Statistics and Its Interface Volume 3 (2010) 357–368

Computation- and space-efficient implementation of SSA

Anton Korobeynikov

The computational complexity of different steps of the Basic SSA is discussed. It is shown that the use of the general-purpose "blackbox" routines which can be found in packages like `LAPACK` leads to a huge waste of time since the Hankel structure of the trajectory matrix is not taken into account.

We outline several state-of-the-art algorithms including the Lanczos-based truncated Singular Value Decomposition (SVD) which can be modified to exploit the structure of the trajectory matrix. The key components here are Hankel matrix-vector multiplication and the hankelization operator. We show that both operations can be computed efficiently by means of the Fast Fourier Transform.

The use of these methods yields the reduction of the worst-case computational complexity from $O(N^3)$ to $O(kN \log N)$, where N is the series length and k is the number of desired eigentriples.

AMS 2000 subject classifications: 65C60, 68Q17.
Keywords and phrases: Basic SSA, Computational complexity, Truncated SVD, Hankelization, Fast Fourier Transform.

1. INTRODUCTION

Despite the recent growth of SSA-related techniques it seems that little attention was paid to their efficient implementations, e.g. the singular value decomposition of the Hankel trajectory matrix, which is a dominating time-consuming step of the Basic SSA.

In almost all SSA-related applications only a few leading eigentriples are desired, thus the use of general-purpose "blackbox" SVD implementations causes a huge waste of time and memory resources. This almost prevents the use of the optimum window lengths even for time series of moderate length (say, few thousand). The problem is much more severe for 2D-SSA [15] or subspace-based methods [2, 8, 13], where a window size is typically large.

We show that the proper use of the state-of-the-art singular value decomposition algorithms can significantly reduce the amount of operations needed to compute the truncated SVD suitable for SSA purposes. This is possible because the information about the leading singular triples becomes available long before the costly full decomposition.

These algorithms are usually based on the Lanczos iterations or related methods. Such methods can be modified to exploit the Hankel structure of the data matrix effectively. Thus the computational burden of SVD can be considerably reduced.

In Section 2 we outline the computational and storage complexity of the steps of the Basic SSA algorithm. Section 3 contains an overview of well-known results connected with the Lanczos singular value decomposition algorithm. Generic implementation issues in finite-precision arithmetics are discussed as well. In Section 4 the derivation of the efficient matrix-vector multiplication method for the Hankel SSA trajectory matrices is presented. This method is a key component of the fast truncated SVD for the Hankel SSA trajectory matrices. In Section 5 we exploit the special structure of the rank 1 hankelization operator which allows its efficient implementation. Finally, in Sections 6 and 7 the performance of algorithms is studied on simulated and real data.

2. COMPLEXITY OF THE BASIC SSA ALGORITHM

Let $N > 2$. Denote by $F = (f_1, \dots, f_N)$ a real-valued time series of length N. Fix the *window length* L, $1 < L < N$. The Basic SSA algorithm for the decomposition of a time series consists of four steps [14]. We consider each step separately and discuss its computational and storage complexity.

2.1 Embedding

The embedding step maps the original time series to a sequence of lagged vectors. The embedding procedure forms $K = N - L + 1$ *lagged vectors* $X_i \in \mathbb{R}^L$:

$$X_i = (f_i, \dots, f_{i+L-1})^T, \quad 1 \le i \le K.$$

These vectors form the *L-trajectory* (or *trajectory*) matrix X of the series F:

$$X = \begin{pmatrix} f_1 & f_2 & \cdots & f_{K-1} & f_K \\ f_2 & f_3 & \cdots & f_K & f_{K+1} \\ \vdots & \vdots & & \vdots & \vdots \\ f_L & f_{L+1} & \cdots & f_{N-1} & f_N \end{pmatrix}.$$

Note that this matrix has equal values along the antidiag-

arXiv: 0911.4498v2

onals, thus it is a *Hankel matrix*. The computational complexity of this step is negligible since it consists of simple data copying which is usually considered as a cheap procedure. However, if the matrix is stored as-is, then this step obviously has the storage complexity of $O(LK)$ with the worst case being $O(N^2)$ when $K \sim L \sim N/2$.

2.2 Singular value decomposition

At this step the singular value decomposition of the $L \times K$ trajectory matrix X is performed. Without loss of generality we assume that $L \le K$. Then the SVD can be written as

$$X = U^T \Sigma V,$$

where $U = (u_1, \dots, u_L)$ is a $L \times L$ orthogonal matrix, $V = (v_1, \dots, v_K$ is a $K \times K$ orthogonal matrix, and Σ is a $L \times K$ diagonal matrix with nonnegative real diagonal entries $\Sigma_{ii} = \sigma_i$, for $i = 1, \dots, L$. The vectors u_i are called *left singular vectors*, the v_i are the *right singular vectors*, and the σ_i are the *singular values*. Let the singular values be arranged in the descending order: $\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_L$. Note that the singular value decomposition of X can be represented in the form

$$X = X_1 + \cdots + X_L,$$

where $X_i = \sigma_i u_i v_i^T$. The matrices X_i have rank 1, therefore we call them *elementary matrices*. The triple (σ_i, u_i, v_i) will be called an *eigentriple*.

Usually the SVD is computed by means of the Golub and Reinsh algorithm [12] which requires $O(L^2K + LK^2 + K^3)$ multiplications having the worst-case computational complexity $O(N^3)$ when $K \sim L \sim N/2$. Note that all the data must be processed at once and SVDs even for moderate length time series (say, when $N > 5,000$) are essentially unfeasible.

2.3 Grouping

The grouping procedure partitions the set of indices $\{1, \dots, L\}$ into m disjoint sets $I_1, \dots, I_m$. For the index set $I = \{i_1, \dots, i_p$ one can define the *resultant matrix* $X_I = X_{i_1} + \cdots + X_{i_p}$. Such matrices are computed for $I = I_1, \dots, I_m$; this leads to the decomposition

$$X = X_{I_1} + \cdots + X_{I_m}.$$

For the sake of simplicity we assume that $m = L$ and $I_j = \{j\}$ for all j later on. The computation of each elementary matrix X_j costs $O(LK)$ multiplications and $O(LK)$ storage yielding $O(L^2K)$ for the overall computation complexity and $O(L^2K)$ for storage. Again the worst case is achieved at $L \sim K \sim N/2$ with the complexity $O(N^3)$ for both computation and storage.

2.4 Diagonal averaging (hankelization)

At the last step of the Basic SSA algorithm each matrix X_{I_j} is transformed into a new time series of length N. This is performed by means of the *diagonal averaging procedure* (or *hankelization*). It transforms an arbitrary $L \times K$ matrix Y to the series $g_1, \dots, g_N$ by the formula:

$$(1)\quad g_k = \begin{cases} \dfrac{1}{k} \displaystyle\sum_{j=1}^{k} y_{j,k-j+1}, & 1 \le k \le L, \\ \dfrac{1}{L} \displaystyle\sum_{j=1}^{L} y_{j,k-j+1}, & L < k < K, \\ \dfrac{1}{N-k+1} \displaystyle\sum_{j=k-K+1}^{L} y_{j,k-j+1}, & K \le k \le N. \end{cases}$$

Note that the third and forth steps are usually combined. One just forms the elementary matrices one by one and immediately applies the hankelization operator. This considerably reduces the storage requirements.

Note that $O(N)$ multiplications and $O(LK)$ additions are required for performing the diagonal averaging of one elementary matrix. Summing these values for L elementary matrices we obtain the overall computation complexity being $O(L^2K)$ additions and $O(NL)$ multiplications. The worst-case complexity is $O(N^3)$ additions and $O(N^2)$ multiplications.

Summing the complexity values altogether for all four steps we obtain the worst case computational complexity to be $O(N^3)$. The worst case occurs when the window length L equals $N/2$ (the optimal window length for the asymptotic separability [14]).

3. TRUNCATED SINGULAR VALUE DECOMPOSITION

The typical approach to the problem of computing the eigentriples (σ_i, u_i, v_i) of A is to use the Schur decomposition of some matrices related to A, namely

1. the *cross product* matrices AA^T, A^TA, or
2. the *cyclic* matrix $C = \left(\begin{smallmatrix} 0 & A \\ A^T & 0 \end{smallmatrix}\right)$.

The proof of the following theorem can be found in [11]:

Theorem 1. *Let A be an $L \times K$ matrix and assume without loss of generality that $K \ge L$. Let the singular value decomposition of A be*

$$U^T A V = \operatorname{diag}(\sigma_1, \dots, \sigma_L).$$

Then

$$V^T (A^T A) V = \operatorname{diag}(\sigma_1^2, \dots, \sigma_L^2, \underbrace{0, \dots, 0}_{K-L}),$$

$$U^T (A A^T) U = \operatorname{diag}(\sigma_1^2, \dots, \sigma_L^2).$$

Moreover, if V is partitioned as

$$V = (V_1, V_2), \quad V_1 \in \mathbb{R}^{K\times L}, V_2 \in \mathbb{R}^{K\times(K-L)},$$

then the orthonormal columns of the $(L+K)\times(L+K)$ matrix

$$Y = \frac{1}{\sqrt{2}}\begin{pmatrix} U & -U & 0 \\ V_1 & V_1 & \sqrt{2}V_2 \end{pmatrix}$$

form an eigenvector basis for the cyclic matrix C and

$$Y^T C Y = \operatorname{diag}(\sigma_1, \ldots, \sigma_L, -\sigma_1, \ldots, -\sigma_L, \underbrace{0, \ldots, 0}_{K-L}).$$

This theorem tells us that we can obtain the singular values and vectors of A by computing the eigenvalues and corresponding eigenvectors of one of the symmetric matrices. This fact forms the basis of any SVD algorithm.

In order to find all eigenvalues and eigenvectors of the real symmetric matrix one usually transforms it into a similar tridiagonal matrix by means of orthogonal transformations (Householder rotations or alternatively via Lanczos recurrences). Then, for example, another series of orthogonal transformations can be applied to the latter tridiagonal matrix which converges to a diagonal matrix, see [11].

This approach, which in its original form is due to Golub and Reinsh [12] and is used in e.g. `LAPACK` [1], computes the SVD by implicitly applying the QR algorithm to the symmetric eigenvalue problem for A^TA.

For large and sparse matrices the Golub-Reinsh algorithm is impractical. The algorithm itself starts out by applying a series of transformations directly to the matrix A to reduce it to the special bidiagonal form. Therefore it requires the matrix to be stored explicitly, which may be impossible simply due to its size. Moreover, it may be difficult to take advantage of any structure (e.g. Hankel in the SSA case) or sparsity in A since it is quickly destroyed by the transformations applied to the matrix. The same argument applies to the SVD computation via the Jacobi algorithm.

Bidiagonalization was proposed by Golub and Kahan [10] as a way of tridiagonalizing the cross product matrix without forming it explicitly. The method yields the decomposition

$$A = P^T B Q,$$

where P and Q are orthogonal matrices and B is an $L\times K$ lower bidiagonal matrix. Here BB^T is similar to AA^T. Additionally, there are well-known SVD algorithms for real bidiagonal matrices, for example, the QR method [11], the divide-and-conquer algorithm [16], and the twisted method [7].

3.1 Lanczos bidiagonalization

For a rectangular $L\times K$ matrix A the Lanczos bidiagonalization computes a sequence of *left Lanczos vectors* $u_j \in \mathbb{R}^L$ and *right Lanczos vectors* $v_j \in \mathbb{R}^K$ and scalars α_j and β_j, $j = 1, \ldots, k$ as follows:

Algorithm 1: Golub-Kahan-Lanczos Bidiagonalization

1 Choose a starting vector $p_0 \in \mathbb{R}^L$
2 $\beta_1 \longleftarrow \|p_0\|_2$, $u_1 \longleftarrow p_0/\beta_1$, $v_0 \longleftarrow 0$
3 **for** $j \leftarrow 1$ **to** k **do**
4 $\quad r_j \longleftarrow A^T u_j - \beta_j v_{j-1}$
5 $\quad \alpha_j \longleftarrow \|r_j\|_2, \quad v_j \longleftarrow r_j/\alpha_j$
6 $\quad p_j \longleftarrow A v_j - \alpha_j u_j$
7 $\quad \beta_{j+1} \longleftarrow \|p_j\|_2, \quad u_{j+1} \longleftarrow p_j/\beta_{j+1}$
8 **end**

In what follows we refer to one iteration of the **for** loop at line 3 as a *Lanczos step* (or *iteration*). After k steps we have the lower bidiagonal matrix

$$B_k = \begin{pmatrix} \alpha_1 & & & & \\ \beta_2 & \alpha_2 & & & \\ & \beta_3 & \ddots & & \\ & & \ddots & \alpha_k & \\ & & & \beta_{k+1} & \end{pmatrix}.$$

If all the computations performed are exact, then the Lanczos vectors are orthonormal and satisfy the recurrences

$$\alpha_j v_j = A^T u_j - \beta_j v_{j-1}, \tag{2}$$

$$\beta_{j+1} u_{j+1} = A v_j - \alpha_j u_j. \tag{3}$$

Moreover, $u_k \in \mathcal{K}_k(AA^T, u_1)$, $v_k \in \mathcal{K}_k(A^TA, v_1)$, where

$$\mathcal{K}_k\left(AA^T, u_1\right) = \left\{u_1, AA^T u_1, \ldots, \left(AA^T\right)^k u_1\right\}, \tag{4}$$

$$\mathcal{K}_k\left(A^TA, v_1\right) = \left\{v_1, A^TA v_1, \ldots, \left(A^TA\right)^k v_1\right\}, \tag{5}$$

and therefore $u_1, u_2, \ldots, u_k$ and $v_1, v_2, \ldots, v_k$ form an orthogonal basis for these two Krylov subspaces.

3.2 Lanczos bidiagonalization in inexact arithmetics

When the Lanczos bidiagonalization is carried out in finite precision arithmetics the Lanczos recurrence becomes

$$\alpha_j v_j = A^T u_j - \beta_j v_{j-1} + f_j,$$
$$\beta_{j+1} u_{j+1} = A v_j - \alpha_j u_j + g_j,$$

where $f_j \in \mathbb{R}^K$ and $g_j \in \mathbb{R}^L$ are error vectors accounting for the rounding errors at the j-th step. Usually, the rounding terms are small, thus after k steps the equations (2) and (3) still hold to almost machine accuracy. In contrast, the orthogonality among the left and right Lanczos vectors is lost.

However, the Lanczos algorithm possesses a remarkable feature that the accuracy of the converged Ritz values is not

affected by the loss of orthogonality, but while the largest singular values of B_k are still accurate approximations of the largest singular values of A, the spectrum of B_k additionally contains false multiple copies of converged Ritz values (this happens even if the corresponding true singular values of A are isolated). Moreover, spurious singular values ("ghosts") periodically appear between the already converged Ritz values.

There are two different approaches which can be used to derive a robust method in finite arithmetics.

3.2.1 Lanczos bidiagonalization with no orthogonalization

One approach is to apply the simple Lanczos process "as-is" and subsequently use some criterion to detect and remove spurious singular values.

The advantage of this method is that it completely avoids any extra work connected with the reorthogonalization, and the storage requirements are very low since only a few latest Lanczos vectors have to be remembered. The disadvantage is that many iterations are wasted on simply generating multiple copies of the large values [22]. The number of extra iterations required compared to that when executing the Lanczos process in exact arithmetics can be very large: up to six times the original number as reported in [23]. Another disadvantage is that the criterion mentioned above can be rather difficult to implement, since it depends on the correct choice of different thresholds.

3.2.2 Lanczos bidiagonalization using reorthogonalization

Another way to deal with loss of orthogonality among Lanczos vectors is to apply a *reorthogonalization* scheme. The simplest one is the so-called *full reorthogonalization* (FRO) where each new Lanczos vector u_{j+1} (or v_{j+1}) is orthogonalized against all the Lanczos vectors generated so far using, e.g., the modified Gram-Schmidt algorithm.

This approach is often too expensive, since for a large window length the running time needed for reorthogonalization quickly dominates the execution time, unless the necessary number of iterations k is very small compared to the dimension of the problem (see Section 3.3 for the full analysis of the complexity). The storage requirements of FRO might also be a limiting factor since all the generated Lanczos vectors need to be saved.

A number of different schemes for reducing the work associated with keeping the Lanczos vectors orthogonal was developed (see [3] for an extensive review). The main idea is to estimate the *level of orthogonality* among the Lanczos vectors and perform the orthogonalization only when necessary.

For example, the so-called *partial reorthogonalization scheme* (PRO) described in [26, 25] uses the fact that the level of orthogonality among the Lanczos vectors satisfies some recurrence relation which can be derived from the recurrence used to generate the vectors themselves.

3.3 The complexity of the Lanczos bidiagonalization

The computational complexity of the Lanczos bidiagonalization is determined by two matrix-vector multiplications, thus the overall complexity of k Lanczos iterations in exact arithmetics is $O(kLK)$ multiplications.

The computational cost increases when one deals with the loss of orthogonality due to inexact computations. For the FRO scheme the additional $O(k^2(L+K))$ operations required for the orthogonalization quickly dominate the execution time, unless the necessary number of iterations k is very small compared to the dimension of the matrix. The storage requirements of FRO may also be a limiting factor since all the generated Lanczos vectors need to be saved.

In general, there is no way to determine in advance how many steps will be needed to provide the singular values of interest within a specified accuracy. Usually the number of steps is determined by the distribution of the singular numbers and the choice of the starting vector p_0. In the case when the singular values form a cluster the convergence does not occur until k, the number of iterations, gets very large. In such a situation the method also suffers from the increased storage requirements: it is easy to see that they are $O(k(L+K))$.

These computational and storage problems can be usually solved via limiting the dimension of the Krylov subspaces (4) and (5) and using *restarting schemes*: restarting the iterations after a number of steps by replacing the starting vector with some "improved" starting vector (ideally, we would like p_0 to be a linear combination of the right singular vectors of A associated with the desired singular values). Thus, if we limit the dimension of the Krylov subspaces to d the storage complexity drops to $O(d(L+K))$ (surely the number of desired eigentriples should be greater than d). See [3] for the review of the restarting schemes proposed so far and introduction to the thick-restarted Lanczos bidiagonalization algorithm. In the papers [29, 30] different strategies of choosing the best dimension d of Krylov subspaces are discussed.

If the matrix being decomposed is sparse or structured (for example, Hankel in the Basic SSA case or Hankel with Hankel blocks in 2D-SSA) then the computational complexity of the bidiagonalization can be significantly reduced if the efficient matrix-vector multiplication routine is available.

4. EFFICIENT MATRIX-VECTOR MULTIPLICATION FOR HANKEL MATRICES

In this section we present the efficient matrix-vector multiplication algorithm for the Hankel SSA trajectory matrix. By means of the Fast Fourier Transform (FFT) the cost of the multiplication drops from ordinary $O(LK)$ multiplications to $O((L+K)\log(L+K))$. The idea of using the FFT for computing a Hankel matrix-vector product is not new: it

was probably first described by Bluestein in 1968 (see [27]), later references are [20, 5].

This approach reduces the computational complexity of the singular value decomposition of Hankel matrices from $O(kLK+k^2(L+K))$ to $O(k(L+K)\log(L+K)+k^2(L+K))$. The complexity in the worst case when $K \sim L \sim N/2$ drops significantly from $O(kN^2+k^2N)$ to $O(kN\log N+k^2N)$.[1]

Also the described algorithm provides an efficient way to store the Hankel SSA trajectory matrix reducing the storage requirements from $O(LK)$ to $O(L+K)$.

4.1 Matrix representation of discrete Fourier transform and circulant matrices

The 1-d discrete Fourier transform (DFT) of a (complex) vector $(f_k)_{k=0}^{N-1}$ is defined by:

$$\hat{f}_l = \sum_{k=0}^{N} e^{-2\pi i k l/N} f_k, \quad l = 0, \dots, N-1.$$

Denote by ω the primitive N-root of unity, $\omega = e^{-2\pi i/N}$. Introduce the DFT matrix $\mathbb{F}_N$:

$$\mathbb{F}_N = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \cdots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{N-1} & \cdots & \omega^{(N-1)(N-1)} \end{pmatrix}.$$

Then the 1-d DFT can be written in the matrix form: $\hat{f} = \mathbb{F}_N f$. The inverse of the DFT matrix is given by

$$\mathbb{F}_N^{-1} = \frac{1}{N}\mathbb{F}_N^*,$$

where $\mathbb{F}_N^*$ is the adjoint (conjugate transpose) of DFT matrix $\mathbb{F}_N$. Thus, the inverse 1-d discrete Fourier transform (IDFT) is given by

$$f_k = \frac{1}{N}\sum_{l=0}^{N-1} e^{2\pi i k l/N}\hat{f}_l, \quad k = 0, \dots, N-1.$$

The fast Fourier transform is an efficient algorithm to compute the DFT (and IDFT) with $O(N\log N)$ complex multiplications instead of N^2 complex multiplications in direct implementation of the DFT [19, 9].

Definition 1. *An $N \times N$ matrix C of the form*

$$C = \begin{pmatrix} c_1 & c_N & c_{N-1} & \cdots & c_3 & c_2 \\ c_2 & c_1 & c_N & \cdots & c_4 & c_3 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ c_{N-1} & c_{N-2} & c_{N-3} & \cdots & c_1 & c_N \\ c_N & c_{N-1} & c_{N-2} & \cdots & c_2 & c_1 \end{pmatrix}$$

is called a **circulant matrix.**

[1] Compare with $O(N^3)$ for full SVD via the Golub-Reinsh method.

A circulant matrix is fully specified by its first column $c = (c_1, c_2, \dots, c_N)^T$. The remaining columns of C are each cyclic permutations of the vector c with offset equal to the column index. The last row of C is the vector c in reverse order, and the remaining rows are each cyclic permutations of the last row.

The eigenvectors of a circulant matrix of a given size are the columns of the DFT matrix of the same size. Thus a circulant matrix is diagonalized by the DFT matrix:

$$C = \mathbb{F}_N^{-1}\,\mathrm{diag}\,(\mathbb{F}_N c)\mathbb{F}_N,$$

so the eigenvalues of C are given by $\mathbb{F}_N c$.

This factorization can be used to perform efficient matrix-vector multiplication. Let $v \in \mathbb{R}^N$ and C be an $N \times N$ circulant matrix. Then

$$Cv = \mathbb{F}_N^{-1}\,\mathrm{diag}\,(\mathbb{F}_N c)\mathbb{F}_N v = \mathbb{F}_N^{-1}\left(\mathbb{F}_N c \odot \mathbb{F}_N v\right),$$

where $\odot$ denotes the element-wise vector multiplication. This can be computed efficiently using the FFT by first computing the two DFTs, $\mathbb{F}_N c$ and $\mathbb{F}_N v$, and then computing the IDFT $\mathbb{F}_N^{-1}(\mathbb{F}_N c \odot \mathbb{F}_N v)$. If the matrix-vector multiplication is performed repeatedly with the same circulant matrix and different vectors, then the DFT $\mathbb{F}_N c$ needs to be computed only once.

In this way the overall computational complexity of matrix-vector product for the circulant matrices drops from $O(N^2)$ to $O(N\log N)$.

4.2 Toeplitz and Hankel matrices

Definition 2. *A $L \times K$ matrix of the form*

$$T = \begin{pmatrix} t_K & t_{K-1} & \cdots & t_2 & t_1 \\ t_{K+1} & t_K & \cdots & t_3 & t_2 \\ \vdots & \vdots & & \vdots & \vdots \\ t_{K+L-1} & t_{K+L-2} & \cdots & t_{L+1} & t_L \end{pmatrix}$$

is called a (non-symmetric) ***Toeplitz matrix****.*

A Toeplitz matrix is completely determined by its last column and last row, and thus depends on $K+L-1$ parameters. The entries of T are constant down the diagonals parallel to the main diagonal.

Given an algorithm for the fast matrix-vector product for circulant matrices, it is easy to obtain the algorithm for a Toeplitz matrix, since a Toeplitz matrix can be embedded into a circulant matrix C_{K+L-1} of size $K+L-1$ with the first column equals to $(t_K, t_{K+1}, \dots, t_{K+L-1}, t_1, t_2, \dots, t_{K-1})^T$. Obviously, the leading $L \times K$ principal submatrix of C_{K+L-1} is T. This technique of embedding a Toeplitz matrix in a larger circulant matrix to achieve fast computation is widely used in preconditioning methods [6].

Using this embedding, the Toeplitz matrix-vector product Tv can be computed efficiently in $O((K+L)\log(K+L))$

time and $O(K+L)$ memory using the FFT as described in Section 4.1.

A Hankel matrix H of size $L \times K$ is completely determined by its first column and last row, and thus depends on $K + L - 1$ parameters. The entries of H are constant along the antidiagonals.

One can easily convert a Hankel matrix into a Toeplitz matrix by reversing its columns. Indeed, let P be the $K \times K$ matrix whose elements on the antidiagonals are 1 and all other elements are 0. Then HP is a Toeplitz matrix for any Hankel matrix H, and TP is a Hankel matrix for any Toeplitz matrix T. Note that $P = P^T = P^{-1}$ as well. Now for the product Hv of a Hankel matrix H and a vector v we have

$$Hv = HPPv = (HP)\,Pv = Tw,$$

where T is a Toeplitz matrix and vector w is obtained as a vector v with entries in reversed order. The product Tw can be computed using the circulant embedding procedure as described earlier.

4.3 Hankel SSA trajectory matrices

Now we are ready to exploit the connection between the time series $F = (f_j)_{j=1}^N$ under decomposition and the corresponding trajectory Hankel matrix to derive the matrix-vector multiplication algorithm in terms of the series itself. In this way we effectively skip the embedding step of the SSA algorithm and reduce the computational and storage complexity.

The entries of the trajectory matrix H of the series are $h_j = f_j$, $j = 1, \ldots, N$. The entries of Toeplitz matrix $T = HP$ are $t_j = h_j = f_j$, $j = 1, \ldots, N$. The corresponding first column of the circulant embedding matrix C_{K+L-1} is

$$c_j = \begin{cases} t_{N-L+j} = f_{N-L+j}, & 1 \le j \le L; \\ t_{j-L+1} = f_{j-L+1} & L < j \le N, \end{cases}$$

and we obtain the following algorithm for the matrix-vector multiplication for the trajectory matrix:

Algorithm 2: Matrix-Vector Multiplication for the SSA Trajectory Matrix

Input: A time series $F = (f_j)_{j=1}^N$, a window length L, a vector v of length $N - L + 1$.
Output: $p = Xv$, where X is a trajectory matrix for F given window length L.

1. $c \longleftarrow (f_{N-L+1}, \ldots, f_N, f_1, \ldots, f_{N-L})^T$
2. $\hat{c} \longleftarrow FFT_N(c)$
3. $w \longleftarrow (v_{N-L+1}, \ldots, v_1, 0, \ldots, 0)^T$
4. $\hat{w} \longleftarrow FFT_N(w)$
5. $p' \longleftarrow IFFT_N(\hat{w} \odot \hat{c})$
6. $p \longleftarrow (p'_1, \ldots, p'_L)^T$

where $(\hat{w} \odot \hat{c})$ denotes the element-wise vector multiplication.

If the matrix-vector multiplication Xv is performed repeatedly with the same matrix X and different vectors v then steps 1 and 2 should be performed only once at the beginning and the resulting vector $\hat{c}$ should be reused later on.

The overall computational complexity of the matrix-vector multiplication is $O(N \log N)$. Storage of size $O(N)$ is required to store the precomputed vector $\hat{c}$.

Note that the matrix-vector multiplication of the transposed trajectory matrix X^T can be performed using the same vector $\hat{c}$. Indeed, the bottom-right $K \times L$ submatrix of circulant embedding matrix C_{K+L-1} contains the Toeplitz matrix $X^T P$ and we can easily modify Algorithm 2 as follows:

Algorithm 3: Matrix-Vector Multiplication for the transpose of SSA Trajectory Matrix

Input: A time series $F = (f_j)_{j=1}^N$, a window length L, a vector v of length L.
Output: $p = X^T v$, where X is a trajectory matrix for F given window length L.

1. $c \longleftarrow (f_{N-L+1}, \ldots, f_N, f_1, \ldots, f_{N-L})^T$
2. $\hat{c} \longleftarrow FFT_N(c)$
3. $w \longleftarrow (0, \ldots, 0, v_L, \ldots, v_1)^T$
4. $\hat{w} \longleftarrow FFT_N(w)$
5. $p' \longleftarrow IFFT_N(\hat{w} \odot \hat{c})$
6. $p \longleftarrow (p'_{L-1}, \ldots, p'_N)^T$

5. EFFICIENT RANK 1 HANKELIZATION

Let us recall the *diagonal averaging* procedure as described in Section 2.4 which transforms an arbitrary $L \times K$ matrix Y into a Hankel one and therefore into a series g_k, $k = 1, \ldots, N$, using the formula (1).

Without loss of generality we consider only *rank 1 hankelization* when matrix Y is elementary and can be represented as $Y = \sigma u v^T$ with vectors u and v of length L and K, respectively. Then the equation (1) becomes

$$(6)\quad g_k = \begin{cases} \dfrac{\sigma}{k} \displaystyle\sum_{j=1}^{k} u_j v_{k-j+1}, & 1 \le k \le L, \\ \dfrac{\sigma}{L} \displaystyle\sum_{j=1}^{L} u_j v_{k-j+1}, & L < k < K, \\ \dfrac{\sigma}{N-k+1} \displaystyle\sum_{j=k-K+1}^{L} u_j v_{k-j+1}, & K \le k \le N. \end{cases}$$

This gives a hint how the diagonal averaging can be efficiently computed.

Indeed, consider the infinite series u'_n, $n \in \mathbb{Z}$ such that $u'_n = u_n$ when $1 \le n \le L$ and 0 otherwise. The series v'_n is defined in the same way, so $v'_n = v_n$ when $1 \le n \le K$ and 0

otherwise. The linear convolution $u'_n * v'_n$ of u'_n and v'_n can be written as follows:

(7)

$$(u'_n * v'_n)_k = \sum_{j=-\infty}^{+\infty} u'_j v'_{k-j+1} = \sum_{j=1}^{L} u'_j v'_{k-j+1}$$
$$= \begin{cases} \sum_{j=1}^{k} u_j v_{k-j+1}, & 1 \le k \le L, \\ \sum_{j=1}^{L} u_j v_{k-j+1}, & L < k < K, \\ \sum_{j=k-K+1}^{L} u_j v_{k-j+1}, & K \le k \le N. \end{cases}$$

Comparing the equations (1) and (7) we deduce that

$$g_k = c_k \left(u'_n * v'_n\right)_k,$$

where the constants c_k are known in advance.

The linear convolution $u*v$ can be defined in terms of the *circular convolution* as follows. Expand the vectors u and v to the length $L+K-1$ by filling them up with zeros. Then the linear convolution of the original vectors u and v is the same as the circular convolution of the extended series of length $L+K-1$, namely

$$(u * v)_k = \sum_{j=1}^{L+K-1} u_j v_{k-j+1},$$

where the index $k-j+1$ is evaluated by the module $(L+K-1)$. The resulting circular convolution can be calculated efficiently via the FFT [4]:

$$(u * v)_k = IFFT_N \left(FFT_N(u') \odot FFT_N(v')\right),$$

where $N = L+K-1$, and u', v' denote the zero-padded vectors u and v, respectively.

Thus, we obtain the following algorithm for the hankelization via convolution:

Algorithm 4: Rank 1 Hankelization via Linear Convolution

Input: A vector u of length L, a vector v of length K, a singular value σ.

Output: Time series $G = (g_j)_{j=1}^{N}$ corresponding to the matrix $Y = \sigma u v^T$ after hankelization.

1 $u' \longleftarrow (u_1, \ldots, u_L, 0, \ldots, 0)^T \in \mathbb{R}^N$
2 $\hat{u} \longleftarrow FFT_N(u')$
3 $v' \longleftarrow (v_1, \ldots, v_K, 0, \ldots, 0)^T \in \mathbb{R}^N$
4 $\hat{v} \longleftarrow FFT_N(v')$
5 $g' \longleftarrow IFFT_N(\hat{v} \odot \hat{u})$
6 $w \longleftarrow (1, \ldots, L, L, \ldots, L, L, \ldots, 1) \in \mathbb{R}^N$
7 $g \longleftarrow \sigma(w \odot g')$

The computational complexity of this algorithm is $O(N \log N)$ multiplications versus $O(LK)$ for the naïve direct approach. Basically this means that the worst-case complexity of the diagonal averaging drops from $O(N^2)$ to $O(N \log N)$.

6. IMPLEMENTATION COMPARISON

The algorithms presented in this paper have much better asymptotic complexity than standard ones, but obviously they might have much bigger overhead and thus would not be suitable for the real-world problems unless the series length and/or the window length become considerably large.

The presented algorithms were implemented by means of the `Rssa`[2] package for the `R` system for statistical computing [24]. All the comparisons were carried out on the 4 core AMD Opteron 2210 workstation running Linux. Where possible we tend to use state-of-the-art implementations of various computational algorithms (e.g. highly optimized `ATLAS` [28] and `ACML` implementations of `BLAS` and `LAPACK` routines). We used `R` version 2.8.0 throughout the tests.

We compare the performance of such key components of the fast SSA implementation as rank 1 diagonal averaging and the Hankel matrix-vector multiplication. The performance of the methods which use the whole SSA method, namely the bootstrap confidence interval construction and structure change detection, is considered as well.

6.1 Fast Hankel matrix-vector product

The fast Hankel matrix-vector multiplication is the key component for the Lanczos bidiagonalization. The algorithm outlined in Section 4.3 was implemented in two different ways: one using the core `R` FFT implementation and another one using FFT from the FFTW library [9]. We selected the window length equal to one half of the series length since this corresponds to the worst case both in terms of computational and storage complexity. All initialization times (for example, Hankel matrix construction or circulant precomputation times) were excluded from the timings since they are performed only once at the beginning. So, we compare the performance of the generic `DGEMV` [1] matrix-vector multiplication routine versus the performance of the special routine exploiting the Hankel structure of the matrix.

The running times for 100 matrix-vector products with different series lengths are presented in Figure 1. Note that we had to use the logarithmic scale for the y-axis in order to outline the difference between the generic routines and our implementations properly. One can easily see that all the routines possess the expected asymptotic complexity behaviour. Also the special routines are almost always faster than generic ones, thus we recommend to use them for any window lengths (the overhead for small series lengths can be neglected).

[2]The package will be submitted to CRAN soon, yet it can be obtained from author.

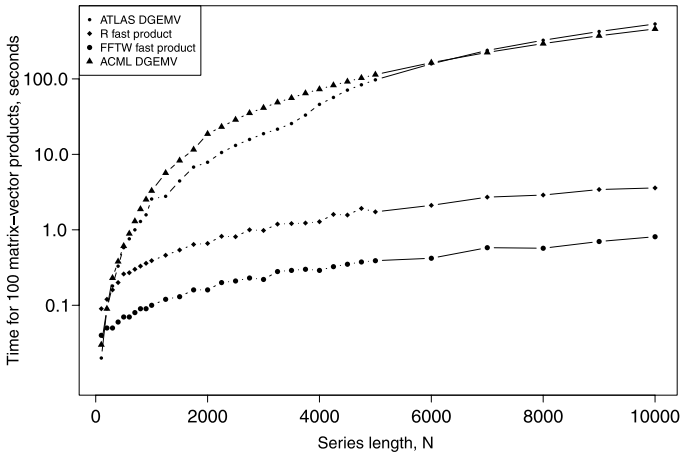

Figure 1. Dependence of the Hankel matrix-vector multiplication time on N for different methods.

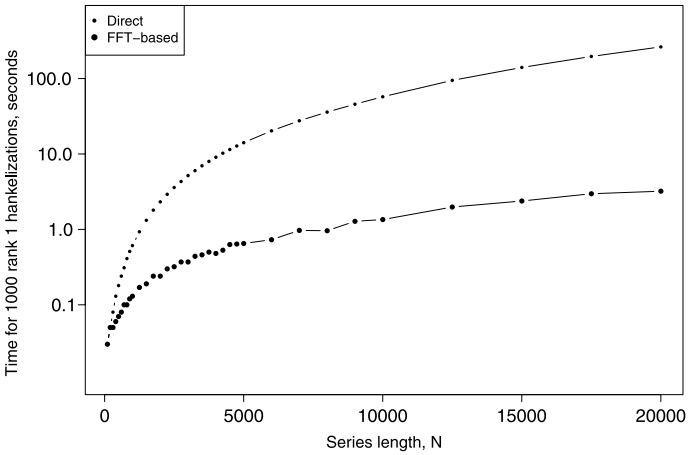

Figure 2. Dependence of the rank 1 hankelization time on N for different methods.

6.2 Rank 1 hankelization

The data access during the straightforward implementation of rank 1 hankelization is not so regular, thus the pure `R` implementation turned out to be slow and we haven't considered it at all. In fact, two implementations are under comparison: the straightforward `C` implementation and the FFT-based one as described in Section 5. For the latter the FFTW library was used. The window length was equal to half of the series length to outline the worst possible case.

The results are shown in Figure 2. As before, the logarithmic scale was used for the y-axis. From this figure one can see that the computational complexity of the direct implementation of the hankelization operation quickly dominates the overhead of the more complex FFT-based one and thus the latter can be readily used for all non-trivial series lengths.

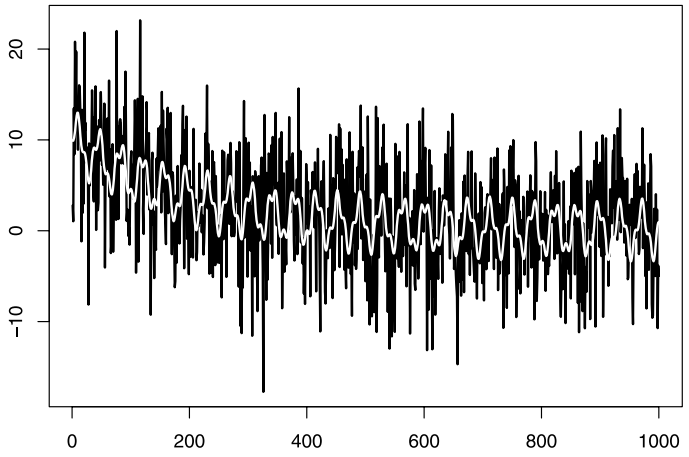

Figure 3. The series F_{1000} and F'_{1000}.

6.3 Bootstrap confidence intervals construction

For the comparison of the implementations of the whole SSA algorithm the problem of bootstrap confidence intervals construction was selected. It can be described as follows: consider the series F_N of finite rank d. Form the series $F'_N = F_N + \sigma\varepsilon_N$, where ε_i denotes the white noise (thus the series F'_N is of full rank). Fix the window length $L < N$ and let G_N denote the series reconstructed from first d eigentriples of series F'_N. Thus G_N is a series of rank d as well.

Since the original series F_N is considered to be known, we can study large-sample properties of the estimate G_N: bias, variance and mean square error. Having the variance estimate one can, for example, construct bootstrap confidence intervals for the mean of the reconstructed series G_N.

Such simulations are usually quite time-consuming since we need to perform a few dozen reconstructions for the given length N in order to estimate the variance.

For the simulation experiments we consider the series $F_N = (f_1, \dots, f_N)$ of rank 5 with

$$f_n = 10e^{-5n/N} + \sin\left(\frac{2\pi}{13}\frac{n}{N}\right) + 2.5\sin\left(\frac{2\pi}{37}\frac{n}{N}\right)$$

and $\sigma = 5$. The series F_{1000} (white) and F'_{1000} (black) are shown in Figure 3.

We compare three different SSA implementations: using the full SVD via `DGESDD` routine from the `ATLAS` library, using the full SVD via the same routine from the `ACML` library and using the fast Hankel truncated SVD and FFT-based rank-1 hankelization. Again, the `FFTW` library was used to perform the FFT. The window length for SSA was always fixed to be one half of the series length.

The running times for the rank 5 reconstruction are presented in Figure 4. Note that the logarithmic scale was used for the y-axis. From Figure 4 we can easily see that the running times for the SSA implementation via the fast Hankel

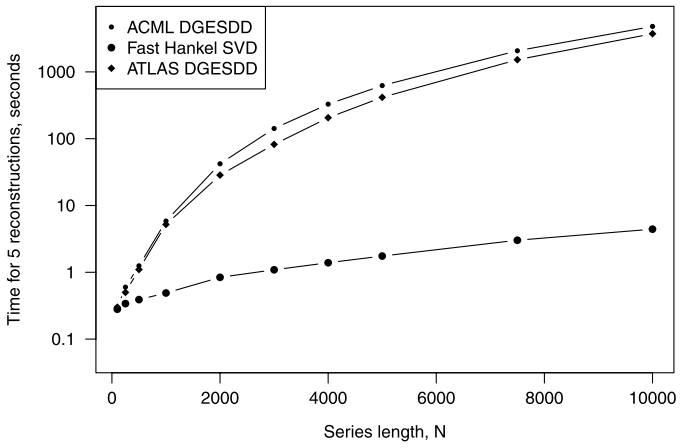

Figure 4. Dependence of the computational time on N for the series reconstruction.

SVD and FFT-based rank-1 hankelization are dramatically smaller compared to the running times of other implementations for any non-trivial series length.

6.4 Detection of structural changes

SSA can be used to detect the structural changes in the series. The main instrument for detection is the so-called *heterogeneity matrix* (H-matrix). We briefly describe the algorithm for constructing of such matrix, discuss the computation complexity of the construction and compare the performances of different implementations.

Exhaustive exposition of the detection of structural changes by means of SSA can be found in [14].

Consider two time series $F^{(1)}$ and $F^{(2)}$. Let N_1 and N_2 denote their lengths respectively. Take integer $L < \min(N_1 - 1, N_2)$. Let $U_j^{(1)}$, $j = 1, \dots, L$ denote the eigenvectors of the SVD of the trajectory matrix of the series $F^{(1)}$. Fix I to be a subset of $\{1, \dots, L\}$ and denote $\mathcal{L}^{(1)} = \operatorname{span}(U_i,\ i \in I)$. Denote by $X_1^{(2)}, \dots, X_{K_2}^{(2)}$ ($K_2 = N_2 - L + 1$) the L-lagged vectors of the series $F^{(2)}$.

Now we introduce the *heterogeneity index* [14] which measures the discrepancy between the series $F^{(2)}$ and the structure of the series $F^{(1)}$ (this structure is described by the subspace $\mathcal{L}^{(1)}$):

$$g(F^{(1)}, F^{(2)}) = \frac{\sum_{j=1}^{K_2} \operatorname{dist}^2(X_j^{(2)}, \mathcal{L}^{(1)})}{\sum_{j=1}^{K_2} \|X_j^{(2)}\|^2}, \tag{8}$$

where $\operatorname{dist}(X, \mathcal{L})$ denotes the Euclidean distance between the vector X and the subspace $\mathcal{L}$. One can easily see that $0 \le g \le 1$.

Note that since the vectors U_i form an orthonormal basis of the subspace $\mathcal{L}^{(1)}$ the equation (8) can be rewritten as

$$g(F^{(1)}, F^{(2)}) = 1 - \frac{\sum_{j=1}^{K_2} \sum_{i \in I} (U_i^T X_j^{(2)})^2}{\sum_{j=1}^{K_2} \|X_j^{(2)}\|^2}.$$

Having this discrepancy measure for two arbitrary series in the hand one can obviously construct the method of detection of structural changes in the single time series. Indeed, it is sufficient to calculate the heterogeneity index g for different pairs of subseries of the series F.

The *heterogeneity matrix* (H-matrix) [14] provides a consistent view on the structural discrepancy between different parts of the series. Denote by $F_{i,j}$ the subseries of F of the form: $F_{i,j} = (f_i, \dots, f_j)$. Fix two integers $B > L$ and $T \ge L$. Let these integers denote the lengths of *base* and *test* subseries, respectively. Introduce the H-matrix $G_{B,T}$ with the elements g_{ij} as follows:

$$g_{ij} = g(F_{i,i+B}, F_{j,j+T}),$$

for $i = 1, \dots, N - B + 1$ and $j = 1, \dots, N - T + 1$, that is we split the series F into subseries of lengths B and T and calculate the heterogeneity index between all possible pairs of the subseries.

The computation complexity of the calculation of the H-matrix is determined by the complexity of the SVDs for series $F_{i,i+B}$ and the complexity of calculation of all heterogeneity indices g_{ij} as defined by the equation (8).

The worst-case complexity of the SVD for the series $F_{i,i+B}$ corresponds to the case when $L \sim B/2$. Then the single decomposition costs $O(B^3)$ for the full SVD via the Golub-Reinsh method and $O(kB \log B + k^2 B)$ via the fast Hankel SVD as presented in Sections 3 and 4. Here k denotes the number of elements in the index set I. Since we need to make $N - B + 1$ decompositions we obtain the following complexities in the worst case (when $B \sim N/2$): $O(N^4)$ for the full SVD and $O(kN^2 \log N + k^2 N^2)$ for the fast Hankel SVD.

Having all the desired eigenvectors U_i for all the subseries $F_{i,i+B}$ in hand one can calculate the H-matrix $G_{B,T}$ in $O(kL(T - L + 1)(N - T + 1) + (N - T + 1)L)$ multiplications. The worst case corresponds to $L \sim T/2$ and $T \sim N/2$ yielding the $O(kN^3)$ complexity for this step. We should note that this complexity can be further reduced with the help of the fast Hankel matrix-vector multiplication, but for the sake of simplicity we omit this analysis.

Summing the complexities we obtain $O(N^4)$ multiplications for the H-matrix computation via the full SVD and $O(kN^3 + k^2 N^2)$ via the fast Hankel SVD.

For the implementation comparison we consider the series $F_N = (f_1, \dots, f_N)$ of the form

$$f_n = \begin{cases} \sin\left(\dfrac{2\pi}{10} n\right) + 0.01\varepsilon_n, & n < Q, \\ \sin\left(\dfrac{2\pi}{10.5} n\right) + 0.01\varepsilon_n, & n \ge Q, \end{cases} \tag{9}$$

where ε_n denotes the white noise.

The typical H-matrix for such series is shown in Figure 5 (compare with Figure 3.3 in [14]). The parameters used for

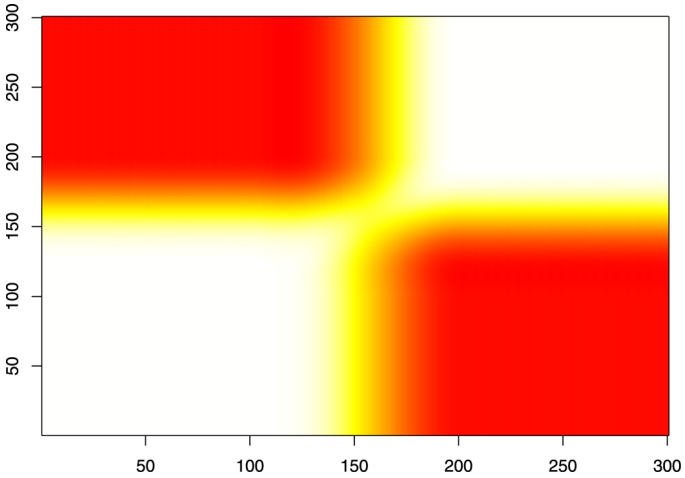

Figure 5. The H-matrix for the series (9).

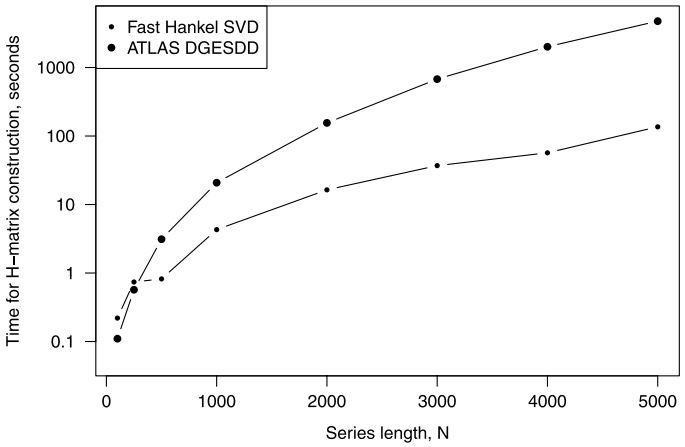

Figure 6. Dependence of the H-matrix construction time on N for different methods.

this matrix are $N = 400$, $Q = 200$, $B = 100$, $T = 100$, $L = 50$ and $I = \{1, 2\}$.

To save some computational time we compare not the complexities in the worst case, but some intermediate situation. The parameters used were $Q = N/2$, $B = T = N/4$, $L = B/2$ and $I = \{1, 2\}$. For the full SVD `DGESDD` routine from the `ATLAS` library [28] was used (as it was in Figure 1); it is the fastest full SVD implementation available for the workstation we used.

The obtained results are shown in Figure 6. As before, the logarithmic scale was used for the y-axis. Notice that the implementation of structure change detection using the fast Hankel SVD is more than 10 times faster even for the series of moderate length.

7. REAL-WORLD TIME SERIES

The fast SSA implementation enables us to use a large window length during the decomposition which was computationally unfeasible earlier. This may be a crucial point in achieving of asymptotic separability between different components of the series [14].

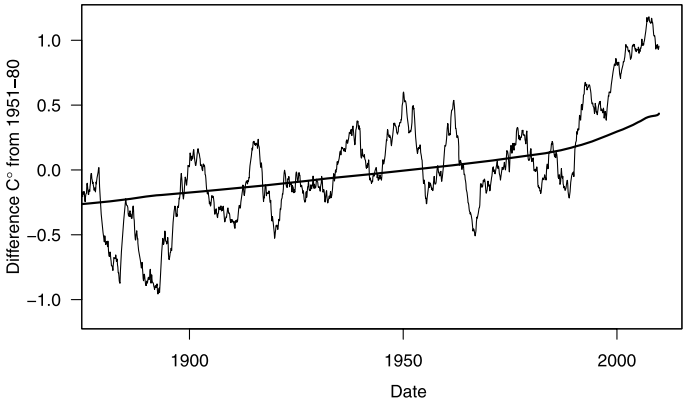

Figure 7. Hadley CET series and extracted trend.

7.1 Trend extraction for HadCET dataset

The Hadley Centre Central England Temperature (HadCET) dataset [21] is the longest instrumental record of temperature in the world. The mean daily data began in 1772 and is representative of a roughly triangular area of the United Kingdom enclosed by Lancashire, London and Bristol. The total series length is 86,867 (up to October, 2009).

We applied SSA with the window length of 43,433 to obtain the trend and a few seasonal components. The decomposition took 22.5 seconds and 50 eigentriples were computed. The first eigentriple was obtained as a representative for the trend of the series. The resulting trend on top of a 5-year running mean temperature anomaly relative to 1951–1980 is shown in Figure 7 (compare with Figure 1 in [17]).

7.2 Quebec birth rate series

The series obtained from [18] contains the number of daily births in Quebec, Canada, from January 1, 1977 to December 31, 1990. This series was discussed in [14] and it was shown that in addition to a smooth trend, two cycles of different ranges (the one-year periodicity and the one-week periodicity) can be extracted from the series.

However, the extraction of the one-year periodicity was incomplete since the series seems to be too smooth and thus it does not uncover the complicated structure of the annual periodicity (for example, it does not contain the birth rate drop during Christmas). The total series length is 5,113, which suggests the window length of 2,556 to be used to achieve the optimal separability. This seems to have been computationally expensive at the time when the book [14] was published (even nowadays the full SVD of such matrix using `LAPACK` takes several minutes and requires a decent amount of RAM).

We apply the sequential SSA approach: first we do the decomposition using the window length of 365. This allows

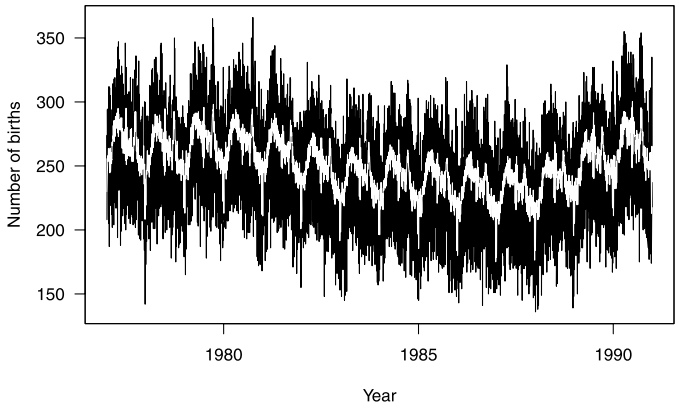

Figure 8. Quebec birth rate: trend and year periodicity.

us to extract the trend and weekly periodics in the same way as described in [14]. After this, we performed the decomposition of the residual series using the window length of $2,556$. 100 eigentriples were computed in 3.2 seconds. The decomposition clearly revealed the complicated structure of the annual periodicity: we identified many components corresponding to the periods of one-year, half-year, one third of the year, etc. In the end, the eigentriples $1-58$ excluding $22-24$ were used during the reconstruction. The reconstruction containing the trend and one-year periodicity is shown in Figure 8 (compare with Figure 1.9 in [14]) which explains the complicated series structure much better.

8. CONCLUSION

The stages of the Basic SSA algorithm were reviewed and their computational and space complexities were discussed. It was shown that the worst-case computational complexity of $O(N^3)$ can be reduced to $O(kN\log N + k^2N)$, where k is the number of desired eigentriples.

The comparison of implementations was presented and we have shown the superiority of the outlined algorithms in terms of the running time. This improvement in speed enables us to use the SSA algorithm for the decomposition of quite a long time series using the optimal window lengths for achieving the asymptotic separability.

ACKNOWLEDGEMENTS

Author would like to thank N.E. Golyandina and anonymous referees for thoughtful suggestions and comments that led to the significantly improved presentation of the paper.

Received 30 November 2009

REFERENCES

[1] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. (1999). *LAPACK Users' Guide*, Third ed. SIAM, Philadelphia, PA.

[2] Badeau, R., Richard, G., and David, B. (2008). Performance of ESPRIT for estimating mixtures of complex exponentials modulated by polynomials. *Signal Processing, IEEE Transactions on* **56**, 2 (Feb.), 492–504. MR2445532

[3] Baglama, J. and Reichel, L. (2005). Augmented implicitly restarted Lanczos bidiagonalization methods. *SIAM J. Sci. Comput.* **27**, 1, 19–42. MR2201173

[4] Briggs, W. L. and Henson, V. E. (1995). *The DFT: An Owner's Manual for the Discrete Fourier Transform*. SIAM, Philadelphia. MR1322049

[5] Browne, K., Qiao, S., and Wei, Y. (2009). A Lanczos bidiagonalization algorithm for Hankel matrices. *Linear Algebra and Its Applications 430*, 1531–1543. MR2490695

[6] Chan, R. H., Nagy, J. G., and Plemmons, R. J. (1993). FFT-Based preconditioners for Toeplitz-block least squares problems. *SIAM Journal on Numerical Analysis* **30**, 6, 1740–1768. MR1249041

[7] Dhillon, I. S. and Parlett, B. N. (2004). Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices. *Linear Algebra and its Applications 387*, 1–28. MR2069265

[8] Djermoune, E.-H. and Tomczak, M. (2009). Perturbation analysis of subspace-based methods in estimating a damped complex exponential. *Signal Processing, IEEE Transactions on* **57**, 11 (Nov.), 4558–4563.

[9] Frigo, M. and Johnson, S. G. (2005). The design and implementation of FFTW3. *Proceedings of the IEEE* **93**, 2, 216–231.

[10] Golub, G. and Kahan, W. (1965). Calculating the singular values and pseudo-inverse of a matrix. *SIAM J. Numer. Anal 2*, 205–224. MR0183105

[11] Golub, G. and Loan, C. V. (1996). *Matrix Computations*, 3 ed. Johns Hopkins University Press, Baltimore. MR1417720

[12] Golub, G. and Reinsch, C. (1970). Singular value decomposition and least squares solutions. *Numer. Math. 14*, 403–420. MR1553974

[13] Golyandina, N. (2010). On the choice of parameters in Singular Spectrum Analysis and related subspace-based methods. *Statistics and Its Interface* **3**, 259–279.

[14] Golyandina, N., Nekrutkin, V., and Zhiglyavsky, A. (2001). *Analysis of time series structure: SSA and related techniques*. CRC Press. MR1823012

[15] Golyandina, N. and Usevich, K. (2009). 2D-extensions of singular spectrum analysis: algorithm and elements of theory. In *Matrix Methods: Theory, Algorithms, Applications*. World Scientific Publishing, 450–474.

[16] Gu, M. and Eisenstat, S. C. (1995). A divide-and-conquer algorithm for the bidiagonal SVD. *SIAM J. Matrix Anal. Appl.* **16**, 1, 79–92. MR1311419

[17] Hansen, J., Sato, M., Ruedy, R., Lo, K., Lea, D. W., and Elizade, M. M. (2006). Global temperature change. *Proceedings of the National Academy of Sciences* **103**, 39 (September), 14288–14293.

[18] Hipel, K. W. and Mcleod, A. I. (1994). *Time Series Modelling of Water Resources and Environmental Systems*. Elsevier, Amsterdam. http://www.stats.uwo.ca/faculty/aim/1994Book/.

[19] Loan, C. V. (1992). *Computational frameworks for the fast Fourier transform*. SIAM, Philadelphia, PA, USA. MR1153025

[20] O'Leary, D. and Simmons, J. (1981). A bidiagonalization-regularization procedure for large scale discretizations of ill-posed problems. *SIAM J. Sci. Stat. Comput.* **2**, 4, 474–489. MR0639013

[21] Parker, D. E., Legg, T. P., and Folland, C. K. (1992). A new daily central England temperature series. *Int. J. Climatol 12*, 317–342.

[22] Parlett, B. (1980). *The Symmetric Eigenvalue Problem*. Prentice-Hall, New Jersey. MR0570116

[23] Parlett, B. and Reid, J. (1981). Tracking the progress of the Lanczos algorithm for large symmetric eigenproblems. *IMA J. Numer. Anal.* **1**, 2, 135–155. MR0616327

[24] R Development Core Team. (2009). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, http://www.R-project.org.

[25] Simon, H. (1984a). Analysis of the symmetric Lanczos algorithm with reothogonalization methods. *Linear Algebra Appl. 61*, 101–131. MR0755252

[26] Simon, H. (1984b). The Lanczos algorithm with partial reorthogonalization. *Math. Comp.* **42**, 165, 115–142. MR0725988

[27] Swarztrauber, P., Sweet, R., Briggs, W., Hensen, V. E., and Otto, J. (1991). Bluestein's FFT for arbitrary N on the hypercube. *Parallel Comput. 17*, 607–617. MR1128977

[28] Whaley, R. C. and Petitet, A. (2005). Minimizing development and maintenance costs in supporting persistently optimized BLAS. *Software: Practice and Experience* **35**, 2, 101–121.

[29] Wu, K. and Simon, H. (2000). Thick-restart Lanczos method for large symmetric eigenvalue problems. *SIAM J. Matrix Anal. Appl.* **22**, 2, 602–616. MR1781506

[30] Yamazaki, I., Bai, Z., Simon, H., Wang, L.-W., and Wu, K. (2008). Adaptive projection subspace dimension for the thick-restart Lanczos method. Tech. rep., Lawrence Berkeley National Laboratory, University of California, One Cyclotron road, Berkeley, California 94720.

Anton Korobeynikov
Botanicheskaya 70/3-306
198504, Saint Petersburg
Russia
E-mail address: asl@math.spbu.ru