Computer Physics Communications 230 (2018) 135–144

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

MXPFIT: A library for finding optimal multi-exponential approximations[☆]

Hidekazu Ikeno [*]

NanoSquare Research Institute, Research Center for the 21st Century, Organization for Research Promotion, Osaka Prefecture University, 1-2 Gakuen-cho, Naka-ku, Sakai, Osaka 599-8570, Japan
Precursory Research for Embryonic Science and Technology (PRESTO), Japan Science and Technology Agency (JST), 4-1-8 Hon-cho, Kawaguchi, Saitama, 332-0012, Japan

ARTICLE INFO

Article history:
Received 19 April 2017
Received in revised form 6 March 2018
Accepted 9 April 2018
Available online 25 April 2018

Keywords:
Exponential sum approximation
ESPRIT method
Balanced truncation
Con-eigenvalue problem

ABSTRACT

`mxpfit` is a library implemented in C++ to find optimal approximations of functions by multi-exponential sums with complex-valued parameters. The library provides an interface for evaluating exponents and coefficients from sampling data on a uniform grid using the fast Estimation of Signal Parameters via Rotational Invariance Techniques (ESPRIT) algorithm originally proposed by Potts and Tasche (2015). The parameters can be estimated efficiently from a sampling data even including noise. A modified balanced truncation algorithm to find the multi-exponential sum with a smaller order is also provided. These features are useful for finding optimal exponential sum approximations of analytic functions or large-scale numerically sampled data set.

Program summary
Program Title: mxpfit
Program Files doi: http://dx.doi.org/10.17632/vjygkwpss4.1
Licensing provisions: MIT
Programming language: C++
Nature of problem: Nonlinear curve fitting by multi-exponential function, frequency estimation of signals, and reduction of multi-exponential sum
Solution method: The modified fast Estimation of Signal Parameters via Rotational Invariance Techniques (ESPRIT) algorithm originally proposed in [1] is utilized for the parameter estimation of multi-exponential sums from sampled data on a uniform grid. In this method, partial Lanczos bidiagonalization is used to find a low-rank approximation of a Hankel matrix constructed from sampled data. An overdetermined Vandermonde system is solved by the conjugate gradient least square (CGLS) method with preconditioner to obtain the coefficients of the multi-exponential sum. The truncation of the multi-exponential sum is performed by a modified balanced truncation method in [2]. The projection matrix to obtain the truncated system is determined by computing con-eigenvalues of a Gramian matrix, which can be computed efficiently with highly relative accuracy using the algorithm in [3].
Additional comments: A C++ compiler supporting the C++11 standard is required to compile programs using the mxpfit library. mxpfit depends on the Eigen (http://eigen.tuxfamily.org/) and FFTW3 (http://www.fftw.org/) libraries. CMake (https://cmake.org/) is used for building example programs.
References:
[1] D. Potts and M. Tasche, Appl. Numer. Math. 88 (2015) 31.
[2] K. Xu and S. Jiang, J. Sci. Comput. 55 (2013) 16.
[3] T. S. Haut and G. Beylkin, SIAM J. Matrix Anal. Appl. 33 (2013) 1101.

1. Introduction

We consider functions expressed as a sum of exponential functions given as

$$f(t) = \sum_{i=1}^{M} c_i e^{-a_i t} \quad (t \geq 0) \tag{1}$$

[☆] This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

[*] Correspondence to: NanoSquare Research Institute, Research Center for the 21st Century, Organization for Research Promotion, Osaka Prefecture University, 1-2 Gakuen-cho, Naka-ku, Sakai, Osaka 599-8570, Japan
E-mail address: ikeno@mtr.osakafu-u.ac.jp.

https://doi.org/10.1016/j.cpc.2018.04.015
0010-4655/© 2018 Elsevier B.V. All rights reserved.

where $a_j \in \mathbb{C}$, $\mathrm{Re}(a_j) > 0$ are distinct complex numbers, and $c_j \in \mathbb{C} \setminus \{0\}$. The function $f(t)$ is a smooth, exponentially decaying function, and it oscillates if $\mathrm{Im}(a_j) \neq 0$. Recovering the parameters in Eq. (1) from noisy sampled data is a common problem in the field of signal processing. Exponential data fitting was already applied in electrical engineering and in many mathematical physics applications, e.g., fitting data in magnetic resonance imaging (MRI) and modeling spectroscopic data [1].

Prony's method is a well-known algorithm for finding the best M-term approximation of $f(t)$ from uniformly sampled signals. However, this method may be unstable, especially when the case signal is embedded in noise. Various modifications of Prony's method to improve its numerical stability have been proposed [2,3]. Among them, the Estimation of Signal Parameters via Rotational Invariance Techniques (ESPRIT) method and its variants are commonly used for their robustness against noise [4,5]. In the original ESPRIT method, we need a singular value decomposition (SVD) of the Hankel matrix constructed from sampled data. Fast decomposition algorithms relying on the special structure of the Hankel matrix are available, and were used in the fast ESPRIT method [6].

Approximating analytic functions by exponential sums has also been studied in applied mathematics. Beylkin and Monzón proposed a Prony-like method for finding the best approximation of analytic functions on finite intervals from function values on equally spaced mesh points [7]. This method is based on the theory of Adamjan, Arov, and Kreǐn (AAK theory). Andersson et al. made further mathematical considerations for this method [8]. They also proposed a similar algorithm (the so-called fast alternating projection method) for parameter estimation [9].

Beylkin and Monzón introduced another approach to obtain exponential sum approximations of analytic functions by discretizing the integral representations using a numerical quadrature rule [10]. The order of the exponential sum obtained by this procedure is usually suboptimal. The modified Prony's method for obtaining the optimal number of terms was also discussed in this literature. Alternatively, balanced truncation, which is an algorithm used for model order reduction (MOR), can also be used to find optimal approximations by exponential sums [11,12]. These techniques are suitable for approximating functions with singularities, such as power functions $r^{-\beta}$ ($\beta > 0$). This approximation was applied to construct Green's functions of Laplace equations appearing in quantum chemistry and fluid dynamics [13–15]. For example, Genovese et al. developed a fast Laplace equation solver that was used to evaluate integrals for electron–electron interaction energies [16]. The exponential sum approximation is also useful for efficient evaluation of integral transformation with an oscillatory kernel [17].

The `mxpfit` library was developed for numerically computing the exponential sum approximation of functions efficiently and with controllable accuracy. This library provides application programming interfaces (APIs) for (1) recovering parameters in Eq. (1) from sampled data on a uniform grid, and (2) reducing the order (number of terms) M of a given exponential sum. The modified fast ESPRIT algorithm was adopted for (1). A fast least squares solver for a rectangular Vandermonde matrix, which is required for computing coefficients $\{c_i\}$, was also developed. The balanced truncation method specialized for an exponential sum was designed and implemented for (2). The con-eigensolver of a quasi-Cauchy matrix developed by Haut and Beylkin was adopted to execute the truncation efficiently and with high relative accuracy [18].

Recently, the present author has applied this library to evaluate the spherical Bessel transform (SBT), which is defied as,

$$F(k) = \int_0^\infty r^2 f(r) j_l(kr) dr, \tag{2}$$

where the $j_l(r)$ is the spherical Bessel function. This can be evaluated efficiently and accurately if exponential sum approximation of residual function $r^2 f(r)$ and oscillating kernel $j_l(r)$ are obtained in high-accuracy. The details about the procedure can be found elsewhere [19].

2. Fast ESPRIT method

In this section, a fast ESPRIT algorithm to estimate the parameters a_i, c_i and number of terms M of exponential sums in Eq. (1) is briefly introduced. A detailed description of the fast ESPRIT algorithm can be found in the literature [6]. For the mathematical background about the ESPRIT and other Prony-like methods, we refer to [2,3,7].

Let $\boldsymbol{f} = (f_0, f_1, \ldots, f_{N-1})^T$ be the values of $f(t)$ sampled on a uniform grid $t_k = t_0 + hk$, $(h > 0,\ k = 0, 1, \ldots, N-1)$ with a sufficiently large number of $N \geq 2M$. Each element f_k can be written as

$$f_k = \sum_{j=1}^{M} c_j e^{-a_j(t_0+hk)} = \sum_{j=1}^{M} w_j z_j^k, \tag{3}$$

where $z_j = e^{-a_j h}$ and $w_j = c_j e^{-a_j t_0}$. The problem of obtaining the best exponential sum approximation of the form (1) becomes the goal of finding complex weights $\boldsymbol{w} = (w_0, \ldots, w_M)^T$ and complex nodes $\boldsymbol{z} = (z_1, \ldots, z_M)^T$ with a minimal number of terms M. Then, we introduce a rectangular Hankel matrix

$$H = \begin{pmatrix} f_0 & f_1 & \cdots & f_{K-1} \\ f_1 & f_2 & \cdots & f_K \\ \vdots & \vdots & \ddots & \vdots \\ f_{L-1} & f_L & \cdots & f_{L+K-2} \end{pmatrix} \in \mathbb{C}^{L\times K}, \tag{4}$$

where $M \leq L \leq N - M + 1$ and $K = N - L + 1$. Note that $L \geq K$. In the data processing application, the sequence $\boldsymbol{f}$ is called a time series of length N, the matrix H is called a L-trajectory matrix with a window length of L. The Hankel matrix H is usually singular when N is sufficiently large. Prony-like methods use the fact that the rank of H coincides with the number of terms M in Eq. (3). The ESPRIT method uses the singular value decomposition (SVD) of the Hankel matrix

$$H = U\Sigma W^*, \tag{5}$$

where $U \in \mathbb{C}^{L\times L}$ and $W \in \mathbb{C}^{K\times K}$ are unitary matrices, and Σ is the diagonal matrix holding the singular values as diagonal elements in ascending order $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_M > \sigma_{M+1} = \cdots = \sigma_K = 0$. In practice, the rank M is determined so that $\sigma_{M+1} < \epsilon$, where $\epsilon > 0$ is the required accuracy for the approximation. The nodes z_j are computed as the eigenvalues of the matrix

$$F_{\mathrm{SVD}} = W_M^{\dagger}(0) W_M(1) \in \mathbb{C}^{M\times M}, \tag{6}$$

where the matrices $W_M(0)$ and $W_M(1)$ are defined as $W_M(0) = W[1:K-1, 1:M]$ and $W_M(1) = W[2:K, 1:M]$, respectively, and the dagger (†) indicates the Moore–Penrose pseudoinverse of the matrix.

One of the drawbacks in the original ESPRIT method is the high computational cost of order $O(N^3)$ for computing the full SVD of the Hankel matrix. This makes it difficult to apply the method to a large number of sample data. In the fast ESPRIT algorithm, a partial Lanczos bidiagonalization is utilized to obtain a low-rank approximation of the Hankel matrix (4).

The Golub–Kahan–Lanczos bidiagonalization decomposes the general matrix $A \in \mathbb{C}^{L\times K}$ in the form

$$A = P_K B_K Q_K^*, \tag{7}$$

where $P_n = [\boldsymbol{p}_1, \dots, \boldsymbol{p}_n] \in \mathbb{C}^{L\times n}$ and $Q_n = [\boldsymbol{q}_1, \dots, \boldsymbol{q}_n] \in \mathbb{C}^{K\times n}$ are matrices with orthonormal columns, and B_n is the bidiagonal real matrix

$$B_n = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ & \alpha_2 & \beta_2 & & \\ & & \ddots & \ddots & \\ & & & \alpha_{K-1} & \beta_{K-1} \\ & & & & \alpha_K \end{pmatrix} \in \mathbb{R}^{K\times K}. \tag{8}$$

The column vectors $\boldsymbol{p}_j$ and $\boldsymbol{q}_j$ are called the left and right Lanczos vectors, respectively. The bidiagonal elements α_i, β_i and the Lanczos vectors are obtained using the recurrence relation

$$\begin{aligned} \alpha_1 \boldsymbol{p}_1 &= A\boldsymbol{q}_1, \\ \alpha_{j+1}\boldsymbol{p}_{j+1} &= A\boldsymbol{q}_{j+1} - \beta_j \boldsymbol{p}_j, \\ \beta_j \boldsymbol{q}_{j+1} &= A^*\boldsymbol{p}_j - \alpha_j \boldsymbol{q}_j, \end{aligned} \tag{9}$$

starting from an arbitrary selected unit vector $\boldsymbol{q}_1$. When computing the recurrence (9) in finite precision arithmetic, the orthogonality between the Lanczos vectors might be lost. In order to recover the orthogonality, several reorthogonalization schemes have been proposed. See Ref. [20] for more details about the Lanczos method and the reorthogonalization scheme. In the present implementation of `mxpfit`, the full-reorthogonalization scheme was adopted following [6] for numerical robustness.

Let us consider the matrix $\tilde{A}_M = P_M B_M Q_M^*$, where P_M, Q_M and B_M are obtained by applying the bidiagonalization step M times ($M \le K$). As discussed in [6], P_M and Q_M contain good approximations of the singular vectors related to the first M dominant singular values. Hence, $\tilde{A}_M$ could be regarded as a low-rank approximation of matrix A. The approximation error of $\tilde{A}_M$ with respect to A is defined in the Frobenius norm as $\Delta_M = \|A - \tilde{A}_M\|_F$. The approximation error can be determined recursively as

$$\begin{aligned} \Delta_{M+1}^2 &= \Delta_M^2 - \alpha_{M+1}^2 - \beta_M^2, \ (2 \le M \le K-1) \\ \Delta_1^2 &= \|A\|_F - \alpha_1^2, \end{aligned} \tag{10}$$

and $\Delta_K = 0$. In addition,

$$\|\tilde{A}_M\|_F^2 = \|B_M\|_F^2 = \alpha_1^2 + \sum_{j=1}^{M-1} (\alpha_{j+1}^2 + \beta_j^2) \tag{11}$$

converges monotonically to $\|A\|_F^2$. Thus, we can utilize the magnitude of $\alpha_{M+1}^2 + \beta_M^2$ as an estimator of the approximation error. In practice, we monitor the value in each bidiagonalization step and stop the recurrence if the condition $\alpha_{M+1}^2 + \beta_M^2 < \epsilon^2 \|\tilde{A}_M\|_F^2$ is met.

The low-rank approximation of the Hankel matrix $H \in \mathbb{C}^{L\times K}$ is computed using the partial Lanczos bidiagonalization described above. In this process, two Hankel matrix–vector multiplication operations are required in each bidiagonalization step. The operation can be performed efficiently using a fast Fourier transform (FFT) [6,21]. The algorithm can be written as follows:

Algorithm 1 (*General Fast Hankel Matrix–Vector Product*). Input: $L, K \in \mathbb{N}$ and a sequence $\boldsymbol{f} = (f_0, \dots, f_{L+K-1})^T$ to form a $L \times K$ Hankel matrix H, $P \in \mathbb{N}$ the length of the FFT with $P \ge L + K - 1$, and a vector $\boldsymbol{x}$ of length K ($\boldsymbol{x}'$ of length L). Output: $\boldsymbol{y} := H\boldsymbol{x}$ of length L ($\boldsymbol{y}' := H^*\boldsymbol{x}'$ of length K).

1. Form the vector $\tilde{\boldsymbol{f}}$ of length P
$$\tilde{\boldsymbol{f}} = (f_{K-1}, f_K, \dots, f_{L+K-2}, \underbrace{0, \dots, 0}_{P-L-K+1}, f_0, f_1, \dots, f_{K-2}).$$
2. Form the vector $\tilde{\boldsymbol{x}}$ of length P
$$\tilde{\boldsymbol{x}} = (x_{K-1}, x_{K-2}, \dots, x_0, \underbrace{0, \dots, 0}_{P-K}) \quad \text{for multiplying } H$$
$$\tilde{\boldsymbol{x}} = (\underbrace{0, \dots, 0}_{P-L}, x_{L-1}'^*, x_{L-2}'^*, \dots, x_0'^*) \quad \text{for multiplying } H^*.$$
3. Compute the vector $\tilde{\boldsymbol{y}}$ of length P as
$$\tilde{\boldsymbol{y}} := \mathrm{IFFT}(\mathrm{FFT}(\tilde{\boldsymbol{f}}) * \mathrm{FFT}(\tilde{\boldsymbol{x}}))$$
where '$*$' denotes an element-wise multiplication of two vectors.
4. Form the vector $\boldsymbol{y}$ ($\boldsymbol{y}'$) as
$$\boldsymbol{y} = \tilde{\boldsymbol{y}}[0 : L-1] \quad \text{for multiplying } H$$
$$\boldsymbol{y}' = \tilde{\boldsymbol{y}}[P-K : P-1] \quad \text{for multiplying } H^*.$$

Here, the FFT and IFFT in Algorithm 1 denote a one-dimensional FFT and inverse FFT, respectively. In the present case, steps 2–4 are repeated $2M$ times during the M bidiagonalization step with $\boldsymbol{f}$ fixed. Therefore, once the vector $\tilde{\boldsymbol{f}}$ in step 1 of Algorithm 1 is constructed, $\mathrm{FFT}(\tilde{\boldsymbol{f}})$ is computed and stored in a temporary vector so that we can reuse it in step 3. The computational cost of the Hankel matrix–vector operation is $O(P \log P)$ per single product, while that for the general matrix–vector product is $O(LK)$. Note that the optimal FFT size is in the range $L + K - 1 \le P < 2(L + K - 1)$. The computational complexity for the M-step Lanczos bidiagonalization for the Hankel matrix becomes $O(MP \log P)$, which is much smaller than that of the full SVD.

Once a low-rank approximation of the Hankel matrix $H = P_M B_M Q_M^*$ is obtained, the nodes z_j are computed as the eigenvalues of the matrix

$$F_{\mathrm{LBD}} = Q_M^\dagger(0) Q_M(1) \in \mathbb{C}^{M\times M}, \tag{12}$$

where $Q_M(0) = Q_M[1 : K-1, 1 : M]$ and $Q_M(1) = Q_M[2 : K, 1 : M]$, respectively. The computation of the Moore–Penrose pseudoinverse $Q_M^\dagger$ is expensive because it requires computing the SVD of Q_M. In the original paper of the fast ESPRIT method, this step is bypassed by solving a generalized eigenvalue problem yielding the same eigenvalues. Badeau et al., used a different method of computing F_{LBD} without explicitly computing the pseudoinverse as

$$F_{\mathrm{LBD}} = \Psi + \frac{1}{1 - \|\boldsymbol{\nu}\|_2^2} \boldsymbol{\nu} (\Psi^* \boldsymbol{\nu})^* \tag{13}$$

with $\Psi = Q_M^*(0) Q_M(1)$ and $\boldsymbol{\nu} = Q_M[K, 1 : M]$ [22]. This method relies on the orthonormality between column vectors $\boldsymbol{q}_j$ of matrix Q_M. The `mxpfit` library adopted the latter approach.

After obtaining nodes z_j, weights w_j in (3) are computed as the least squares solution of the linear system:

$$V(\boldsymbol{z})\boldsymbol{w} = \boldsymbol{f}, \tag{14}$$

where $V(\boldsymbol{z})$ is a rectangular Vandermonde matrix defined by nodes $\boldsymbol{z}$ as

$$V(\boldsymbol{z}) = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ z_1 & z_2 & \cdots & z_M \\ z_1^2 & z_2^2 & \cdots & z_M^2 \\ \vdots & \vdots & \ddots & \vdots \\ z_1^{N-1} & z_2^{N-1} & \cdots & z_M^{N-1} \end{pmatrix} \in \mathbb{C}^{N\times M}. \tag{15}$$

There is little discussion in literature about the computational complexity for solving this overdetermined Vandermonde system (14) related to parameter estimation. The least squares problem for the complex general matrix can be solved using subroutine `ZGELSD` implemented in the linear algebra package (`LAPACK`). This subroutine internally uses the SVD of the matrix. The computational cost for the SVD of $N\times M$ matrix $V(\boldsymbol{z})$ is $O(N^2M+NM^2+M^3)$. Although the number of terms M is usually small, the computational time for solving (14) becomes significant if the number of sample data N is large.

In the present work, the conjugate gradient least square (CGLS) method was adopted to solve the overdetermined Vandermonde system (14). The convergence of the CGLS method can be accelerated if an appropriate preconditioner, which approximately solves a linear system

$$V^*(\boldsymbol{z})V(\boldsymbol{z})\boldsymbol{x}=V^*(\boldsymbol{z})\boldsymbol{b}, \tag{16}$$

is provided. In the present work, we designed a preconditioner using the fast decomposition algorithm for a Vandermonde matrix. In practice, the Cholesky decomposition

$$V^*(\boldsymbol{z})V(\boldsymbol{z})=X\Sigma^2X^* \tag{17}$$

is computed, where $X\in\mathbb{C}^{M\times M}$ is a unit lower triangular matrix, and $\Sigma\in\mathbb{R}^{M\times M}$ is a diagonal matrix with nonnegative entries. The preconditioning linear system can be solved as

$$\boldsymbol{x}=X^{-*}\Sigma^{-2}X^{-1}(V^*(\boldsymbol{z})\boldsymbol{b}), \tag{18}$$

with $O(M^2)$ computational cost. The computational complexity of the Cholesky decomposition (17) is only $7M^2/2+O(n)$, using the algorithm proposed by Demeure (Algorithm C in Ref. [23]). The least squares solution of Eq. (14) can usually be obtained with a small number of iterations in CGLS (fewer than 10 in most cases). Each CG step requires two matrix–vector products, which cost $O(NM)$ for each product. Thus, the total complexity of the new Vandermonde least squares solver is $O(NM+M^2)$, which is significantly smaller than that for a conventional least squares solver using the SVD. Note that this CGLS algorithm might not converge if ϵ is too small. In such a case, the present code automatically falls back to the standard least square algorithm using the SVD to achieve the correct weights.

Once nodes z_j and w_j are computed, the parameters a_j and c_j for the exponential sum can be obtained as $a_j=-(\log z_j)/h$ and $c_j=w_je^{-a_jt_0}$. Our modified fast ESPRIT algorithm is summarized in Algorithm 2.

Algorithm 2 (*Modified Fast ESPRIT Method*)**.** Input: Sequence of function values f_k sampled on an equispaced grid $t_k=t_0+kh$ for $k=0,1,\dots,N-1$, the window length $L\le N$, M_b the upper bound of number of terms M, and prescribed accuracy $\epsilon>0$.
Output: parameters a_j with $\mathrm{Re}(a_j)>0$ and c_j $(j=1,\dots,M)$ for the exponential sum $f(t)$ such that $\|f(t_k)-f_k\|<\|\boldsymbol{f}\|_2\epsilon$

1. Compute M-step partial Lanczos bidiagonalization of the Hankel matrix H in (4) such that $\|H-P_MB_MQ_M^*\|<\epsilon$. All Hankel matrix–vector products are computed using Algorithm 1.
2. Form matrix F_{LDB} using (13), and compute its eigenvalues z_j $(j=1,\dots,M)$. Set $a_j=-(\log z_j)/h$ $(j=1,\dots,M)$.
3. Compute w_j $(j=1,\dots,M)$ as a least squares solution of the overdetermined Vandermonde system (14) using a CGLS solver with a preconditioner (18) (automatically fall back to the least squares method with SVD if not converged). Set $c_j=w_je^{-a_jt_0}$ $(j=1,\dots,M)$.

3. Balanced truncation algorithm for exponential sum

For a prescribed accuracy $\epsilon>0$, we consider the approximation of $f(t)$ with another exponential sum with a smaller number of terms, such that

$$\hat{f}(t)=\sum_{j=1}^{M'}\hat{c}_je^{-\hat{a}_jt},\quad \|f(t)-\hat{f}(t)\|<\epsilon, \tag{19}$$

where $M'\le M$. Let us consider the Laplace transform of $f(t)$, which can be evaluated analytically as

$$F(s)=\mathcal{L}[f(t)]=\int_0^\infty e^{-st}f(t)dt=\sum_{i=1}^{M}\frac{c_i}{s+a_i}. \tag{20}$$

Thus, the exponents a_i and coefficients c_i are the poles and residues of the Laplace transform $F(s)$. The problem of finding an optimal exponential sum (19) becomes a problem of finding a shorter sum-of-poles approximation of (20), such that

$$\hat{F}(s)=\sum_{i=1}^{M'}\frac{\hat{c}_i}{s+\hat{a}_i},\quad \|F(s)-\hat{F}(s)\|<\epsilon \tag{21}$$

for all $s>0$.

Let $A=\mathrm{diag}(a_1,a_2,\dots,a_M)$, $B=\left(\sqrt{c_1},\sqrt{c_2},\dots,\sqrt{c_M}\right)^T$, and $C=B^T$. Then, $f(t)$ and $F(s)$ can be expressed as

$$f(t)=Ce^{-tA}B, \tag{22}$$

and

$$F(s)=C(sI+A)^{-1}B, \tag{23}$$

respectively. Using the terminology of *control theory*, $F(s)$ is a transfer function of a linear time-invariant (LTI) system, which is described by a differential equation given as

$$\begin{aligned}\frac{d\boldsymbol{x}(t)}{dt}&=-A\boldsymbol{x}(t)+Bu(t),\ \boldsymbol{x}(0)=\boldsymbol{0}\\ y(t)&=C\boldsymbol{x}(t),\end{aligned} \tag{24}$$

where $u(t)$ is the control (input), $y(t)$ is the output, and $\boldsymbol{x}(t)=(x_1(t),\dots,x_M(t))$ are the so-called state variables. By applying the Laplace transform to Eq. (24), we obtain

$$\begin{aligned}s\boldsymbol{X}(s)&=-A\boldsymbol{X}(s)+BU(s),\\ Y(s)&=C\boldsymbol{X}(s).\end{aligned} \tag{25}$$

The transfer function $F(s)$ defines the linear mapping between the Laplace transform of the input $U(s)=\mathcal{L}[u(t)]$ and that of the output $Y(s)=\mathcal{L}[y(t)]$ as $Y(s)=F(s)U(s)$. Approximating such a dynamic system with a smaller number of state variables is a common problem of the MOR in control theory [24].

Among many techniques of MOR, balanced truncation is a common technique to approximate an LTI system using projection [25]. This method relies on the fact that applying a linear transformation to a state vector $T\tilde{\boldsymbol{x}}=\boldsymbol{x}$ does not change the input–output behavior of the system. The controllability Gramian W_C and observability Gramian W_O associated with the LTI system realized by (A,B,C) are defined as

$$W_C=\int_0^\infty e^{At}BB^*e^{A^*t}dt, \tag{26}$$

and

$$W_O=\int_0^\infty e^{A^*t}C^*Ce^{At}dt, \tag{27}$$

respectively. We look for the linear transformation on the state vector space that "balances" the system. In other words, we look for a projection matrix T such that

$$\begin{aligned}W_C'&=T^{-1}W_CT^{-*}=\Sigma,\\ W_O'&=T^*W_OT=\Sigma\end{aligned} \tag{28}$$

where $\Sigma=\mathrm{diag}(\sigma_1,\dots,\sigma_M)$ with $\sigma_1\ge\sigma_2\ge\cdots\ge\sigma_M\ge0$. The $\{\sigma_i\}$ are Hankel singular values of the dynamic system.

For a general matrix A and $C\ne B^T$, the transformation matrix T is computed as follows. First, Gramian matrices are computed by solving the Lyapunov equations $AW_C+W_CA^*=BB^*$ and

$A^*W_O + W_O A = C^*C$. Since the Gramian matrices W_C and W_O are positive definite, the Cholesky decomposition of those matrices can be computed as $W_O = L_O L_O^*$ and $W_O = L_O L_O^*$ where L_C and L_O are $M \times M$ lower triangular matrices. The Hankel singular values of the system are derived as singular values of $L_C^* L_O$, which are equal to the square root of the eigenvalues of $W_O W_C$ and $L_C^* W_O L_C$. Consider the eigenvalue decomposition

$$L_C^* W_O L_C = U^* \Sigma^2 U. \tag{29}$$

Then, the transformation matrix is constructed as

$$T = L_C^* U^* \Sigma^{-\frac{1}{2}}, \quad T^{-1} = \Sigma^{\frac{1}{2}} U L_C. \tag{30}$$

In the present case of a system with $F(s)$ in (20) as the transfer function, matrix elements of W_C and W_O can be analytically obtained as

$$\begin{aligned}[W_C]_{ij} &= \int_0^\infty e^{-a_i t}\sqrt{c_i c_j^*} e^{-a_j^* t} dt = \frac{\sqrt{c_i c_j^*}}{a_i + a_j^*}, \\ [W_O]_{ij} &= \int_0^\infty e^{-a_i t}\sqrt{c_i^* c_j} e^{-a_j t} dt = \frac{\sqrt{c_i^* c_j}}{a_i^* + a_j} \\ &= [W_C]_{ij}^* = [W_C]_{ji}.\end{aligned} \tag{31}$$

Thus, W_C is a self-adjoint quasi-Cauchy matrix, and $W_O = \overline{W}_C = W_C^T$, where the overline indicates the element-wise complex conjugate of the matrix. The W_C can be decomposed as

$$W_C = \overline{U} \Sigma U^T, \tag{32}$$

where $U_M \in \mathbb{C}^{M \times M}$ with $U^T U = I$, and $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_M)$ with $\sigma_i \geq \sigma_2 \geq \cdots \geq \sigma_M > 0$. A decomposition of the form (32) is referred to as a con-eigenvalue decomposition. Each column of U is called a con-eigenvector, and σ_i is called a con-eigenvalue. The con-eigenvalues $\{\sigma_i\}\,(i = 1, \ldots, M)$ coincide with the Hankel singular values of the LTI system with $F(s)$ as the transfer function, which can be verified as

$$W_O W_C = W_C^T W_C = U \Sigma U^* \overline{U} \Sigma U^T = U \Sigma^2 U^{-1}. \tag{33}$$

A fast and accurate con-eigenvalue decomposition algorithm for a quasi-Cauchy matrix was developed by Haut and Beylkin [18]. In this algorithm, a rank-revealing Cholesky decomposition, $W_C = (PL)D^2(PL)^*$, is computed using the highly accurate Gaussian elimination with a complete pivoting (GECP) algorithm for a quasi-Cauchy matrix in [26]. The con-eigenvalue decomposition is computed using the Demmel's SVD algorithm for a matrix with rank-revealing decomposition (RRD) [26,27]. This method allows us to compute con-eigenvalues and con-eigenvectors with high relative accuracy.

In the mxpfit library, a modified balanced truncation algorithm combined with the fast con-eigenvalue solver described above was implemented, as shown in Algorithm 3. The truncation is made in step 2, where the order of truncated system, M', is determined by the error bound computed from the Hankel singular values. Then, the first linear transformation is performed in step 3. The second transformation is made in steps 4 and 5 so that the transfer function of the reduced system has the form (21).

Algorithm 3 (*Modified Balanced Truncation Method*). Input: Sequences $\{a_i\}\,(\mathrm{Re}(a_i) > 0)$ and $\{c_i\}$ for $i = 1, \ldots, M$ to define transfer function $F(s)$ in (20), and a prescribed accuracy $\epsilon > 0$. Output: Sequences $\{\hat{a}_i\}$ and $\{\hat{c}_i\}$ for $i = 1, \ldots, M'\,(M' \leq M)$ defining the transfer function of reduced system $\hat{F}(s)$ in (21).

1. Compute the con-eigenvalue decomposition of a quasi-Cauchy matrix $W_C = \overline{U} \Sigma U^T$ using the fast and accurate algorithm in [18].
2. Find M' such that $2\sum_{i=M'+1}^{M} \sigma_i \leq \epsilon$.
3. Compute an $M' \times M'$ matrix $A' = U_{M'}^* A \overline{U}_{M'}$ and a vector of length M', $B' = U_{M'}^* B$, where $U_{M'} = U[1:M, 1:M']$.
4. Compute the eigenvalue decomposition of $A' = X \Lambda X^{-1}$, and set $\hat{a}_i = \Lambda_{ii}$ for $i = 1, \ldots, M'$. Since matrix A' is symmetric, eigenvectors can be chosen such that $X^{-1} = X^T$.
5. Compute $B'' = X^{-1}B' = X^T B'$ and set $\hat{c}_i = (B_i'')^2$ for $i = 1, \ldots, M'$.

4. Description of the program

The mxpfit library is a pure template and header-only library written in C++. The library depends on the Eigen linear algebra library (version 3.3 or later) [28], and FFTW3 library for fast Fourier transforms [29,30]. The reason for using the Eigen library as the backend is that it provides a generic application programming interface (API) for matrix vector arithmetic and matrix decompositions using C++ templates. This feature enables us to conduct a simulation using various scalar types including single/double/quadruple precision floating-point numbers and custom scalar types such as multiprecision numbers implemented in the GNU Multi-Precision Library (GMP).

One can use this library by including just header files. To compile the source code using mxpfit, a modern C++ compiler supporting the C++11 standard is necessary. The FFTW3 library with the appropriate scalar type has to be linked using the fast ESPRIT algorithm. For building an example and test codes, CMake and related build tools such as GNU make are also required.

In this library, an exponential sum function of the form (1) is represented by ExponentialSum class in which arrays to store parameters a_j and c_j are dynamically allocated. Scalar types for an exponent a_j and weight c_j should be provided as template arguments (the scalar type for a weight is same as that of an exponent by default). Alternatively, one can view two separated arrays as an exponential function via ExponentialSumWrapper class. A utility function to make an instance of this class easily is provided. An instance of ExponentialSumWrapper cannot be resized or modified. Users are responsible to manage the memory allocation and to set valid values of input arrays. Both the ExponentialSum and ExponentialSumWrapper classes have common interface to access those parameters and that to evaluate function value at a given coordinate (t) which is inherited from ExponentialSum-Base class. See the right-half part of Fig. 2 for relationship between these classes. These classes can be used for instance,

```
// Create a five-term exponential sum with real exponents
// and real weights.
mxpfit::ExponentialSum<double> expsum(5);
assert(expsum.size() == 5);
// You can change the number of terms dynamically.
expsum.resize(2);
assert(expsum.size() == 2);
// Access to parameters
expsum.exponent(0) = 1.0;  expsum.weight(0) = 0.5;
expsum.exponent(1) = 2.0;  expsum.weight(1) = 1.5;
// Evaluate function value
auto f = expsum(2.5);

// Create a view of a complex exponential sum from  pre-existing
// arrays of exponents and weights
using Complex = std::complex<double>;
Eigen::ArrayXcd a(2), c(2);
a << Complex(1.0, 0.0), Complex(2.0, 0.0); // exponents
c << Complex(0.5, 0.0), Complex(1.5, 0.0); // weights
// This will return an instance of
// mxpfit::ExponentialSumWrapper.
auto expsum_wrapper = mxpfit::makeExponentialSum(a, c);
assert(expsm_wrapper.size() == 2);
```

```
// Evaluate function value
auto f = expsum(2.5);

// You cannot modify ExponentialSumWrapper directly.
// Modify the arrays (a,c) and re-create an instance.
// expsum_wrapper.resize(3);         /* not allowed */
// expsum_wrapper.exponent(0) = 2.0; /* not allowed */
```

The library provides two main APIs: `FastESPRIT` class for finding the best parameters in (1) from sampled data on a uniform grid via the modified fast ESPRIT algorithm, and `BalancedTruncation` class for finding optimal parameters from a given exponential sum function via the modified balanced truncation method. Both classes return an instance of `ExponentialSum` class. The `FastESPRIT` class internally uses the partial Lanczos bidiagonalization algorithm and the CGLS method with Vandermonde preconditioner as has been described in 2, which have been implemented in separate classes. Precisely speaking, the `FastESPRIT` class depends on the classes described below. The relationships between those classes are visualized in Fig. 1 using a unified modeling language (UML) class diagram.

- `PartialLanczosBidiagonalization`: computes a low-rank approximation of a given matrix via a partial Lanczos bidiagonalization algorithm.
- `HankelMatrix`: represents a rectangular Hankel matrix (4), with an interface for the Hankel matrix–vector product described in Algorithm 2. This class internally uses the FFTW3 library. `MatrixFreeGEMV<HankelMatrix>` is provided as an argument of PartialLanczosBidiagonalization.
- `VandermondeLeastSquareSolver` solves an overdetermined Vandermonde system (14) using the CGLS method. This is a realization of the CGLS solver provided by Eigen, where `MatrixFreeGEMV<VandermondeMatrix>` and `VandermondePreconditioner` are used as the matrix type and preconditioner type, respectively.
- `VandermondePreconditioner`: solves a preconditioning linear system for a Vandermonde matrix (18).
- `VandermondeMatrix`: represents a rectangular Vandermonde matrix (15), with an interface for the Vandermonde matrix–vector product.
- `MatrixFreeGEMV`: provides interface overloading `operator*` for the product of the `HankelMatrix/VandermondeMatrix` and any vector type in Eigen.

A typical usage of the `FastESPRIT` class is as follows:

```
using Complex = std::complex<double>
using ESPRIT  = mxpfit::FastESPRIT<double>;
// 1. Preparation:
//   N: number of grid points (N >= 1)
//   L: window length (1 <= L <= N)
//   M_max: upper bound of M, the number of terms
//          (1 <= M_max <= min(N, N - L + 1))
ESPRIT esprit(N, L, M_max);
// 2. Fitting:
//   f:  sample data on uniform grid
//       (Eigen's vector of length N)
//   t0: first grid point
//   h:  interval between two grid points (h > 0)
//   epsilon: prescribed accuracy (0 < epsilon << 1)
// * Result type is mxpfit::ExponentialSum<Complex>
auto expsum = esprit.compute(f, t0, h, epsilon);
// You can re-use ‘esprit’ for fitting other data with
// same length
auto expsum2 = esprit.compute(f2, t0, h, epsilon);
```

As the first step, one needs to create an object of the `FastESPRIT` class with a number of grid points N, window length L, and upper bound of M for exponential sum (1). In this step, memory space for internal arrays is allocated, and optimal "plans" for an FFT and inverse FFT of length N are generated. Then, parameters of the exponential sum approximation are computed by calling the `compute` method of this object with sample data $\boldsymbol{f}$, parameters t_0 and h which define a uniform grid $t_k = t_0 + hk\,(k = 0, \dots, N-1)$, and a prescribed accuracy ϵ as input arguments. Any vector expressions available in the `Eigen` library can be used as an argument type of $\boldsymbol{f}$.

In the `BalancedTruncation` class, a rank-revealing decomposition $W_C = (PL)D^2(PL)^*$ of a self-adjoint quasi-Cauchy matrix (31) is computed, then coneigenvalue decomposition (32) is performed. These algorithms are also implemented in separate classes described as follows. The relationships between those classes are shown in Fig. 2 as a UML class diagram.

- `SelfAdjointQuasiCauchyRRD`: computes a rank-revealing Cholesky decomposition of a self-adjoint quasi-Cauchy matrix with high relative accuracy.
- `SelfAdjointConeigensolver`: computes the coneigenvalue decomposition of a self-adjoint matrix having an RRD, $A = XD^2X^*$, with high relative accuracy.

The `BalancedTruncation` class can be used as follows:

```
using Complex = std::complex<double>;
mxpfit::ExponentialSum<Complex> f_orig(N);
// ... set 'f_orig' here ...
mxpfit::BalancedTruncation<Complex> truncation;
// This returns a 'ExponentialSum' object with same scalar type
//   epsilon: prescribed accuracy (0 < epsilon << 1)
auto f_truncated = truncation.compute(f_orig, epsilon);
```

The `BalancedTruncation` class can be applied to an exponential sum with real parameters as long as $a_j > 0$ and $c_j > 0$ are satisfied. Note that in case any $c_j < 0$, the truncation fails because a domain error occurs in the built-in square-root function (`std::sqrt`).

5. Numerical examples

In this section, numerical examples for the fast ESPRIT method and balanced truncation method are provided. The calculations were made on a laptop computer with an Intel Core i7 processor @ 3.3 GHz. Programs were built using `clang++` compiler with `'-std=c++11 -stdlib=libc++ -O3 -march=native -DNDEBUG'` flags.

5.1. Example 1 (Fast ESPRIT applied to analytical function)

We consider a nonlinear approximation of the function on a finite interval,

$$g(t) = \mathrm{sinc}(t) = \frac{\sin(t)}{t} \quad (t \in [0, t_{\max}]) \tag{34}$$

by an exponential sum. The fast ESPRIT algorithm was applied with varying numbers of sample data N from 2^8 to 2^{18} for sampled values $g(t_k)$ with $t_k = hk$, $h = 1/16\,(k = 0, \dots, N-1)$. Other arguments were chosen as $L = N/2$, $M_{\max} = \min(L, 500)$, and $\epsilon = 10^{-12}$. The exponential sum of the form (1) was obtained with complex exponents a_j and coefficients c_j.

In the case of $N = 2^{12} = 4096$, we obtained the exponential sum $\tilde{g}(t) = \sum_{j=1}^{M} c_j e^{-a_j t}$ for a sinc function of order $M = 30$ on the interval $t \in [0, 256]$. The function $f(t)$ and absolute error $|g(t) - \tilde{g}(t)|$ on this interval are shown in Fig. 3. The absolute error is kept smaller than the prescribed accuracy 10^{-12}.

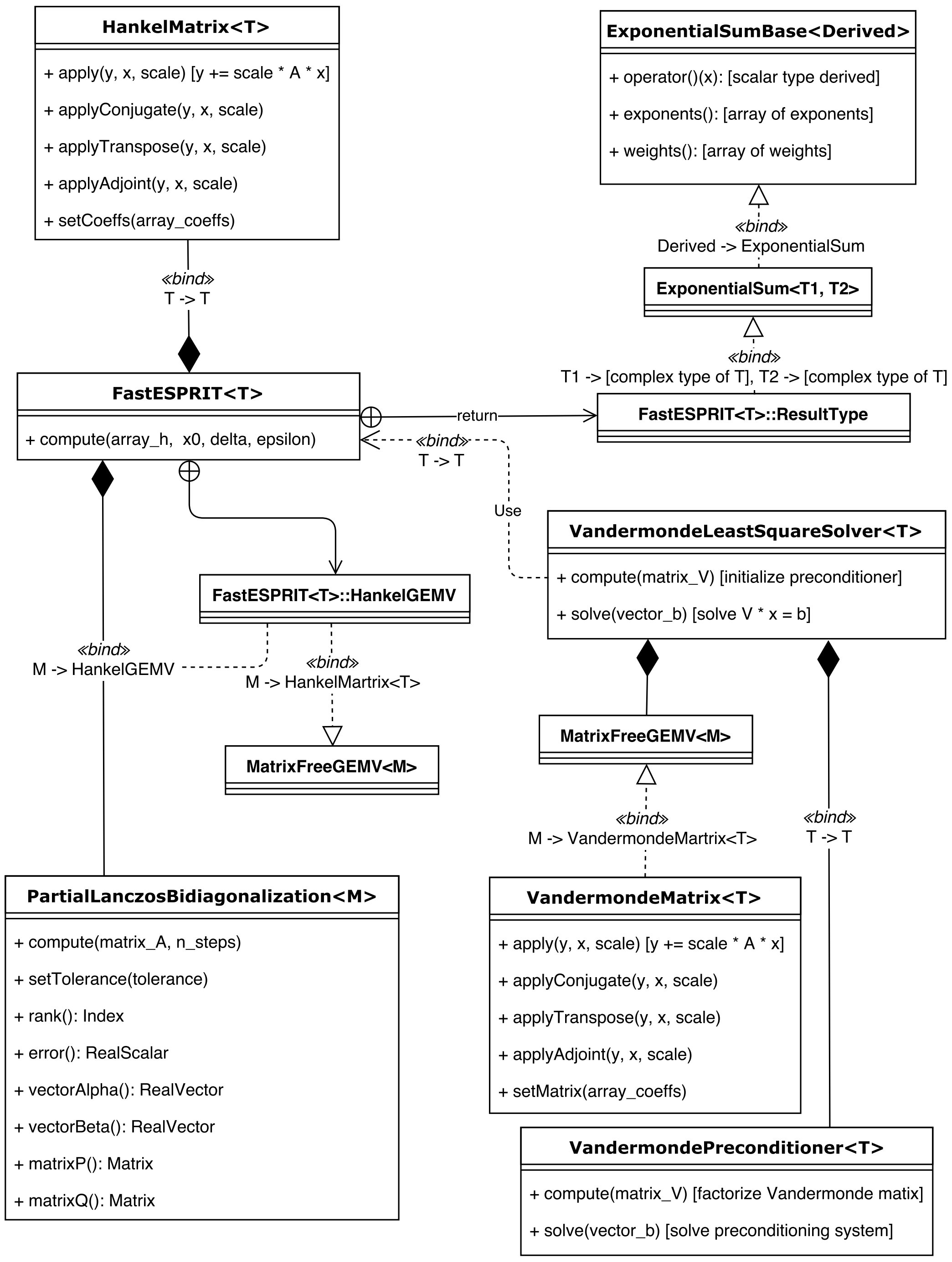
HankelMatrix<T>
+ apply(y, x, scale) [y += scale * A * x]
+ applyConjugate(y, x, scale)
+ applyTranspose(y, x, scale)
+ applyAdjoint(y, x, scale)
+ setCoeffs(array_coeffs)
«bind» T -> T
ExponentialSumBase<Derived>
+ operator()(x): [scalar type derived]
+ exponents(): [array of exponents]
+ weights(): [array of weights]
«bind» Derived -> ExponentialSum
ExponentialSum<T1, T2>
«bind» T1 -> [complex type of T], T2 -> [complex type of T]
FastESPRIT<T>::ResultType
FastESPRIT<T>
+ compute(array_h, x0, delta, epsilon)
return
«bind» T -> T
Use
VandermondeLeastSquareSolver<T>
+ compute(matrix_V) [initialize preconditioner]
+ solve(vector_b) [solve V * x = b]
FastESPRIT<T>::HankelGEMV
«bind» M -> HankelGEMV
«bind» M -> HankelMartrix<T>
MatrixFreeGEMV<M>
MatrixFreeGEMV<M>
«bind» M -> VandermondeMartrix<T>
«bind» T -> T
PartialLanczosBidiagonalization<M>
+ compute(matrix_A, n_steps)
+ setTolerance(tolerance)
+ rank(): Index
+ error(): RealScalar
+ vectorAlpha(): RealVector
+ vectorBeta(): RealVector
+ matrixP(): Matrix
+ matrixQ(): Matrix
VandermondeMatrix<T>
+ apply(y, x, scale) [y += scale * A * x]
+ applyConjugate(y, x, scale)
+ applyTranspose(y, x, scale)
+ applyAdjoint(y, x, scale)
+ setMatrix(array_coeffs)
VandermondePreconditioner<T>
+ compute(matrix_V) [factorize Vandermonde matix]
+ solve(vector_b) [solve preconditioning system]

Fig. 1. Fast ESPRIT class diagram.

The order of exponential sum, approximation errors and the running time of the fast ESPRIT algorithm for a sinc function with different N are summarized in Table 1. The maximum absolute error is defined as $\max|g(t_k) - \tilde{g}(t_k)|$. The running time for preparing a `FastESPRIT` object and that for the fitting data are listed separately. As can be seen, most of the CPU time is spent on the preparation step. In fact, most of the CPU time is spent on creating plans for the `FFTW3` library. Once a `FastESPRIT` object is prepared, a nonlinear fitting can be obtained efficiently. Even in the case of $N = 2^{18} = 262\,144$, it only takes about 2 s to obtain the fitting data. The present implementation of the fast ESPRIT algorithm is feasible for fitting many numbers of data sets with the same length N.

5.2. Example 2 (Fast ESPRIT applied to numerically sampled data)

Let us consider the exponential sum of order $M = 5$,

$$h(t) = \sum_{j=1}^{5} c_j e^{-if_j t} \tag{35}$$

with

$$\boldsymbol{f} = (0, \pi/4, -\pi/4, \pi/2, -\pi/2)^T, \tag{36}$$

$$\boldsymbol{c} = (34, 300, 300, 1, 1)^T, \tag{37}$$

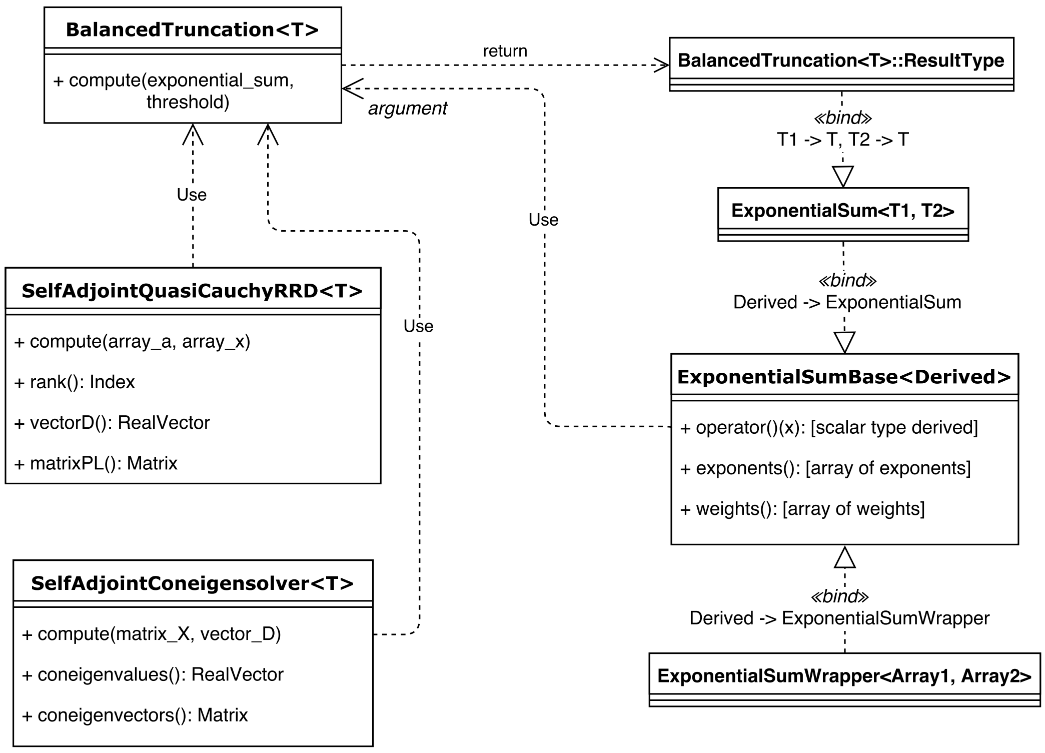
BalancedTruncation<T>
+ compute(exponential_sum, threshold)
return
argument
BalancedTruncation<T>::ResultType
«bind» T1 -> T, T2 -> T
ExponentialSum<T1, T2>
«bind» Derived -> ExponentialSum
ExponentialSumBase<Derived>
+ operator()(x): [scalar type derived]
+ exponents(): [array of exponents]
+ weights(): [array of weights]
«bind» Derived -> ExponentialSumWrapper
ExponentialSumWrapper<Array1, Array2>
Use
Use
Use
SelfAdjointQuasiCauchyRRD<T>
+ compute(array_a, array_x)
+ rank(): Index
+ vectorD(): RealVector
+ matrixPL(): Matrix
SelfAdjointConeigensolver<T>
+ compute(matrix_X, vector_D)
+ coneigenvalues(): RealVector
+ coneigenvectors(): Matrix

Fig. 2. Balanced truncation class diagram.

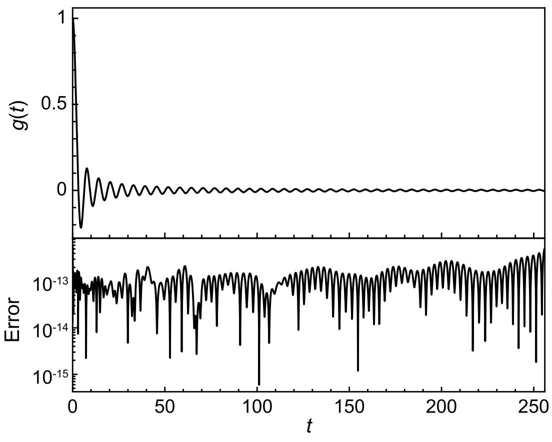

Fig. 3. Function $g(t) = \operatorname{sinc}(t)$ (upper) and absolute error between $g(t)$ and exponential sum approximation $\tilde{g}(t)$ of order 30 on the interval $[0, 256]$.

Table 1
Errors and running time of fast ESPRIT algorithm for $g(t_k) = \operatorname{sinc}(t_k)$ with $t_k = k/16$, $(k = 0, \dots, N-1)$. Running time during preparation step and fitting step are listed separately (in milliseconds).

N	M	Max abs. error	Time (prep.)	Time (fit)
2^8	12	1.1×10^{-11}	38	0.3
2^9	16	2.4×10^{-12}	50	0.4
2^{10}	22	2.1×10^{-13}	79	1
2^{11}	26	3.0×10^{-13}	131	3
2^{12}	30	3.5×10^{-13}	248	6
2^{13}	34	4.8×10^{-13}	498	15
2^{14}	38	2.1×10^{-13}	966	43
2^{15}	42	1.7×10^{-13}	1,841	163
2^{16}	46	3.4×10^{-13}	3,747	421
2^{17}	49	4.2×10^{-13}	7,923	792
2^{18}	53	2.9×10^{-13}	17,742	1,958

and construct sampling data on uniform grid as

$$h_k = h(k) + e_k = 34 + 600 \cos\left(\frac{\pi t}{4}\right) + 2\cos\left(\frac{\pi t}{2}\right) + e_k \quad (k = 0, 1, \dots, N-1), \tag{38}$$

where e_k are random noise uniformly distributed in $[-3, 3]$. The same sequence has been investigated in Example 6.1 in [6]. It should be noted that the noise is higher than the amplitude of coefficient related to the frequencies $\pm\pi/2$. The frequencies $\tilde{\boldsymbol{f}}$ and coefficients $\tilde{\boldsymbol{c}}$ were estimated from the sequence $\{h_k\}$ by applying modified fast ESPRIT algorithm with the number of sampling points, N, varying from 2^8 to 2^{18}. The window size was chosen $L = N/2$ and target accuracy was set to $\epsilon = 10^{-5}$. The maximum error of frequencies, $\max\left(|f_1 - \tilde{f}_1|, \dots, |f_5 - \tilde{f}_5|\right)$, as well as the maximum errors of coefficients, $\max\left(|c_1 - \tilde{c}_1|, \dots, |c_5 - \tilde{c}_5|\right)$, where the maximum are formed over 10 trials. The same measurements were also performed by original (sub-optimal) ESPRIT algorithm with N up to 2^{13}. The results are summarized in Table 2. The maximum errors observed in the results of fast ESPRIT algorithm and original one are almost same. The maximum error of frequencies occurs at $f_4 = \pi/2$ and $f_5 = -\pi/2$. The errors of other frequencies are smaller about two order of magnitude. Both the errors of frequencies and coefficients decrease with increase of N.

The CPU time of fast ESPRIT algorithm and original ESPRIT algorithm averaged over 10 trials are also shown in Table 2, in which the running time for preparation step of fast ESPRIT algorithm is excluded. We estimated computational complexities of these two algorithms by performing curve fittings. The CPU time of original ESPRIT algorithm is found to be proportional to N^b with $b = 2.93$, as has been expected from the complexity of the dense SVD. The CPU time of fast ESPRIT algorithm is much smaller than that of original ESPRIT algorithm. The complexity of fast ESPRIT algorithm is proportional to $N \log N$. The result suggests that most of CPU time is spent for the Hankel matrix–vector multiplications in the partial Lanczos bidiagonalization step.

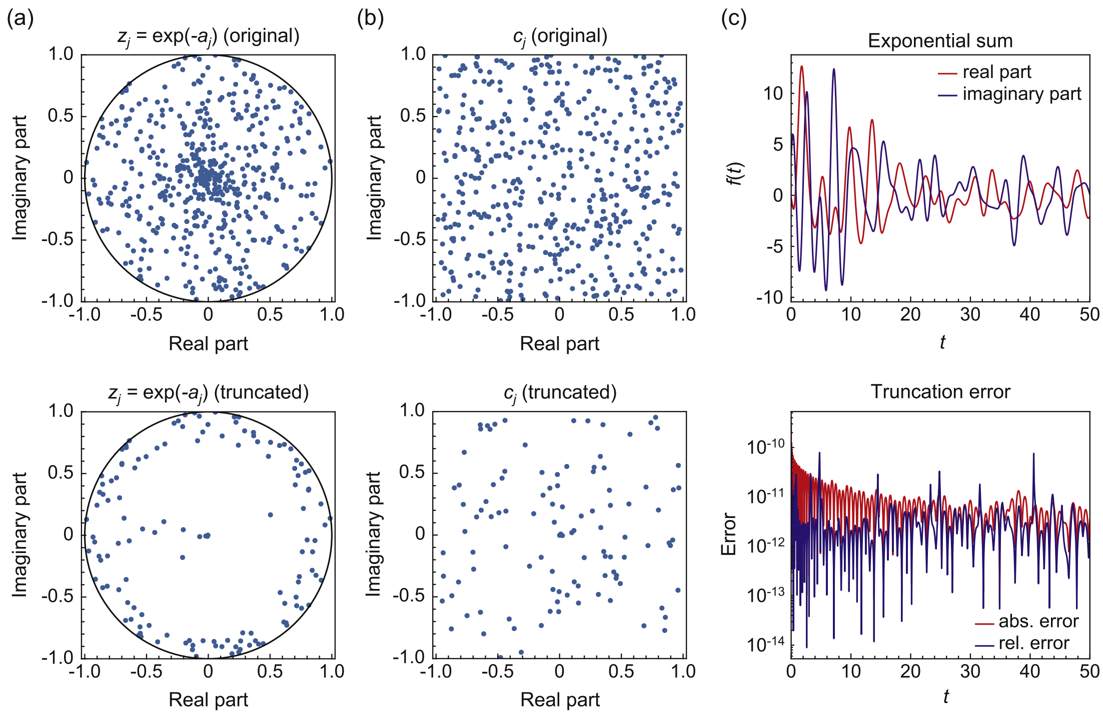

Fig. 4. Results of the modified balanced truncation applied to a randomly generated exponential sum of order 500. (a) Distribution of $z_j = e^{-a_j}$ for original and truncated exponential sum. (b) Distribution of coefficients c_j for original and truncated exponential sum. (c) The function $f(t)$, the absolute truncation error $|f(t) - \hat{f}(t)|$, and relative truncation error $|f(t) - \hat{f}(t)|/|f(t)|$ are on the interval $t \in [0, 50]$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 2
Errors and running time of fast ESPRIT algorithm for Example 5.2 with different number of sampling points, N, which are compared with the values obtained by the original ESPRIT algorithm using full SVD. The window size is chosen to be $L = N/2$, and accuracy is set to $\epsilon = 10^{-5}$.

N	Max error of frequencies		Max error of coefficients		Time (ms)	
	Fast	Original	Fast	Original	Fast	Original
2^8	2.81×10^{-3}	2.95×10^{-3}	4.40×10^{-1}	4.21×10^{-1}	0.1	5.8
2^9	9.10×10^{-4}	9.22×10^{-4}	3.61×10^{-1}	3.52×10^{-1}	0.3	18.8
2^{10}	2.75×10^{-4}	3.08×10^{-4}	1.93×10^{-1}	2.31×10^{-1}	0.4	87.6
2^{11}	1.07×10^{-4}	1.08×10^{-4}	1.76×10^{-1}	1.75×10^{-1}	0.7	561.3
2^{12}	5.72×10^{-5}	4.03×10^{-5}	1.34×10^{-1}	1.35×10^{-1}	1.3	4748.9
2^{13}	1.78×10^{-5}	1.76×10^{-5}	7.64×10^{-2}	7.63×10^{-2}	2.5	36003.7
2^{14}	4.98×10^{-6}	–	4.42×10^{-2}	–	5.9	–
2^{15}	1.85×10^{-6}	–	4.99×10^{-2}	–	15.8	–
2^{16}	7.12×10^{-7}	–	2.33×10^{-2}	–	32.1	–
2^{17}	1.99×10^{-7}	–	1.96×10^{-2}	–	67.7	–
2^{18}	6.44×10^{-8}	–	1.60×10^{-2}	–	144.3	–

5.3. *Example 3 (modified balanced truncation)*

The modified balanced truncation algorithm was applied to a randomly generated exponential sum in (1) of order $M = 500$. The complex exponents were chosen such that $z_j = e^{-a_j} = r_j e^{i\delta_j}$ with uniformly distributed $r_j \in [0, 1]$, and $\delta_j \in [0, 2\pi]$. All z_j were inside the unit disk on a complex plane. The complex coefficients c_j were uniformly distributed on $[-1, 1] + i[-1, 1]$. The truncation was made with the prescribed accuracy $\epsilon = 10^{-12}$.

Fig. 4 summarizes the results of a typical run. A truncated exponential sum $\hat{f}(t)$ of order $M' = 133$ was obtained. Parameters $z_j = e^{-a_j}$ and c_j before and after the truncation are shown in Fig. 4(a) and (b). The function $f(t)$ is shown in the upper panel of Fig. 4(c). The real and imaginary parts of $f(t)$ are indicated by red and blue lines, respectively. The absolute truncation error $|f(t) - \hat{f}(t)|$ and relative truncation error $(|f(t) - \hat{f}(t)|)/|f(t)|$ are given at the bottom panel of Fig. 4(c). The relative error is kept below 10^{-11} at most points, which is slightly higher than the prescribed accuracy ϵ. The CPU time spent on the truncation was 2.1 s.

Acknowledgments

This work was supported by PRESTO, Grant No. JPMJPR16N1 16815006 from Japan Science and Technology Agency (JST), and the Grant-in-Aid for Scientific Research on Innovative Areas, Grant No. 26106518 from the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of Japan.

References

[1] V. Pereyra, G. Scherer (Eds.), Exponential Data Fitting and its Applications, Bentham Science, 2010. http://dx.doi.org/10.2174/97816080504821100101.
[2] T. Peter, D. Potts, M. Tasche, SIAM J. Sci. Comput. 33 (2011) 1920–1947. http://dx.doi.org/10.1137/100790094.
[3] D. Potts, M. Tasche, Linear Algebra Appl. 439 (2013) 1024–1039. http://dx.doi.org/10.1016/j.laa.2012.10.036.
[4] R. Roy, T. Kailath, IEEE Trans. Acoust. Speech Signal Process. 37 (7) (1989) 984–995. http://dx.doi.org/10.1109/29.32276.
[5] D. Potts, M. Tasche, in: D.A. Bini, T. Ehrhardt, A.Y. Karlovich, I. Spitkovsky (Eds.), Large Truncated Toeplitz Matrices, Toeplitz Operators, and Related

Topics, in: Operator Theory: Advances and Applications, vol. 259, Springer International Publishing, 2017, pp. 621–648. http://dx.doi.org/10.1007/978-3-319-49182-0_25.
[6] D. Potts, M. Tasche, Appl. Numer. Math. 88 (2015) 31–45. http://dx.doi.org/10.1016/j.apnum.2014.10.003.
[7] G. Beylkin, L. Monzón, Appl. Comput. Harmon. Anal. 19 (2005) 17–48. http://dx.doi.org/10.1016/j.acha.2005.01.003.
[8] F. Andersson, M. Carlsson, M.V. de Hoop, J. Approx. Theory 163 (2011) 213–248. http://dx.doi.org/10.1016/j.jat.2010.09.005.
[9] F. Andersson, M. Carlsson, Proc. Geo-Math. Imaging Group 1 (2011) 21–36.
[10] G. Beylkin, L. Monzón, Appl. Comput. Harmon. Anal. 28 (2010) 131–149. http://dx.doi.org/10.1016/j.acha.2009.08.011.
[11] K. Xu, S. Jiang, J. Sci. Comput. 55 (2013) 16–39. http://dx.doi.org/10.1007/s10915-012-9620-9.
[12] W. McLean, Exponential sum approximations and fast evaluation of fractional integrals, 2017. URL http://arxiv.org/abs/1606.00123.
[13] T. Yanai, G.I. Fann, Z. Gan, R.J. Harrison, G. Beylkin, J. Chem. Phys. 121 (2004) 2866–2876. http://dx.doi.org/10.1063/1.1768161.
[14] R.J. Harrison, G.I. Fann, T. Yanai, Z. Gan, G. Beylkin, J. Chem. Phys. 121 (2004) 11587–11598. http://dx.doi.org/10.1063/1.1791051.
[15] G. Beylkin, T.S. Haut, Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. 469 (2013) 20130231. http://dx.doi.org/10.1098/rspa.2013.0231.
[16] L. Genovese, T. Deutsch, A. Neelov, S. Goedecker, G. Beylkin, J. Chem. Phys. 125 (7) (2006) 074105. http://dx.doi.org/10.1063/1.2335442.
[17] L. Monzn, G. Beylkin, Discrete Contin. Dyn. Syst. 36 (8) (2016) 4077–4100. http://dx.doi.org/10.3934/dcds.2016.36.4077.
[18] T.S. Haut, G. Beylkin, SIAM J. Matrix Anal. Appl. 33 (2012) 1101–1125. http://dx.doi.org/10.1137/110821901.
[19] H. Ikeno, J. Comput. Phys. 355 (2018) 426–435. http://dx.doi.org/10.1016/j.jcp.2017.11.016.
[20] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, H. van der Vorst (Eds.), Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide, SIAM, Philadelphia, 2000. http://dx.doi.org/10.1137/1.9780898719581.
[21] K. Browne, S. Qiao, Y. Wei, Linear Algebra Appl. 430 (2009) 1531–1543. http://dx.doi.org/10.1016/j.laa.2008.01.012.
[22] R. Badeau, G. Richard, B. David, Statistical Signal Processing, 2005 IEEE/SP 13th Workshop on, 2005, pp. 289–294. http://dx.doi.org/10.1109/SSP.2005.1628608.
[23] C.J. Demeure, Linear Algebra Appl. 122–124 (1989) 165–194.
[24] W.H. Schilders, H.A.v.d. Vorst, J. Rommes (Eds.), Model Order Reduction: Theory, Research Aspects and Applications, Springer Verlag, Berlin Heiderberg, 2008.
[25] S. Gugercin, A.C. Antoulas, Internat. J. Control 77 (8) (2004) 748–766. http://dx.doi.org/10.1080/00207170410001713448.
[26] J. Demmel, SIAM J. Matrix Anal. Appl. 21 (2) (1999) 562–580.
[27] J. Demmel, M. Gu, S. Eisenstat, I. Slapničar, K. Veseli, Z. Drmač, Linear Algebra Appl. 299 (1999) 21–80.
[28] G. Guennebaud, B. Jacob, et al., Eigen v3, 2010. URL http://eigen.tuxfamily.org/.
[29] M. Frigo, S.G. Johnson, S. Kral, R. Dolbea, E. Lindahl, FFTW, 2003. URL http://www.fftw.org/.
[30] M. Frigo, S.G. Johnson, Proc. IEEE 93 (2) (2005) 216–231.