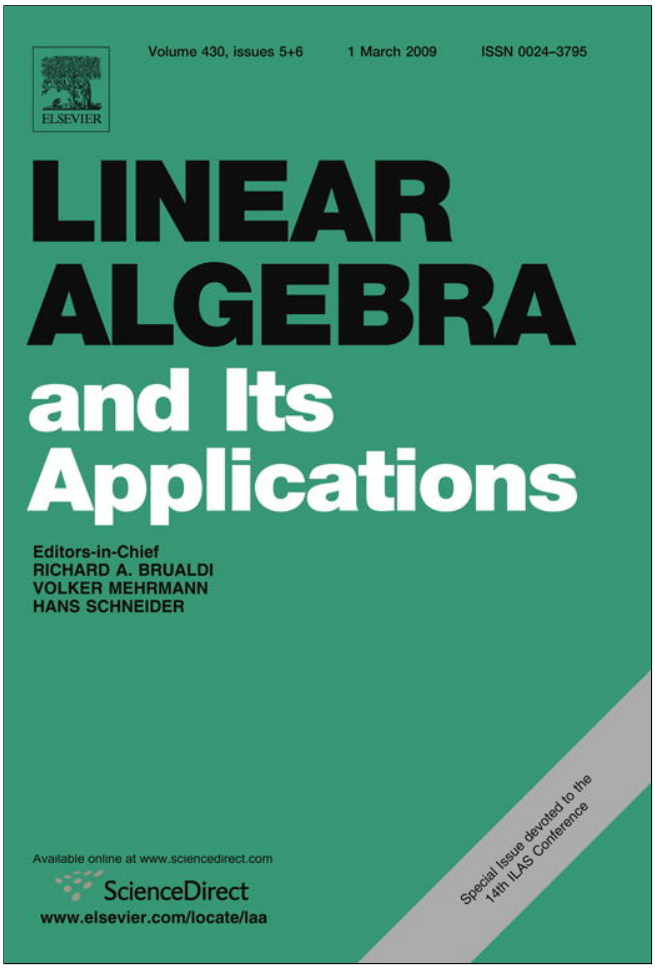

Available online at www.sciencedirect.com
ScienceDirect
Linear Algebra and its Applications 430 (2009) 1531–1543
LINEAR ALGEBRA AND ITS APPLICATIONS
www.elsevier.com/locate/laa

A Lanczos bidiagonalization algorithm for Hankel matrices☆

Kevin Browne [a], Sanzheng Qiao [a,*], Yimin Wei [b,c]

[a] *Department of Computing and Software, McMaster University, Hamilton, Ont., Canada L8S 4K1*
[b] *Institute of Mathematics, School of Mathematical Science, Fudan University, Shanghai 200433, PR China*
[c] *Key Laboratory of Nonlinear Science, Fudan University, Ministry of Education, PR China*

Received 31 October 2007; accepted 15 January 2008
Available online 4 March 2008
Submitted by Pei Yuan Wu

Abstract

This paper presents a fast algorithm for bidiagonalizing a Hankel matrix. An $m \times n$ Hankel matrix is reduced to a real bidiagonal matrix in $\mathcal{O}((m+n)n\log(m+n))$ floating-point operations (flops) using the Lanczos method with modified partial orthogonalization and reset schemes to improve its stability. Performance improvement is achieved by exploiting the Hankel structure, as fast Hankel matrix–vector multiplication is used. The accuracy and efficiency of the algorithm are demonstrated by our numerical experiments.

Keywords: Fast Hankel SVD; Hankel matrix; Lanczos bidiagonalization

1. Introduction

The singular value decomposition (SVD) of structured matrices such as the Hankel matrix plays an important role in signal processing, among other applications. Previous work has dealt with a special form of the SVD called the Takagi Factorization [10] of a square complex Hankel matrix. That is that for any square complex Hankel matrix A of order n, there exist a unitary $Q \in \mathbb{C}^{n\times n}$

☆ Sanzheng Qiao was supported in part by the Natural Sciences and Engineering Research Council of Canada and partially supported by Shanghai Key Laboratory of Contemporary Applied Mathematics of Fudan University. Yimin Wei is supported by the National Natural Science Foundation of China under Grant 10471027 and Shanghai Education Committee.
* Corresponding author.
E-mail addresses: brownek@mcmaster.ca (K. Browne), qiao@mcmaster.ca (S. Qiao), ymwei@fudan.edu.cn (Y. Wei).

0024-3795/$ - see front matter
doi:10.1016/j.laa.2008.01.012

and an order n nonnegative diagonal $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$, where $\sigma_1 \geqslant \sigma_2 \geqslant \cdots \sigma_n \geqslant 0$, such that

$$A = Q\Sigma Q^{\mathrm{T}} \quad \text{or} \quad Q^{\mathrm{H}} A \overline{Q} = \Sigma,$$

where $\overline{Q}$ is the complex conjugate of Q. This was computed in two stages: a Lanczos symmetric tridiagonalization followed by a symmetric Takagi Factorization of a symmetric tridiagonal matrix. In the general case, however, a Lanczos bidiagonalization is computed followed by the SVD of a real square bidiagonal matrix. In this paper, we will focus on the first step, the Lanczos bidiagonalization of a general m-by-n Hankel matrix where $m \geqslant n$ without loss of generality. There are well-known SVD algorithms for real bidiagonal matrices, such as the QR method [3], divide-and-conquer [4], and twisted [1,2] methods. In particular, the twisted method can efficiently compute both singular values and singular vectors, suitable for the second step in Hankel SVD.

We compute the real bidiagonal B of a general Hankel matrix A by finding $U \in \mathbb{C}^{m\times n}$ with orthonormal columns and unitary $V \in \mathbb{C}^{n\times n}$ such that

$$A = UBV^{\mathrm{H}}.$$

This is described in Section 2 using the Lanczos bidiagonalization algorithm [3]. Fast Hankel matrix–vector multiplication improves the performance of this algorithm by exploiting the Hankel structure.

As we know, re-orthogonalization is necessary for a practical Lanczos method. In the previous work dealing with complex symmetric square matrices, the Lanczos method has been improved upon by incorporating orthogonality estimate calculations, which are used to determine when vectors must be re-orthogonalized [8]. As these calculations were previously dependent upon the matrix involved being square and symmetric, of particular concern in this paper is how the orthogonality estimates used to determine when vectors must be re-orthogonalized are computed for the general case of rectangular Hankel matrices. The major difference is that in the symmetric case, only one set of vectors, namely the columns of Q, is dealt with, whereas in the general case two sets, the columns of U and V, must be considered. This is covered in Section 3.

Another important technique in the Lanczos algorithm is the reset, which may occur when dealing with multiple/clustered singular values. In the case of multiple/clustered singular values a small superdiagonal entry may be encountered, which would make the algorithm numerically unstable left as is. This requires a vector of the algorithm used to compute the superdiagonal elements in B and matrix V to be *reset* by generating a new vector and re-orthogonalizing the vector against all previously computed vectors.

In Section 4, we demonstrate these new techniques in the algorithm with numerical experiments to show the accuracy and efficiency of our algorithm. For square matrix input we compare the performance of the general Hankel matrix Lanczos algorithms with those specifically designed for square matrices.

2. Bidiagonalization

For any m-by-n, $m \geqslant n$, general matrix A we can find an m-by-n matrix U with orthonormal columns and an n-by-n unitary matrix V such that

$$B = U^{\mathrm{H}} A V \tag{1}$$

is real, upper bidiagonal and square. Let

$$B = \begin{bmatrix} \alpha_1 & \beta_1 & & \dots & 0 \\ & \ddots & \ddots & & \vdots \\ & & \ddots & \ddots & \\ \vdots & & & \ddots & \beta_{n-1} \\ 0 & \dots & & & \alpha_n \end{bmatrix} \tag{2}$$

and rewrite (1) as

$$AV = UB \tag{3}$$

and

$$A^{\mathrm{H}}U = VB^{\mathrm{T}}. \tag{4}$$

Comparing the jth columns of both sides of (3) and (4), we have

$$A\mathbf{v}_j = \alpha_j \mathbf{u}_j + \beta_{j-1}\mathbf{u}_{j-1}$$

and

$$A^{\mathrm{H}}\mathbf{u}_j = \alpha_j \mathbf{v}_j + \beta_j \mathbf{v}_{j+1},$$

which lead us to a Lanczos recursion:

$$\mathbf{r}_j = \alpha_j \mathbf{u}_j = A\mathbf{v}_j - \beta_{j-1}\mathbf{u}_{j-1}, \tag{5}$$

$$\mathbf{p}_j = \beta_j \mathbf{v}_{j+1} = A^{\mathrm{H}}\mathbf{u}_j - \alpha_j \mathbf{v}_j. \tag{6}$$

Orthonormality tells us that $\alpha_j = \pm\|\mathbf{r}_j\|_2$, $\beta_j = \pm\|\mathbf{p}_j\|_2$, and $\mathbf{u}_j = \mathbf{r}_j/\alpha_j$, $\mathbf{v}_{j+1} = \mathbf{p}_j/\beta_j$. Thus one can construct a Lanczos bidiagonalization algorithm for general matrices.

Algorithm 1 (*General Lanczos bidiagonalization* [3, *p.* 495]). Given a starting vector $\mathbf{r}$ and a subroutine for matrix–vector multiplication $\mathbf{y} = A\mathbf{x}$ for any $\mathbf{x}$, where A is an m-by-n, $m \geqslant n$, general matrix. This algorithm computes the diagonals of the real square upper bidiagonal matrix B in (1) and U, V such that $A = UBV^{\mathrm{H}}$.

$\mathbf{u}_0 = 0$; $\beta_0 = 1$;
$\mathbf{p} = \mathbf{r}/\|\mathbf{r}\|_2$;
for $j = 1$ to n
    $\mathbf{v}_j = \mathbf{p}/\beta_{j-1}$;
    $\mathbf{r} = A\mathbf{v}_j - \beta_{j-1}\mathbf{u}_{j-1}$;
    $\alpha_j = \|\mathbf{r}\|_2$;
    $\mathbf{u}_j = \mathbf{r}/\alpha_j$;
    if $j < n$
        $\mathbf{p} = A^{\mathrm{H}}\mathbf{u}_j - \alpha_j \mathbf{v}_j$;
        $\beta_j = \|\mathbf{p}_j\|_2$;
        if $\beta_j = 0$, quit; end
    end
end

As the most computationally expensive operation in the Lanczos method is matrix–vector multiplication, when A is a Hankel matrix, one can exploit the Hankel structure of A to improve performance [6].

The Hankel matrix structure is exploited by first changing the Hankel matrix to a Toeplitz matrix by reversing the columns. Given the first column $\bar{\mathbf{c}} = [c_1, \ldots, c_m]^{\mathrm{T}}$ and last row $\bar{\mathbf{r}} = [c_m, r_2, \ldots, r_n]$ of an m-by-n Hankel, the first column of the Toeplitz is

$$[c_n, \ldots, c_m, r_2, \ldots, r_n]^{\mathrm{T}}$$

and the first row of the Toeplitz is

$$[c_n, c_{n-1}, \ldots, c_1].$$

The Toeplitz matrix may then be expanded into a Toeplitz-circulant matrix C, whose first column is

$$\hat{\mathbf{c}} = [c_n, \ldots, c_m, r_2, \ldots, r_n, c_1, \ldots, c_{n-1}]^{\mathrm{T}}, \tag{7}$$

if $m \geqslant n$, otherwise it becomes

$$\hat{\mathbf{c}} = [r_{n-m+1}, \ldots, r_n, c_1, \ldots, c_m, r_2, \ldots, r_{n-m}]^{\mathrm{T}}. \tag{8}$$

One can now efficiently compute the matrix–vector product

$$\bar{\mathbf{p}} = A\bar{\mathbf{x}} \tag{9}$$

as follows, where $\bar{\mathbf{x}}$ is a given n-element vector:

$$\bar{\mathbf{x}} = [x_1, x_2, \ldots, x_n]^{\mathrm{T}}. \tag{10}$$

A new $(n + m - 1)$-length vector can now be created:

$$\hat{\mathbf{x}} = [x_n, x_{n-1}, \ldots, x_1, 0, \ldots, 0]^{\mathrm{T}}, \tag{11}$$

which is derived from $\bar{\mathbf{x}}$ by reversing its entries and appending $m - 1$ zeros. Given the product of the circulant matrix C and $\hat{\mathbf{x}}$:

$$\bar{\mathbf{y}} \equiv C\hat{\mathbf{x}}, \tag{12}$$

then $\bar{\mathbf{p}}$ is given by the first m elements of $\bar{\mathbf{y}}$.

How do we compute $C\hat{\mathbf{x}}$ efficiently? We remind the reader that the impressive efficiency with which $C\hat{\mathbf{x}}$ may be computed is based on the special spectral factorization of a circulant C. Let F denote the Fourier matrix of appropriate order and let $\hat{\mathbf{c}}$ denote the first column of C, then the circulant matrix–vector product

$$C\mathbf{x} = \mathrm{inv}(F)\mathrm{diag}(F\hat{\mathbf{c}})F\mathbf{x}.$$

Thus, in our case, the product $C\hat{\mathbf{x}}$ can be efficiently computed by the Fast Fourier Transform [5] (FFT):

$$\bar{\mathbf{y}} = \mathtt{ifft}(\mathtt{fft}(\hat{\mathbf{c}}) \dot{*} \mathtt{fft}(\hat{\mathbf{x}})).$$

Note that:

- $\mathtt{fft}(\mathbf{v})$ denotes a one-dimensional FFT of vector $\mathbf{v}$.
- $\mathtt{ifft}(\mathbf{v})$ denotes a one-dimensional inverse FFT of $\mathbf{v}$.
- $\dot{*}$ denotes a componentwise multiplication of two vectors.

Algorithm 2 (*General fast Hankel matrix–vector product*). Given a vector $\bar{\mathbf{x}}$ in (10) and the first column $\bar{\mathbf{c}}$ and last row $\bar{\mathbf{r}}$ of a Hankel matrix, this algorithm computes the product vector $\bar{\mathbf{p}}$ of (9) by using the fast Fourier transform.

% Construct the $(m+n-1)$-element vector $\hat{\mathbf{c}}$ as in Eq. (7) or (8)
if $m \geqslant n$
  $\hat{\mathbf{c}} = [c_n, \ldots, c_m, r_2, \ldots, r_n, c_1, \ldots, c_{n-1}]^{\mathrm{T}}$
else
  $\hat{\mathbf{c}} = [r_{n-m+1}, \ldots, r_n, c_1, \ldots, c_m, r_2, \ldots, r_{n-m}]^{\mathrm{T}}$
end
% Construct the $(m+n-1)$-element vector $\hat{\mathbf{x}}$ as in Eq. (11)
$\hat{\mathbf{x}} = [x_n, x_{n-1}, \ldots, x_{n-1}, 0, \ldots, 0]^{\mathrm{T}}$
% Compute the $(m+n-1)$-element vector $\bar{\mathbf{y}}$
$\bar{\mathbf{y}} = \mathtt{ifft}(\mathtt{fft}(\hat{\mathbf{c}}) .* \mathtt{fft}(\hat{\mathbf{x}}))$
% extract the desired m-element product vector $\bar{\mathbf{p}}$ from $\bar{\mathbf{y}}$
$\bar{\mathbf{p}} = [y_1, y_2, \ldots, y_m]^{\mathrm{T}}$

The complexity of this algorithm can be derived as follows. Note that an FFT of a vector size n costs $5n\log(n)$ flops (floating-point additions and multiplications). As such in Algorithm 2, each of the two FFT operations requires $5(m+n-1)\log(m+n-1)$ flops, the pointwise multiplication costs $6(m+n-1)$ flops, the inverse FFT costs $5(m+n-1)\log(m+n-1)$ flops. The total cost of computing $\mathtt{ifft}(\mathtt{fft}(\hat{\mathbf{c}}) .* \mathtt{fft}(\hat{\mathbf{x}}))$ is about $15(m+n)\log(m+n)+6(m+n)$ flops. As general complex matrix–vector multiplication involves $8mn$ flops, Algorithm 2 is more efficient than general matrix–vector multiplication when $m \geqslant n \geqslant 16$ [6].

To improve efficiency, in our implementation, when using Algorithm 2 to perform the Hankel matrix–vector products in the Lanczos Algorithms 1, 3 and 4, instead of generating the vector $\hat{\mathbf{c}}$ representing the Toeplitz-circulant matrix for each matrix–vector product, we compute $\hat{\mathbf{c}}$ once for each of A and A^{H} outside the loop when the algorithm is initiated. Then, inside the loop, only the remaining steps of Algorithm 2 are required at each iteration to perform the fast Hankel matrix–vector multiplication.

3. Orthogonalization

The Lanczos method given in Algorithm 1 suffers from loss of orthogonality of the computed U and V. In order to correct this problem, one could re-orthogonalize each computed $\mathbf{u}_j$ and $\mathbf{v}_j$ against all previously computed vectors $\mathbf{u}_i$ and $\mathbf{v}_i$, respectively, which is called complete orthogonalization. In the case of square complex matrices, a technique of selectively re-orthogonalizing vectors based on estimates of the orthogonalities of the Takagi vectors is presented in [8]. This has the effect of reducing the prohibitive cost of the complete orthogonalization, and it is extended to general matrices below.

As in the square case, we first establish recursions on the estimates for the orthogonalities of U and V. We can then base a partial re-orthogonalization algorithm on having these estimates.

Denoting the orthogonality measurements $\mu_{k,j} = \mathbf{u}_k^{\mathrm{H}}\mathbf{u}_j$ and $\nu_{k,j} = \mathbf{v}_k^{\mathrm{H}}\mathbf{v}_j$, we can construct recursions on them as follows. We first incorporate round-off error vectors $\mathbf{f}_j$ and $\mathbf{g}_k$ into (5) and (6):

$$\alpha_j\mathbf{u}_j = A\mathbf{v}_j - \beta_{j-1}\mathbf{u}_{j-1} - \mathbf{f}_j,$$
$$\beta_k\mathbf{v}_{k+1} = A^{\mathrm{H}}\mathbf{u}_k - \alpha_k\mathbf{v}_k - \mathbf{g}_k.$$

Premultiplying the above equations by $\mathbf{u}_k^{\mathrm{H}}$ and $\mathbf{v}_j^{\mathrm{H}}$, respectively, gives us

$$\alpha_j \mu_{k,j} = \mathbf{u}_k^{\mathrm{H}} A \mathbf{v}_j - \beta_{j-1} \mu_{k,j-1} - \mathbf{u}_k^{\mathrm{H}} \mathbf{f}_j,$$
$$\beta_k \nu_{j,k+1} = \mathbf{v}_j^{\mathrm{H}} A^{\mathrm{H}} \mathbf{u}_k - \alpha_k \nu_{j,k} - \mathbf{v}_j^{\mathrm{H}} \mathbf{g}_k.$$

We also note that $\mathbf{v}_j^{\mathrm{H}} A^{\mathrm{H}} \mathbf{u}_k$ and $\nu_{j,k}$ are complex conjugates of $\mathbf{u}_k^{\mathrm{H}} A \mathbf{v}_j$ and $\nu_{k,j}$, respectively, and α_j and β_j are real. Then, by subtracting the complex conjugate of the second equation from the first one above, we obtain a recursion on $\mu_{k,j}$:

$$\alpha_j \mu_{k,j} = \beta_k \nu_{k+1,j} + \alpha_k \nu_{k,j} - \beta_{j-1} \mu_{k,j-1} + \mathbf{g}_k^{\mathrm{H}} \mathbf{v}_j - \mathbf{u}_k^{\mathrm{H}} \mathbf{f}_j. \tag{13}$$

As the round-off terms $\mathbf{f}_j$ and $\mathbf{g}_k$ are unknown, we define $\theta_{k,j} := \mathbf{g}_k^{\mathrm{H}} \mathbf{v}_j - \mathbf{u}_k^{\mathrm{H}} \mathbf{f}_j$ for now.

Similarly, combining

$$\mathbf{u}_{j-1}^{\mathrm{H}} (\alpha_k \mathbf{u}_k) = \mathbf{u}_{j-1}^{\mathrm{H}} A \mathbf{v}_k - \mathbf{u}_{j-1}^{\mathrm{H}} (\beta_{k-1} \mathbf{u}_{k-1}) - \mathbf{u}_{j-1}^{\mathrm{H}} \mathbf{f}_k$$

and

$$\mathbf{v}_k^{\mathrm{H}} (\beta_{j-1} \mathbf{v}_j) = \mathbf{v}_k^{\mathrm{H}} A^{\mathrm{H}} \mathbf{u}_{j-1} - \mathbf{v}_k^{\mathrm{H}} (\alpha_{j-1} \mathbf{v}_{j-1}) - \mathbf{v}_k^{\mathrm{H}} \mathbf{g}_{j-1},$$

we can derive a recursion on $\nu_{k,j}$:

$$\beta_{j-1} \nu_{k,j} = \alpha_k \mu_{k,j-1} + \beta_{k-1} \mu_{k-1,j-1} - \alpha_{j-1} \nu_{k,j-1} + \mathbf{f}_k^{\mathrm{H}} \mathbf{u}_{j-1} - \mathbf{v}_k^{\mathrm{H}} \mathbf{g}_{j-1}. \tag{14}$$

Also, we define $\phi_{k,j-1} = \mathbf{f}_k^{\mathrm{H}} \mathbf{u}_{j-1} - \mathbf{v}_k^{\mathrm{H}} \mathbf{g}_{j-1}$. Note that $\phi_{k,j-1}$ is the complex conjugate of $\theta_{j-1,k}$. Since $\mathbf{f}_j$ is the rounding error introduced in the computation of $\alpha_j \mathbf{u}_j$, its norm is associated with α_j. Likewise, the norm of $\mathbf{g}_k$ is associated with β_k. Now, we are able to estimate $\theta_{k,j}$ and $\phi_{k,j-1}$ based on a statistical study [9] as

$$\theta_{k,j} = \epsilon(\beta_k + \alpha_j)(\Psi_r + \mathrm{i}\Psi_i) \quad \text{and} \quad \phi_{k,j-1} = \epsilon(\alpha_k + \beta_{j-1})(\Psi_r + \mathrm{i}\Psi_i), \tag{15}$$

where ϵ is the machine precision and Ψ_r, $\Psi_i \in N(0, 0.6)$ and $N(0, v)$ defines a normal distribution with zero mean and variance v.

Now when $\mu_{k,j+1}$ exceeds the threshold $\sqrt{\epsilon}$ for any $1 \leqslant k \leqslant j$, we re-orthogonalize $\mathbf{u}_{j+1}$ against all previously computed $\mathbf{u}_k$, $k = 1, \ldots, j$. Similarly when $\nu_{k,j+1}$ exceeds the threshold $\sqrt{\epsilon}$ for any k, we re-orthogonalize $\mathbf{v}_{j+1}$ against all previously computed $\mathbf{v}_k$, $k = 1, \ldots, j$. If re-orthogonalization is required of either $\mathbf{u}_{j+1}$ or $\mathbf{v}_{j+1}$ in a given iteration, then in the following iteration we will again perform re-orthogonalization of that same vector. In theory after a re-orthogonalization of $\mu_{k,j+1}$ we have that $\mu_{k,j+1} = 0$ for $k = 1, \ldots, j$, and after a re-orthogonalization of $\nu_{k,j+1}$ we have that $\nu_{k,j+1} = 0$ for $k = 1, \ldots, j$. In order to incorporate the rounding error after re-orthogonalization of $\nu_{k,j}$ or $\mu_{k,j}$, we set

$$\nu_{k,j+1} = \epsilon(\Psi_r + \mathrm{i}\Psi_i) \tag{16}$$

and

$$\mu_{k,j+1} = \epsilon(\Psi_r + \mathrm{i}\Psi_i), \tag{17}$$

where Ψ_r, $\Psi_i \in N(0, 1.5)$.

This gives us all the information we need to carry out the algorithm for the partial re-orthogonalization. Let us now outline how the algorithm proceeds in order to make clear how what we have derived above comes together.

The starting values are the same as in Algorithm 1, and in addition we initialize $\mu_{1,2} = \epsilon m(\Psi_r + \mathrm{i}\Psi_i)$ where Ψ_i, $\Psi_r \in N(0, 0.6)$, as we expect the first two vectors $\mathbf{u}_1$ and $\mathbf{u}_2$ have

good orthogonality. Similarly, we set $\nu_{1,2} = \epsilon n(\Psi_r + \mathrm{i}\Psi_i)$. Also, apparently, from the definitions, $\mu_{j,j} = \nu_{j,j} = 1$ for all j. The algorithm then proceeds through its first iteration exactly as Algorithm 1 (as of course we do not need re-orthogonalization at this point). However, in the second iteration, $j = 2$, since we have initialized $\mu_{1,2}$ and $\mu_{2,2} = 1$, we only compute $\nu_{1,3}$ and $\nu_{2,3}$ using (14), where $\mu_{0,2} = 0$. From the third iteration forward, right after we compute α_j and β_j, we now compute (13) and (14), which give us the error estimates for $\mu_{k,j}$ and $\nu_{k,j+1}$, respectively. Then as described above, we check for the possibility that an element of either μ or ν has exceeded an error threshold $\sqrt{\epsilon}$. If this is the case we perform re-orthogonalization of $\mathbf{u}_j$ or $\mathbf{v}_{j+1}$ on both this iteration and the next. If we have performed re-orthogonalization we must also set $\mu_{k,j}$ and/or $\nu_{k,j+1}$ as in (16) and (17). Also note that in the case we have performed re-orthogonalization we must again compute α_j or β_j to reflect the re-orthogonalized values. The algorithm we have just described is given in pseudo-code below.

Algorithm 3 (*Lanczos bidiagonalization with general partial orthogonalization*). Given a starting vector $\mathbf{r}$ and a subroutine for matrix–vector multiplication $\mathbf{y} = A\mathbf{x}$ for any $\mathbf{x}$, where A is an m-by-n general matrix. This algorithm computes the diagonals of the square bidiagonal matrix B in (1) and U, V such that $B = UAV^{\mathrm{H}}$. It re-orthogonalizes $\mathbf{u}_i$ and $\mathbf{v}_i$ when loss of orthogonality is detected.

$\mathbf{u}_0 = 0$; $\beta_0 = 1$; $\alpha_0 = 0$;
$\mathbf{p} = \mathbf{r}/\|\mathbf{r}\|_2$;
Initialize $\mu_{1,2}$, $\mu_{2,2}$, $\nu_{1,2}$, $\nu_{2,2}$;
for $j = 1$ to n
    $\mathbf{v}_j = \mathbf{p}/\beta_{j-1}$;
    $\mathbf{r} = A\mathbf{v}_j - \beta_{j-1}\mathbf{u}_{j-1}$;
    $\alpha_j = \|\mathbf{r}\|_2$;
    if $j > 2$
        Compute $\mu_{k,j}$ for $k = 1, 2, \ldots, j-1$ using (13);
        Set $\mu_{j,j} = 1.0$;
        if $\mathtt{max}_{1 \leqslant k \leqslant j-1}(|\mu_{k,j}|) > \sqrt{\epsilon}$
            Orthogonalize $\mathbf{r}$ against $\mathbf{u}_1, \ldots, \mathbf{u}_{j-1}$;
            Perform orthogonalization in the next iteration;
            Set $\mu_{k,j}$ using (17);
            Re-calculate $\alpha_j = \|\mathbf{r}\|_2$;
        end
    end
    $\mathbf{u}_j = \mathbf{r}/\alpha_j$;
    if $j < n$
        $\mathbf{p} = A^{\mathrm{H}}\mathbf{u}_j - \alpha_j\mathbf{v}_j$;
        $\beta_j = \|\mathbf{p}\|_2$;
        if $j > 2$
            Compute $\nu_{k,j+1}$ for $k = 1, 2, \ldots, j$ using (14), where $\mu_{0,j} = 0$;

```
            Set ν_{j+1,j+1} = 1.0;
        if max_{1≤k≤j}(|ν_{k,j+1}|) > √ε
            Orthogonalize p against v_1, ..., v_j;
            Perform orthogonalization in the next iteration;
            Set ν_{k,j+1} using (16);
            Re-calculate β_j = ‖p‖_2;
        end
    end
end
```

This algorithm can be taken a step forward by re-orthogonalizing $\mathbf{r}$ and $\mathbf{p}$ against only some selected $\mathbf{u}_j$ and $\mathbf{v}_j$ instead of all previously computed vectors. In particular, subintervals are identified as being in need of re-orthogonalization. The algorithm is the same as Algorithm 3, except instead of re-orthogonalizing all vectors when μ or ν indicates that we should, we search for the intervals of the index k where the value $\mu_{k,j}$ or $\nu_{k,j}$ is greater than, say, $\sqrt{\epsilon^3}$. We refer to this technique as *modified partial orthogonalization*. It is shown in the following Algorithm 4.

When dealing with multiple/clustered singular values, we may encounter a small superdiagonal entry β_j causing numerical problems. This indicates that $\mathbf{p}$ lies in an invariant subspace. Thus we reset the vector $\mathbf{p}$ when this occurs by generating a new random $\mathbf{p}$ and re-orthogonalizing it against all previous $\mathbf{v}_j$.

The important question becomes: what do we consider a small subdiagonal? Based on our experiments, we choose the tolerance:

$$\text{RSTOL} = \sqrt{\epsilon}(\|A\|_{\text{F}}/(mn)), \tag{18}$$

$\|M\|_{\text{F}}$ is the Frobenius norm of a matrix M.

With this information in hand, we present a new algorithm that incorporates modified partial orthogonalization and the reset technique.

Algorithm 4 (*Lanczos bidiagonalization with modified partial orthogonalization and reset*). Given a starting vector $\mathbf{r}$ and a subroutine for matrix–vector multiplication $\mathbf{y} = A\mathbf{x}$ for any $\mathbf{x}$, where A is an m-by-n general matrix. This algorithm computes the diagonals of the square bidiagonal matrix B in (1) and U, V such that $B = UAV^{\text{H}}$. This algorithm incorporates the reset technique such that when a small β is detected $\mathbf{p}$ is reset. As well this algorithm incorporated modified partial orthogonalization in that when $\mathbf{u}$, $\mathbf{v}$ are deemed to require re-orthogonalization they are only re-orthogonalized when and where it is considered necessary as described above.

```
u_0 = 0; β_0 = 1; α_0 = 0;
p = r/‖r‖_2;
for j = 1 to n
    v_j = p/β_{j-1};
    r = Av_j − β_{j-1}u_{j-1};
    α_j = ‖r‖_2;
    if j > 2
        Compute μ_{k,j} for k = 1, 2, ..., j − 1
```

for all k such that $|\mu_{k,j}| > \sqrt{\epsilon}$
Find an interval I_k, such that $k \in I_k$ and $\mu_{i,j} > \sqrt{\epsilon^3}$ for all $i \in I_k$;
Orthogonalize $\mathbf{r}$ against $\mathbf{u}_i$, $i \in I_k$;
Perform orthogonalization in the next iteration, same intervals;
Set $\mu_{i,j}$ using (17);
Re-calculate $\alpha_j = \|\mathbf{r}\|_2$;
end
end
$\mathbf{u}_j = \mathbf{r}/\alpha_j$;
if $j < n$
$\mathbf{p} = A^{\mathrm{H}}\mathbf{u}_j - \alpha_j \mathbf{v}_j$;
$\beta_j = \|\mathbf{p}\|_2$;
if $\|\mathbf{p}\|_2 <$ RSTOL
Reset a random $\mathbf{p}$;
Orthogonalize $\mathbf{p}$ against $\mathbf{v}_1, \ldots, \mathbf{v}_j$;
Perform orthogonalization in the next iteration;
Set $\nu_{k,j+1}$ using (16);
Re-calculate $\beta_j = \|\mathbf{p}\|_2$;
else if $j > 2$
Compute $\nu_{k,j+1}$ for $k = 1, 2, \ldots, j$
for all k such that $|\nu_{k,j+1}| > \sqrt{\epsilon}$
Find an interval I_k, such that
$k \in I_k$ and $\mu_{i,j+1} > \sqrt{\epsilon^3}$ for all $i \in I_k$;
Orthogonalize $\mathbf{p}$ against $\mathbf{v}_i$, $i \in I_k$;
Perform orthogonalization in the next iteration, same intervals;
Set $\nu_{i,j+1}$ using (16);
Re-calculate $\beta_j = \|\mathbf{p}\|_2$;
end
end
end
end

4. Performance experiments

Algorithm 4 was programmed in MATLAB in three versions. A version was created for Hankel matrices specifically which utilizes Algorithm 2 to perform fast Hankel matrix–vector multiplications. Two versions were implemented for general matrices, one of which lacked the reset method and another incorporated reset in order to allow us to show the effect of reset. These experiments were run on a machine with a 1.4 GHz Intel Celeron M processor and 256 MB of RAM running Windows XP.

The Hankel matrices in all test cases were specified by their first columns and last rows. A first column or last row vector was generated by a random complex vector $\mathbf{r} + \mathrm{i}\mathbf{c}$ with the entries of $\mathbf{r}$ and $\mathbf{c}$ uniformly distributed on $[-1, 1]$.

In the case of experiments involving a general matrix we generated an m-by-n complex matrix U with orthonormal columns, an n-by-n unitary matrix V, and a diagonal singular value matrix S, then we formed $A = USV^{\mathrm{H}}$, a random m-by-n matrix with predetermined singular values.

All timing results were obtained using MATLAB's `cputime` function; the results are thus presented in the time unit seconds. The 2-norm of X is denoted by $\|X\|$.

Example 1. In this example, we demonstrate the accuracy of the Hankel version of Algorithm 4. We do so by examining the factorization error and orthogonality errors in U and V. As shown in Table 1, the accuracy is about the square root of the machine precision as expected, since tolerances, such as the tolerance for reset, are set to the square root of the machine precision.

Example 2. Next we examine the performance of the Hankel version of Algorithm 4 for various values of m and n. We look at both the running time compared to $n(m+n)\log(m+n)$ and the total number of orthogonalizations required. The results in Tables 2 and 3 are the average of 10 runs for each case and the run time results have been normalized for the case where $m = n = 50$.

We note from these experiments that the cost of re-orthogonalization does not affect the overall complexity of the function which is shown to be roughly $n(m+n)\log(m+n)$.

Table 1
Accuracy of Algorithm 4 for Hankel matrices

$m = n$	$\|A - UBV^{\mathrm{H}}\|$	$\|I - UU^{\mathrm{T}}\|$	$\|I - VV^{\mathrm{T}}\|$
200	1.822E−07	2.186E−08	7.078E−09
400	6.842E−07	4.121E−08	6.133E−08
800	6.181E−07	1.175E−07	4.431E−08

Table 2
Performance conforming to expected $n(m+n)\log(m+n)$ run time as both m and n increase

m	n	$n(m+n)\log(m+n)$	Run time (s)	Total number of orthogonalizations
50	50	1.000	1.000	695.0
100	100	4.602	3.608	2922.1
200	200	20.816	14.404	12832.8
400	400	92.898	82.670	52599.4

Table 3
Performance conforming to expected $n(m+n)\log(m+n)$ run time as m increases

m	n	$n(m+n)\log(m+n)$	Run time (s)	Total number of orthogonalizations
50	50	1.000	1.000	672.7
100	50	1.632	1.344	632.9
200	50	2.997	1.750	602.6
400	50	5.970	1.812	552.4

Example 3. Next using non-Hankel versions of Algorithm 4 we examine the numerical stability of the algorithms as clustered singular values are introduced. One version of the algorithm includes reset and the other does not. We introduce an increasing number of random clustered singular values uniformly distributed in $[1, 1 + 10^{-13}]$. The rest of the singular values are uniformly distributed in [0,1]. Note that in all cases for the results presented in Tables 4 and 5, $m = n = 300$.

In Table 4, we present the results for the version of the algorithm with reset and in Table 5, we present the results for the version of the algorithm without reset. We notice that as the number of multiple singular values increases, the accuracy of the algorithm becomes very poor.

Example 4. Next we look at how Algorithm 4 performs against a version of the algorithm for square Hankel matrices presented in [7]. Modified partial orthogonalization and reset were incorporated into the square matrix Hankel algorithm in order to make the comparison fair. Normalized for the case that $m = n = 50$, the results are presented in Table 6.

We observe that the version of the algorithm for square matrices outperforms the general matrix version. Specifically, as shown in Table 6, the square algorithm costs about a half of the general one as expected, since the square algorithm exploits the symmetric structure.

Example 5. Finally, we compare the performance of our fast Lanczos bidiagonalization Algorithm 4 against the Householder bidiagonlization method [3, p. 252] implemented in MATLAB. In order to make the comparison fair, in our implementation of the Householder bidiagonalization, the unitary U and V are computed explicitly and the diagonal entries in the bidiagonal B are scaled to real and nonnegative. Also, for the Householder bidiagonalization in the case when $m = 600$ and $n = 200$, for efficiency, a QR decomposition was performed on the original Hankel matrix

Table 4
Accuracy of reset enabled algorithm as clustered singular values increase

Clustered singular values	$\|A - UBV^{\mathrm{H}}\|$	$\|I - UU^{\mathrm{T}}\|$	$\|I - VV^{\mathrm{T}}\|$
12	7.394E−07	5.841E−06	6.964E−07
16	6.165E−07	5.904E−07	1.029E−06
20	6.784E−08	6.870E−08	5.624E−08

Table 5
Accuracy of algorithm without reset as clustered singular values increase

Clustered singular values	$\|A - UBV^{\mathrm{H}}\|$	$\|I - UU^{\mathrm{T}}\|$	$\|I - VV^{\mathrm{T}}\|$
12	3.682E−08	3.193E−08	8.042E−08
16	1.770E−05	1.120E−04	7.786E−06
20	2.011E+00	1.109E+00	1.052E+00

Table 6
Performance of square vs. general algorithms

$m = n$	Square version run time (s)	General version run time (s)
50	1.000	1.000
100	2.410	3.032
200	10.013	13.097
400	54.487	129.548

Table 7
Performance of fast Hankel bidiagonalization Algorithm 4 vs. Householder bidiagonalization

m	n	Algorithm 4 run time (s)	Householder bidiagonalization run time (s)
10	10	0.005	0.004
20	20	0.013	0.009
50	50	0.050	0.054
100	100	0.201	0.295
200	200	0.719	2.484
600	200	1.571	28.568

by calling MATLAB function `qr`. Then the Householder bidiagonalization was applied to the upper triangular matrix resulted from the QR decomposition. Table 7 shows that our Algorithm 4 outperforms the Householder bidiagonalization algorithm, except for small matrices. We observe that the performance gap expands exponentially as m and n increase.

5. Conclusion

In this paper, we have presented a Lanczos algorithm for bidiagonalizing a general Hankel matrix. An orthogonalization scheme is developed and incorporated to improve stability and a reset method is used to handle the case of multiple/clustered singular values. As well, a matrix–vector multiplication is used, which exploits the Hankel structure, for fast multiplication. Experimental results show that the numerical stability and accuracy of these new algorithms are sound. Performance tests indicate that when dealing with square matrices, however, it is preferable to use the previously derived square Lanczos algorithm. Finally, we wish to note that this algorithm could be used as the first step of an SVD computation followed by an SVD algorithm for real bidiagonal matrices, such as the twisted method [1,2].

Acknowledgement

We would like to thank the anonymous reviewer's constructive suggestions.

References

[1] J.S. Dhillon, A new O(n^2) algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem, Ph.D. Thesis, University of California, Berkeley, 1997.
[2] J.S. Dhillon, B.N. Parlett, Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices, Linear Algebra Appl., 387 (2004) 1–28.
[3] G.H. Golub, C.F. Van Loan, Matrix Computations, third ed., The Johns Hopkins University Press, Baltimore, MD, 1996.
[4] M. Gu, S.C. Eisenstat, A divide-and-conquer algorithm for the bidiagonal SVD, SIAM J. Matrix Anal. Appl. 16 (1995) 79–92.
[5] C.F. Van Loan, Computational Frameworks for the Fast Fourier Transform, SIAM, 1992.
[6] Franklin T. Luk, Sanzheng Qiao, A fast eigenvalue algorithm for Hankel matrices, in: Franklin T. Luk (Ed.), Advance Signal Processing Algorithms, Architectures, and Implementations VIII, Proceedings of SPIE, vol. 3461, The International Society for Optical Engineering, 1998, pp. 249–256.
[7] F.T. Luk, S. Qiao, A fast singular value algorithm for Hankel matrices, in: V. Olshevsky (Ed.), Fast Algorithms for Structured Matrices: Theory and Applications, Contemporary Mathematics, vol. 323, American Mathematical Society, 2003.

[8] S. Qiao, Orthogonalization techniques for the Lanczos tridiagonalization of complex symmetric matrices, in: Franklin T. Luk (Ed.), Advanced Signal Processing Algorithms, Architectures, and Implementations XIV, Proceedings of SPIE, vol. 5559, 2004, pp. 423–434.
[9] H.D. Simon, The Lanczos algorithm with partial reorthogonalization, Math. Comp. 42 (1984) 15–142.
[10] Wei Xu, Sanzheng Qiao, A fast SVD algorithm for square Hankel matrices, Linear Algebra Appl. 482 (2+3) (2008) 550–563.