
    I = strcmpi('M_B',names); I=find(I>0);
    if ~isempty(I), m_b = OPTS.(names{I}); m_reorth = m_b; end
    I = strcmpi('M_REORTH',names); I=find(I>0);
    if ~isempty(I), m_reorth = OPTS.(names{I}); end
    if m_reorth < m_b, error('alsqr: Requires m_reorth >= m_b'); end
    I = strcmpi('REORTH012',names); I=find(I>0);
    if ~isempty(I), reorth012 = OPTS.(names{I}); end
end

%--------------------------%
% END: PARSE INPUT VALUES %
%--------------------------%

%--------------------------------%
% BEGIN: SET UP INPUT VARIABLES %
%--------------------------------%

% Preallocate memory for all large matrices and vectors.
% Resizing will cause an increase in cpu time.
if m_reorth <= m_b+1
    Q = zeros(m,m_b+1);
    P = zeros(n,m_b+1);
else
    Q = zeros(m,m_reorth); P = zeros(n,m_reorth);
end
w = zeros(n,1);         % Storage vector required for LSQR.

% Initialize the ouput variable Arrnorm.
Arrnorm = [ones(maxitp+maxitl,2) zeros(maxitp+maxitl,1)];

% Compute the initial residual if x is not zero.
% Q(:,1) is the starting vector for the routine.
% The matrix-vector product computed here is not counted
% in Arrnorm.
if nnz(x) ~= 0
    if Anumeric
```

```
        Q(:,1) = b - A*x;
    else
        Q(:,1) = b - A(x,1);
    end
else
    Q(:,1) = b;
end

% Initialize the output value flag.
flag = zeros(1,4);

%-------------------------------%
% END: SET UP INPUT VARIABLES %
%-------------------------------%

%-------------------------------------------------------%
% BEGIN: DESCRIPTION AND INITIALIZATION OF VARIABLES %
%-------------------------------------------------------%

alpha = [];             % alpha_{m+1} in the GK bidiagonalization.
B = [];                 % Bidiagonal matrix.
hcount = 0;             % Count of converged harmonic Ritz vectors.
k_org = k;              % Holds initial value of k.
nrcount = 0;            % Count for residual norms.
resHarm = [];           % Residual norms of harmonic Ritz vectors.
piter = 1;              % Main loop iteration count for part I.
f = [];                 % Initial right hand side.
f_last = [];            % Last element of the right hand side,
                               % used to compute residual.
s = [];                 % Place holder.
Smax = -1;              % Holds the maximum value of all
                               % computed singular values of B.
```

```
Smin = Inf;               % Holds the minimum value of all
                                 % computed singular values of B.
S_B = [];                 % Singular values of B.
U_B = [];                 % Left singular vectors of B.
U_old = [];               % Place holder of left singular vectors of B
                                 % for updating harmonic Ritz vectors.
U_B_last = [];            % Holds the last column of U_B.
V_B = [];                 % Right singular vectors of B.
Ar0 = norm(A'*Q(:,1)); % For checking stopping criteria.
r0  = norm(Q(:,1));      % Value needed in Arrnorm.

%-----------------------------------------------------%
% END: DESCRIPTION AND INITIALIZATION OF  VARIABLES %
%-----------------------------------------------------%

%-------------------------------------------------------%
% BEGIN: PART I BUILDING SPACES AND UPDATING SOLUTION %
%-------------------------------------------------------%

% Increase the number of desired values by ADJUST to help
% speed up convergence. If k = 0 then there is no augmenting and
% nonrestarted LSQR is called.
if k > 0,
    k = k + adjust;
end

% Initial call to function augbidiaglsqr.
% Set k = 0 (no vectors for augmenting). On exit, if orginal k > 0
% we will have B an m_b+1 x m_b, Q an m x m_b+1, P an n x m_b+1.
if k > 0
    [alpha,Arrnorm,B,flag,f_last,nrcount,P,Q,w,x] = ...
        augbidiaglsqr(A,Anumeric,Arrnorm,B,f,flag,0,...
        ltol,m_b,m_b,m,n,nrcount,P,Q,reorth012,w,x,Ar0,r0);
end

% Main iteration loop to build up the spaces.
while (piter <= maxitp && ~flag(1) && k > 0)

    % Compute SVD of the rectangular lower bidiagonal matrix B.
    % MATLAB orders largest to smallest so we must reorder.
    % S_B: m_b+1 x m_b, U_B: m_b+1 x m_b+1, V_B: m_b x m_b
    [U_B,S_B,V_B] = svd(B);
    U_B_last = U_B(:,m_b+1); % Hold last column of U_B.
    U_B      = U_B(:,1:m_b); % Reset to economy size for reordering.
    S_B      = S_B(1:m_b,:); % Reset to economy size for reordering.

    % Write the columns in reverse order.
    U_B      = U_B(:,end:-1:1);
    S_B(:)   = S_B(end:-1:1);
    V_B      = V_B(:,end:-1:1);

    % Estimate ||A|| and condition number of A using the largest
    % and smallest singular values over all iterations.
    Smax = max(Smax,S_B(m_b,m_b)); Smin = min(Smin,S_B(1,1));
    flag(2) = Smax/Smin; % Approximate condition number of A.

    % Holds the old value of U_B that is used to compute new B.
    U_old = U_B;

    % Compute the harmonic Ritz vectors.
    s   = -U_B_last(1:m_b,1)/(U_B_last(m_b+1,1));
    U_B =  U_B(1:m_b,:)+s*U_B(m_b+1,:);

    % Compute QR of U_B and the residual vector using MATLAB's
    % internal QR function.
    [U_B,~] = qr([ [U_B(:,1:k); zeros(1,k)] [-s; 1] ],0);

    % Compute the vector f.
    f = f_last*[-s; 1];

    % Update Q, P and f.
    Q(:,1:k+1) = Q(:,1:m_b+1)*U_B;
    P(:,1:k+1) = [P(:,1:m_b)*V_B(:,1:k) P(:,m_b+1)];
    f = U_B'*f;

    % Reset the counter of converged harmonic Ritz vectors.
    hcount = 0;

    % Check convergence of harmonic Ritz vectors.
    for j=1:k_org
        if (sqrt((S_B(j,j)^2)*(norm(U_old(:,j)-U_B(:,j)))^2  + ...
                 (norm(B'*U_B(:,j)-S_B(j,j)*V_B(:,j)))^2     + ...
                 (alpha*U_B(m_b+1,j))^2) < htol*Smax)
                    hcount = hcount + 1;
        end
    end

    % Update B and alpha. MALTAB command TRIL is used to ensure
    % triangular part of B.
    B = tril([S_B(1:k,1:k)*U_old(:,1:k)'*U_B;...
              alpha*U_B(m_b+1,:)]');

    % Break from part I and continue to nonrestarted lsqr part II
    % if all harmonic Ritz vectors have converged.
    if (hcount >= k_org || piter >= maxitp), break; end

    % Update iteration count.
    piter   = piter + 1;
    flag(3) = piter;

    % Call the restarted augmented harmonic method to build
    % the spaces, approximate the solution, and update the
    % residual norms. On exit we will have B an m_b+1 x m_b,
    % Q an m x m_b+1, and P an n x m_b+1.
    [alpha,Arrnorm,B,flag,f_last,nrcount,...
        P,Q,w,x] = augbidiaglsqr(A,Anumeric,Arrnorm,B,f,flag,...
        k,ltol,m_b,m_b,m,n,nrcount,P,Q,reorth012,w,x,Ar0,r0);

end

%—————————————————————————————————————————————————————%
% END: PART I BUILDING SPACES AND UPDATING SOLUTION %
%—————————————————————————————————————————————————————%

%———————————————————————————————————%
% BEGIN: PART II NONRESTARTED LSQR %
%———————————————————————————————————%

% Set the number of iterations of nonrestarted LSQR to zero.
flag(4) = 0;

% Reset reorth to be one—sided for nonrestarted LSQR.
if reorth012 == 3, reorth012 = 1; end

% Call the nonrestarted lsqr part of function augbidiaglsqr.
if ~flag(1)
    [alpha,Arrnorm,B,flag,f_last,nrcount,...
        P,Q,w,x] = augbidiaglsqr(A,Anumeric,Arrnorm,B,f,flag,...
        k,ltol,maxitl,m_reorth,m,n,nrcount,P,Q,reorth012,...
        w,x,Ar0,r0);
end

%————————————————————————————————————%
% END: PART II NONRESTARTED LSQR %
%————————————————————————————————————%

%——————————————————————————%
% FUNCTION: AUGBIDIAGLSQR %
%——————————————————————————%

function [alpha,Arrnorm,B,flag,f_last,nrcount,...
            P,Q,w,x] = augbidiaglsqr(A,Anumeric,Arrnorm,B,f,...
            flag,k,ltol,maxitl,m_b,m,n,nrcount,P,Q,reorth012,...
            w,x,Ar0,r0)

% Function computes the GK Bidiagonalization and updates LSQR
% solution and residual. The function also can continue onto
% nonrestarted LSQR.
%
% INPUT:
%              A — (m x n) matrix
%       Anumeric — logic variable indicates whether A is a function
%                  call or numeric variable.
%        Arrnorm — (3 x 1) array contains the ||A'*r||, ||r|| and #
%                  of matrix—vector products.
%              B — (k+1 x k) bidiagonal matrix used in the GK
%                  bidiagonalization.
%              f — (k+1 x 1) vector right hand side used for
%                  least—squares solution.
%              k — scalar, number of augmenting vectors.
%           ltol — scalar, tolerance used for convergence of LSQR.
%         maxitl — scalar, maximum number of iteration for LSQR,
%                  set to m_b during part I.
%            m_b — scalar, size of the GK bidiagonalization
%                  and the number of reorthogonalization vectors
%                  during the LSQR stage.
%              m — scalar, number of rows of A.
%              n — scalar, number of columns of A.
%        nrcount — scalar, count used for Arrnorm.
%              P — (n x k+1) orthogonal matrix used in the GK
%                  bidiagonalization.
%              Q — (m x k+1) orthogonal matrix used in the GK
%                  bidiagonalization.
%      reorth012 — scalar indicates no, one, or two sided
%                  reorthogonalization of GK vectors.
%              w — (n x 1) vector used to update solution x in LSQR.
%              x — (n x 1) vector the LSQR solution.
%
% OUTPUT:
%          alpha — scalar, holds the last diagonal element of B.
%        Arrnorm — (3 x 1) updated array that contains ||A'*r||,
%                  ||r||, and # of matrix—vector products.
%              B — (m_b+1 x m_b) bidiagonal matrix used in the
%                  GK bidiagonalization.
%           flag — (4 x 1) array holds convegence indicators.
%         f_last — scalar, holds the last element of the right side.
%                  vector f that is used for least—squares solution.
%        nrcount — scalar, count used for Arrnorm.
%              P — (n x k+1) orthogonal matrix used in the GK
%                  bidiagonalization.
%              Q — (m x k+1) orthogonal matrix used in the GK
%                  bidiagonalization.
%      reorth012 — scalar, indicates no, one, or two sided
%                  reorthogonalization.
%              w — (n x 1) vector used to update solution x in LSQR.
%              x — (n x 1) vector the LSQR solution.
%
% Updated: 1/31/2012
% Authors: James Baglama and Daniel Richmond
%
% Function is called directly from alsqr.
%
% First iteration, set all initial values.
if k == 0
    beta = norm(Q(:,1)); f(1,1) = beta; Q(:,1) = Q(:,1)/beta;
    if Anumeric
        P(:,1) = (Q(:,1)'*A)';
    else
        P(:,1) = A(Q(:,1),2);
    end
    alpha = norm(P(:,1)); P(:,1) = P(:,1)/alpha; B(1,1) = alpha;
    Arrnorm(1,1)  = alpha*beta/Ar0;
    Arrnorm(1,2)  = beta/r0;
    Arrnorm(1,3)  = 1;
    nrcount       = 1;
end

% This part assumes entry is of the following form
%
%   A * P(:,1:k)    = Q(:,1:k+1)*B(1:k+1,1:k)
%   A'* Q(:,1:k+1)  = P(:,1:k)*B((1:k+1,1:k)'
%                          +alpha_{k+1}*P(:,k+1)*e_{k+1}'
%
% No checks are performed here and the code will produce
% erroneous results if the above equations do not hold on entry.
% Note k = 0 on first iteration.

% Set the B(k+1,k+1). On input when k > 0, B(k+1,k+1) holds
% the alpha value.
alpha = B(k+1,k+1);

% Compute A*P(:,k+1) - alpha_(k+1)*Q(:,k+1).
if Anumeric
    Q(:,k+2) = A*P(:,k+1) - Q(:,k+1)*alpha;
else
    Q(:,k+2) = A(P(:,k+1),1) - Q(:,k+1)*alpha;
end

% Reorthogonalization of Q vectors.
if (reorth012 >= 2 || (reorth012 <= 1 && m <= n ) )
    for i=1:k+1
        Q(:,k+2) = Q(:,k+2) - Q(:,i)*(Q(:,i)' * Q(:,k+2));
    end
end

% Set the B(k+2,k+1) entry beta_{k+2}.
beta       = norm(Q(:,k+2));
Q(:,k+2)   = Q(:,k+2)/beta;
B(k+2,k+1) = beta;

% Compute A'*Q(:,k+2) - beta_{k+2}*P(:,k+1).
if Anumeric
   P(:,k+2) = (Q(:,k+2)'*A)' - P(:,k+1)*beta;
else
   P(:,k+2) = A(Q(:,k+2),2) - P(:,k+1)*beta;
end
```

```

% Reorthogonalization of P.
if ((reorth012 >= 2) || (reorth012 <= 1 && m > n ))
    for i=1:k+1
        P(:,k+2) = P(:,k+2) - P(:,i)*(P(:,i)'*P(:,k+2));
    end
end

% Set the B(k+2,k+2) entry alpha_{k+2}.
alpha      = norm(P(:,k+2));
B(k+2,k+2) = alpha;
P(:,k+2)   = P(:,k+2)/alpha;

% We now have the following relationship
%
%   A *P(:,1:k+1) = Q(:,1:k+2)*B(1:k+2,1:k+1)
%   A'*Q(:,1:k+2) = P(:,1:k+1)*B((1:k+2,1:k+1)'
%                        +alpha_{k+2}*P(:,k+2)*e_{k+2}'
%
% Right hand side vector f is given from the previous iteration
% and is comuputed in the main loop. Notice that
% f is in the range of Q(:,1:k+1) and Q(:,k+2)'f = 0.
% We need the right hand side to be k+2 x 1 vector.
f(k+2,1) = 0;  f_last=0;

% Use MATLAB's internal QR. This gives +- values on the diagonal.
[Q_B,R_B] = qr(B(1:k+2,1:k+1));

% Setup of transition values for LSQR.
rho_bar = alpha*Q_B(k+2,k+2);
theta   = alpha*Q_B(k+2,k+1);

```

```
% Replace f(1:k+2,1) with Q'*f.
f = Q_B'*f;

phi_bar = f(k+2,1); % Intermediate right hand side value.

% Compute the solution y of ||f(k+2,1) - B(k+2,k+1)*y||.
y = R_B(1:k+1,1:k+1)\f(1:k+1);

% Update the LSQR solution vector.
x = x + P(:,1:k+1)*y;

% Check the stopping criteria and update Arrnorm
nrcount = nrcount + 1;
Arrnorm(nrcount,1) = abs(alpha*beta*y(k+1))/Ar0;
Arrnorm(nrcount,2) = abs(phi_bar)/r0;
Arrnorm(nrcount,3) = Arrnorm(nrcount-1,3) + 2;
if Arrnorm(nrcount,1) <= ltol
    flag(1) = 1;
    Arrnorm = Arrnorm(1:nrcount,1:3);
    return;
end

% Update w vector
w = P(:,k+2) - P(:,1:k+1)*(y/f(k+1,1)*theta);

% Set up the last vectors to hold the current iteration value
% for running the LSQR iteration.
P(:,m_b+1) = P(:,k+2); Q(:,m_b+1) = Q(:,k+2);

for i = k+2:maxitl

    % Set the next alpha value to the diagonal value B(i,i).
    if i <= m_b
        B(i,i) = alpha;
    end

    % Compute A*p - alpha*q.
    if Anumeric
        Q(:,m_b+1) = A*P(:,m_b+1) - alpha*Q(:,m_b+1);
    else
        Q(:,m_b+1) = A(P(:,m_b+1),1) - alpha*Q(:,m_b+1);
    end

    % Reorthogonalization of Q vectors.
    if i <= m_b
        if (reorth012 >= 2 || (reorth012 <= 1 && m <= n ) )
            for j=1:i
                Q(:,m_b+1) = Q(:,m_b+1) - ...
                    (Q(:,j)'*Q(:,m_b+1))*Q(:,j);
            end
        elseif(reorth012 >= 2 || (reorth012 == 1 && m <= n ) )
            for j=1:min(i,m_b)
                Q(:,m_b+1) = Q(:,m_b+1) - ...
                    (Q(:,j)'*Q(:,m_b+1))*Q(:,j);
            end
        end
    end

    % Computes beta_{i} of the B matrix
    beta = norm(Q(:,m_b+1)); Q(:,m_b+1) = Q(:,m_b+1)/beta;

    % Set the beta value to B(i+1,i).
    if i <= m_b, B(i+1,i) = beta; end

    % Compute A'*q - beta*p.
    if Anumeric
        P(:,m_b+1) = (Q(:,m_b+1)'*A)' - beta*P(:,m_b+1);
    else
        P(:,m_b+1) = A(Q(:,m_b+1),2) - beta*P(:,m_b+1);
    end

    % Reorthogonalization of P vectors.
    if i <= m_b
        if (reorth012 >= 2 || (reorth012 <= 1 && m > n ) )
            for j=1:i
                P(:,m_b+1) = P(:,m_b+1) - ...
                    (P(:,j)'*P(:,m_b+1))*P(:,j);
            end
     elseif(reorth012 >= 2 || (reorth012 == 1 && m > n ) )
            for j=1:min(i,m_b)
                P(:,m_b+1) = P(:,m_b+1) - ...
                    (P(:,j)'*P(:,m_b+1))*P(:,j);
            end
        end
    end

    % Computes alpha_{i} of the B matrix
    alpha      = norm(P(:,m_b+1));
    P(:,m_b+1) = P(:,m_b+1)/alpha;

    % Store only Q(:,1:m_b) and P(:,1:m_b)
    % and the matrix B(1:m_b+1,1:m_b).
    % Storage of the Q and P vectors is overwritten.
    % and reorthogonalization occcurs with all k harmonic Ritz
    % vectors and the last generated m_b vectors.
    if i < m_b
```

```
        Q(:,i+1) = Q(:,m_b+1); P(:,i+1) = P(:,m_b+1);
    elseif m_b ~= maxitl
        Q(:,mod(i,m_b)+1) = Q(:,m_b+1);
        P(:,mod(i,m_b)+1) = P(:,m_b+1);
    end

    % Start LSQR solution procedure.
    rho = pythag(rho_bar,beta); c1 = rho_bar/rho; s1 = beta/rho;

    % Transform B from lower bidiagonal to upper bidiagonal
    % matrix R. Super diagonal element in R is theta_{i} and
    % is computed from applying Givens to B. The diagonal value
    % of R is rho_{i} and is given above as two norm
    % of [rho_bar,beta].
    theta = s1*alpha;

    % Updating the last diagonal element of R.
    rho_bar = -c1*alpha;

    % Updating elements in f (two elements up).
    phi     = c1*phi_bar;
    phi_bar = s1*phi_bar;
    f_last  = -c1*phi_bar;

    % Update the solution and work vector.
    x = x + (phi/rho)*w; w = P(:,m_b+1) - (theta/rho)*w;

    % Check the stopping criteria and update Arrnorm.
    nrcount = nrcount + 1; flag(4) = i;
    Arrnorm(nrcount,1) = abs(phi_bar*rho_bar)/Ar0;
    Arrnorm(nrcount,2) = abs(phi_bar)/r0;
    Arrnorm(nrcount,3) = Arrnorm(nrcount-1,3) + 2;
    if Arrnorm(nrcount,1) <= ltol
        flag(1) = 1;
        Arrnorm = Arrnorm(1:nrcount,1:3);
        return;
    end
end

%----------------------%
% END AUGBIDIAGLSQR %
%----------------------%

%---------------------%
% FUNCTION: PYTHAG %
%---------------------%

function x = pythag(y,z)
% PYTHAG Computes sqrt( y^2 + z^2 ).
%
% x = pythag(y,z)
%
% Returns sqrt(y^2 + z^2) but is careful to scale to avoid overflow.

% Christian H. Bischof, Argonne National Laboratory, 03/31/89.

[m n] = size(y);
if m>1 || n>1
    y = y(:); z=z(:);
    rmax = max(abs([y';z']))';
    id=find(rmax==0);
    if length(id)>0
        rmax(id) = 1;
        x = rmax.*sqrt((y./rmax).^2 + (z./rmax).^2);
        x(id)=0;
    else
        x = rmax.*sqrt((y./rmax).^2 + (z./rmax).^2);
    end
    x = reshape(x,m,n);
else
    rmax = max(abs([y;z]));
    if (rmax==0)
        x = 0;
    else
        x = rmax*sqrt((y/rmax)^2 + (z/rmax)^2);
    end
end
```

B.2 Demo MATLAB script for using alsqr.m

```
% This script is easily modified to run alsqr.m for any matrix A
% and vector b from the matrix market collection or Univ. of
% Florida Collection. MATLAB function mmread.m is used here.
% It will plot the residual curves ||A'r||/||A'r_0||
% and ||r||/||r_0|| against the number of matrix-vector products
% on separate figures.

clear all; close all;

% Choose matrix and vector b.
A = mmread('illc1850.mtx');
b = mmread('illc1850_rhs1.mtx');

% Set the options for alsqr.
OPTS.ADJUST     = 40;
OPTS.K          = 20;
OPTS.m_b        = 100;
OPTS.M_REORTH  = OPTS.m_b;
OPTS.REORTH012 = 1;
OPTS.LTOL      = 1e-12;
OPTS.MAXITL    = 5000;
OPTS.MAXITP    = 1000;
OPTS.HTOL      = 5e-2;

[x,FLAG,Arrnorm] = alsqr(A,b,zeros(size(A,2),1),OPTS);
semilogy(Arrnorm(:,3),Arrnorm(:,1), 'k'); hold on

title('illc1850');
xlabel('matrix-vector products with A and A^T')
ylabel('$\frac{||A^Tr||}{||A^Tr_0||}$','interpreter','latex')
set(get(gca,'YLabel'),'Rotation',0.0)
h_legend = legend('ALSQR (100,20)');
set(h_legend,'FontSize',10);

% Plot ||r_m||/||r_0||.
figure

semilogy(Arrnorm(:,3),Arrnorm(:,2), 'k'); hold on

title('illc1850');
xlabel('matrix-vector products with A and A^T')
ylabel('$\frac{||r||}{||r_0||}$','interpreter','latex')
set(get(gca,'YLabel'),'Rotation',0.0)
h_legend = legend('ALSQR (100,20)');
set(h_legend,'FontSize',10);
```

B.3 MATLAB function irlsqr.m

```
function [x,Arrnorm] = irlsqr(A,b,x,OPTS)
% IRLSQR is an implictly restarted LSQR method. The solution
% is computed via an implictly restarted Golub—Kahan (GK)
% bidiagonalization method with harmonic Ritz values as shifts.
% IRLSQR solves the least—squares problem:
%
%                   min || b — A*x ||
%                    x
%
% where A is an (m x n) matrix, b an (m x 1) vector, and
% x is an (n x 1) vector. b is not assumed to be in span(Col(A)).
% ||.|| always denotes the 2 norm.

%———————————————————————————————————————————————————————————————————
%
% INPUT OPTIONS:
% ——————————————
%
% I.)   [x,Arrnorm] = IRLSQR(A,b,x)
%           The first input argument into ... = irlsqr(A,b,x)
%           is a numeric matrix A. The second input is the vector b
%           and third input is the vector x, an initial
%           approximation to min ||b — A*x||, typically
%           chosen to be 0.
%
%       [x,Arrnorm] = IRLSQR(A,b,x,OPTS)
%           OPTS is a structure containing input parameters.
%           The input parameters can be given in any order.
%           The structure OPTS may contain some or all of the
%           following input parameters. The string for the input
%           parameters can contain upper or lower case characters.
%           irlsqr(A,b,x) without the structure OPTS will use all
%           of the default values for OPTS.
%
```

```
%
% INPUT PARAMETER          DESCRIPTION
%
% OPTS.LTOL          Tolerance used for stopping rule.
%                    ||A'*r|| <= LTOL*||A'*r_0||
%                    DEFAULT VALUE:  LTOL = 1d-10
%
% OPTS.MAXITL        Maximum number of restarts for IRLSQR.
%                    DEFAULT VALUE:  MAXITL = min(m,n).
%
% OPTS.M_B           Size of the GK bidiagonal matrix. The program
%                    may increase M_B to ensure certain
%                    requirements are satisfied. A warning message
%                    will be displayed if M_B is increased.
%                    DEFAULT VALUE:   M_B = 90
%
% OPTS.REORTH012     Performs zero, one, or 2-sided
%                    reorthogonalization within the  GK vectors.
%                    DEFAULT VALUE:  REORTH012 = 1
%
% OPTS.P             Number of shifts to apply.
%                    DEFAULT VALUE:  P = 20
%
% OPTS.GAP_VAL       apply p +- GAP_VAL number of shifts based on
%                    largest gap between harmonic Ritz values.
%                    DEFAULT VALUE:  GAP_VAL = 0
%
% OPTS.BIDIAG_RED    String for returning the matrix B to lower
%                    bidiagonal form after the application of the
%                    shifts. A value of 1 returns B back to
%                    bidiagonal form.
%                    DEFAULT VALUE:  BIDIAG_RED = 0
```

```
%
%----------------------------------------------------------------------
%
% OUTPUT OPTIONS:
% ---------------
%
% I.)   x = IRLSQR(A,b,x,OPTS)
%           Returns only an approximate solution vector
%           x to min || b - A*x ||.
%
% II.) [x,Arrnorm] = IRLSQR(A,b,x,OPTS)
%            x - An approximate solution of min || b - A*x ||.
%
%      Arrnorm - Three dimensional array such that
%                Arrnorm(j,1) = ||A'*(b - A*x_j)|| = ||A'*r_j||,
%                Arrnorm(j,2) = ||b - A*x_j|| = ||r_j||,
%                and Arrnorm(j,3) is the total number of
%                matrix-vector products with A and A' required
%                to compute the solution x_j.
%
%----------------------------------------------------------------------
%
% DATE:  10/1/13
% VER:  1.0
%
% AUTHORS:
% James Baglama
% University of Rhode Island
% EMAIL: jbaglama@math.uri.edu
%
% Daniel Richmond
% University of Rhode Island
% EMAIL: dan@math.uri.edu

% REFERENCE:
%     J. Baglama and D. Richmond
%        "Implictly Restarting the LSQR Algorithm",
%         Electronic Transaction on Numerical Analysis,
%         Accepted for publication 2/7/14.
%
%---------------------------------------------------------------------

%----------------------------%
% BEGIN: PARSE INPUT VALUES %
%----------------------------%

% Wrong number of input or output values.
if (nargin < 3 || nargin > 4 || nargout < 1 || nargout > 2 )
    error('irlsqr: Incorrect number of input or output arguments');
end

% Matrix A must be input as numeric matrix.
if isa(A,'numeric')
    [m,n] = size(A);
else
    error('irlsqr: Unknown type for matrix A.');
end

%-------------------------------%
% BEGIN: PARSE INPUT VARIABLES %
%-------------------------------%

% Set all input options to default values.
 ltol = 1d-10; maxitl = min(m,n); m_b = 100; reorth012 = 1;
 p = 20; gap_val = 0; bidiag_red = 0;

if nargin == 4
    if ~isa(OPTS,'struct')
        error('irlsqr:OptionsNotStructure',...
        'Options argument must be a structure.')
    end
    names = fieldnames(OPTS);
    I = strcmpi('LTOL',names); I=find(I>0);
    if ~isempty(I), ltol = OPTS.(names{I}); end
    I = strcmpi('MAXITL',names); I=find(I>0);
    if ~isempty(I), maxitl = OPTS.(names{I}); end
    I = strcmpi('M_B',names); I=find(I>0);
    if ~isempty(I), m_b = OPTS.(names{I}); end
    I = strcmpi('REORTH012',names); I=find(I>0);
    if ~isempty(I), reorth012 = OPTS.(names{I}); end
    I = strcmpi('P',names); I=find(I>0);
    if ~isempty(I), p = OPTS.(names{I}); end
    I = strcmpi('GAP_VAL',names); I=find(I>0);
    if ~isempty(I), gap_val = OPTS.(names{I}); end
    I = strcmpi('BIDIAG_RED',names); I=find(I>0);
    if ~isempty(I), bidiag_red = OPTS.(names{I}); end
end

%-----------------------------------------------------%
% BEGIN: DESCRIPTION AND INITIALIZATION OF VARIABLES %
%-----------------------------------------------------%

rlsqr = [];                % Residual of LSQR.
alpha = [];                % alpha_{m+1} in the GK
                               % bidiagonalization.
B = [];                    % Bidiagonal matrix from GK
```

```
                                 % bidiagonalization.
Pres = [];                   % Last vector in P space.
f = [];                      % Initial right hand side.
S_B = [];                    % Singular values of B.
U_B = [];                    % Left singular vectors of B.
V_B = [];                    % Right singular vectors of B.
gap = [];                    % Value used in gap strategy.
Q(:,1) = b-A*x;              % Starting vector.
nrcount = 0;                 % Variable used to keep track of current
                                 % iteration.
w = [];                      % Work vector for LSQR solution procedure.
f1 = [];                     % Used in computation of vector rlsqr.
qres = 0;                    % q_{m_b+1,k+1} element of Q_L used in
                                 % restart vector.
iter = 1;                    % counter for main iteration loop.
flag = 0;                    % used for when method has converged.
ptemp = 0;                   % holds the number of shifts after gap
                                 %strategy.
Ar0 = norm(A'*Q(:,1));       % Used in plotting relative residual.
r0 = norm(Q(:,1));           % Used in plotting relative residual.

%--------------------------------------------------%
% END: DESCRIPTION AND INITIALIZATION OF VARIABLES %
%--------------------------------------------------%

%---------------------%
% BEGIN: MAIN METHOD %
%---------------------%

% Starting iteration, set all initial values.
beta    = norm(Q(:,1));
f1      = zeros(m_b+1,1);
```

```
f(1,1)  = beta;
f1(1,1) = beta;
Q(:,1)  = Q(:,1)/beta;

% Compute first vector in second Krylov space
P(:,1) = A'*Q(:,1);
alpha  = norm(P(:,1));
P(:,1) = P(:,1)/alpha;
B(1,1) = alpha;

% Compute initial values for Arrnorm
Arrnorm(1,1) = alpha*beta/Ar0; %||A'r||/||A'r_0||.
Arrnorm(1,2) = beta/r0;        %||r||/||r_0||.
Arrnorm(1,3) = 1;
nrcount      = 1;

% Initial call to GK bidiagonalization with K=0.
    [alpha,Q,P,B,nrcount,x,r0,Ar0,...
        Arrnorm,flag] = gkbd(A,Q,P,B,0,m_b,nrcount,...
        f,x,r0,Ar0,Arrnorm,ltol,flag,reorth012,m,n);

while (iter <= maxitl && ~flag)

    % hold last vector in P space before re-sizing.
    Pres = P(:,m_b+1);
    P    = P(:,1:m_b);

    % The following GK equations must be satisfied on exit:
    %   A * P(:,1:m_b)    =   Q(:,1:m_b+1)*B(1:m_b+1,1:m_b)
    %   A'* Q(:,1:m_b+1)  =   P(:,1:m_b)*B(1:k+1,1:k)'...
    %                         +alpha*P(:,m_b+1)*flipud(eye(m_b+1,1))

    % Compute the residual vector from LSQR.
    [q,~]   = qr(B);
    phi_bar = flipud(eye(m_b+1,1))'*q'*f1;
    rlsqr   = Q*phi_bar*q*flipud(eye(m_b+1,1));

    % Compute SVD of B to get updated U_B, S_B, and V_B matrices.
    [U_B,S_B,V_B] = svd(B);

    % Implement gap strategy.
    gap_old = 0;
    ptemp   = p;
    for i = (p-gap_val):(p+gap_val)
       gap = abs(S_B(i,i)-S_B(i+1,i+1));
        if gap > gap_old
            gap_old = gap;
            p = i; % Update the number of shifts to apply.
        end
    end
    k = m_b-p;

    % Apply p harmonic Ritz values as shifts.
    [Q_L,B,Q_R] = hbsQR(B,U_B,V_B,m_b,p);

    % Update P, Q, B matrices and truncate.
    Q = Q*Q_L(:,1:k+1);
    P = P*Q_R(:,1:k);
    B = B(1:k+1,1:k);

    % B may not be a bidiagonal matrix, can use rowwise Householder
    % to get back bidiagonal structure
    if bidiag_red == 1
        [B,Q_LB,Q_RB] = rowHH(B);
```

```
        P = P*Q_RB;
        Q = Q*Q_LB';
        qres = Q_L(m_b+1,k+1)*Q_LB(k+1,k+1);
    else
        % Hold value of Q_L_{m_b+1,k+1}.
        qres = Q_L(m_b+1,k+1);
    end

    % The following GK equations must be satisfied after
    % 'p' shifts have been applied
    %  A * P(:,1:k)   = Q(:,1:k+1)*B(1:k+1,1:k)
    %  A'* Q(:,1:k+1) = P(:,1:k)*B(1:k+1,1:k)'...
    %                     +qres*alpha*P(:,m_b+1)*flipud(eye(k+1,1))'

    % Compute restart vector and next element of B matrix.
    rk          = alpha*qres*Pres;
    rknorm      = norm(rk);
    P           = [P(:,1:k) rk/rknorm];
    B(k+1,k+1)  = rknorm;

    % Create f vector needed for restart in solving for LSQR
    % solution. rlsqr must be in col(Q) to restart correctly.
    f = Q'*rlsqr;

    % Need to update f1 for computation of rlsqr after restart.
    f1 = [f; zeros(p,1)];

    p = ptemp; % Reset value of p to pre-gap value.

    % Call GK bidiagonalization to build up space to size m_b.
    [alpha, Q,P,B,nrcount,x,r0,Ar0,...
        Arrnorm,flag] = gkbd(A,Q,P,B,k,m_b,nrcount,...
        f,x,r0,Ar0,Arrnorm,ltol,flag,reorth012,m,n);
end

%-------------------%
% END: MAIN METHOD %
%-------------------%

%-------------------------------------%
% BEGIN: GK BIDIAGONALIZATION / LSQR %
%-------------------------------------%
function [alpha,Q,P,B,nrcount,x,r0,Ar0,...
    Arrnorm,flag] = gkbd(A,Q,P,B,k,m_b,nrcount,...
    f,x,r0,Ar0,Arrnorm,ltol,flag,reorth012,m,n)

alpha = B(k+1,k+1);

% Compute next vector in Q.
Q(:,k+2) = A*P(:,k+1)- Q(:,k+1)*alpha;

% Reorthogonalization of Q vectors.
if (reorth012 == 2 || (reorth012 == 1 && m < n ))
    for j=1:k+1
        Q(:,k+2) = Q(:,k+2) - Q(:,j)*(Q(:,j)' * Q(:,k+2));
    end
end

% Computes beta_{k+2} of the B(k+2,k+1) matrix.
beta = norm(Q(:,k+2));
Q(:,k+2)   = Q(:,k+2)/beta;
B(k+2,k+1) = beta;

% Compute A'*Q(:,k+2) - beta_{k+2}*P(:,k+1).
```

```
P(:,k+2) = (Q(:,k+2)'*A)' - P(:,k+1)*beta;

% Reorthogonalization of P vectors.
if (reorth012 == 2 || (reorth012 == 1 && m >= n ))
    for j=1:k+1
        P(:,k+2) = P(:,k+2) - P(:,j)*(P(:,j)'*P(:,k+2));
    end
end

alpha = norm(P(:,k+2));

% Needed for applying only 1 shift.
if k~=m_b-1
    B(k+2,k+2) = alpha;
end
P(:,k+2) = P(:,k+2)/alpha;

% Update vector f.
f(k+2,1) = 0;

% Use MATLAB's internal QR. This gives +- values on the diagonal.
[Q_B,R_B] = qr(B(1:k+2,1:k+1));

% Set up transition values for LSQR
rho_bar = alpha*Q_B(k+2,k+2);
theta   = alpha*Q_B(k+2,k+1);

% Solve ||f(k+2,1) - B(k+2,k+1)*y||.
% Replace B(k+2,k+1) with Q(1:k+2,1:k+2)*R(1:k+2,1:k).
% Replace f(k+2,1) with Q'*f.
f = Q_B'*f;
phi_bar = f(k+2,1);
```

```

% Compute the solution y of ||f(k+2,1) - B(k+2,k+1)*y||.
y = R_B(1:k+1,1:k+1)\f(1:k+1);

% Update the solution vector.
x = x + P(:,1:k+1)*y;

% Compute norms of residuals / check stopping criteria.
nrcount              = nrcount + 1;
Arrnorm(nrcount,1)  = abs(alpha*beta*y(k+1))/Ar0;
Arrnorm(nrcount,2)  = abs(phi_bar)/r0;
Arrnorm(nrcount,3)  = Arrnorm(nrcount-1,3) + 2;
if Arrnorm(nrcount,1) <= ltol
    Arrnorm = Arrnorm(1:nrcount,1:3); flag=1;
    return;
end

% Update w vector.
w = P(:,k+2) - P(:,1:k+1)*(y/f(k+1,1)*theta);

for i = k+2:m_b
    Q(:,i+1) = A*P(:,i) - alpha*Q(:,i);

    % Reorthogonalization of Q vectors.
    if ((reorth012 == 2) || (reorth012 == 1 && m < n ))
        for j=1:i
            Q(:,i+1) = Q(:,i+1) - Q(:,j)*(Q(:,j)' * Q(:,i+1));
        end
    end

    % Normalize the q_i vector.
    beta     = norm(Q(:,i+1));
    Q(:,i+1) = Q(:,i+1)/beta;

    % Update the B matrix.
    B(i+1,i) = beta;

    % Continue the bidiagonalization.
    P(:,i+1) = A'*Q(:,i+1) - beta*P(:,i);

    % Reorthogonalization of P vectors.
    if ((reorth012 == 2) || (reorth012 == 1 && m >= n ))
        for j=1:i
            P(:,i+1) = P(:,i+1) - P(:,j)*(P(:,j)' * P(:,i+1));
        end
    end

    % Update the B matrix.
    alpha      = norm(P(:,i+1));
    P(:,i+1)   = P(:,i+1)/alpha;
    B(i+1,i+1) = alpha;
    if i==m_b
        B = B(1:i+1,1:i);
    end

    % Givens values.
    rho     = pythag(rho_bar,beta);
    c1      = rho_bar/rho;
    s1      = beta/rho;
    theta   = s1*alpha;
    rho_bar = -c1*alpha;
    phi     = c1*phi_bar;
    phi_bar = s1*phi_bar;

    % Update the solution.
    x = x + (phi/rho)*w; w = P(:,i+1) - (theta/rho)*w;

    % Compute norms of residual / check stopping criteria
    nrcount             = nrcount + 1;
    Arrnorm(nrcount,1) = abs(phi_bar*rho_bar)/Ar0;
    Arrnorm(nrcount,2) = abs(phi_bar)/r0;
    Arrnorm(nrcount,3) = Arrnorm(nrcount-1,3) + 2;
    if Arrnorm(nrcount,1) <= ltol
        Arrnorm = Arrnorm(1:nrcount,1:3); flag = 1;
        return;
    end
end

%-----------------------------------%
% END: GK BIDIAGONALIZATION / LSQR %
%-----------------------------------%

%---------------------------------------%
% FUNCTION: HARMONIC BIDIAGONAL METHOD %
%---------------------------------------%

function [Q_L, B_plus, Q_R] = hbsQR(B,U_B,V_B,m_b,p)
% This function computes the orthogonal matrices Q_L and Q_R
% such that B_plus = Q_L'*B*Q_R is the matrix obtained after p
% implicit shifts have been performed. B_plus may no longer be
% lower bidiagonal due to numerical error.
%
% INPUT:
%                B - m_b+1 x m_b lower bidiagonal matrix from GK
%                    bidiagonalization.
%              U_B - matrix of left singular vectors of B.
%              V_B — matrix of right singular vectors of B.
%              m_b — size of the matrix B.
%                p — number of shifts to be applied.
%
% OUTPUT:
%              Q_L — (m_b+1) x (m_b+1) orthogonal upper Hessenberg
%                    matrix.
%           B_plus — updated B matrix.
%              Q_R — m_b x m_b orthogonal upper Hessenberg matrix.
%

% Initialize values.
sL = m_b+1;
sR = m_b;
Q_L = zeros(m_b+1); Q_R = zeros(m_b); Is = p:−1:1;

% Take QR of the p+1 x p principal submatrix of U_B' to get
% first column of Q_L.
[Q,~] = qr(U_B(1:p+1,1:p)); Q_L(1:p+1,1) = Q(:,p+1);

% Continue process to get first m_b+1−p columns of Q_L.
for i = 2:sL−p
    [Q,~] = qr([Q_L(1:i+p,1:i−1) U_B(1:i+p,1:p)]);
    Q_L(1:i+p,i) = Q(:,p+i);
end

% Last p columns of Q_L are the first p columns of U_B.
Q_L(:,sL−p+1:sL) = U_B(:,Is);

% Take QR of p+1 x p principal submatrix of V_B' to get
% first column of Q_R.
[Q,~] = qr(V_B(1:p+1,1:p)); Q_R(1:p+1,1) = Q(:,p+1);
```

```

% Continue process to get first m_b-p columns of Q_R.
for i = 2:sR-p
    [Q,~] = qr([Q_R(1:i+p,1:i-1) V_B(1:i+p,1:p)]);
    Q_R(1:i+p,i) = Q(:,i+p);
end

% Last p columns of Q_R are the first p columns of V_B.
Q_R(:,sR-p+1:sR) = V_B(:,Is);

% Update B.
B_plus = Q_L'*B*Q_R;

%--------------------------------------%
% END HARMONIC BIDIAGONAL METHOD %
%--------------------------------------%

%--------------------------------------%
% FUNCTION: ROWWISE HOUSEHOLDER %
%--------------------------------------%

function [B,U,V] = rowHH(A)
% This function produces a lower bidiagonal matrix B
% such that B = U A V.
% Function adapted from David S. Watkins' uphess.m 03/09/2007.
%
% INPUT:
%                A - (m_b+1) x m_b matrix.
%
% OUTPUT:
%                B - (m_b+1) x m_b lower bidiagonal matrix.
%                U - Orthogonal (m_b+1) x (m_b+1) matrix.
%                 V - Orthogonal m_b x m_b matrix.

[m,n] = size(A);
B = A; QL=eye(m); QR=eye(n);
U=eye(m); V=eye(n);
for k = m:-1:3
    % Obtain Householder reflector to zero out rows of B.
    % b*G = alpha*e_k^T
    [v,beta,~] = reflector(B(k,1:k-1)');
    QR(1:k-1,1:k-1) = eye(k-1)-beta*(v*v');
    V = V*QR;
    B = B*QR;

    % Obtain Householder reflector to zero out columns of B.
    [v,beta,~] = reflector(B(1:k-1,k-1));
    QL(1:k-1,1:k-1) = eye(k-1)-beta*(v*v');
    U = QL*U;
    B = QL*B;

    %Reset matrices.
    QR = eye(n); QL = eye(m);
end

%-----------------------------%
% END ROWWISE HOUSEHOLDER %
%-----------------------------%

%------------------------%
% FUNCTION: REFLECTOR %
%------------------------%

function [u,beta,alpha] = reflector(x)
```

```
% This function generates a reflector Q = eye - beta*u*u'
% such that Q*x = alpha*e_n.
% David S. Watkins 03/09/2007.

m = size(x,1);

% Rescale x so that x_1 is nonnegative and norm(x) = 1.
scale = norm(x);
if scale == 0
    u = x; beta = 0; alpha = 0;
else
    x = x/scale;
    if x(m) ~= 0
        phase = x(m)/abs(x(m));
        x = x*conj(phase); x(m) = real(x(m));
    else
        phase = 1;
    end

    % Build u and beta.
    u = x; u(m) = u(m) + 1;
    beta = 1/u(m);

    % Rescale.
    alpha = -scale*phase;
end

%----------------%
% END REFELCTOR %
%----------------%
```

B.4 Demo MATLAB script for using `irlsqr.m`

```
% This script is easily modified to run irlsqr.m for any matrix A
% and vector b from the matrix market collection, or Univ. Florida
% Collection. It will plot the residual curves ||A'r||/||A'r_0||
% and ||r||/||r_0|| against the number of matrix-vector products
% on separate figures.

clear all; close all

%Choose matrix A, vector b, and initial guess x.
A = mmread('illc1850.mtx');
b = mmread('illc1850_rhs1.mtx');
x = zeros(size(A,2),1);

%set the options.
OPTS.P          = 30;
OPTS.M_B        = 100;
OPTS.GAP_VAL    = 5;
OPTS.BIDIAG_RED = 0;
OPTS.REORTH012  = 1;
OPTS.LTOL       = 1e-12;
OPTS.MAXITL     = 1000;

[x,Arrnorm] = irlsqr(A,b,x,OPTS);
semilogy(Arrnorm(:,3),Arrnorm(:,2),'k'); hold on

title('illc1850');
xlabel('matrix-vector products with A and A^T')
ylabel('$\frac{||r||}{||r_0||}$','interpreter','latex')
set(get(gca,'YLabel'),'Rotation',0.0)
h_legend = legend('IRLSQR(100,30)');
set(h_legend, 'FontSize',8);

```

```
% Plot ||r||/||r_0||.
figure

semilogy(Arrnorm(:,3),Arrnorm(:,1),'k'); hold on

title('illc1850');
xlabel('matrix-vector products with A and A^T')
ylabel('$\frac{||A^Tr||}{||A^Tr_0||}$','interpreter','latex')
set(get(gca,'YLabel'),'Rotation',0.0)
h_legend = legend('IRLSQR(100,30)');
set(h_legend,'FontSize',8);
```

BIBLIOGRAPHY

Baglama, J., Calvetti, D., Golub, G. H., and Reichel, L., "Adaptively preconditioned GMRES algorithms," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 243–269, 1998.

Baglama, J., Reichel, L., and Richmond, D., "An augmented LSQR method," *Numerical Algorithms*, vol. 64, no. 2, pp. 263–293, 2013.

Baglama, J. and Reichel, L., "Augmented implicitly restarted Lanczos bidiagonalization methods," *SIAM Journal on Scientific Computing*, vol. 27, no. 1, pp. 19–42, 2005.

Baglama, J. and Reichel, L., "Restarted block Lanczos bidiagonalization methods," *Numerical Algorithms*, vol. 43, no. 3, pp. 251–272, 2006.

Baglama, J. and Reichel, L., "An implicitly restarted block Lanczos bidiagonalization method using Leja shifts," *BIT Numerical Mathematics*, vol. 53, no. 2, pp. 285–310, 2013.

Baker, A. H., Jessup, E. R., and Manteuffel, T., "A technique for accelerating the convergence of restarted GMRES," *SIAM Journal on Matrix Analysis and Applications*, vol. 26, no. 4, pp. 962–984, 2005.

Baruchel, J., Buffiere, J.-Y., and Maire, E., *X-ray tomography in material science*. Hermes Science, 2000.

Benzi, M., "Preconditioning techniques for large linear systems: a survey," *Journal of Computational Physics*, vol. 182, no. 2, pp. 418–477, 2002.

Benzi, M. and Tuma, M., "A robust preconditioner with low memory requirements for large sparse least squares problems," *SIAM Journal on Scientific Computing*, vol. 25, no. 2, pp. 499–512, 2003.

Björck, Å. and Yuan, J., "Preconditioners for least squares problems by LU factorization," *Electronic Transactions on Numerical Analysis*, vol. 8, pp. 26–35, 1999.

Björck, Å., *Numerical methods for least squares problems*. Siam, 1996.

Björck, Å., Grimme, E., and Van Dooren, P., "An implicit shift bidiagonalization algorithm for ill-posed systems," *BIT Numerical Mathematics*, vol. 34, no. 4, pp. 510–534, 1994.

Boisvert, R. F., Pozo, R., Remington, K. A., Barrett, R. F., and Dongarra, J., "Matrix Market: a web resource for test matrix collections,," in *Quality of Numerical Software*, 1996, pp. 125–137.

Choi, S.-C. T., "Iterative methods for singular linear equations and least-squares problems," Ph.D. dissertation, Stanford University, 2006.

Davis, T. A. and Hu, Y., "The University of Florida sparse matrix collection," *ACM Transactions on Mathematical Software*, vol. 38, no. 1, p. 1, 2011.

Duff, I. S., Grimes, R. G., and Lewis, J. G., "Users' guide for the Harwell-Boeing sparse matrix collection (Release I)," Report RAL-92-086, Atlas Centre, Rutherford Appleton Laboratory, Didcot, Oxon, UK, Tech. Rep., 1992.

Erhel, J., Burrage, K., and Pohl, B., "Restarted GMRES preconditioned by deflation," *Journal of computational and applied mathematics*, vol. 69, no. 2, pp. 303–318, 1996.

Fong, D. C.-L. and Saunders, M., "LSMR: An iterative algorithm for sparse least-squares problems," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2950–2971, 2011.

Golub, G. and Kahan, W., "Calculating the singular values and pseudo-inverse of a matrix," *Journal of the Society for Industrial & Applied Mathematics, Series B: Numerical Analysis*, vol. 2, no. 2, pp. 205–224, 1965.

Golub, G. H. and Van Loan, C. F., *Matrix computations.* Johns Hopkins University Press, 2012, vol. 3.

Hayami, K., Yin, J.-F., and Ito, T., "GMRES methods for least squares problems," *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 5, pp. 2400–2430, 2010.

Hochstenbach, M. E., "Harmonic and refined extraction methods for the singular value problem, with applications in least squares problems," *BIT Numerical Mathematics*, vol. 44, no. 4, pp. 721–754, 2004.

Ito, T. and Hayami, K., "Preconditioned GMRES methods for least squares problems," NII Technical Report, NII-2004-006E, National Institute of Informatics, Tokyo, Japan, Tech. Rep., 2004.

Jia, Z., "Some properties of LSQR for large sparse linear least squares problems," *Journal of Systems Science and Complexity*, vol. 23, no. 4, pp. 815–821, 2010.

Jia, Z. and Niu, D., "An implicitly restarted refined bidiagonalization Lanczos method for computing a partial singular value decomposition," *SIAM journal on matrix analysis and applications*, vol. 25, no. 1, pp. 246–265, 2003.

Jia, Z. and Niu, D., “A refined harmonic Lanczos bidiagonalization method and an implicitly restarted algorithm for computing the smallest singular triplets of large matrices,” *SIAM Journal on Scientific Computing*, vol. 32, no. 2, pp. 714–744, 2010.

Karimi, S., Salkuyeh, D. K., and Toutounian, F., “A preconditioner for the LSQR algorithm,” *Journal of Applied Mathematics and Informatics*, vol. 26, no. 1-2, pp. 213–222, 2008.

Kokiopoulou, E., Bekas, C., and Gallopoulos, E., “Computing smallest singular triplets with implicitly restarted Lanczos bidiagonalization,” *Applied numerical mathematics*, vol. 49, no. 1, pp. 39–61, 2004.

Larsen, R. M., “Lanczos bidiagonalization with partial reorthogonalization,” *DAIMI Report Series*, vol. 27, no. 537, 1998.

Larsen, R. M., “Combining implicit restarts and partial reorthogonalization in Lanczos bidiagonalization,” *Program in Scientific Computing and Computational Mathematics, Stanford University*, 2001.

Maire, E., Fazekas, A., Salvo, L., Dendievel, R., Youssef, S., Cloetens, P., and Letang, J. M., “X-ray tomography applied to the characterization of cellular materials. Related finite element modeling problems,” *Composites Science and Technology*, vol. 63, no. 16, pp. 2431–2443, 2003.

MATLAB, *version R2011a*. Natick, Massachusetts: The MathWorks Inc., 2011.

Morgan, R. B., “Computing interior eigenvalues of large matrices,” *Linear Algebra and its Applications*, vol. 154, pp. 289–309, 1991.

Morgan, R. B., “A restarted GMRES method augmented with eigenvectors,” *SIAM Journal on Matrix Analysis and Applications*, vol. 16, no. 4, pp. 1154–1171, 1995.

Morgan, R. B., “Implicitly restarted GMRES and Arnoldi methods for nonsymmetric systems of equations,” *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1112–1135, 2000.

Morgan, R. B., “GMRES with deflated restarting,” *SIAM Journal on Scientific Computing*, vol. 24, no. 1, pp. 20–37, 2002.

Paige, C. C., Parlett, B. N., and van der Vorst, H. A., “Approximate solutions and eigenvalue bounds from Krylov subspaces,” *Numerical linear algebra with applications*, vol. 2, no. 2, pp. 115–133, 1995.

Paige, C. C., “Bidiagonalization of matrices and solution of linear equations,” *SIAM Journal on Numerical Analysis*, vol. 11, no. 1, pp. 197–209, 1974.

Paige, C. C. and Saunders, M. A., "LSQR: An algorithm for sparse linear equations and sparse least squares," *ACM Transactions on Mathematical Software*, vol. 8, no. 1, pp. 43–71, 1982.

Parlett, B. N., *The symmetric eigenvalue problem.* SIAM, 1980, vol. 7.

Reichel, L. and Ye, Q., "A generalized LSQR algorithm," *Numerical Linear Algebra with Applications*, vol. 15, no. 7, pp. 643–660, 2008.

Saad, Y. and Schultz, M. H., "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on scientific and statistical computing*, vol. 7, no. 3, pp. 856–869, 1986.

Saad, Y., *Iterative methods for sparse linear systems.* SIAM, 2003.

Simon, H. D. and Zha, H., "Low-rank matrix approximation using the Lanczos bidiagonalization process with applications," *SIAM Journal on Scientific Computing*, vol. 21, no. 6, pp. 2257–2274, 2000.

Stewart, G., "A Krylov–Schur algorithm for large eigenproblems," *SIAM Journal on Matrix Analysis and Applications*, vol. 23, no. 3, pp. 601–614, 2002.

Stoll, M., "A Krylov–Schur approach to the truncated svd," *Linear Algebra and its Applications*, vol. 436, no. 8, pp. 2795–2806, 2012.

Watkins, D. S., *The matrix eigenvalue problem: GR and Krylov subspace methods.* Siam, 2007.

Zavorin, I., O'Leary, D. P., and Elman, H., "Complete stagnation of GMRES," *Linear Algebra and its Applications*, vol. 367, pp. 165–183, 2003.

Zilkoski, D. B., Richards, J. H., and Young, G. M., "Special report: Results of the general adjustment of the North American vertical datum of 1988," *Surveying and Land Information Systems*, vol. 52, no. 3, pp. 133–149, 1992.