THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

A KRYLOV-SCHUR METHOD FOR COMPUTING SINGULAR VALUES

AMUTHAN NARTHANA
SPRING 2014

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degrees
in Computer Science
and Mathematics
with honors in Computer Science

Reviewed and approved* by the following:

Jesse Barlow
Professor of Computer Science and Engineering
Thesis Supervisor / Honors Adviser

Kamesh Madduri
Assistant Professor of Computer Science and Engineering
Faculty Reader

* Signatures are on file in the Schreyer Honors College.

Abstract

Given a large sparse matrix, it is often useful to compute a small number of its singular values and vectors. This computation is based on Golub-Kahan-Lanczos bidiagonalization and the Krylov-Schur method for finding singular values and vectors of a matrix. The Krylov-Schur approach to finding a small number of singular values will be described, and various reorthogonalization strategies will be discussed. Numerical tests will compare variations of this method on several matrices, and the results will be analyzed.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

The singular value decomposition (SVD) of a matrix $X \in \mathbb{R}^{m \times n}$ $(m \geq n)$ is a factorization of the form

$$X = P\Sigma Z^T \tag{1}$$

where $P \in \mathbb{R}^{m \times n}$ is left orthogonal, $\Sigma \in \mathbb{R}^{n \times n}$ is diagonal, and $Z \in \mathbb{R}^{n \times n}$ is orthogonal. The diagonal entries of Σ are called singular values of X, and the column vectors of P and Z are called left and right singular vectors of X, respectively.

If $m < n$, then the SVD of X satisfies (1) where $P \in \mathbb{R}^{m \times m}$ is orthogonal, $\Sigma \in \mathbb{R}^{m \times m}$ is diagonal, and $Z \in \mathbb{R}^{m \times n}$ is right orthogonal. Note that if $m < n$, then

$$X = P\Sigma Z^T \iff X^T = Z\Sigma P^T$$

so the SVD of X is interchangeable with the SVD of X^T. Thus without loss of generality, this paper will always assume $m \geq n$.

The SVD of X can also be written in the outer product form

$$X = \sum_{i=1}^{n} \sigma_i u_i v_i^T$$

where σ_i is the i-th largest singular value of X, and u_i and v_i are the associated left and right singular vectors, respectively.

One way to compute the SVD of a matrix was introduced by Golub and Kahan in [6]. They defined a process called Golub-Kahan-Lanczos (GKL) bidiagonalization, which produces a factorization of $X \in \mathbb{R}^{m \times n}$ satisfying

$$X = UBV^T \tag{2}$$

where $U \in \mathbb{R}^{m \times n}$ is left orthogonal, $B \in \mathbb{R}^{n \times n}$ is upper bidiagonal, and $V \in \mathbb{R}^{n \times n}$ is orthogonal. Any bidiagonal SVD routine (e.g. the QR algorithm [7, §7.5]) can quickly generate the SVD of B:

$$B = Q\Sigma S^T \tag{3}$$

Defining

$$P = UQ$$
$$Z = VS$$

the SVD of X can be computed by combining (2) and (3) to get

$$X = UBV^T = U(Q\Sigma S^T)V^T = (UQ)\Sigma(VS)^T = P\Sigma Z^T. \tag{4}$$

In [11], Stewart defined the Krylov-Schur method for solving the generalized eigenvalue problem. Baglama and Reichel [2] demonstrated a restarted algorithm that uses GKL bidiagonalization to compute a small number (e.g. 10) of singular values and associated singular vectors of a large sparse matrix. Stoll [12] illustrated a similar method and showed how this method is a variation of the Krylov-Schur method introduced by Stewart [11]. A variation of Stoll's [12] Krylov-Schur algorithm is presented in this paper.

1.2 Applications

There are many applications for computing a small number of singular values and associated singular vectors of a large sparse matrix. These include total least squares problems [2], image analysis [5, 10], and signal tracking [4]. In addition, the largest k singular values/vectors can be used to create the rank k matrix $\overline{X}$ closest to a matrix X in the Frobenius norm. The Schmidt-Mirsky Theorem [7, §2.4] shows that

$$\overline{X} = \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

where σ_i is the i-th largest singular value of X, and u_i and v_i are the associated left and right singular vectors, respectively. Thus by computing the largest singular values and vectors of a matrix, the Krylov-Schur algorithm can be used to generate the best low-rank approximation of a large matrix X.

When a less accurate low-rank approximation of X will suffice, Simon and Zha show in [10] how GKL bidiagonalization can also be used to inexpensively produce the low-rank approximation

$$\overline{X} = U_k B_k V_k^T$$

where $k \ll n$, $U_k \in \mathbb{R}^{m\times k}$ and $V_k \in \mathbb{R}^{n\times k}$ are left orthogonal, and $B_k \in \mathbb{R}^{k\times k}$ is upper bidiagonal.

1.3 Outline

This paper will describe the Krylov-Schur approach to finding a small number of singular values and associated singular vectors of a matrix. Section 2 will describe a variant of the chasing algorithm, a basic operation used in the Krylov-Schur method. Section 3 will define the Krylov-Schur method, with section 3.1 providing an overview of GKL bidiagonalization and section 3.2 describing how the Krylov-Schur method generates singular values/vectors. Section 4 will describe various reorthogonalization techniques, and section 5 will present and analyze some numerical experiments for this Krylov-Schur method.

2 The Chasing Algorithm

The "chasing algorithm" is one of the basic operations used in the Krylov-Schur method for computing singular values and singular vectors. Given a diagonal matrix $\Sigma_w \in \mathbb{R}^{\ell\times\ell}$ and a vector $d_w \in \mathbb{R}^{\ell}$, the "chasing algorithm" generates orthogonal matrices $Q_w \in \mathbb{R}^{\ell\times\ell}$ and $S_w \in \mathbb{R}^{\ell\times\ell}$, and an upper bidiagonal matrix $B_w \in \mathbb{R}^{\ell\times\ell}$ satisfying

$$Q_w^T \Sigma_w S_w = B_w \tag{5}$$

$$Q_w^T d_w = \hat{\phi}_{\ell+1} e_\ell \tag{6}$$

where e_ℓ is the ℓ-th row of the identity matrix and $\hat{\phi}_{\ell+1} = \|d_w\|$.

The "chasing algorithm" is a variation of the chasing scheme described in [13], which reduces a symmetric arrowhead matrix to tridiagonal form. The orthogonal matrices Q_w and S_w will each consist of a product of Givens rotations. To satisfy (5), applying these rotations to Σ_w needs to produce a matrix B_w in bidiagonal form. To satisfy (6), applying the Q_w row rotations to d_w needs to produce a multiple of e_ℓ. Thus applying the Q_w row rotations and S_w column rotations to $[\Sigma_w \;\; d_w]$ should cause the following conversion:

$$[\Sigma_w \;\; d_w] = \begin{pmatrix} x & & & & x \\ & x & & & x \\ & & \ddots & & \vdots \\ & & & x & x \\ & & & x & x \end{pmatrix} \Rightarrow \begin{pmatrix} x & x & & & & \\ & x & x & & & \\ & & \ddots & \ddots & & \\ & & & x & x & \\ & & & & x & x \end{pmatrix}$$

Demonstrating the algorithm for the case of $\ell = 4$ should be sufficient for illustrating the algorithm for any ℓ. In the case of $\ell = 4$,

$$[\Sigma_w \;\; d_w] = \begin{pmatrix} x & & & x \\ & x & & x \\ & & x & x \\ & & & x \end{pmatrix}$$

The entry at $d_w(1)$ must be converted to a zero. To place a zero at $d_w(1)$, apply a Givens rotation to rows 1 and 2:

$$\begin{pmatrix} x & & & x \\ & x & & x \\ & & x & x \\ & & & x \end{pmatrix} \xrightarrow{\text{rows 1,2}} \begin{pmatrix} \boldsymbol{x} & \boldsymbol{x} & & \boldsymbol{0} \\ \boldsymbol{x} & \boldsymbol{x} & & \boldsymbol{x} \\ & & x & x \\ & & & x \end{pmatrix}$$

This created an undesired nonzero entry at row 2 column 1. That entry can be eliminated by rotating columns 1 and 2:

$$\begin{pmatrix} x & x & & & \\ x & x & & & x \\ & & x & & x \\ & & & x & x \end{pmatrix} \xrightarrow{\text{cols 1,2}} \begin{pmatrix} \boldsymbol{x} & \boldsymbol{x} & & & \\ \mathbf{0} & \boldsymbol{x} & & & x \\ & & x & & x \\ & & & x & x \end{pmatrix}$$

Placing a zero at $d_w(2)$ involves rotating rows 2 and 3:

$$\begin{pmatrix} x & x & & & \\ & x & & & x \\ & & x & & x \\ & & & x & x \end{pmatrix} \xrightarrow{\text{rows 2,3}} \begin{pmatrix} x & x & & & \\ & \boldsymbol{x} & \boldsymbol{x} & & \mathbf{0} \\ & \boldsymbol{x} & \boldsymbol{x} & & \boldsymbol{x} \\ & & & x & x \end{pmatrix}$$

Eliminating the undesired bulge at row 3 column 2 involves three rotations that "chase" the bulge out of the matrix:

$$\begin{pmatrix} x & x & & & \\ & x & x & & \\ & x & x & & x \\ & & & x & x \end{pmatrix} \xrightarrow{\text{cols 2,3}} \begin{pmatrix} x & \boldsymbol{x} & \boldsymbol{x} & & \\ & \boldsymbol{x} & \boldsymbol{x} & & \\ & \mathbf{0} & \boldsymbol{x} & & x \\ & & & x & x \end{pmatrix}$$

$$\xrightarrow{\text{rows 1,2}} \begin{pmatrix} \boldsymbol{x} & \boldsymbol{x} & \mathbf{0} & & \\ \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & & \\ & & x & & x \\ & & & x & x \end{pmatrix} \xrightarrow{\text{cols 1,2}} \begin{pmatrix} \boldsymbol{x} & \boldsymbol{x} & & & \\ \mathbf{0} & \boldsymbol{x} & x & & \\ & & x & & x \\ & & & x & x \end{pmatrix}$$

The final entry to eliminate is $d_w(3)$. Rotating rows 3 and 4 yields:

$$\begin{pmatrix} x & x & & & \\ & x & x & & \\ & & x & & x \\ & & & x & x \end{pmatrix} \xrightarrow{\text{rows 3,4}} \begin{pmatrix} x & x & & & \\ & x & x & & \\ & & \boldsymbol{x} & \boldsymbol{x} & \mathbf{0} \\ & & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \end{pmatrix}$$

The bulge at row 4 column 3 can be eliminated with the following five rotations:

$$\begin{pmatrix} x & x & & & \\ & x & x & & \\ & & x & x & \\ & & x & x & x \end{pmatrix} \xrightarrow{\text{cols 3,4}} \begin{pmatrix} x & x & & & \\ & x & \boldsymbol{x} & \boldsymbol{x} & \\ & & \boldsymbol{x} & \boldsymbol{x} & \\ & & \mathbf{0} & \boldsymbol{x} & x \end{pmatrix}$$

$$\xrightarrow{\text{rows 2,3}} \begin{pmatrix} x & x & & & \\ & \boldsymbol{x} & \boldsymbol{x} & \mathbf{0} & \\ & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & \\ & & & x & x \end{pmatrix} \xrightarrow{\text{cols 2,3}} \begin{pmatrix} x & \boldsymbol{x} & \boldsymbol{x} & & \\ & \boldsymbol{x} & \boldsymbol{x} & & \\ & \mathbf{0} & \boldsymbol{x} & x & \\ & & & x & x \end{pmatrix}$$

$$\xrightarrow{\text{rows 1,2}} \begin{pmatrix} \boldsymbol{x} & \boldsymbol{x} & \mathbf{0} & & \\ \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & & \\ & & x & x & \\ & & & x & x \end{pmatrix} \xrightarrow{\text{cols 1,2}} \begin{pmatrix} \boldsymbol{x} & \boldsymbol{x} & & & \\ \mathbf{0} & \boldsymbol{x} & x & & \\ & & x & x & \\ & & & x & x \end{pmatrix}$$

This is the desired form of $[B_w \;\; \hat{\phi}_{\ell+1} e_\ell]$. Thus if Q_w is the product of all row rotations applied to $[\Sigma_w \;\; d_w]$, and S_w is the product of all column rotations applied to $[\Sigma_w \;\; d_w]$, then

$$Q_w^T [\Sigma_w \;\; d_w] \begin{pmatrix} S_w & 0 \\ 0 & 1 \end{pmatrix} = [B_w \;\; \hat{\phi}_{\ell+1} e_\ell]$$

which implies both (5) and (6).

3 Krylov-Schur Method

The following algorithm, based on [12], uses Golub-Kahan-Lanczos (GKL) bidiagonalization to compute a small number (e.g. 10) of singular values and associated singular vectors of a matrix. In practice, this algorithm would most commonly be used to compute the largest singular values, the smallest singular values, or the singular values within a specified interval. These desired singular values will be referred to as "wanted", and other singular values will be referred to as "unwanted". Unless otherwise specified, $\|\cdot\|$ refers to the vector/matrix 2-norm.

3.1 Golub-Kahan-Lanczos Bidiagonalization

Start with a matrix $X \in \mathbb{R}^{m\times n}$ $(m \geq n)$, and a starting vector $v_1 \in \mathbb{R}^n$ satisfying $\|v_1\| = 1$. GKL bidiagonalization is an iterative procedure defined as follows:

$$\gamma_1 u_1 = X v_1 \tag{7}$$
$$\phi_i v_i = X^T u_{i-1} - \gamma_{i-1} v_{i-1} \tag{8}$$
$$\gamma_i u_i = X v_i - \phi_i u_{i-1} \tag{9}$$

where $\|u_i\| = \|v_i\| = 1$. The vectors u_i and v_i are called left and right Lanczos vectors, respectively. A common choice for the starting vector v_1 is e_1 (the first column of the identity matrix).

The following factorization of X is produced by k iterations of GKL bidiagonalization (or Lanczos iterations):

$$XV_k = U_k B_k$$
$$X^T U_k = V_k B_k^T + \phi_{k+1} v_{k+1} e_k^T$$

where $U_k = [u_1, u_2, \ldots, u_k]$, $V_k = [v_1, v_2, \ldots, v_k]$, U_k and V_k are left orthogonal, and

$$B_k = \begin{pmatrix} \gamma_1 & \phi_2 & & & & \\ & \gamma_2 & \phi_3 & & & \\ & & \ddots & \ddots & & \\ & & & \gamma_{k-2} & \phi_{k-1} & \\ & & & & \gamma_{k-1} & \phi_k \\ & & & & & \gamma_k \end{pmatrix}.$$

One way to compute the SVD of X would be to run $k = n$ Lanczos iterations. This would produce the factorization (2), from which the SVD of X can be generated using (4). If the objective is to compute a small number (e.g. 10) of singular values, then computing the full SVD of X would be unnecessarily expensive. For very large n, running $k = n$ Lanczos iterations wouldn't even be feasible. A better solution would be to choose $k \ll n$. Running k Lanczos iterations would yield the low-rank approximation

$$X \approx U_k B_k V_k^T = U_k (Q \Sigma S^T) V_k^T = (U_k Q) \Sigma (V_k S)^T.$$

As k increases, the singular values/vectors of $U_k B_k V_k^T$ reveal some singular values/vectors of X. Thus to obtain a small number of singular values/vectors, using $k \ll n$ would suffice, and would be more efficient than computing the full SVD using $k = n$ Lanczos iterations.

3.2 Reduction Step

As k increases, the sizes of $U_k \in \mathbb{R}^{m \times k}$ and $V_k \in \mathbb{R}^{n \times k}$ increase. Thus for large k, U_k and V_k require a lot of memory. In addition, the memory for U_k and V_k may need to be dynamically reallocated as k increases. To alleviate these issues, Baglama and Reichel [2] suggest periodically pausing GKL bidiagonalization, reducing the sizes of U_k, B_k, and V_k, and restarting bidiagonalization. This method is described below.

3.2.1 Computing Singular Values and Vectors

GKL bidiagonalization is paused once k reaches a predetermined capacity K. At this point, bidiagonalization has produced the following factorization:

$$X V_k = U_k B_k \tag{10}$$

$$X^T U_k = V_k B_k^T + \phi_{k+1} v_{k+1} e_k^T \tag{11}$$

The QR algorithm [7, §7.5] can quickly produce the SVD of B_k:

$$B_k = Q \Sigma S^T$$

Define

$$P_k = U_k Q, \tag{12}$$

$$Z_k = V_k S, \tag{13}$$

$$d = \phi_{k+1} Q^T e_k. \tag{14}$$

Multiplying (10) by S from the right and multiplying (11) by Q from the right yields

$$XZ_k = P_k\Sigma, \tag{15}$$

$$X^T P_k = Z_k\Sigma + v_{k+1}d^T. \tag{16}$$

The entries of the diagonal matrix $\Sigma \in \mathbb{R}^{k\times k}$ are the singular values of B_k (and the approximations of the singular values of X). The columns of $P_k \in \mathbb{R}^{m\times k}$ and $Z_k \in \mathbb{R}^{n\times k}$ are the approximations of the left and right singular vectors of X, respectively. Note that if for some i, $d(i) = 0$, then the approximations are exact (i.e. $\Sigma(i,i)$ is a singular value of X, while p_i and z_i are corresponding singular vectors of X). Thus if $d(i)$ is close enough to 0, one can conclude that the corresponding singular value has converged to a singular value of X. These singular values will be referred to as "converged".

3.2.2 Discarding Unwanted Singular Values

Recall that some of the singular values are considered "wanted" (e.g. the six largest singular values, the six smallest singular values, or the singular values within a specified interval), and the remaining singular values are considered "unwanted". The singular values can be partitioned such that

$$\Sigma = \begin{pmatrix} \Sigma_c & & \\ & \Sigma_w & \\ & & \Sigma_u \end{pmatrix}$$

where Σ_c contains converged singular values, Σ_w contains the unconverged "wanted" singular values, and Σ_u contains the unconverged "unwanted" singular values. Similar partitions can be formed for the left singular vectors $P_k = [P_c \;\; P_w \;\; P_u]$, the right singular vectors $Z_k = [Z_c \;\; Z_w \;\; Z_u]$, and

$$d = \begin{pmatrix} d_c \\ d_w \\ d_u \end{pmatrix}.$$

The columns of Z_c (the converged right singular vectors) must be saved in order to ensure that future columns of V_k are orthogonal to Z_c (see section 4.1). The "unwanted" singular values and vectors can be discarded. After recording Z_c and the converged "wanted" singular values and vectors, the

converged singular values and vectors can also be discarded. This leaves the unconverged "wanted" singular values and vectors, reducing (15) and (16) to

$$XZ_w = P_w\Sigma_w, \tag{17}$$
$$X^TP_w = Z_w\Sigma_w + v_{k+1}d_w^T. \tag{18}$$

3.2.3 Restarting GKL bidiagonalization

The "chasing algorithm" (see Section 2) can be applied to $\Sigma_w \in \mathbb{R}^{\ell\times\ell}$ and $d_w \in \mathbb{R}^\ell$ in order to generate orthogonal matrices $Q_w \in \mathbb{R}^{\ell\times\ell}$ and $S_w \in \mathbb{R}^{\ell\times\ell}$ that satisfy

$$Q_w^T\Sigma_w S_w = B_w$$
$$Q_w^T d_w = \hat{\phi}_{\ell+1}e_\ell$$

where $B_w \in \mathbb{R}^{\ell\times\ell}$ is bidiagonal, and $\hat{\phi}_{\ell+1} = \|d_w\|$.

Let $U_w = P_wQ_w$ and $V_w = Z_wS_w$. Multiplying (17) by S_w from the right and multiplying (18) by Q_w from the right yields

$$XV_w = U_wB_w,$$
$$X^TU_w = V_wB_w^T + \hat{\phi}_{\ell+1}v_{k+1}e_\ell^T.$$

Redefine u_i and v_i to be the i-th columns of U_w and V_w, respectively. Also redefine γ_i and ϕ_i to be the corresponding entries of B_w. This sets up a new instance of GKL bidiagonalization (with ℓ Lanczos iterations already completed). Letting $\phi_{\ell+1} \stackrel{def}{=} \hat{\phi}_{\ell+1}$ and $v_{\ell+1} \stackrel{def}{=} v_{k+1}$, GKL bidiagonalization can be resumed with the $(\ell+1)$-th Lanczos iteration.

3.2.4 Implementation Note

Many of the columns of P_k and Z_k are never used (e.g. the columns of P_u and Z_u), so an implementation of this algorithm should only compute the columns of P_k and Z_k that are needed. Let $\hat{Q}$ and $\hat{S}$ contain the "wanted" columns of Q and S, satisfying $P_w = U_k\hat{Q}$ and $Z_w = V_k\hat{S}$. To compute U_w and V_w efficiently and avoid computing P_w and Z_w directly, observe that

$$U_w = P_wQ_w = (U_k\hat{Q})Q_w = U_k(\hat{Q}Q_w),$$
$$V_w = Z_wS_w = (V_k\hat{S})S_w = V_k(\hat{S}S_w).$$

Since $m \gg k \geq \ell$, computing U_w as $U_w = U_k(\hat{Q}Q_w)$ is more efficient than computing P_w directly. Similarly, computing V_w as $V_w = V_k(\hat{S}S_w)$ is preferable.

4 Reorthogonalization

4.1 Reorthogonalizing the Right Lanczos Vectors

Although theoretically U_k and V_k should be orthogonal, in numerical computation the column vectors of U_k and V_k lose orthogonality very quickly [2]. One solution would be to reorthogonalize the columns of both U_k and V_k. A computationally less expensive solution proposed in [10] would be to reorthogonalize just one of them (say V_k). This solution will be explained below.

Recall that (8) defines the right Lanczos vector v_{k+1} as

$$\phi_{k+1} v_{k+1} = X^T u_k - \gamma_k v_k.$$

Let $r_0 = X^T u_k - \gamma_k v_k$. To ensure that V_{k+1} is orthogonal, r_0 must be reorthogonalized against the existing k columns of V_k. Note that each time GKL bidiagonalization is restarted (see Section 3.2), many right Lanczos vectors are discarded. To maintain orthogonality during a restarted bidiagonalization routine, Barlow [3] describes complete reorthogonalization, a method in which r_0 is reorthogonalized against all previously computed right Lanczos vectors. This can be a costly strategy since the history of right Lanczos vectors can get to be very large. Barlow [3] notes that to avoid reconverging to previously found singular values, r_0 only needs to be reorthogonalized against all previously found right singular vectors (the columns of Z_c) [8, 9]. Combining V_k and Z_c into a matrix $G = [V_k \ Z_c]$, r_0 must now be reorthogonalized against the columns of G. The reorthogonalized vector r_i will be orthogonal to G, and $\phi_{k+1} v_{k+1}$ will be set equal to r_i. The vector r_i can be computed using Gram-Schmidt reorthogonalization as shown in [3] and described below.

For $i \geq 1$, take the orthogonal decomposition of r_{i-1} with respect to G to get

$$r_{i-1} = r_i + G(G^T r_{i-1})$$

where r_i is theoretically (but not necessarily numerically) orthogonal to G. So

$$r_i = r_{i-1} - G(G^T r_{i-1}) = (I - GG^T) r_{i-1}. \tag{19}$$

Let $\zeta = \|I - G^T G\|$, so ζ is a measure of the orthogonality of G. To determine whether r_i is sufficiently orthogonal to G, check if $\|G^T r_i\|$ is smaller than some threshold dependent on ζ. Letting $0.5\zeta\|r_i\|$ be the threshold, if

$$\|G^T r_i\| \leq 0.5\zeta\|r_i\| \tag{20}$$

then r_i is sufficiently orthogonal to G, and r_i can be accepted as the reorthogonalized vector (i.e. $\phi_{k+1} v_{k+1} \stackrel{def}{=} r_i$).

Since computing ζ involves computing a matrix norm, it may be preferable to use a similar but less expensive test of orthogonality for r_i. Taking the norm of the orthogonal decomposition of r_{i-1} yields

$$\|r_{i-1}\|^2 = \|r_i\|^2 + \|G^T r_{i-1}\|^2$$

which implies that

$$\|G^T r_{i-1}\| = \sqrt{\|r_{i-1}\|^2 - \|r_i\|^2}.$$

Multiplying (19) by G^T yields

$$\begin{aligned} G^T r_i &= G^T(I - GG^T) r_{i-1} \\ &= (G^T - G^T G G^T) r_{i-1} \\ &= (I - G^T G) G^T r_{i-1}. \end{aligned}$$

Take the norm of $G^T r_i$ to get

$$\|G^T r_i\| \leq \|I - G^T G\| \|G^T r_{i-1}\| = \zeta \sqrt{\|r_{i-1}\|^2 - \|r_i\|^2}.$$

Reusing the threshold $0.5\zeta\|r_i\|$ from (20) yields $\zeta \sqrt{\|r_{i-1}\|^2 - \|r_i\|^2} \leq 0.5\zeta\|r_i\|$ which simplifies to

$$\|r_i\| \geq \sqrt{0.8}\|r_{i-1}\|. \tag{21}$$

If (21) is satisfied, then r_i is accepted as the reorthogonalized vector. Since (21) is less expensive than computing ζ explicitly, it is the preferred method for testing the orthogonality of r_i.

4.2 Loss of Orthogonality in Left Lanczos Vectors

Since the columns of U_k are not explicitly reorthogonalized during GKL bidiagonalization, the left singular vectors ($P_k = U_k Q$ from (12)) may have lost accuracy. Constructing a reorthogonalized U_k called $\widehat{U}_k$ and redefining P_k as

$$P_k = \widehat{U}_k Q \tag{22}$$

would yield more accurate left singular vectors, as shown in [3].

To construct $\widehat{U}_k$, first define Householder transformations W_j where $W_j = I - w_j w_j^T$ and

$$w_j = \begin{matrix} n \\ m \end{matrix} \begin{pmatrix} -e_j \\ u_j \end{pmatrix}.$$

Let $\widehat{W}_k = W_1 W_2 \cdots W_k \in \mathbb{R}^{(m+n)\times(m+n)}$. The following equation is stated and proven in [3]:

$$\begin{pmatrix} 0_{n\times k} \\ X V_k \end{pmatrix} = \widehat{W}_k \begin{pmatrix} B_k \\ 0_{(m+n-k)\times k} \end{pmatrix}. \tag{23}$$

Taking the first k columns and last m rows of (23) yields

$$X V_k = \widehat{U}_k B_k$$

where $\widehat{U}_k = \widehat{W}_k$ ((n+1):(m+n), 1:k) in MATLAB notation (so $\widehat{U}_k \in \mathbb{R}^{m\times k}$). Instead of computing P_k explicitly, one can extend (22) to

$$\begin{pmatrix} C_{n\times k} \\ P_k \end{pmatrix} = \widehat{W}_k \begin{pmatrix} Q \\ 0_{(m+n-k)\times k} \end{pmatrix} \tag{24}$$

and apply the k Householder transformations of $\widehat{W}_k = W_1 W_2 \cdots W_k$ one at a time. To do this, define

$$P^{(k+1)} = 0_{m\times k} \tag{25}$$

$$P^{(j)} = P^{(j+1)} - u_j (u_j^T P^{(j+1)} - Q(j,:)) \tag{26}$$

$$Q^{(j+1)} = \begin{pmatrix} Q(1:j,:) \\ 0_{(n-j)\times k} \\ P^{(j+1)} \end{pmatrix} \tag{27}$$

Applying the Householder transformation W_j to $Q^{(j+1)}$ yields

$$\begin{aligned}
W_j Q^{(j+1)} &= (I - w_j w_j^T) Q^{(j+1)} \\
&= Q^{(j+1)} - w_j w_j^T Q^{(j+1)} \\
&= Q^{(j+1)} - w_j (u_j^T P^{(j+1)} - Q(j,:)) \\
&= \begin{pmatrix} Q(1:(j-1),:) \\ Q(j,:) \\ 0_{(n-j)\times k} \\ P^{(j+1)} \end{pmatrix} - \begin{pmatrix} 0_{(j-1)\times k} \\ Q(j,:) \\ 0_{(n-j)\times k} \\ u_j (u_j^T P^{(j+1)} - Q(j,:)) \end{pmatrix} \\
&= \begin{pmatrix} Q(1:(j-1),:) \\ 0_{(n-j+1)\times k} \\ P^{(j)} \end{pmatrix} \\
&= Q^{(j)}.
\end{aligned}$$

Note that

$$Q^{(k+1)} = \begin{pmatrix} Q \\ 0_{(m+n-k)\times k} \end{pmatrix}$$

so by induction, $W_1 W_2 \cdots W_k Q^{(k+1)} = Q^{(1)}$. Combining this with (24) yields $P_k = P^{(1)}$. Thus (26) provides an inexpensive recurrence that generates $P_k = \widehat{U}_k Q$ without explicitly computing $\widehat{U}_k$. A simple MATLAB implementation of (26) is shown in [3] and provided below:

```
function P_k = UvecsMult(U_k, Q)
    [m, k] = size(U_k);
    P_k = U_k(:, k) * Q(k, :);
    for i = k - 1 : -1 : 1
        g = U_k(:, i)' * P_k;
        P_k = P_k - U_k(:, i) * (g - Q(i, :));
    end;
end;
```

5 Numerical Experiments

The following numerical tests were performed using MATLAB R2013a Student Edition (32-bit) on a Surface Pro running Windows 8.1 (64-bit).

5.1 Usefulness of Restarting Bidiagonalization

Periodically restarting GKL bidiagonalization has its pros and cons. Since computing the singular values of a large matrix may take hundreds of Lanczos iterations, frequent restarts can keep U_k and V_k from getting too large. This not only makes reorthogonalization faster, but also allows a greater portion of these matrices to reside in fast memory. However, the overhead of restarting bidiagonalization could conceivably outweigh these benefits. A numerical experiment was performed to confirm that restarting bidiagonalization can improve execution time.

The following test matrices were created in MATLAB:

$$\begin{aligned} A &= \text{sprandn}(40000, 40000, 0.001); \\ B_0 &= \text{delsq}(\text{numgrid}('\text{S}', 120)); \\ B &= \text{chol}(B_0(\text{amd}(B_0), \text{amd}(B_0))); \\ C &= \text{randn}(8000, 8000); \end{aligned}$$

$A \in \mathbb{R}^{40000\times 40000}$ was a random sparse matrix, $B \in \mathbb{R}^{13924\times 13924}$ was the Cholesky factor of a permutation of the Laplacian matrix, and $C \in \mathbb{R}^{8000\times 8000}$ was a random dense matrix. Restarted and non-restarted bidiagonalization were used to compute the 6 largest singular values of each matrix. The restarted approach paused bidiagonalization whenever k reached the capacity $K = 36$, guaranteeing that at least $K - 6 = 30$ Lanczos iterations would be executed between restarts. The non-restarted approach checked the 6 largest singular values for convergence every 30 Lanczos iterations. The following execution times were recorded:

	With Restarts	Without Restarts
A (sparse)	7.7 sec	20.0 sec
B (sparse)	3.5 sec	21.6 sec
C (dense)	16.3 sec	18.4 sec

Table 1: Restarted versus Non-Restarted Bidiagonalization

For the sparse matrices, restarted bidiagonalization clearly outperformed its non-restarted counterpart. The dense matrix yielded less convincing results, showing how restarted bidiagonalization is optimal for large sparse matrices.

5.2 Optimal Frequency of Krylov-Schur Restarts

Section 3.2 begins by describing how GKL bidiagonalization is stopped after k (the number of columns in U_k and V_k) reaches some predetermined capacity K. Choosing the right capacity can improve the performance of the algorithm, and the choice is non-trivial. Intuitively, if K is too small, then bidiagonalization would be stopped and restarted more often, and the overhead of these frequent restarts would be high. If K is too large, then both reorthogonalizing the Lanczos vectors and restarting bidiagonalization would involve operations with large matrices, also adversely affecting performance. This implies the possible existence of a "sweet spot" where K yields the lowest execution time. This optimal K likely depends on many factors, including the computer's architecture, the target matrix, and the number of desired singular values. A numerical experiment was performed to measure K's effect on execution time.

The following test matrices were created in MATLAB:

$$\begin{aligned} D_0 &= \text{delsq}(\text{numgrid}('\text{S}', 150)); \\ D &= \text{chol}(D_0(\text{amd}(D_0), \text{amd}(D_0))); \\ E &= \text{sprandn}(50000, 40000, 0.001); \\ F &= \text{randn}(8000, 8000); \end{aligned}$$

$D \in \mathbb{R}^{21904 \times 21904}$ was the Cholesky factor of a permutation of the Laplacian matrix, $E \in \mathbb{R}^{50000 \times 40000}$ was a random sparse matrix, and $F \in \mathbb{R}^{8000 \times 8000}$ was a random dense matrix. Various values of K were used to compute the six largest singular values, and the execution times were recorded.

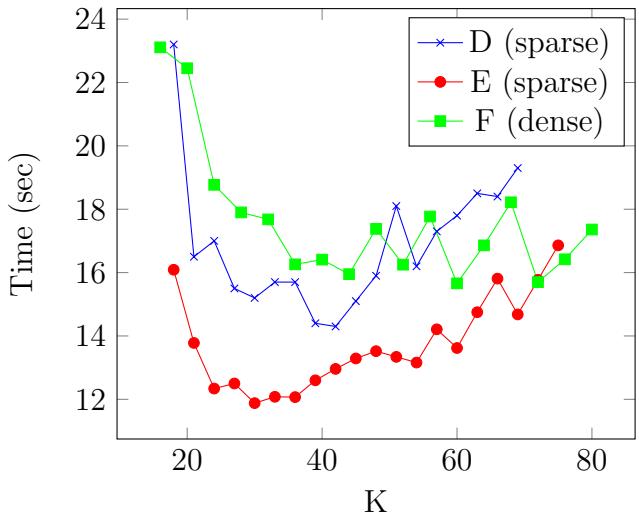

Figure 1: Effect of Capacity (K) on Execution Time

For the sparse matrices D and E, the optimal capacity lied roughly between 30 and 40. Performance was clearly hindered for K less than 20 or greater than 60, supporting the existence of a range of optimal capacities. For the dense matrix F, the execution time was worse for sufficiently small K but not for sufficiently large K. For K greater than 40, changes in K seem to have little impact on performance. This challenges the existence of an optimal capacity, but is consistent with the results of the dense matrix C in section 5.1, where restarts did not significantly improve performance. Thus a range of optimal capacities might only exist for sparse matrices.

6 Conclusion

The Krylov-Schur method uses GKL bidiagonalization to compute a few singular values of a matrix. To conserve memory and improve performance, bidiagonalization is periodically restarted and converged/unwanted singular values are periodically removed. In numerical computation, the Lanczos vectors tend to lose orthogonality very quickly. To correct for this, the right Lanczos vectors are explicitly reorthogonalized, and computations involving the left Lanczos vectors are adjusted.

Restarting bidiagonalization seems to improve the performance of the algorithm, especially when targeting large sparse matrices. Numerical evidence suggests that a range of optimal capacities (maximum number of Lanczos vectors stored) exists, where the overhead of applying a Krylov-Schur restart is minimized and the benefits of restarting bidiagonalization are maximized. Ultimately, numerical experiments have shown how the Krylov-Schur method performs well for large sparse matrices.

References

[1] Abdallah, A., Hu, Y., Parallel VLSI computing array implementation for signal subspace updating algorithm. IEEE Trans. Acoust., Speech, Signal Processing 37, 742-748 (1989)

[2] Baglama, J., Reichel L., Augmented implicitly restarted Lanczos bidiagonalization methods. SIAM J. Sci. Comput. 27 (1), 19–42 (2005)

[3] Barlow, J.L., Reorthogonalization for the Golub–Kahan–Lanczos bidiagonal reduction. Numerische Mathematik 124 (2), 237-278 (2013)

[4] Comon, P., Golub, G.H., Tracking a few extreme singular values and vectors in signal processing. Proc. IEEE, 78 1327–1343 (1990)

[5] Geladi, P., Grahn, H., Multivariate Image Analysis. John Wiley, New York, 1996.

[6] Golub, G.H., Kahan, W.M., Calculating the singular values and pseudoinverse of a matrix. SIAM J. Numer. Anal. Ser. B 2, 205–224 (1965)

[7] Golub, G.H., Van Loan, C.F., Matrix Computations, 4th Edition. The Johns Hopkins University Press, 2013.

[8] Paige, C.C., The Computation of Eigenvalues and Eigenvectors of Very Large Sparse Matrices. PhD thesis, University of London, 1971.

[9] Parlett, B.N., Scott, D.S., The Lanczos algorithm with selective reorthogonalization. Math. Comput. 33, 217–238 (1979)

[10] Simon, H., Zha, H., Low rank matrix approximation using the Lanczos bidiagonalization process. SIAM J. Sci. Stat. Comput. 21, 2257–2274 (2000)

[11] Stewart, G., A Krylov–Schur algorithm for large eigenproblems. SIAM J. Matrix Anal. Appl. 23 (4), 601–614 (2001)

[12] Stoll, M., A Krylov–Schur approach to the truncated SVD. Linear Algebra Appl. 436, 2795-2806 (2012)

[13] Zha, H., A Two-Way chasing scheme for reducing a symmetric arrowhead matrix to tridiagonal form. J. Numerical Linear Algebra with Applications 1, 49-57 (1992)

ACADEMIC VITA

Amuthan Narthana

EDUCATION

The Pennsylvania State University, University Park, PA — May 2014
Bachelor of Science in Computer Science
Bachelor of Science in Mathematics
Honors in Computer Science

EXPERIENCE AND HONORS

Software Development Engineer Intern — May-August 2013
Microsoft in Redmond, WA

Co-op Application Developer — January-July 2012
IBM Rational in Littleton, MA

ITS Lab Consultant — 2011-2013
The Pennsylvania State University in University Park, PA

Evan Pugh Scholar Award — 2014

President's Freshman Award — 2011