Numer Algor (2015) 69:875–891
DOI 10.1007/s11075-014-9930-0

ORIGINAL PAPER

A fast SVD for multilevel block Hankel matrices with minimal memory storage

Ling Lu · Wei Xu · Sanzheng Qiao

Received: 2 January 2014 / Accepted: 6 October 2014 / Published online: 28 October 2014

Abstract Motivated by the Cadzow filtering in seismic data processing, this paper presents a fast SVD method for multilevel block Hankel matrices. A seismic data presented as a multidimensional array is first transformed into a two dimensional multilevel block Hankel (MBH) matrix. Then the Lanczos process is applied to reduce the MBH matrix into a bidiagonal or tridiagonal matrix. Finally, the SVD of the reduced matrix is computed using the twisted factorization method. To achieve high efficiency, we propose a novel fast MBH matrix-vector multiplication method for the Lanczos process. In comparison with existing fast Hankel matrix-vector multiplication methods, our method applies 1-D, instead of multidimensional, FFT and requires minimum storage. Moreover, a partial SVD is performed on the reduced matrix, since complete SVD is not required by the Caszow filtering. Our numerical experiments show that our fast MBH matrix-vector multiplication method significantly improves both the computational cost and storage requirement. Our fast MBH SVD algorithm is particularly efficient for large size multilevel block Hankel matrices.

This work was supported by the Natural Science Foundation of China (Project No: 11101310) and Shanghai Key Laboratory of Contemporary Applied Mathematics of Fudan University.

L. Lu · W. Xu (✉)
Department of Mathematics, Tongji University,
Shanghai, 200092, People's Republic of China
e-mail: wdxu@tongji.edu.cn

L. Lu
e-mail: 073413@tongji.edu.cn

S. Qiao
Department of Computing and Software,
McMaster University, Hamilton, Ontario, L8S 4K1, Canada
e-mail: qiao@mcmaster.ca

Keywords Multi-level Block Hankel Matrix · SVD · Cadzow filtering · Seismic data processing

Mathematics Subject Classification (2010) 15B05 · 15A18 · 65F20 · 65F25 · 65F50

1 Introduction

A matrix is called Hankel if its entries on each antidiagonal are equal. Hankel matrices arise in many applications, such as digital information processing, system theory and automatic control theory. In particular, in seismic data processing, Cadzow filtering [5] involves Hankel matrices. Denoising and completion (regularization) are possible by iteratively finding a low-rank approximation that honors the original observations. Cadzow filtering is one kind of rank-reduction methods which can be employed to attenuate noise and for prestack seismic data regularization. It is equivalent to time series analysis called multichannel singular spectrum analysis [4, 12]. The application of Cadzow filtering approach for random noise attenuation on seismic data started from the works of Ulrych et al. [24] on eigenimage filtering of seismic data.

For one dimensional case, the Cadzow filtering consists of the following three steps: (1) Convert the original 1D data into the frequency domain and form a Hankel matrix; (2) find a low-rank approximation of the Hankel matrix; (3) recover the signal by averaging the elements on each antidiagonal of the low-rank approximation and transforming into the time domain. Note that the Hankel structure can be destroyed during step (2) and restored by the averaging process at step (3). These two steps are performed only once to produce a Hankel matrix, which need not be the nearest approximation to the original Hankel matrix. There are some other structure-preserving low-rank approximation methods which guarantee both structure and "nearest" approximation [7, 15]. However, they require a number of iterations involving the computation of the singular value decomposition (SVD), introducing significant additional computational cost. Thus they are impractical for large data matrices in seismic data processing.

In the absence of noise, the Hankel marix formed by seismic data is rank deficient [21, 22]. However, due to noise in the observed data, the rank of the data Hankel matrix is higher than it should be. Cadzow filtering removes the noise by reducing the rank of the Hankel matrix and then recovers trace values from the rank-reduced matrix. Thus, seismic data reconstruction and noise attenuation can be posed as a Hankel matrix rank-reduction problem.

Trickett furthered Cadzow filtering by applying eigenimage filtering to 3D data frequency slices and later extended F-x Cadzow filtering to F-xy Cadzow filtering by forming a larger Hankel matrix of Hankel matrices (Level-2 Block Hankel matrix) in multiple spatial dimensions [21–23]. In 2013, Gao et al. [13] developed a rank reduction denoising and reconstruction scheme that is used to reconstruct prestack data that depend on four spatial dimensions by forming a Level-4 Block Toeplitz matrix. This is often called 5D interpolation because reconstruction algorithms operate on

volumes that depend on four spatial dimensions and time or frequency. Falkovskiy et al. [9] introduced an additional dimension for composing the Hankel matrices. Instead of using only spatial dimensions for composing these matrices, they proposed to add a frequency dimension to create an extended matrix from a series of frequency slices. This Frequency Extension approach improved filter quality through better statistics by utilizing these multi-frequency slices. If using Cadzow filtering with Frequency Extension, we need to construct a Level-5 Block Hankel matrix. In practice, the seismic signal data are in dozens even hundreds of gigabytes, whose generated Hankel matrix size is usually in hundreds of thousands. Thus, fast and memory efficient matrix reduction methods are necessary.

The most stable matrix rank-reduction method is the singular value decomposition (SVD). The subroutine ZGESDD in LAPACK, adopted by MATLAB function svd, computes the SVD of a general matrix. But neither the truncated classical SVD nor the randomized SVD algorithm [14, 16, 17] take advantage of the special structure of the Multilevel Block Hankel (MBH) matrix arising from Cadzow filtering methods. A general SVD method, like ZGESDD, is too expensive for large size matrices occured in seismic data processing. By exploiting the Hankel structure, Xu and Qiao [27] proposed a fast SVD algorithm for (level-1) Hankel matrices. The fast SVD algorithm for Hankel matrices in [27] consists of two stages. First, the Hankel matrix is bidiagonalized through the Lanczos method. When the Hankel matrix is square, it is tridiagonalized to maintain its symmetry. Second, the SVD of the bidiagonal or tridiagonal matrix is computed using the twisted factorization method in [2, 26]. It is well known that the computational cost of the Lanczos method is dominated by matrix-vector multiplications. Thus, efficient matrix-vector multiplication is crucial.

Motivated by the seismic application, in this paper, we consider the Cadzow filtering for multidimensional data arrays and propose an efficient MBH matrix-vector multiplication in the Lanczos method. A multidimensional seismic data array is first transformed into a two-dimensional MBH matrix, then the 1-D FFT [1] is applied to perform a fast MBH matrix-vector multiplication. In contrast to other fast MBH matrix-vector multiplication methods employing multidimensional FFTs in [6, 13], our algorithm only requires 1-D FFT and minimal memory storage. Therefore, the main contribution of this article is our fast MBH matrix-vector multiplication with minimal memory storage requirement and its application to fast MBH SVD. In this paper, we focus on complex MBH matrices arising from the seismic data denoising. Our method is also applicable to real MBH matrices and multi-level block Teoplitz matrices from other applications, for example, the vibration signal measured on ship or mechanical equipment[29] and discrete dipole approximation (DDA) from electromagnetic scattering [20].

The rest of the paper is organized as follows. In Section 2, we present an algorithm for constructing a two-dimensional MBH from a signal data array of multiple spatial dimensions. In Section 3, we propose a fast algorithm for the MBH matrix-vector multiplication. Then, based on the fast MBH matrix-vector multiplication, the Lanczos method and the twisted factorization is given in Section 4. In Section 5 we present our numerical experimental results on some random MBH matrices and the experiments on seismic signal data. Finally, we conclude this paper with some remarks.

2 Constructing multilevel block Hankel matrices

In this section, we present our algorithm for constructing a two dimensional MBH matrix from a signal data array S of multiple spatial dimensions in frequency domain. The two-dimensional MBH matrix allows us to develop a fast MBH matrix-vector multiplication using 1-D FFT. For clarity, we present a recursive version of our algorithm.

In the trivial case when the data signal array S is one dimensional, that is, an n-by-1 complex column vector $[s_1\ s_2\ ...\ s_n]^{\mathrm{T}}$, representing an array of n traces along a constant-frequency slice. The Cadzow filtering for this signal model can be achieved by constructing the Hankel matrix [17]:

$$H = \begin{bmatrix} s_1 & s_2 & \cdots & s_{n-p+1} \\ s_2 & s_3 & \cdots & s_{n-p+2} \\ s_3 & s_4 & \cdots & s_{n-p+3} \\ \vdots & \vdots & \cdots & \vdots \\ s_p & s_{p+1} & \cdots & s_n \end{bmatrix}. \tag{2.1}$$

Apparently, a Hankel matrix is determined by its first column and last row. In this paper, we set the Hankel matrix as near to square as possible. Specifically, in (2.1), we choose $p = \lfloor n/2 \rfloor + 1$, where $\lfloor a \rfloor$ denotes the largest integer that is not greater than a. Thus H is either square when n is odd or p-by-$(p-1)$ when n is even. Then, given a trace vector $S = [s_1\ s_2\ ...\ s_n]^{\mathrm{T}}$, the Hankel matrix H is uniquely determined. Moreover, a longer main diagonal is desirable in the denoising process, since the rank of a matrix is limited by the length of its main diagonal.

Now, we consider the general case when S is a data signal array of k-dimensions. Suppose that the number of traces in the ith dimension is n_i, for $i = 1, ..., k$. Similar to the one-dimension case, we choose $p_i = \lfloor n_i/2 \rfloor + 1$ and set $q_i = n_i - p_i + 1$. To reduce the dimension, for each index j, $j = 1, ..., n_k$, in the kth dimension, we construct the $(k-1)$-dimensional arrays

$$S_j = S(:, ..., :, j), \qquad \text{for} \quad j = 1, ..., n_k.$$

Then, we apply this procedure recursively to each S_j to obtain a lower level MBH matrix H_j. Finally, we construct the level-k block Hankel matrix:

$$H = \begin{bmatrix} H_1 & H_2 & \cdots & H_{q_k} \\ H_2 & H_3 & \cdots & H_{q_k+1} \\ \vdots & \vdots & \cdots & \vdots \\ H_{p_k} & H_{p_k+1} & \cdots & H_{n_k} \end{bmatrix}. \tag{2.2}$$

In particular, when $k = 2$, each block H_i is a (level-1) Hankel matrix and H is a block Hankel matrix with Hankel blocks (BHHB).

Algorithm 1 (Constructing 2-D MBH) Given a k-dimensional signal array S, where the size in the ith dimension is n_i. This algorithm constructs the corresponding 2-D MBH matrix

1. if $k = 1$
2. Construct the Hankel matrix H in (2.1), where $n = n_1$;
3. return
4. end
5. for $i = 1$ to n_k
6. Construct MBH H_i by applying this algorithm to the $(k-1)$-dimensional array $S(:, ..., :, i)$;
7. Assign H_i to the corresponding antidiagonal of H as shown in (2.2);
8. end

3 Fast MBH matrix-vector multiplication

In this section, we present an efficient method for computing MBH matrix-vector multiplications. Our method applies 1-D, instead of multidimensional FFTs, and requires minimal memory by storing only necessary entries in the data matrix.

We describe a recursive version of our method. First, we consider the trivial case when the dimension of the data array S is one [11]. For example, when $n = 3$ in (2.1), we consider the Hankel matrix-vector multiplication

$$\mathbf{y} = \begin{bmatrix} s_1 & s_2 \\ s_2 & s_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} s_1x_1 + s_2x_2 \\ s_2x_1 + s_3x_2 \end{bmatrix}. \tag{3.3}$$

Recall that the Hankel matrix in the above multiplication is determined by the data array $S = [s_1\ s_2\ s_3]^{\mathrm{T}}$. Now we construct an n-by-n circulant matrix whose first column is S:

$$\mathsf{circ}(S) = \begin{bmatrix} s_1 & s_3 & s_2 \\ s_2 & s_1 & s_3 \\ s_3 & s_2 & s_1 \end{bmatrix}.$$

Accordingly, we expand the vector $\mathbf{x} = [x_1\ x_2]^{\mathrm{T}}$ of length $q = n - p + 1 = 2$ into an n-vector by padding $p - 1 = 1$ zero on the top of $\mathbf{x}$ followed by reversing its entries. Specifically,

$$\hat{\mathbf{x}} = [0\ x_1\ x_2]^{\mathrm{T}} \quad \text{and} \quad \mathsf{rev}(\hat{\mathbf{x}}) = [x_2\ x_1\ 0]^{\mathrm{T}}.$$

Then the product of the circulant matrix and the vector $\mathsf{rev}(\hat{\mathbf{x}})$ is

$$\begin{bmatrix} s_1 & s_3 & s_2 \\ s_2 & s_1 & s_3 \\ s_3 & s_2 & s_1 \end{bmatrix} \begin{bmatrix} x_2 \\ x_1 \\ 0 \end{bmatrix} = \begin{bmatrix} s_3x_1 + s_1x_2 \\ s_1x_1 + s_2x_2 \\ s_2x_1 + s_3x_2 \end{bmatrix} = \hat{\mathbf{y}}. \tag{3.4}$$

The circulant matrix-vector multiplication can be efficiently computed by applying the 1-D FFT to the first column of the circulant matrix and the vector $\mathsf{rev}(\hat{\mathbf{x}})$ [11], specifically,

$$\hat{\mathbf{y}} = \mathrm{ifft}(\mathrm{fft}(S) .* \mathrm{fft}(\mathsf{rev}(\hat{\mathbf{x}}))).$$

From (3.4), we can see that the Hankel matrix-vector product $\mathbf{y}$ in (3.3) can be obtained by extracting the bottom $p = 2$ elements of the product $\hat{\mathbf{y}}$ in (3.4).

Now we consider the general case when S is a k-dimensional array, where the size of the ith dimension is n_i. From the previous section, the corresponding MBH matrix H is $(p_1 \cdots p_k)$-by-$(q_1 \cdots q_k)$. Thus the size of the vector $\mathbf{x}$ in the MBH matrix-vector multiplication

$$\mathbf{y} = H\mathbf{x}$$

is $q_1 \cdots q_k$. To transform this problem into a circulant matrix-vector multiplication and then apply the 1-D FFT, we vectorize the k-dimensional data array S. The vectorization operation is recursively defined by

$$\mathsf{vec}(S) = \begin{cases} S, & \text{if} \quad k = 1, \\ \begin{bmatrix} \mathsf{vec}(S[:, ..., :, 1]) \\ \vdots \\ \mathsf{vec}(S[:, ..., :, n_k]) \end{bmatrix}, & \text{otherwise.} \end{cases}$$

Then, in the fast multiplication, the 1-D FFT is applied to the $(n_1 \cdots n_k)$-vector $\mathsf{vec}(S)$, which is the first column of the circulant matrix. Note that $\mathsf{vec}(S)$ contains only the information from S, no additional zeros.

How is the $(q_1 \cdots q_k)$-vector $\mathbf{x}$ extended to an $(n_1 \cdots n_k)$-vector to be applied by the 1-D FFT? In the trivial case when $k = 1$, the q_1-vector $\mathbf{x}$ is extended to an n_1-vector $\hat{\mathbf{x}}$ by padding a $(p_1 - 1)$-vector of zeros on the top of $\mathbf{x}$. In the general case, we partition $\mathbf{x}$ into

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_{q_k} \end{bmatrix},$$

where each $\mathbf{x}_i$, $1 \le i \le q_k$, is a $(q_1 \cdots q_{k-1})$-vector. Thus the problem size is reduced by one, from k to $k-1$. So, we apply our operation recursively to each subvector $\mathbf{x}_i$ to obtain an extended vector $\hat{\mathbf{x}}_i$ of dimension $n_1 \cdots n_{k-1}$ to obtain an $(n_1 \cdots n_k)$-vector

$$\hat{\mathbf{x}} = \begin{bmatrix} \mathbf{0} \\ \hat{\mathbf{x}}_1 \\ \vdots \\ \hat{\mathbf{x}}_{q_k} \end{bmatrix},$$

where $\mathbf{0}$ is a zero vector of size $(p_k - 1)(n_1 \cdots n_{k-1})$. Then the 1-D FFT is applied to $\mathsf{rev}(\hat{\mathbf{x}})$. In summary, using the notations above, we have the following recursive operation:

$$\mathsf{pad}(\mathbf{x}) = \begin{cases} \begin{bmatrix} \mathbf{0} \\ \mathbf{x} \end{bmatrix}, & \text{when} \quad k = 1, \\ \begin{bmatrix} \mathbf{0} \\ \mathsf{pad}(\mathbf{x}_1) \\ \vdots \\ \mathsf{pad}(\mathbf{x}_{q_k}) \end{bmatrix}, & \text{otherwise.} \end{cases}$$

However, the result

$$\hat{\mathbf{y}} = \mathrm{ifft}(\mathrm{fft}(\mathsf{vec}(S)) .* \mathrm{fft}(\mathsf{rev}(\mathsf{pad}(\mathbf{x}))))$$

is an $(n_1 \cdots n_k)$-vector. How do we extract the MBH matrix-vector product $\mathbf{y}$, which is a $(p_1 \cdots p_k)$-vector, from $\hat{\mathbf{y}}$? In the trivial case when $k = 1$, we extract the bottom p_1 elements from $\hat{\mathbf{y}}$. In the general case, we partition

$$\hat{\mathbf{y}} = \begin{bmatrix} \hat{\mathbf{y}}_0 \\ \hat{\mathbf{y}}_1 \\ \vdots \\ \hat{\mathbf{y}}_{p_k} \end{bmatrix},$$

where $\hat{\mathbf{y}}_0$ is a $(q_k - 1)(n_1 \cdots n_{k-1})$-vector and $\hat{\mathbf{y}}_i$, for $1 \le i \le p_k$, is an $(n_1 \cdots n_{k-1})$-vector. We then apply this procedure recursively to $\hat{\mathbf{y}}_i$, for $i = 1, ..., p_k$. In summary, we have the following recursive operation

$$\mathsf{extract}(\hat{\mathbf{y}}) = \begin{cases} \hat{\mathbf{y}}_{q_1:n_1}, & \text{when} \quad k = 1, \\ \begin{bmatrix} \mathsf{extract}(\hat{\mathbf{y}}_1) \\ \vdots \\ \mathsf{extract}(\hat{\mathbf{y}}_{p_k}) \end{bmatrix}, & \text{otherwise.} \end{cases}$$

Next, we present the fast MBH matrix-vector multiplication algorithm.

Algorithm 2 (MBH matrix-vector multiplication) Given a multi-dimensional complex data array S this algorithm multiplies the MBH matrix H in (2.2) determined by S with a vector $\mathbf{x}$ with compatible size, using the 1-D FFT and minimal storage

1. $\hat{\mathbf{y}} = \mathsf{ifft}(\mathsf{fft}(\mathsf{vec}(S)) .* \mathsf{fft}(\mathsf{rev}(\mathsf{pad}(\mathbf{x}))))$;
2. return $\mathsf{extract}(\hat{\mathbf{y}})$.

Alternatively, fast MBH matrix-vector multiplication can be achieved by first transforming an MBH matrix into a Multilevel Block Circulant (MBC) matrix then applying multi-dimensional FFT [6, 13]. Zero entries are introduced when transforming an MBH matrix into an MBC matrix, resulting in more storage requirement. Still another approach is to form a circulant block for each Hankel block at each level without padding zeros to construct a multilevel block circulant (MBC) matrix. Then fast MBC matrix-vector multiplication can be computed by applying multi-dimensional FFT. Although this approach requires the same memory as our method, it is not as efficient as our method. As we know, the FFT performs most efficiently on vectors of lengths of power of two. In this approach, the FFT is applied to multiple vectors at multiple levels. In particular, when the lengths of those multiple vectors are prime numbers, the performance suffers, even the complexity $O(N \log N)$ is invalid, worse than the multi-dimension FFT with zero padding method, which changes prime numbers to even numbers. In contrast, our method applies 1DFFT to a single long vector whose length is the product of the lengths of those multiple vectors. As shown in Table 1, where MDFFTNZ denotes this multi-dimension FFT without zero padding, MDFFT the multi-dimension FFT with zero padding, 1DFFT our one-dimension FFT method, and MD/1D and MDNZ/1D the ratios, the MDFFTNZ

Table 1 Running time (in seconds) comparison of the three fast MBH matrix-vetor multiplication methods: 1DFFT, MDFFT and MDFFTNZ

$p_1 \times p_2 \times \cdots \times p_k$	$P = Q$	1DFFT	MDFFT	MDFFTNZ	MD/1D	MDNZ/1D
Case k=2						
35×119	4165	0.0047	0.0047	0.0094	1.00	2.00
68×96	6528	0.0109	0.0156	0.0281	1.43	2.57
40×100	4000	0.0062	0.0109	0.0094	1.75	1.50
75×95	7125	0.0094	0.0172	0.0156	1.83	1.67
Case k=4						
$7 \times 3 \times 21 \times 7$	3087	0.014	0.0203	0.0296	1.44	2.11
$7 \times 19 \times 7 \times 7$	6517	0.0343	0.0374	0.0640	1.09	1.86
$10 \times 15 \times 5 \times 6$	4500	0.0218	0.0265	0.0281	1.21	1.29
$16 \times 12 \times 5 \times 8$	7680	0.0452	0.0655	0.0577	1.45	1.28
Case k=5						
$3 \times 7 \times 5 \times 11 \times 3$	3465	0.0125	0.0390	0.0406	3.12	3.25
$7 \times 10 \times 3 \times 5 \times 7$	7350	0.0562	0.1045	0.1061	1.86	1.89
$5 \times 6 \times 8 \times 3 \times 5$	3600	0.0156	0.0546	0.0593	3.50	3.80
$5 \times 7 \times 6 \times 9 \times 4$	7560	0.0624	0.1186	0.1014	1.90	1.63

method performs worse than the MDFFT method when the vector lengths p_i are prime numbers. Our method always outperforms both methods.

Table 2 compares three matrix-vector multiplication methods on storage and floating-point operation (flop) counts. The first column lists the the methods in comparison: 1DFFT is our fast MBH matrix-vector multiplication method; MDFFT is the fast MBH matrix-vector multiplication using multidimensional FFT [6]; MATLAB is the Matlab general matrix-vector multiplication. The second column shows the major variables in the methods in the first column, where $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, $\bar{S}$, $\bar{\mathbf{x}}$, and $\bar{\mathbf{y}}$ are extended vectors from S, $\mathbf{x}$, and $\mathbf{y}$. The third column "Sizes" shows the sizes of the corresponding variables in the second column, where $N = n_1 n_2 \cdots n_k$, $M = (n_1 + 1)(n_2 + 1) \cdots (n_k + 1)$, $P = p_1 p_2 \cdots p_k$ and $Q = q_1 q_2 \cdots q_k$. Our 1DFFT method is the most efficient method requiring $3N + P + Q$ storage and

Table 2 Complexity comparison of the three methods for MBH matrix-vector multiplication on storage and floating-point operation (flop) counts

Method	Variables	Sizes	Storage	Flops
1DFFT	$S, \mathbf{x}, \mathbf{y}, \hat{\mathbf{x}}, \hat{\mathbf{y}}$	N, Q, P, N, N	$3N + P + Q$	$15N \log_2 N + 6N$
MDFFT	$\bar{S}, \mathbf{x}, \mathbf{y}, \bar{\mathbf{x}}, \bar{\mathbf{y}}$	M, Q, P, M, M	$3M + P + Q$	$15M \log_2 M + 6M$
MATLAB	$H, \mathbf{x}, \mathbf{y}$	PQ, Q, P	$PQ + P + Q$	$8PQ - P - Q$

$15N \log_2 N + 6N$ flops. Specifically, the first column of the circulant matrix in our method contains the information entirely from the data S. Thus, the storage requirement of our method is minimal. Our experimental results presented in Section 5 confirms the theoretical comparison.

Table 3 depicts the calculated storage requirements of the three methods for various sizes and levels of MBH matrices, where MD/1D represents the ratio of MDFFT/1DFFT and MAT/1D is the ratio of MATLAB/1DFFT. The table shows that as the level k increases our 1DFFT method utilizes memory more efficiently than the MDFFT method.

4 Lanczos reduction and twisted factorization

In this section, we describe the application of our fast MBH matrix-vector multiplication to the fast MBH SVD. The first stage of our fast SVD algorithm is to reduce a complex MBH matrix into a bidiagonal matrix using the Lanczos method. When the MBH matrix is square, it is reduced to a complex symmetric tridiagonal matrix. For simplicity, we only describe the Lanczos bidiagonalization process for complex rectangle MBH matrices. The tridiagonalization process is analogous to the bidiagonalization. The second stage is to compute the SVD of the bidiagonal/tridiagonal matrix using the twisted factorization method. We also only describe the second stage for bidiagonal matrices.

Given an $m \times n$, $m \geq n$, complex symmetric matrix A, we can find an $m \times n$ matrix U with orthonormal columns and an $n \times n$ unitary matrix V such that

$$U^{\mathrm{H}} A V = B, \tag{4.5}$$

Table 3 Storage comparison of three methods for MBH matrix-vector multiplication. The storage here means the quantity of complex numbers

$p_1 \times \cdots \times p_k$	$P = Q$	1DFFT	MDFFT	MATLAB	MD/1D	MAT/1D
Case k=2						
512×25	12,800	175,981	179,200	163,865,600	1.02	931.16
512×40	20,480	283,411	286,720	419,471,360	1.01	1480.08
200×128	25,600	356,435	358,400	655,411,200	1.01	1838.80
Case k=4						
$8 \times 64 \times 5 \times 5$	12,800	488,515	640,000	163,865,600	1.31	335.44
$16 \times 32 \times 8 \times 5$	20,480	831,925	1,024,000	419,471,360	1.23	504.22
$5 \times 40 \times 16 \times 8$	25,600	1,043,045	1,280,000	655,411,200	1.23	628.36
Case k=5						
$8 \times 8 \times 8 \times 5 \times 5$	12,800	845,725	1,254,400	163,865,600	1.48	193.76
$16 \times 8 \times 4 \times 8 \times 5$	20,480	1,359,235	2,007,040	419,471,360	1.48	308.61
$5 \times 5 \times 8 \times 16 \times 8$	25,600	1,746,125	2,508,800	655,411,200	1.44	375.35

where U^{H} denotes the Hermitian of U and B is a real bidiagonal matrix:

$$B = \begin{bmatrix} a_1 & b_1 & & \\ & a_2 & b_2 & \\ & & \ddots & \ddots \\ & & & b_{n-1} \\ & & & a_n \end{bmatrix}.$$

Rewriting (4.5) as

$$AV = UB, \tag{4.6}$$

and

$$A^{\mathrm{H}}U = VB^{\mathrm{T}} \tag{4.7}$$

and comparing the jth columns of both sides of (4.6) and (4.7), we have

$$A\mathbf{v}_j = a_j\mathbf{u}_j + b_{j-1}\mathbf{u}_{j-1},$$

and

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

for $j = 1, \ldots, n$, which leads to the inner Lanczos recursion,

$$\mathbf{r}_j = a_j\mathbf{u}_j = A\mathbf{v}_j - b_{j-1}\mathbf{u}_{j-1}, \tag{4.8}$$

$$\mathbf{p}_j = b_j\mathbf{v}_{j+1} = A^{\mathrm{H}}\mathbf{u}_j - a_j\mathbf{v}_j, \tag{4.9}$$

Because of the orthonormality of the columns of U and V, we have $a_j = \|\mathbf{r}_j\|_2$, $b_j = \|\mathbf{p}_j\|_2\mathbf{u}_j = \mathbf{r}_j/a_j$, and $\mathbf{v}_{j+1} = \mathbf{p}_j/b_j$ Thus, we have the following algorithm for the Lanczos bidiagonalization of general complex matrices.

Algorithm 3 (Lanczos bidiagonalization) Given a starting vector $\mathbf{r} \in \mathbb{C}^n$ and a complex matrix $A \in \mathbb{C}^{m\times n}$, $m \geq n$, this algorithm computes the diagonals of the real square upper bidiagonal matrix B in (4.5) and U, V such that $A = UBV^{\mathrm{H}}$.[2]

1. $\mathbf{u}_0 = 0$; $b_0 = 1$; $\mathbf{p} = \mathbf{r}/\|\mathbf{r}\|_2$;
2. **for** $j = 1 : n$
3. $\quad \mathbf{v}_j = \mathbf{p}/b_{j-1}$;
4. $\quad \mathbf{r} = A\mathbf{v}_j - b_{j-1}\mathbf{u}_{j-1}$;
5. $\quad a_j = \|\mathbf{r}\|_2$;
6. $\quad \mathbf{u}_j = \mathbf{r}/a_j$;
7. $\quad$ **if** $j < n$.
8. $\quad\quad \mathbf{p} = A^{\mathrm{H}}\mathbf{u}_j - a_j\mathbf{v}_j$;
9. $\quad\quad b_j = \|\mathbf{p}\|_2$;
10. $\quad\quad$ **if** $b_j = 0$, quit; **end**
11. $\quad$ **end**
12. **end**

The Lanczos bidiagonalization described above suffers from loss of orthogonality when computing U and V due to the rounding errors. So, a selective reorthogonalizion technique based on estimates of the orthogonalities of the orthornormal vectors is proposed in [18]. Compared to the complete orthogonalization in [11],

the partial orthogonalization is more economic and can be extended to general complex matrices as well. In our implementation, we adopt the Lanczos bidiagonalization with modified partial orthogonalization described in [18].

For general MBH matrices, the Lanczos bidiagonalization Algorithm 3 can be accelerated by carrying out the matrix vector multiplications in line 4 and line 8 using the efficient MBH matrix-vector multiplication descibed in Section 1. To minimize the storage requirement, the input matrix is the signal matrix S, rather than the full MBH matrix. Moreover, it is unnecessary to construct $\mathsf{vec}(S)$ and apply the 1-D FFT to $\mathsf{vec}(S)$ each time when computing the product. In practice, the $\mathsf{vec}(S)$ and $\mathsf{fft}(\mathsf{vec}(S))$ can be pre-computed outside the for loop in line 2 of Algorithm 3 when the algorithm is initiated. For square MBH matrices, the algorithm for the Lanczos tridiagonalization is similar to Algorithm 3. See [18] for details.

After the real bidiagonal matrix B is computed by the Lanczos method in the first stage, the singular values of B are the square roots of the eigenvalues of $B^{\mathrm{T}}B$, which can be computed in $O(n^2)$ flops by the QR method for eigenvalues only. Then the right singular vectors of B are the eigenvectors of $B^{\mathrm{T}}B$, which can be obtained by the twisted factorization method [8] in $O(n^2)$. Then the left singular vectors can be obtained from its right singular vectors.

The matrix rank reduction in seismic data processing requires only the singular vectors corresponding to the singular values that are larger than a predetermined tolerance, instead of a complete set of singular vectors. Thus, the cost of our proposed method, which is based on the Lanczos process, can be further reduced comparing with the MATLAB svd function, especially when the rank after reduction is small.

5 Numerical experiments and seismic examples

In this section, we present our experimental results on the matrix-vector multiplication methods, the SVD methods, and Cadzow filtering. The three matrix-vector multiplication methods in comparison are: our fast method (1DFFT); the fast MBH matrix-vector multiplication using multi-dimensional FFT (MDFFT); general matrix-vector multiplication in MATLAB. The four SVD algorithms in comparison are: our method (1DFFT); the multidimensional FFT Lanczos process combined with the Twisted factorization method (MDFFT); Lanczos process with general matrix-vector multiplication combined with the Twisted factorization (LT); MATLAB built-in svd function, which is implemented by ZGESVD in LAPACK [3]. The Cadzow filtering examples include simulated and real seismic data. All experiments were carried out on a server with an AMD Phenom X6 1090T 3.2 GHz processor and 16GB RAM running MATLAB 7.8 under Windows 7.

In our experiments, the data were generated as arrays of complex entries with random imaginary and real parts normally distributed with zero mean and unit variance. The MBH matrices were constructed by the generated random data arrays. We compare the whole SVD other than the partial SVD because the suitable rank of MBH matrix in Cadzow filtering depends on every singular value. All the blocks are square, that is, $p_i = q_i$, for $i = 1, ..., k$. Thus $P = Q$. Each running time of matrix-vector multiplication in Table 4 is an average of ten random examples.

Table 4 Running time (in seconds) comparison of the three MBH matrix-vetor multiplication methods: 1DFFT, MDFFT and MATLAB

$p_1 \times p_2 \times \cdots \times p_k$	$P = Q$	1DFFT	MDFFT	MATLAB	MD/1D	MAT/1D
Case k=2						
512×25	12800	0.019	0.025	0.830	1.32	43.68
512×40	20480	0.051	0.086	3.655	1.69	71.67
200×128	25600	0.234	0.342	3.643	1.46	15.57
Case k=4						
$8 \times 64 \times 5 \times 5$	12800	0.072	0.114	0.838	1.58	11.64
$16 \times 32 \times 8 \times 5$	20480	0.134	0.192	4.051	1.43	30.23
$5 \times 40 \times 16 \times 8$	25600	0.459	0.877	4.097	1.91	8.93
Case k=5						
$8 \times 8 \times 8 \times 5 \times 5$	12800	0.106	0.236	0.792	2.22	7.47
$16 \times 8 \times 4 \times 8 \times 5$	20480	0.246	0.390	3.693	1.58	14.98
$5 \times 5 \times 8 \times 16 \times 8$	25600	0.568	1.323	4.228	2.33	7.45

Each running time of the SVD in Table 5 and Table 6 is an average of two random examples.

Table 4 shows the running times of the three matrix-vector multiplication methods: 1DFFT, MDFFT, and MATLAB where MD/1D is the ratio of MDFFT/1DFFT and MAT/1D is the ratio of MATLAB/1DFFT. As expected, when the problem size increases, the speedup of our proposed method becomes significant. For the same size, as the level k increases, the ratio MAT/1D decreases, since the matrix is less structured. However, for the same size, as k increases, the ratio MD/1D also increases, since the efficiency of 1DFFT over MDFFT becomes more prominent. In any case, 1DFFT is more efficient than MDFFT.

Table 5 compares the two SVD methods: Lanczos with 1DFFT combined with the Twisted method and Matlab general SVD method. It lists the running times as well as the errors O-Error and F-Error defined by

$$\mathrm{O-Error} = \|I - U^{\mathrm{H}}U\|_F / P^2,$$

Table 5 Comparison of the computational time and accuracy of MATLAB built-in svd with our proposed fast SVD algorithm on square MBH matrices

$p_1 \times p_2 \times p_3$	$P = Q$	MATLAB(s)	1DFFT Time(s)	O-Error	F-Error	MATLAB/1DFFT
$16 \times 10 \times 12$	1920	275.9	128.4	1.24e-14	2.221e-13	2.15
$16 \times 10 \times 20$	3200	1704.8	520.9	1.86e-14	2.808e-13	3.27
$16 \times 15 \times 20$	4800	4991.5	1488.7	3.17e-15	1.484e-13	3.35
$16 \times 20 \times 20$	6400	30872.0	3423.0	6.68e-15	1.325e-13	9.02

$$\text{F} - \text{Error} = \|H - UDU^{\text{T}}\|_F / P^2,$$

for an MBH matrix $H \in \mathbb{C}^{P\times Q}$, where matrix U and D are the singular vector matrix and the diagonal singular value matrix of H, respectively, and matrix I is the identity matrix. It shows that our fast MBH SVD algorithm is accurate. However, the speedup MATLAB/1DFFT is not as dramatic as those in Table 4, since matrix-vector multiplication is only a part of the SVD.

The Matlab SVD method is very different from our fast MBH SVD, which is a combination of the Lancosz process and the Twisted method. So, we compared three similar Lancosz-Twisted combination SVD methods. They differ only in the matrix-vector multiplication in the Lancosz process. Table 6 lists their running times, where LT uses general matrix-vector multiplication, MDFFT uses MDFFT in MBH matrix-vector multiplication, and 1DFFT uses 1D FFT in MBH matrix-vector multiplication. LT/1D is the ratio LT/1DFFT and MD/1D is the ratio MDFFT/1DFFT. As shown in Table 6, for the same k the ratio LT/1D stays about the same and the ratio MD/1D decreases as the matrix size increases. For the same size, the ratio LT/1D decreases as the level k increases, since the matrix is less structured, whereas the ratio MD/1D increases as k increases, since the efficiency of 1D over MD is more prominent for larger k.

Finally, two seismic examples are investigated: simulated and practical. It is vital to determine the optimal rank of the MBH matrix in the seismic denoising. There are several approaches to rank determination. A common approach is to set a threshold based on the inflection point of singular value curve or singular entropy increment curve. In 2005, Sanliturk and Cakar presented a method based on the SVD of measured frequency response functions (FRFs)[19]. In our examples, we adopt the recent method [29] for the determination of the optimal rank according to the relationship between the extremum points of the noise elimination signal and the rank of the matrix. In the simulated example, the dimension of the original signal is three. We extracted a segment of the data to maintain a constant frequency and performed the F-xy Cadzow filtering [21] with the BHHB matrix. Figure 1 shows an inline section

Table 6 Running time (in seconds) comparison of the three SVD methods: LTSVD, MDFFT and 1DFFT

$p_1 \times p_2 \times \cdots \times p_k$	$P = Q$	LT	MDFFT	1DFFT	LT/1D	MD/1D
Case k=4						
$4 \times 3 \times 6 \times 30$	2160	196	182	157	1.25	1.16
$5 \times 8 \times 12 \times 10$	4800	1862	1445	1336	1.39	1.08
$8 \times 6 \times 6 \times 25$	7200	5829	4316	4102	1.42	1.05
$12 \times 16 \times 12 \times 4$	9216	11741	8766	8425	1.39	1.04
Case k=5						
$4 \times 3 \times 6 \times 6 \times 5$	2160	196	212	141	1.39	1.50
$5 \times 5 \times 4 \times 8 \times 6$	4800	1843	1672	1285	1.43	1.30
$8 \times 6 \times 6 \times 5 \times 5$	7200	5666	4836	4160	1.36	1.16
$8 \times 8 \times 9 \times 4 \times 4$	9216	12280	9733	8950	1.37	1.09

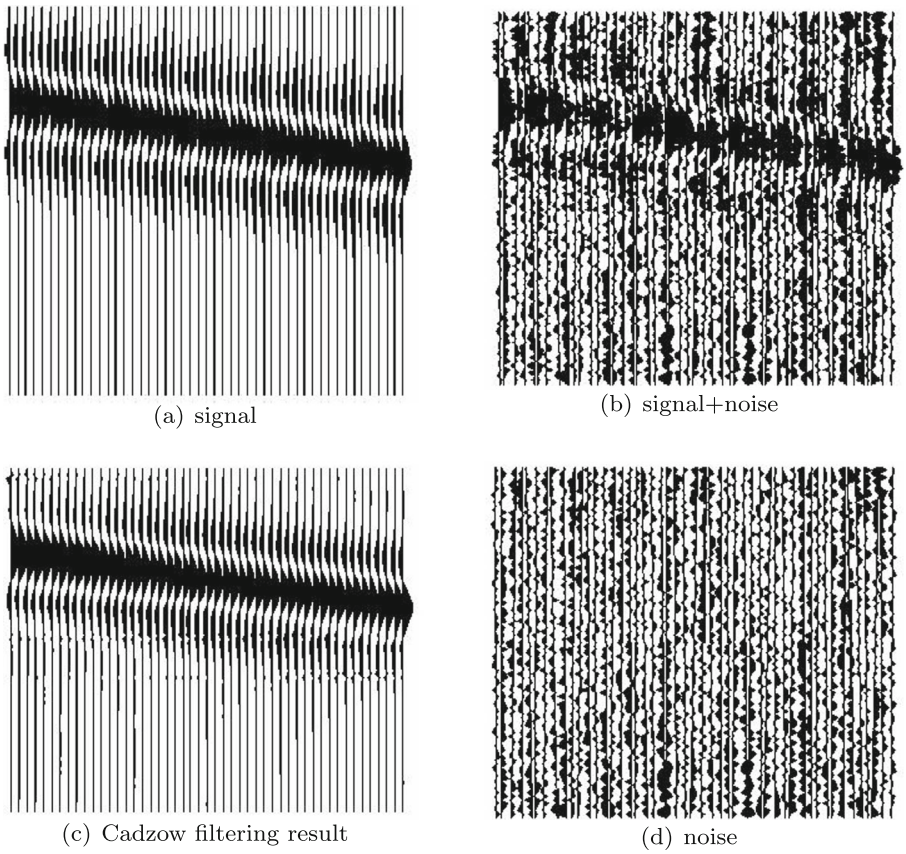
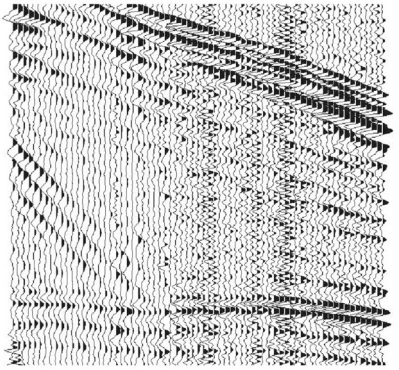
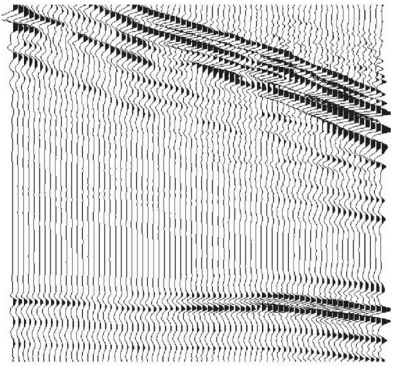


Fig. 1 The inline section of a 99-trace by 49-trace stacked volume with 1 event of Ricker wavelet is shown, having distinct dips in both spatial directions. The original signal is displayed in **a**. The noisy signal added by random noise is shown in **b**. The F-xy Cadzow filtering on the corresponding 1250×1250 BHHB matrix with rank $R_k = 5$ removes most of the noise and preserves much of the signal. Figure **c** shows the signal recovered from **b** by the F-xy Cadzow filtering. The noise removed from **b** is shown in **d**

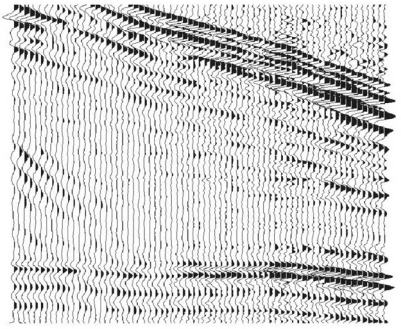
of a 99-trace by 49-trace stacked volume with 1 event of Ricker wavelet, having distinct dips in both spatial directions. The original signal is displayed in Fig. 1a. Random noise was added to the signal. The noisy signal is shown in Fig. 1b. The F-xy Cadzow filtering on the corresponding 1250×1250 BHHB matrix with rank $R_k = 5$ removed most of the noise and preserved much of the signal. Figure 1c shows the signal recovered from Fig. 1b by the F-xy Cadzow filtering. The noise removed from Fig. 1b is shown in Fig. 1d. The comparison of the four pictures in Fig. 1 demonstrates the effectiveness of the Cadzow filtering with our fast SVD algorithms for BHHB matrices. The practical example comes from a geological survey. It is a real seismic signal of 349-trace by 69-trace shown in Fig. 2. We formed a 6125×6125 BHHB matrix and performed the F-xy Cadzow filtering with three different ranks: $R_k = 200$, $R_k = 500$, $R_k = 800$. The results are shown in Figs. 2b, 2c



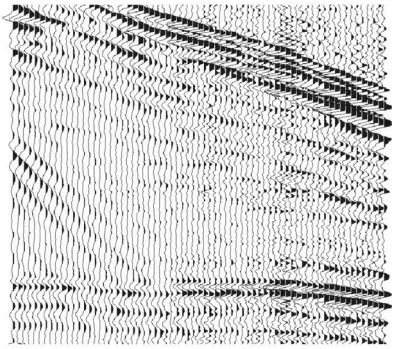
(a) Original signal



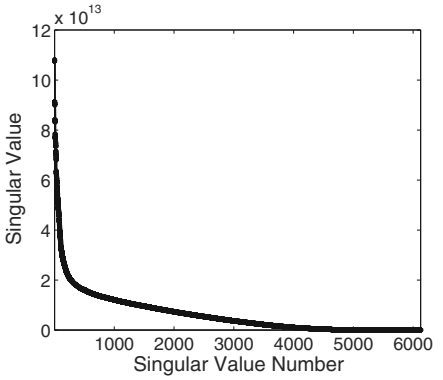
(b) Rank $R_k = 200$



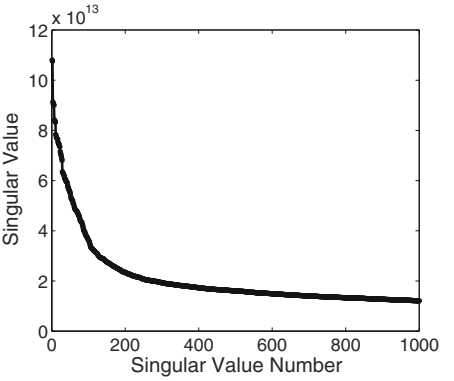
(c) Rank $R_k = 500$



(d) Rank $R_k = 800$



(e) All eigenvalues of BHHB matrix



(f) The 1000 largest eigenvalues

Fig. 2 A real seismic signal of 349-trace by 69-trace which comes from a geological survey is shown. We formed a 6125×6125 BHHB matrix and performed the F-xy Cadzow filtering with three different ranks: $R_k = 200$, $R_k = 500$, $R_k = 800$. The results are listed in subfigures **b**, **c** and **d**. Subfigure **e** displays all the singular values of the BHHB matrix and subfigure **f** shows the 1000 largest singular values

and 2d. A plenty of noise is removed by the F-xy Cadzow filtering and the useful seismic signals are captured. Fig. 2e displays all the singular values of the BHHB matrix and Fig. 2f shows the 1000 largest singular values. The singular value distribution suggests $R_k = 200$ is a good choice. The main signals at the right-top and right-bottom are present but the left-bottom part is absent in Fig. 2 b when $R_k = 200$. By increasing the rank R_k to 500 and 800, the left-bottom part is present in Figs. 2c and 2d, however, more noise is also included. Because the signal data are from real survey, not artiticial simulation, the left-bottom part may be signal or noise. In practice, a geological domain expert is required to determine the rank.

6 Conclusion

In this paper, we propose a fast MBH matrix-vector multiplication method and its application to a fast MBH SVD algorithm. Our fast MBH matrix-vector multiplication method uses the 1-D FFT and requires minimum memory. It is particularly efficient in both computation and storage when the size of the MBH matrix is large and its level is high.

References

1. Barrowes, B.E., Teixeira, F.L., Kong, J.A.: Fast algorithm for matrix-vector multiply of asymmetric multilevel block-toeplitz matrices. IEEE Antennas Propag. Soc. Int. Symp. **4**, 630–633 (2001)
2. Browne, K., Qiao, S., Wei, Y.: A lanczos bidiagonalization algorithm for hankel matrices. Linear Algebra Appl. **430**, 1531–1543 (2009)
3. Barker, V.A., Blackford, L.S., Dongarra, J., Du Croz, J., Hammarling, S., Marinova, M., Wásniewski, J., Yalamov, P.: LAPACK95 Users' Guide", SIAM. Philadelphia, Pennsylvania (2001)
4. Broomhead, D., King, G.: Extracting qualitative dynamics from experimental data. Phys. D Nonlinear Phenom. **20**(2), 217–236 (1986)
5. Cadzow, J.: Signal enhancement a composite property mapping algorithm. IEEE Trans. Acoust., Speech, Sig. Process. **36**, 49–62 (1988)
6. Chan, R.H., Jin, X.: An Introduction to Iterative Toeplitz Solvers, 3rd edition, SIAM. Philadelphia, Pennsylvania (2007)
7. Chu, M.T., Funderlic, R.E., Plemmons, R.J.: Structured low rank approximation. Linear Algebra Appl. **366**, 157–172 (2003)
8. Dhillon, I.S.: A new $O(n^2)$ algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem,PhD thesis, University of California, Berkeley, CA (1997)
9. Falkovskiy, A., Floreani, E., Schlosser, G.: FX Cadzow / SSA random noise filter: frequency extension,2011 CSPG CSEG CWLS Convention, pp. 1–8 (2011)
10. Fernando, F.V.: On computing an eigenvector of a tridiagonal matrix. SIAM Matrix Anal. Appl. **18**, 1013–1034 (1997)
11. Golub, G.H., Van Loan, C.F. Matrix Computations, 3rd edition. Johns Hopkins University Press, Baltimore, MD (2009)
12. Ghil, M., Allen, M., Dettinger, M., Ide, K., Kondrashov, D., Mann, M., Robertson, A., Saunders, A., Tian, Y., Varadi, F., Yiou, P.: Advance spectral methods for climatic time series. Rev. Geophys. **40**, 1–41 (2002)
13. Gao, J., Sacchi, M.D., Chen, X.: A fast reduced-rank interpolation method for prestack seismic volumes that depend on four spatial dimensions. Geophysics **78**(1), V21–V30 (2013)
14. Liberty, E., Woolfe, F., Martinsson, P.G., Rokhlin, V., Tygerts, M.: Randomized algorithms for the low-rank approximation of matrices. Proc. Natl. Acad. Sci. USA **104**(51), 20167–20172 (2007)

15. Markovsky, I.: Low rank approximation: algorithms, implementation, applications. Springer (2012)
16. Oropeza, V.E., Sacchi, M.D.: A randomized SVD for Multichannel Singular Spectrum Analysis (MSSA) noise attenuation,JSEG Denver 2010 Annual Meeting, pp. 3539–3544 (2010)
17. Oropeza, V.E., Sacchi, M.D.: Simultaneous seismic data denoising and reconstruction via multichannel singular spectrum analysis. Geophysics **76**(3), 25–32 (2011)
18. Qiao, S., Liu, G., Xu, W.: Block lanczos tridiagonalization of complex symmetric matrices,*Optics & Photonics 2005*, International Society for Optics and Photonics, 591010-591010 (2005)
19. Sanliturk, K.Y., Cakar, O.: Noise elimination from measured frequency response functions. Mech. Syst. Signal Process. **19**(3), 615–631 (2005)
20. Tsang, L., Ding, K.H., Shih, S.E., Kong, J.A.: Scattering of electromagnetic waves from dense distributions of spheroidal particles based on Monte Carlo simulations. J. Opt. Soc. Amer. A **15**, 2660–2669 (1998)
21. Trickeett, S.: F-xy Cadzow noise suppression,CSPG CSEG CWLS Convention, pp. 303–306 (2008)
22. Trickeett, S., Burroughs, L.: Prestack rank-reduction-based noise suppression. CSEG Recorder **34**, 3193–3196 (2009)
23. Trickeett, S., Burroughs, L., Milton, A., Walton, L., Dack, R.: Rank-reduction-based trace interpolation, 81st Annual International Meeting, SEG,Expanded Abstracts, pp. 1989–1992 (2010)
24. Ulrych, T., Freire, S., Siston, P.: Eigenimage Processing of Seismic Sections. SEG Extended Abstr. **7**, 1261 (1988)
25. van Loan, C.: Computational frameworks for the fast Fourier transform, SIAM. Pennsylvania, Philadelphia (1992)
26. Xu, W., Qiao, S.: A twisted factorization method for symmetric SVD of a complex symmetric tridiagonal matrix. Numer. Linear Algebra Appl. **16**, 801–815 (2009)
27. Xu, W., Qiao, S.: A fast SVD algorithm for square hankel matrices. Linear Algebra Appl. **428**, 550–563 (2008)
28. Zhang, H., Thurber, C.H.: Estimating the model resolution matrix for large seismic tomography problems based on Lanczos bidiagonalization with partial reorthogonalization. Geophys. J. Int. **170**, 337–345 (2007)
29. Zhang, L., Peng, W., Yuan, C.: An improved method for noise reduction based on singular value decomposition. Chin. J. Ship Res. **7**(5), 83–88 (2012)