Sum-of-Exponentials Modeling and Common Dynamics Estimation Using Tensorlab

Ivan Markovsky[1], Otto Debals[2,3], and Lieven De Lathauwer[2,3]

[1] Department ELEC
Vrije Universiteit Brussel (VUB)
Pleinlaan 2, Building K, B-1050
Brussels, Belgium
Ivan.Markovsky@vub.ac.be

[2] ESAT/STADIUS and iMinds Medical IT
K.U.Leuven
Kasteelpark Arenberg 10
3001 Leuven, Belgium
Otto.Debals@esat.kuleuven.be

[3] K.U.Leuven Campus Kortrijk
Group Science, Engineering and Technology
E. Sabbelaan 53
8500 Kortrijk Belgium
Lieven.DeLathauwer@kuleuven-kulak.be

Abstract. Fitting a signal to a sum-of-exponentials model is a basic problem in signal processing. It can be posed and solved as a Hankel structured low-rank matrix approximation problem. Subsequently, local optimization, subspace, and convex relaxation methods can be used for the numerical solution. In this paper, we show another approach, based on the recently proposed concept of structured data fusion. Structured data fusion problems are solved in the Tensorlab toolbox by local optimization methods. The approach allows fitting of signals with missing samples and adding constraints on the model, such as fixed exponents and common dynamics in multi-channel estimation problems. These problems are non-trivial to solve by other existing methods.

Keywords: system identification; low-rank approximation; mosaic Hankel matrix; tensorlab; structured data fusion.

1 Introduction: The Sum-of-Exponentials Modeling Problem

To introduce the sum-of-exponentials modeling problem, let us first consider the exponential function $\exp_z(t) := z^t$. The sequence

$$\big(\ldots, \exp_z(-1), \exp_z(0), \exp_z(1), \ldots\big)$$

is referred to as a discrete-time exponential signal. Given a set of $\mathtt{n}$ complex numbers $z = \big(z_1, \ldots, z_\mathtt{n}\big)$, the $\mathtt{n}$th order sum-of-exponentials model $\mathscr{B}$ is defined as the set

$$\mathscr{B} = \mathscr{B}(z) = \Big\{ \sum_{j=1}^{\mathtt{n}} c_j \exp_{z_j} \mid c \in \mathbb{C}^\mathtt{n} \Big\}. \tag{1}$$

The model $\mathscr{B}$ is real if all signals $y \in \mathscr{B}$ are real valued. In what follows, we consider discrete-time real models. The sum-of-exponentials model (1) is an autonomous linear time-invariant dynamical system [10]:

$$\mathscr{B} = \mathscr{B}(A,C) = \{ y \mid \sigma x = Ax,\ y = Cx \},$$

with σ the shift operator $(\sigma x)(t) := x(t+1)$ and with

$$A = \operatorname{diag}(z_1, \ldots, z_\mathtt{n}), \quad C = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}, \quad \text{and} \quad x(0) = \begin{bmatrix} c_1 & \cdots & c_\mathtt{n} \end{bmatrix}^\top. \tag{2}$$

Alternatively, $y \in \mathscr{B}(z)$ is the impulse response of the linear time-invariant system

$$\mathscr{B} = \mathscr{B}(A,B,C,D) = \{ y \mid \sigma x = Ax + Bu,\ y = Cx + Du \},$$

where A and C are as in (2), and with $B = \begin{bmatrix} c_1 & \cdots & c_\mathtt{n} \end{bmatrix}^\top$ and $D = y(0)$.

Let us now consider trajectories of T samples with $y = \big(y(1), \ldots, y(T)\big)$. We restrict the model $\mathscr{B}$ on the interval $[1, T]$ and denote it by $\mathscr{B}|_T$. We have that

$$\mathscr{B}(z)|_T = \operatorname{image}\big(V_T(z)\big)$$

, where $V_T(z)$ is the (transposed) Vandermonde matrix

$$V_T(z) = \begin{bmatrix} z_1^1 & \cdots & z_\mathtt{n}^1 \\ \vdots & & \vdots \\ z_1^T & \cdots & z_\mathtt{n}^T \end{bmatrix},$$

i.e., for any $y \in \mathscr{B}(z)|_T$, there exists a $c \in \mathbb{C}^\mathtt{n}$ such that

$$y = V_T(z)c. \tag{3}$$

To denote the distance between a signal y and a system $\mathscr{B}$, the following distance measure is used:

$$\operatorname{dist}(y, \mathscr{B}) := \min_{\widehat{y} \in \mathscr{B}} \|y - \widehat{y}\|_2, \tag{4}$$

in which the norm $\|\cdot\|_2$ is the Frobenius norm or 2-norm, defined as the square root of the sum-of-squares of the elements of the vector, matrix or tensor.

The sum-of-exponentials modeling problem is now defined as follows. Given a time series y and order $\mathtt{n}$, find a model $\mathscr{B}(z)$ of order at most $\mathtt{n}$, that is as close as possible to the data y in the 2-norm sense, *i.e.*,

$$\text{minimize} \quad \operatorname{dist}\big(y, \mathscr{B}(z)\big) \quad \text{over } z \in \mathbb{C}^\mathtt{n}. \tag{5}$$

This problem can be viewed as an as autonomous linear time-invariant system identification problem or else as the identification of a linear time-invariant system from impulse response data (a realization problem) [9]. More specifically, problem (5) defines a maximum likelihood estimator in the output error setting, assuming additive zero mean, stationary, white Gaussian noise [8, 12].

In Section 3, problem (5) is generalized to multi-channel sum-of-exponential modeling (vector time series), multiple time series, modeling with missing data, fixed poles, and common poles across different channels. The solution approach to all these generalizations is the structured data fusion framework presented in the next section. For the advanced parts of the algorithms and implementation from Section 2, we refer the reader to Appendix A and B.

2 Sum-of-Exponentials Modeling Using Tensorlab

2.1 Introduction of Tensorlab

Tensorlab is a MATLAB toolbox offering algorithms for tensor decompositions, complex least-squares optimization, global minimization and more [16]. Note that tensors are higher-order generalizations of vectors and matrices, denoted in this paper by calligraphic letters. Structured data fusion is recently proposed as a framework within Tensorlab for rapid prototyping of (coupled) decompositions of (whether dense, incomplete or sparse) tensors with constraints on the factor matrices [13, 15]. Structured data fusion is implemented in Tensorlab by using a domain specific language, enabling the user to choose from various factor structures and from three different tensor decompositions: the canonical polyadic decomposition, the block term decomposition [1] or the low-multilinear rank approximation.

In this paper, tensor decompositions are being used for solving the vector and matrix problems in equation (3). Indeed, the data sets need not necessarily be higher-order. Instead of finding (structured) factor matrices $U^{(1)}$, $U^{(2)}$ and $U^{(3)}$ to decompose a tensor $\mathscr{T}$, we search factor matrices $U^{(1)}$ and $U^{(2)}$ to factorize a matrix T into $U^{(1)}U^{(2)\top}$. Likewise, we can search for a matrix U and vector c (both generally known as 'factors') to write a known vector t as $t = Uc$. Of course, multiple solutions exist for the latter two factorizations. Uniqueness only appears through additional constraints on the factors, unlike for tensor decompositions realizing uniqueness under mild conditions [2–4, 6]. These constraints can be enforced using structured data fusion of which the usefulness becomes apparent from the generalizations proposed in Section 3.

2.2 Structured Data Fusion

Constraints/structures on the factor matrices are typically (non)linear dependencies on some underlying variables z. These underlying variables can be scalars, vectors or matrices. They are transformed to factors through smooth mappings, i.e., the partial derivatives with respect to the elements of the underlying variables and their complex conjugates should exist and be continuous up to all orders (continuous Wirtinger derivatives).

Next, the factors are assigned to each dataset through a chosen decomposition. Tensorlab implements a following least-squares optimization:

$$\min_{z} f = \sum_{d=1}^{D} \omega_d f^{(d)} \quad \text{with } f^{(d)} = \frac{1}{2} \Big|\Big| \underbrace{\mathscr{M}^{(d)}(\mathscr{X}(z)) - \mathscr{T}^{(d)}}_{\mathscr{F}^{(d)}} \Big|\Big|^2_{\mathscr{W}^{(d)}}, \tag{6}$$

where $\|\mathscr{A}\|_{\mathscr{W}} := \|\mathscr{W} * \mathscr{A}\|$ with $*$ the Hadamard (element-wise) product. In (6), z is the set of underlying variables, $\mathscr{X}$ is the collection of the mappings (with the factors as the images), $\mathscr{T}^{(d)}$ is the dth dataset, and $\mathscr{M}^{(d)}$ is the model of the decomposition corresponding to each dataset $\mathscr{T}^{(d)}$. For example, regarding the canonical polyadic decomposition, we have the following model:

$$\mathscr{M}_{\text{CPD}}(U^{(1)}, \ldots, U^{(N)}) := \sum_{r=1}^{R} u_r^{(1)} \otimes \cdots \otimes u_r^{(N)}. \tag{7}$$

Without constraints, the matrices $U^{(i)}$ belong to $\mathscr{X}(z)$. Missing elements from an incomplete tensor are associated with a zero in the corresponding position in the weighting tensor $\mathscr{W}^{(d)}$, then known as the observation tensor. Details about the algorithms used and their implementation are given in Appendix A and B, respectively.

3 Generalizations of the Sum-of-Exponentials Modeling Problem

In Section 3.1 we generalize problem (5) to the multi-channel problem. Multiple trajectories are introduced in Section 3.2. Tensorlab can be used to incorporate missing data, as discussed in Section 3.3. Finally, Section 3.4 discusses the presence of common dynamics, which has a strong connection to blind signal separation [2].

3.1 Multi-channel modeling

In the multi-channel sum-of-exponential modeling problem, the given data y is a $\mathtt{p}$-dimensional vector time series. Each output measurement y_i is a trajectory of the same model $\mathscr{B}(z)$, so that, in general, the exponents $z_1, \ldots, z_{\mathtt{n}}$ are common to all channels. Eq. (3) gives us a structured matrix factorization which can be seen as a structured data fusion problem:

$$\begin{bmatrix} y_1 & \cdots & y_{\mathtt{p}} \end{bmatrix} = V_T(z) \underbrace{\begin{bmatrix} c_1 & \cdots & c_{\mathtt{p}} \end{bmatrix}}_{C}. \tag{8}$$

3.2 Modeling using multiple trajectories

If the data consists of N time series $y^i = \big(y^i(1), \ldots, y^i(T_i)\big)$ generated from the same model $\mathscr{B}(z)$, the modeling problem is to approximate all time series in a least-squares sense:

$$\text{minimize} \quad \sum_{i=1}^{N} \text{dist}^2(y^i, \mathscr{B}(z)) \quad \text{over } z \in \mathbb{C}^{\mathtt{n}}. \tag{9}$$

Note that modeling using multiple trajectories is different from the multi-channel modeling because the lengths $T_1, \ldots, T_N$ of the time series may be different. In addition, it is possible to consider multi-channel modeling with multiple trajectories. Problem (9) is solved in Tensorlab using a coupled factorization with common parameters z:

$$y^i = V_T(z)c^i, \qquad \text{for } i = 1, \ldots, N.$$

3.3 Missing data

Missing data values $y(t)$, for $t \in \mathscr{I}_m$ are handled by excluding them from the calculation of the distance measure, *i.e.*, (4) is calculated only with respect to the given data. In Tensorlab missing values are specified by setting the corresponding elements to `NaN`. The initial approximation is computed by 1) replacing the missing data with the average of the given data, and 2) applying Kung's method on the complete data sequence. Another approach for computing initial approximation is to use the nuclear norm heuristic [5].

3.4 Common dynamics modeling

The common dynamics estimation problem in multi-channel signal processing [11] is aiming to decompose the dynamics of the channels into "common dynamics", *i.e.*, joint exponents, and "individual dynamics", *i.e.*, exponents that appear in a specific channel only. A zero-pattern structure on the matrix C from (8) is imposed. To illustrate we have

$$C = \begin{bmatrix} C_{0,1} & \cdots & C_{0,\mathrm{p}} \\ C_{1,1} & & \\ & \ddots & \\ & & C_{\mathrm{p},\mathrm{p}} \end{bmatrix}, \tag{10}$$

with all missing values being zeros. Let $n_0 := \operatorname{row\,dim}(C_{0,1})$ and $n_i := \operatorname{row\,dim}(C_{i,1})$. The first n_0 exponents are common to all channels, the next n_1 exponents are used in the first channel only, and so on. Such a constraint can be easily imposed in Tensorlab by a Hadamard product of a fully parameterized matrix variable and a mask matrix, *i.e.*, a matrix with ones at the position of the nonzero elements in (10) and zeros elsewhere. The implementation is discussed in Appendix B.

Further discussions and conclusions

In this paper, we considered the basic sum-of-exponentials modeling problem and its generalizations to multi-channel modeling, multiple time series, missing data, modeling with known exponents, and common dynamics identification. All these problems were solved by the structured data fusion functionality of the Tensor toolbox. The solution approach using structured data fusion is based on local optimization methods and requires user specified initial approximation. Simulation results comparing the Tensor toolbox with other solution approaches using local optimization methods show equivalent results in terms of accuracy. The current implementation of the structured data fusion in the Tensor toolbox is simple to generalize to new problems but is slower than the alternative methods.

Acknowledgements

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC Grant agreement number 258581 "Structured low-rank approximation: Theory, algorithms, and applications" and ERC Advanced Grant agreement number 339804 "BIOTENSORS". It is also funded by a Ph.D. grant of the Agency for Innovation by Science and Technology (IWT), by the Research Council KU Leuven: CoE PFV/10/002 (OPTEC), by F.W.O.: projects G.0830.14N and G.0881.14N, and by the Belgian Federal Science Policy Office: IUAP P7/19 (DYSCO II, Dynamical systems, control and optimization, 2012–2017).

References

1. De Lathauwer, L.: Decompositions of a higher-order tensor in block terms — Part II: Definitions and uniqueness. SIAM Journal on Matrix Analysis and Appl. 30(3), 1033–1066 (2008)
2. De Lathauwer, L.: Blind separation of exponential polynomials and the decomposition of a tensor in rank-$(L_r, L_r, 1)$ terms. SIAM Journal on Matrix Analysis and Appl. 32(4), 1451–1474 (2011)
3. Domanov, I., De Lathauwer, L.: On the uniqueness of the canonical polyadic decomposition of third-order tensors — Part I: Basic results and uniqueness of one factor matrix. SIAM Journal on Matrix Analysis and Applications 34(3), 855–875 (2013)
4. Domanov, I., De Lathauwer, L.: On the uniqueness of the canonical polyadic decomposition of third-order tensors — Part II: Uniqueness of the overall decomposition. SIAM Journal on Matrix Analysis and Applications 34(3), 876–903 (2013)
5. Fazel, M.: Matrix rank minimization with applications. Ph.D. thesis, Elec. Eng. Dept., Stanford University (2002)
6. Kruskal, J.B.: Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. Linear Algebra and its Appl. 18(2), 95–138 (1977)
7. Kung, S.: A new identification method and model reduction algorithm via singular value decomposition. In: Proc. 12th Asilomar Conf. Circuits, Systems, Comp. pp. 705–714 (1978)
8. Markovsky, I.: Structured low-rank approximation and its applications. Automatica 44(4), 891–909 (2008)
9. Markovsky, I.: A software package for system identification in the behavioral setting. Control Engineering Practice 21, 1422–1436 (2013),
10. Markovsky, I., Willems, J.C., Van Huffel, S., De Moor, B.: Exact and Approximate Modeling of Linear Systems: A Behavioral Approach. No. 11 in Monographs on Mathematical Modeling and Computation, SIAM (March 2006),
11. Papy, J.M., Lathauwer, L.D., Huffel, S.V.: Common pole estimation in multi-channel exponential data modeling. Signal Processing 86(4), 846–858 (Apr 2006)
12. Pintelon, R., Schoukens, J.: System Identification: A Frequency Domain Approach. IEEE Press, Piscataway, NJ, second edn. (2012)
13. Sorber, L.: Data fusion: Tensor factorizations by complex optimization. Ph.D. thesis, Faculty of Engineering, KU Leuven, Leuven, Belgium (May 2014)
14. Sorber, L., Barel, M., Lathauwer, L.: Unconstrained optimization of real functions in complex variables. SIAM Journal on Optimization 22(3), 879–898 (2012)
15. Sorber, L., Van Barel, M., De Lathauwer, L.: Structured data fusion (2013), internal report 13–177, ESAT-SISTA, KULeuven (Leuven, Belgium)
16. Sorber, L., Van Barel, M., De Lathauwer, L.: Tensorlab v2.0 (January 2014), available online. URL: `http://www.tensorlab.net`

A Algorithms for structured data fusion

We only discuss the nonlinear least squares algorithm, which makes use of first-order derivative information but is able to achieve up to second-order convergence by approximating the second-order derivatives. With z we denote a vector collecting the underlying variables. By approximating the residual tensors $\mathscr{F}^{(d)}$ from (6) with a linear model

$$m_k^{\mathscr{F}^{(d)}}(p) := \mathscr{F}^{(d)}(z_k) + J_k^{(d)} \cdot \overset{c}{p}, \quad \text{where} \qquad J_k^{(d)} := \partial \mathscr{F}^{(d)} / \partial \overset{c}{z}^{\top},$$

one obtains a second-order model of the following form:

$$m_k^f(p) := f(z_k) + \overset{c}{p}^{\top} \cdot \frac{\partial f}{\partial \overset{c}{z}}(z_k) + \frac{1}{2} \overset{c}{p}^{*} \cdot B_k \cdot \overset{c}{p}. \tag{11}$$

Eq. (11) is a second-order complex Taylor series expansion of f from (6) around z_k, and it is minimized to obtain the steps p to the next iterate z_{k+1}. The critical work is to calculate the complex gradient $\partial f / \partial \overset{c}{z}$ and the matrix B_k which is a positive semidefinite approximation of the complex Hessian described by $\partial^2 f / (\partial \overset{c}{\overline{z}} \partial \overset{c}{z}^{\top})$. In the complex gradient, both the cogradient $\partial f / \partial z$ and conjugate cogradient $\partial f / \partial \overline{z}$ are stacked. They are conjugates of each other as f is real. Typically, one only uses the conjugate cogradient.

We further subdivide these components to separate the decomposition models $\mathscr{M}^{(d)}$ from the structure imposed by $\mathscr{X}$. The mappings $\mathscr{X}$ are assumed to be analytical functions, i.e., they are not dependent on the conjugate of z. For non-analytic mappings $\mathscr{X}$, we direct the reader to [14, 15]. The chain rule on the conjugated cogradient can be applied to isolate the derivatives regarding the mappings [14]:

$$\frac{\partial f}{\partial \overline{z}} = \sum_{d=1}^{D} w_d \left(\left(\frac{\partial \operatorname{vec}(\mathscr{X})}{\partial \operatorname{vec}(z)^{\top}} \right)^{*} \cdot \frac{\partial f^{(d)}}{\partial \operatorname{vec}(\overline{\mathscr{X}})} \right). \tag{12}$$

The left-hand factor is related to the mapping and the right-hand factor is related to the decompositions applied. Both can be calculated analytically beforehand, independent of each other.

For the Hessian's approximation, one can write

$$B_k = \left(\sum_{d=1}^{D} w_d \left(\frac{\partial \operatorname{vec}(\mathscr{X})}{\partial \operatorname{vec}(z)^{\top}} \right)^{*} \cdot J_k^{(d)*} J_k^{(d)} \right) \cdot \frac{\partial \operatorname{vec}(\mathscr{X})}{\partial \operatorname{vec}(z)^{\top}}, \tag{13}$$

with the Jacobian $J_k^{(d)} = \left(\partial \operatorname{vec}(\mathscr{F})^{(d)} / \partial \operatorname{vec}(\mathscr{X})^{\top} \right)$ and $J_k^{(d)*} J_k^{(d)}$ called the (Jacobian's) Gramian. Note that this Gramian can be computed very efficiently because of the highly-structured $J_k^{(d)}$; hence, we omit the dot in between.

The first-order information of the mappings can be defined beforehand by deriving two matrix-vector products analytically:

$$\left(\frac{\partial \operatorname{vec}(\mathscr{X})}{\partial \operatorname{vec}(z)^{\top}} \right)^{*} \cdot l \quad \text{and} \quad \frac{\partial \operatorname{vec}(\mathscr{X})}{\partial \operatorname{vec}(z)^{\top}} \cdot r.$$

The latter can be used for equation (13) while the former can be used for both the equations (12) and (13).

B Implementation of the Algorithms

In this section, we elaborate on the implementation for the technique described in Section A. To use custom mappings $\mathscr{X}(z)$ in Tensorlab, only an evaluation method and the two previously described matrix-vector products have to be provided. We discuss the latter for the mappings used in this paper.

The first transformation used is a Vandermonde mapping

$$V_d(z) : z, d \to V, \qquad \text{with} \quad (V)_{ij} = z_i^{d_j}.$$

For general vectors $z \in \mathbb{C}^Z, d \in \mathbb{R}^D$ and given vectors $l \in \mathbb{C}^{ZD}, r \in \mathbb{C}^Z$ we have the following matrix-vector products while supposing $1 \leq i \leq Z$ and $1 \leq j \leq D$:

$$\left(\left(\frac{\partial \operatorname{vec}(V_d(z))}{\partial \operatorname{vec}(z)^\top}\right)^* \cdot l\right)_i = \sum_{j=1}^{D} l_{i+j\widetilde{Z}} \overline{(d_j z_i^{d_j-1})}, \quad \left(\frac{\partial \operatorname{vec}(V_d(z))}{\partial \operatorname{vec}(z)^\top} \cdot r\right)_{i+j\widetilde{Z}} = r_i d_j z_i^{d_j-1},$$

with $\widetilde{Z} = Z - 1$. The returned vectors are of sizes Z and ZD, respectively. In this paper, we have $d = [1, 2, \ldots, T]$.

Another transformation used (in Section 3.4 for common dynamics) is the Hadamard product between a constant matrix and a matrix containing variables: $H_A(Z) : Z, A \to A \odot Z$. For general matrices $Z \in \mathbb{C}^{I \times J}, A \in \mathbb{C}^{I \times J}$ and given vectors $l \in \mathbb{C}^{IJ}, r \in \mathbb{C}^{IJ}$ we have:

$$\frac{\partial \operatorname{vec}(H_A(Z))^*}{\partial \operatorname{vec}(z)^\top} \cdot l = \operatorname{vec}(\overline{A} \odot l), \quad \frac{\partial \operatorname{vec}(H_A(Z))}{\partial \operatorname{vec}(z)^\top} \cdot r = \operatorname{vec}(A \odot r).$$

Sample code for sum-of-exponential modeling using Tensorlab is given below. `zini` and `cini` are the initial approximations for z and C (obtained for example by Kung's method [7]), and `M` is the mask matrix enforcing zero pattern of C in case of common poles estimation.

```
function [sysh, xhini]
 = sem(y, n, zini, cini, M)

V = struct_vander(zini, ...
      [], [1 size(y, 1)])';
model.factorizations.H.data = y;
model.variables.z = zini;
model.variables.c = cini;

s_v = @(z, t)struct_vander(...
        z, t, [1 size(y, 1)]);
s_t = @(V, t) ...
      struct_transpose(V, t);
s_tv = @(z)s_t(s_v(z, []), []);
s_h = @(z, t) ...
      struct_prod_cst(z, t, M);

model.factors.V = {'z', s_v, s_t};
model.factors.c = {'c', s_h};

model.factorizations.H.cpd = ...
                        {'V', 'c'};
sol = sdf_nls(model);
z = sol.variables.z;
xhini = ones(n, 1);
sysh = ss(diag(z), [], ...
    sol.variables.c .* M, [], -1);
```