Software for weighted structured low-rank approximation

Ivan Markovsky and Konstantin Usevich
School of Electronics and Computer Science
University of Southampton
{im,ku}@ecs.soton.ac.uk

Abstract

A software package is presented that computes locally optimal solutions to low-rank approximation problems with the following features:

- mosaic Hankel structure constraint on the approximating matrix,
- weighted 2-norm approximation criterion,
- fixed elements in the approximating matrix,
- missing elements in the data matrix, and
- linear constraints on an approximating matrix’s left kernel basis.

It implements a variable projection type algorithm and allows the user to choose standard local optimization methods for the solution of the parameter optimization problem. For an $m \times n$ data matrix, with $n > m$, the computational complexity of the cost function and derivative evaluation is $O(m^2 n)$. The package is suitable for applications with $n \gg m$. In statistical estimation and data modeling—the main application areas of the package—$n \gg m$ corresponds to modeling of large amount of data by a low-complexity model. Performance results on benchmark system identification problems from the database DAISY and approximate common divisor problems are presented.

Keywords: mosaic Hankel matrix, low-rank approximation, total least squares, system identification, deconvolution, variable projection.

1 Introduction

Structured low-rank approximation is defined as low-rank approximation

$$\text{minimize } \|D - \hat{D}\| \text{ subject to } \text{rank}(\hat{D}) \leq r$$

with the additional constraint that the approximating matrix $\hat{D}$ has the same structure as the data matrix $D$. A typical example where a rank deficient structured matrix arises is when a sequence $p = (p_1, \ldots, p_{n_p})$ satisfies a difference equation with lag $\ell < \lceil n_p/2 \rceil$, i.e.,

$$R_0 p_t + R_1 p_{t+1} + \cdots + R_\ell p_{t+\ell} = 0, \quad \text{for } t = 1, \ldots, n_p - \ell.$$  (DE)

The system of equations (DE) is linear in the vector of parameters $R := \begin{bmatrix} R_0 & R_1 & \cdots & R_\ell \end{bmatrix}$, so that it can be written as $R \mathcal{H}_{\ell+1,n_p-\ell}(p) = 0$, where $\mathcal{H}_{\ell+1,n_p-\ell}(p)$ is a Hankel matrix constructed from $p$. This shows that, for $R \neq 0$, the fact that $p$ satisfies a difference equation (DE) is equivalent to rank deficiency of a Hankel matrix $\mathcal{H}_{\ell+1,n_p-\ell}(p)$.

Many problems in machine learning, system theory, signal processing, and computer algebra can be posed and solved as structured low-rank approximation problem for different types of structures and different approximation
criteria (see Section 5 and [18, 20]). In identification and model reduction of linear time-invariant dynamical systems, the structure is block-Hankel. In the computation of approximate greatest common divisor of two polynomials, the structure is Sylvester. In machine learning, the data matrix is often unstructured but the approximation criterion is a weighted 2-norm (or semi-norm, in the case of missing data).

Despite the academic popularity and numerous applications of the structured low-rank approximation problem, the only efficient publicly available software package for structured low-rank approximation is the one of [24]. The package presented in this paper is a significantly extended version of the software in [24]. The main extensions are summarized in Table 1 and are described in more details in Section 2.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Old version</th>
<th>New version</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix structure</td>
<td>block-Hankel/Toeplitz</td>
<td>matrix × mosaic-Hankel</td>
</tr>
<tr>
<td>cost function</td>
<td>2-norm</td>
<td>weighted 2-norm</td>
</tr>
<tr>
<td>exact data</td>
<td>whole blocks</td>
<td>arbitrary elements</td>
</tr>
<tr>
<td>missing data</td>
<td>not allowed</td>
<td>arbitrary elements</td>
</tr>
<tr>
<td>constraints on the optimization variable</td>
<td>unconstrained</td>
<td>linear constraints</td>
</tr>
<tr>
<td>interface</td>
<td>Matlab</td>
<td>Matlab, Octave, R</td>
</tr>
</tbody>
</table>

Table 1: Comparison of the old and new versions of the software.

The paper is organized as follows. Section 3 defines the considered weighted structured low-rank approximation and presents Matlab/Octave and R interfaces for calling the underlying C++ solver. Section 4 gives details about the solution method for solving the resulting parameter optimization problem. Implementation and software design issues are extracted in Appendix A. Section 5 lists applications of the package and describes in more details an application for solving scalar autonomous linear time-invariant identification and approximate common divisor problems. Appendix ?? defines the structured total least squares problem and explains its relation to structured low-rank approximation. Appendix B lists extra options of the software for choosing the optimization method.

2 Main features of the software

1. Matrix structure specification

The software package supports matrix structures of form

\[ \mathcal{S}(p) := \Phi \mathcal{H}_{m,n}(p), \]  

where \( \Phi \) is a full row rank matrix and \( \mathcal{H}_{m,n} \) is a mosaic Hankel structure [12], i.e., a \( q \times N \) block matrix

\[ \mathcal{H}_{m_1 \cdots m_q, n_1 \cdots n_N}(\mathcal{p}) = \begin{bmatrix} \mathcal{H}_{m_1,n_1}(p^{(1)}) & \cdots & \mathcal{H}_{m_1,n_N}(p^{(1N)}) \\ \vdots & \ddots & \vdots \\ \mathcal{H}_{m_q,n_1}(p^{(q1)}) & \cdots & \mathcal{H}_{m_q,n_N}(p^{(qN)}) \end{bmatrix}, \]  

(\( \mathcal{H}_{m,n} \))
with scalar Hankel blocks

\[ \mathcal{H}_{m,n}(p) := \begin{bmatrix} p_1 & p_2 & p_3 & \cdots & p_n \\ p_2 & p_3 & \cdots & p_{n+1} \\ p_3 & \cdots & \vdots & \ddots \\ \vdots & \ddots & \vdots & \ddots & \ddots \\ p_m & p_{m+1} & \cdots & \cdots & p_{m+n-1} \end{bmatrix} \in \mathbb{R}^{m \times n}. \] 

\((\mathcal{H}_{m,n})\) is more general than the block-Hankel and “flexible structure specification”, used in the old version of the software (see Section 3 in [24]). In fact, the “flexible structure specification” is equivalent to \(\mathcal{H}_{m,n}\) with equal \(n_i\)’s. Mosaic Hankel matrices with blocks of different column dimension allow us to solve, for example, system identification problems with multiple trajectories of different lengths.

The matrix \(\Phi\) further extends the class of mosaic Hankel matrices to (mosaic) Hankel-like matrices. A trivial example is the Toeplitz structure achieved by

\[ \Phi = J_m := \begin{bmatrix} & & & 1 \\ & & \ddots & \vdots \\ & \ddots & & \vdots \\ 1 & \cdots & & \ddots \end{bmatrix} \in \mathbb{R}^{m \times m} \text{ and } \mathcal{H}_{m,n}(p) = \mathcal{H}_{m,n}(p). \]

(Empty spaces in a matrix denote zeros.) A more interesting example is the Toeplitz-plus-Hankel structure, achieved by

\[ \Phi = [I_m \ J_m] = \begin{bmatrix} 1 & \cdots & \cdots & 1 \\ \cdots & \ddots & \cdots & \cdots \\ \cdots & \cdots & \ddots & \vdots \\ 1 & 1 & \cdots & \ddots \end{bmatrix} \in \mathbb{R}^{m \times 2m} \text{ and } \mathcal{H}_{m,n}(p) = \begin{bmatrix} \mathcal{H}_{m,n}(p^{(1)}) \\ \mathcal{H}_{m,n}(p^{(2)}) \end{bmatrix}. \] 

Other examples of using \(\Phi\), including multipolynomial Sylvester matrices [13], can be found in Section 5.

2. Cost function

In the old version of the package, the approximation criterion is the 2-norm (unweighted low-rank approximation). In the new version, the approximation criterion is the weighted 2-norm. Weights are needed for example when the accuracy of the elements of \(p\) vary, see [39]. Weights can be used also to replace the parameter 2-norm with the Frobenius matrix norm of the error matrix \(\mathcal{I}(p - \hat{p})\). As explained in Note 2, the weighted approximation criterion can be viewed alternatively as a modification of the data matrix structure.

3. Exact data

Structure parameters can be fixed to predefined values. (In the old version of the package only subblocks of the structured matrix could be specified as fixed.) This feature allows us to solve problems, such as system identification under exactly known initial conditions, where the upper-left triangles of the Hankel blocks are fixed, and approximate common divisor computation, where the upper-left and lower-right triangles or the Hankel blocks are fixed.

4. Missing data\(^1\)

\(^1\)Currently (January 18, 2013), this feature is implemented only in an experimental Matlab version of the software. It can be simulated in the C++ solver by assigning “small” weights to the missing parameters and replacing the missing values with zeros.
A new feature of the software is the ability to deal with missing data. Due to specific experiment design or data corruption, the data may be incomplete in real-life applications. A well known example from machine learning is prediction of user ratings (recommender system), see, e.g., [2]. A possibility to specify missing values in Hankel low-rank approximation problems allows us to solve system identification problems with incomplete or irregularly sampled data.

5. Total least squares and low-rank approximation

In the old version of the package, the optimization variable is the $X$ matrix in the problem of solving approximately an overdetermined system of linear equations $AX \approx B$. This problem may be viewed as a restriction of the low-rank approximation problem to a subclass of matrices. (For more details see Appendix ??). To preserve backward compatibility, the new version of the software can also solve the structured total least-squares problems, see Note 7.

6. Constraints on the optimization variables

The new version of the package uses as optimization variable a matrix $R$ whose rows form a basis for the left kernel of the approximation $\hat{D}$ and allows specification of linear structure constraints on $R$. An application where such a constraint is needed is multivariate system identification with fixed observability indices [10]. Other applications are given in Section 5.

7. Interfaces to scientific computing environments

The package provides interface for calling the underlying C++ solver from Matlab, Octave, and R, which are among the most often used computing environments for scientists and engineers.

An advantage of unifying structured and weighted low-rank approximation with missing and fixed values and constraints on the optimization variable is that a single algorithm and a piece of software solves a large variety of problems. Table 2 in Section 5 lists special cases of the generic problem solved by the software package. Each special case is motivated by applications, which are considered in the literature and specialized algorithms are developed for their solution. Only a few of the algorithms reported in the literature, however, are implemented in publicly available software.

Fixed values in the approximating matrix can be viewed as an extreme case of weighted low-rank approximation with large weights corresponding to the fixed parameters. Similarly, the missing data case can be viewed as weighted low-rank approximation with small weights corresponding to the missing parameters. Both cases are equivalent to a single singular (zero or infinite weights) weighted low-rank approximation problem and can be solved with the same algorithm.

The generality of the algorithm is not compromised by its computational efficiency. Eliminating the approximation $\hat{D}$ and exploiting the structure in the resulting nonlinear least squares problem, the cost function and derivative evaluation is done in $O(m^2n)$ floating point operations [21]. Indeed, the more general algorithm implemented in the package reduces to the algorithm in [24] in the special case treated by the old version of the software.

The software is written in C++ with Matlab/Octave and R interfaces. The C++ code is complemented by an experimental implementation written in Matlab. The latter supports general linearly structured matrix as and experimental features but is inefficient and can be used only for small size (say, $m \leq n < 100$) problems. In Matlab, the C++ solver can be called directly via the mex-file slra_mex or via a wrapper function, which has the same interface as the mex file but can optionally call the experimental version slra_ext. The package is hosted at the following address:

http://github.com/slra/slra
3 Problem formulation

We denote missing data values by the symbol NaN ("not a number") and define the extended set of real numbers $\mathbb{R}_e$ as the union of the set of the real numbers $\mathbb{R}$ and the symbol NaN:

$$\mathbb{R}_e := \mathbb{R} \cup \text{NaN}.$$  

The considered structured low-rank approximation problem is defined as follows.

**Problem 1.** Given:

- structure specification $\mathcal{S}$,
- vector of structure parameters $p \in \mathbb{R}^n_p$,
- nonnegative vector $w \in (\mathbb{R}_+ \cup \{\infty\})^n_r$, defining a weighted (semi-)norm  

$$\|p\|_w^2 := \sum_{i \mid p_i \neq \text{NaN}} w_ip_i^2,$$

- desired rank $r$,

find a structure parameter vector $\hat{p}$, such that the corresponding matrix $\mathcal{S}(\hat{p})$ has rank at most $r$, and is as close as possible to $p$ in the sense of the weighted semi-norm $\| \cdot \|_w$, i.e.,

$$\text{minimize over } \hat{p} \in \mathbb{R}^n_p \|p - \hat{p}\|_w^2 \text{ subject to } \text{rank}(\mathcal{S}(\hat{p})) \leq r.$$ (SLRA)

Without loss of generality, it is assumed that $m \leq n$. An infinite weight $w_i = \infty$ imposes the equality constraint $\hat{p}_i = p_i$ on the optimization problem (SLRA).

Problem (SLRA) is in general nonconvex. In the software package, it is solved numerically by local optimization methods. The Matlab wrapper function for low-rank approximation is defined as follows:

```matlab
function [ph, info] = slra(p, s, r, varargin)
```

The output argument `ph` of the `slra` function is a locally optimal solution $\hat{p}$ of (SLRA) and `info` is a structure containing additional information about the computed solution:

- `info.Rh` is an $(m - r) \times r$ full row rank matrix $\hat{R}$, such that $\hat{R}\mathcal{S}(\hat{p}) = 0$ (low-rank certificate),
- `info.fmin` is the cost function value $\|p - \hat{p}\|_w^2$ at the computed solution,
- `info.iter` is the number of iterations performed by the optimization solver, and
- `info.time` is the execution time.

Next we describe the required input parameters $p$, $s$, and $r$. Appendix B describes the optional parameter `opt`.

Matrix structure specification

The structure ($\mathcal{S}$) is specified by the two vectors

$$\mathbf{m} := \begin{bmatrix} m_1 & \cdots & m_q \end{bmatrix}^\top \in \mathbb{N}^q \quad \text{and} \quad \mathbf{n} := \begin{bmatrix} n_1 & \cdots & n_N \end{bmatrix}^\top \in \mathbb{N}^N \quad (\mathbf{m}, \mathbf{n})$$
and the matrix $\Phi \in \mathbb{R}^{m \times m'}$, where

$$m' := \sum_{i=1}^{q} m_i.$$ 

The compulsory input arguments $p$, $s$, and $r$ of the `slra` function correspond to the vector of parameters $p$, the problem structure, and the rank $r$, respectively. $s$ should be a structure with a field $m$ and, optionally, a field $n$, containing the vectors $(m, n)$. If $s.n$ is skipped, by default $N = 1$ and

$$n = n = \frac{n_p - \sum_{i=1}^{q} m_i}{q} + 1.$$ 

Optionally, $s$ can have a field `phi`, containing the matrix $\Phi \in \mathbb{R}^{m \times m'}$. The default value for $\Phi$ is the identity matrix of size $m = m'$. The vector $p$ is composed of the structure parameter vectors of all scalar Hankel blocks in $(\mathcal{H}_{m,n})$, ordered first top to bottom and then left to right, i.e.,

$$p = (p^{(11)}, \ldots, p^{(q1)}, \ldots, p^{(1N)}, \ldots, p^{(qN)}).$$

### Weight specification, fixed and missing parameter values

The weight vector $w$ is passed to the `slra` function by the field $w$ of $s$. If $w$ is not specified, its default value is the vector of all ones, corresponding to unweighted low-rank approximation. The parameter $s.w$ can be

- $n_p$-dimensional nonnegative vector, in which case $w = s.w$,
- $q \times N$ nonnegative matrix, in which case the weights corresponding to the structure parameters of the $(i,j)$th Hankel block in $(\mathcal{H}_{m,n})$ are all equal to $w_{ij}$, or
- $q$-dimensional nonnegative vector, in which case the weights corresponding to the structure parameters of the $(i,j)$th Hankel block in $(\mathcal{H}_{m,n})$ are all equal to $w_i$, for all $j$.

The second and third options of specifying $s.w$ evoke a more efficient computational method but are supported only by the C++ solver.

Setting $w_i$ to $\infty$ has the effect of specifying the parameter $p_i$ as exact, i.e., the approximation problem (SLRA) is solved with the additional constraint that $\hat{p}_i$ is equal to the given parameter value $p_i$ (and the weighted norm $\| \cdot \|_w$ is evaluated over the parameters with finite weights only). Setting $w_i$ to 0 has the effect of ignoring the value of the parameter $p_i$ in the solution of the approximation problem. Alternatively, ignored parameter values may be specified by the symbol `NaN`. From a practical point of view, ignored structure parameter values are “missing data”.

**Note 2** (Incorporating the weights $w$ into the matrix structure $\mathcal{S}$). A weighted structured problem (SLRA) can be solved as an equivalent unweighted problem (SLRA), i.e., with weights $w_i = 1$ for all $i$, and a modified structure specification $\mathcal{S}': \mathbb{R}^n \rightarrow \mathbb{R}^{m \times n}$, defined by

$$\mathcal{S}'((p_1, \ldots, p_{n_p})) = \mathcal{S}((w_1 p_1, \ldots, w_{n_p} p_{n_p})).$$

This fact shows that the weighted norm specification can be viewed equivalently as a modification of the data structure.

**Note 3** (Conversion from the structure specification in [24]). In order to facilitate the transition from the old version of the software to the new one, we provide in the package a function `s_old2s` converting the old style structure specification (see [24, page 314]) to the new one. Block structured matrices consisting of block-Hankel and unstructured blocks are specified solely by the $(m,n)$ parameters. Block-Toeplitz matrices are specified as block-Hankel matrices.
with suitable row permutations. In this case, in addition to the parameters \((m, n)\), the transformation matrix \(\Phi\) is used. Finally, fixed blocks are specified by infinite elements in the weight vector \(w\). In general, the transition from the old structure specification format to the new one involves the \((m, n), \Phi,\) and \(w\) parameters.

**Note 4 (R interface)**. Calling the C++ solver from R is done by an R function \(\text{slra}\) with the same parameters as in the Matlab mex function, except that in R lists are used instead of structures, and the list elements are accessed by \(<\text{list}>$.\text{<element>}\) instead of by \(<\text{structure}>$.\text{<element>}.\) The computed result is returned in a list with fields \(\text{ph}\) and \(\text{info}\), which contain, respectively, \(\hat{p}\) and the \(\text{info}\) variable defined above. Optional logical parameters \(\text{compute.ph}\) and \(\text{compute.Rh}\) can be used to disable the computation of \(\hat{p}\) and \(\hat{R}\). By default, \(\hat{p}\) is not computed.

Examples of using the R interface can be found in the directory \text{test_r}\ of the package distribution.

## 4 Solution method

This section outlines the method used in the package for solving the structured low-rank approximation problem (SLRA). Let \(d\) be the rank reduction

\[
d := m - r.
\]

The rank constraint is represented in a kernel form

\[
\text{rank} \left( \mathcal{S}(\hat{p}) \right) \leq r \iff \text{there is full row rank } R \in \mathbb{R}^{d \times m}, \text{ such that } R \mathcal{S}(\hat{p}) = 0,
\]

(KER)

i.e., the left kernel of \(\mathcal{S}(\hat{p})\) is parameterized by the \(d = m - r\) linearly independent rows of the matrix \(R\). Then (SLRA) is equivalent to the following double minimization problem

\[
\text{minimize over full row rank } R \in \mathbb{R}^{d \times m} \quad f(R), \quad \text{(SLRA)}
\]

where

\[
f(R) := \left( \min_{\hat{p} \in \mathbb{R}^n} \|p - \hat{p}\|^2_w \text{ subject to } R \mathcal{S}(\hat{p}) = 0 \right).
\]

(SLRA) is referred to as the outer minimization and \((f(R))\) as the inner minimization. The inner minimization problem is a (generalized) linear least-norm problem and can be solved analytically for given \(R\), resulting in a closed form expression of the cost function \(f\) in the outer minimization problem. The derivative of \(f\) with respect to the elements of \(R\) can also be computed analytically. Fast methods for evaluation of \(f\) and its derivatives are presented in [?].

In data modeling—the main application area of the software—\(R\) can be interpreted as a model parameter. For example, in system identification, \(R\) is related to a difference equation (DE) representation of the system. Typically, a large amount of data is fitted by a simple model, which implies that the dimension of \(R\) is small compared to the dimension of \(\hat{p}\). Problem \((f(R))\) is bilinear in the variables \(R\) and \(\hat{p}\), which makes possible the elimination of \(\hat{p}\) for a fixed \(R\). Many estimation problems have similar bilinear structure and their solution methods are based on elimination of the large dimensional optimization variable.

The elimination of \(\hat{p}\) in the analytic solution of \((f(R))\) is similar to the projection step in the variable projections method [8] for solving separable nonlinear least squares problems. Note, however, that the standard variable projections method is invented as a solution method for \textit{unconstrained} optimization problems, where the cost function is explicitly given. The structured low-rank approximation problem (SLRA) in contrast is a constraint optimization problem and the parameterization (KER) is an implicit function of the optimization variables. As discussed next, (SLRA) is an optimization problem over a Grassmann manifold, which makes it rather different from the standard setup, for the variable projections method.
Note 5 (Feasibility of the minimization problem over \( \hat{\rho} \)). In the analytic solution of \((f(R))\), it is assumed that the problem is feasible for any \( R \) in the search space. A necessary condition for this to hold is that the number of structure parameters is sufficiently large:

\[
n_p > dn. \tag{C}
\]

Condition (C) imposes restrictions on the class of problems that can be solved by the approach implemented in the package. For example, in scalar Hankel low-rank approximation problems the rank reduction \( d \) can be at most one. Interestingly, in applications of problem (SLRA) to approximate realization, system identification, and approximate common divisor computation constraint (C) can always be satisfied.

4.1 Parametrization of the search space and constraints

Problem (SLRA\(_R\)) is an optimization problem on a Grassmann manifold \( \text{Gr}(d,m) \) (set of \( d \)-dimensional subspaces of \( \mathbb{R}^m \)). Indeed, \( f(R_1) = f(R_2) \) if the rows of \( R_1 \) and \( R_2 \) span the same subspace. In the package, we map the optimization on \( \text{Gr}(d,m) \) to an optimization problem on an Euclidean space, see [36]. The problem is turned into optimization of \( f \) over the set of full row rank matrices \( R \in \mathbb{R}^{d \times m} \) that represent all (or a generic part) of the \( d \)-dimensional subspaces.

In some applications, it is necessary to impose linear constraints on \( R \), which can also be incorporated into the parametrization of the search space.

- The experimental Matlab solver uses a general linear constraint on \( R \)

\[
R = \mathcal{R}'(\theta) := \text{vec}^{-1}_d(\theta \Psi), \quad \text{where} \quad \theta \in \mathbb{R}^{n_p}, \tag{\( \theta \mapsto R \)}
\]

defined by a matrix \( \Psi \in \mathbb{R}^{n_p \times dm} \). (Here \( \text{vec}(\cdot) \) is the column-wise vectorization operator and \( \text{vec}^{-1}(\cdot) \) is its inverse.) The rank constraint on \( R \) is imposed by

\[
RR^\top = I_d. \tag{f.r.r. R}
\]

- The C++ solver uses a matrix-product constraint on the matrix \( R \)

\[
R = \mathcal{R}(\Theta) := \Theta \Psi, \quad \text{where} \quad \Theta \in \mathbb{R}^{d \times m''}, \tag{\( \Theta \mapsto R \)}
\]

defined by a matrix \( \Psi \in \mathbb{R}^{m'' \times m} \). The rank constraint on \( R \) is imposed by

\[
\Theta = \begin{bmatrix} X & -I_d \end{bmatrix}, \quad \text{for some} \quad X \in \mathbb{R}^{d \times (m'' - d)}. \tag{\( \Theta \leftrightarrow X \)}
\]

The \( \Psi \) matrix is passed to the \texttt{slra} function by an optimization option \texttt{opt.psi}. The default value of \( \Psi \) is \( I_m \) for the C++ version and \( I_{dm} \) for the Matlab version. The \( \Psi \) matrices—\( \Psi_C \) for the C++ versions and \( \Psi_M \) for the Matlab versions—are generally different. They coincide if and only if the rank reduction is one, i.e., \( d = 1 \). The constraint \( (\theta \mapsto R) \) is more general and includes the constraint \( (\Theta \mapsto R) \) by choosing

\[
\Psi_M = \Psi_C \otimes I_d \quad \text{and} \quad \theta = \text{vec}(\Theta).
\]

The rank constraint (f.r.r. \( R \)) parametrizes \( \text{Gr}(d,m) \), and turns the structured low-rank approximation problem into an optimization problem on \( \mathbb{R}^{md} \) with a quadratic constraint. The rank constraint \( (\Theta \leftrightarrow X) \) parametrizes a generic part of \( \text{Gr}(d,m) \). Its main advantage is that the structured low-rank approximation problem problem is turned into an unconstrained optimization problem on \( \mathbb{R}^{(m'' - d)d} \).
Note 6 (On parametrization of the whole $Gr(d, m)$ in the C++ version). The whole $Gr(d, m)$ can be covered by taking different $\Psi$ (see [36]) or changing $\Psi$ during the optimization, which is done in the optimization algorithms of [17]. Currently, in the C++ version, $\Psi$ is a fixed parameter.

Note 7 (Solution of structured total least squares problems). Setting $\Psi = I_d$ in the C++ version is equivalent to solving a structured total least squares problem. In this case, $\hat{R} = \begin{bmatrix} X & -I_d \end{bmatrix}$.

4.2 Local optimization methods

Due to the elimination of $\hat{p}$, for $n \gg m$, the computation time of the search direction by the optimization method is negligible in comparison with the computation time for the cost function and derivative evaluations. Therefore, we use standard local optimization methods to minimize the cost function $f$.

The C++ solver uses local optimization methods from the GSL library (see Appendix B). One option is to use the Ned-Melder method for minimization without derivatives, which is slow but robust. A faster alternative is to use the BFGS quasi-Newton method. The cost function gradient is computed analytically by

$$df(R,H) = 2y^T d\nu(R,H) - y^T d\Gamma(R,H) y,$$

where $y := \Gamma^{-1}v(R)$.

It can be shown that the gradient evaluation has computational complexity of the same order as the one of the cost function evaluation.

The default optimization method for the C++ solver is the Levenberg-Marquardt algorithm [28, 4], which is typically faster and more robust in practice than the quasi-Newton methods. The Jacobian of $C^{-\top}v$, where $C$ is the Cholesky factor of $\Gamma$ is replaced by the pseudo-Jacobian [11], which computation is of the order of $O(smn)$.

The package provides an interface to the cost function and derivatives evaluation, so that any optimization method (implemented in or callable from C/C++) can be used. More details about the C++ interface can be found in Appendix A. The experimental Matlab solver uses the function fmincon for constrained nonlinear minimization from the Optimization Toolbox of Matlab. In this case, the derivatives are approximated numerically.

All local optimization methods require an initial approximation. By default, unstructured rank-$r$ approximation $\hat{D}_{la}$ of the data matrix $D = \mathcal{A}(p)$ is used, i.e., the initial value for the parameter $R$ is a full row rank matrix $R_{la} \in \mathbb{R}^{d \times m}$, such that $R_{la} \hat{D}_{la} = 0$. The default value can be overwritten by an optional argument opt.Rini.

In order to obtain the $\theta$ or $\Theta$ parameter from $R$, the equation $\mathcal{A}(\theta) \approx R_{la}$ is solved in the least-squares sense. For the C++ solver, the obtained matrix $\Theta$ is then converted to $X$ by

$$X := -P^{-1}Q,$$

where $\Theta := \begin{bmatrix} Q & P \end{bmatrix}$, $Q \in \mathbb{R}^{d \times (m^2 - d)}$, and $P \in \mathbb{R}^{d \times d}$.

The result of the slra function—a locally optimal value of the parameter $R$—is returned in the variable info.R. The corresponding locally optimal approximation $\hat{p}$ is computed by solving the inner minimization problem ($f(R)$). The solution is given by

$$\hat{p}(R) = p + G(R) (\Gamma(R))^{-1} v(R),$$

where $G(R)$ is a linear function in the elements of $R$, defined by

$$\text{vec} \left( R \mathcal{A}(p) \right) = G(R)p.$$
is motivated by applications. (See the references cited in the table.) Section 5.1 demonstrates the efficiency of the package on data sets from the database for system identification DAISY [30]. Section 5.2 shows numerical examples of computing approximate common divisors of two polynomials (example 14), using Sylvester structured low-rank approximation.

<table>
<thead>
<tr>
<th>#</th>
<th>example</th>
<th>application(s)</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>unstructured uniform weights</td>
<td>factor analysis</td>
<td>[9]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>subspace methods</td>
<td>[26, Ch. 7]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>latent semantic analysis</td>
<td>[5]</td>
</tr>
<tr>
<td>2</td>
<td>element-wise weights</td>
<td>chemometrics</td>
<td>[39]</td>
</tr>
<tr>
<td>3</td>
<td>scalar Hankel</td>
<td>model reduction</td>
<td>[1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>autonomous SYSID</td>
<td>[26, Ch. 11]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>linear prediction</td>
<td>[16]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>shape from moments</td>
<td>[29, 7]</td>
</tr>
<tr>
<td>4</td>
<td>Hankel with structured kernel (palindromic)</td>
<td>spectral estimation</td>
<td>[35]</td>
</tr>
<tr>
<td>5</td>
<td>Toeplitz + Hankel</td>
<td>isospectral flow</td>
<td>[32]</td>
</tr>
<tr>
<td>6</td>
<td>Hankel with structured kernel (fixed poles)</td>
<td>SYSID with some predefined poles</td>
<td>[19]</td>
</tr>
<tr>
<td>7</td>
<td>( q \times 1 ) block Hankel</td>
<td>multivariable SYSID</td>
<td>[27]</td>
</tr>
<tr>
<td>8</td>
<td>Hankel with structured kernel (fixed obsrv. indices)</td>
<td>SYSID with fixed obsrv. indices</td>
<td>[10]</td>
</tr>
<tr>
<td>9</td>
<td>( q \times N ) block Hankel</td>
<td>SYSID from multiple time series ( T_i = T )</td>
<td>[27]</td>
</tr>
<tr>
<td>10</td>
<td>Hankel blocks next to each other</td>
<td>SYSID from multiple time series</td>
<td>[31]</td>
</tr>
<tr>
<td>11</td>
<td>Hankel with fixed variables</td>
<td>output error SYSID</td>
<td>[27]</td>
</tr>
<tr>
<td>12</td>
<td>Unstructured beneath Hankel block</td>
<td>deconvolution</td>
<td>[18]</td>
</tr>
<tr>
<td>13</td>
<td>Hankel with first ( \ell ) elements fixed</td>
<td>SYSID with fixed initial conditions</td>
<td>[34]</td>
</tr>
<tr>
<td>14</td>
<td>Sylvester</td>
<td>approximate GCD of two polynomials</td>
<td>[14]</td>
</tr>
<tr>
<td>15</td>
<td>generalized Sylvester</td>
<td>approximate GCD of ( N ) polynomials</td>
<td>[13]</td>
</tr>
<tr>
<td>16</td>
<td>unstructured with missing elements</td>
<td>recommender systems</td>
<td>[33]</td>
</tr>
<tr>
<td>17</td>
<td>Hankel with missing elements</td>
<td>SYSID with missing data</td>
<td>[31]</td>
</tr>
<tr>
<td>18</td>
<td>missing and fixed elements</td>
<td>matrix completion</td>
<td>[3]</td>
</tr>
</tbody>
</table>

Table 2: Summary of examples. (SYSID — system identification, GCD — greatest common divisor)
5.1 Performance comparison with the old version of the package

In [22] and [25], the performance of the old version of the package is tested on benchmark system identification problems from the database DAISY [30]. The problems involve vector time-series and are fitted by multiple-inputs multiple-outputs linear time-invariant systems. The identification problem in this case is equivalent to block-rank low-rank approximation, which is also a special case of the mosaic Hankel low-rank approximation problem (SLRA). For a description of the test examples, we refer the reader to [22].

In Table 3, the execution time of the new and the old version of the package are compared for the same setting of the experiment as in [22]. In all examples, the approximate solutions computed by the two versions of the software are the same. The results show that the efficiency of the new version of the package is improved by an average of 40%. Note that the speedup is achieved by software improvements despite of the fact that the new version treats a more general problem.

| #  | Data set name                                                                 | $T$ | $m$ | $p$ | $\ell$ | $t_{new}$ (sec) | $t_{old}$ (sec) | $\Delta t$ (%) |
|----|-------------------------------------------------------------------------------|-----|-----|-----|-------|----------------|----------------|--------------|}
| 1  | Data of a simulation of the western basin of Lake Erie                        | 57  | 5   | 2   | 1     | 0.01           | 0.01           | 0            |
| 2  | Data of ethane-ethylene distillation column                                  | 90  | 5   | 3   | 1     | 0.02           | 0.03           | 33           |
| 3  | Heating system                                                               | 801 | 1   | 1   | 2     | 0.01           | 0.02           | 50           |
| 4  | Data from an industrial dryer (Cambridge Control Ltd)                         | 867 | 3   | 3   | 1     | 0.16           | 0.25           | 36           |
| 5  | Data of a laboratory setup acting like a hair dryer                          | 1000| 1   | 1   | 5     | 0.02           | 0.04           | 50           |
| 6  | Data of the ball-and-beam setup in SISTA                                     | 1000| 1   | 1   | 2     | 0.01           | 0.02           | 50           |
| 7  | Wing flutter data                                                            | 1024| 1   | 1   | 5     | 0.02           | 0.04           | 50           |
| 8  | Data from a flexible robot arm                                               | 1024| 1   | 1   | 4     | 0.01           | 0.01           | 0            |
| 9  | Data of a glass furnace (Philips)                                            | 1247| 3   | 6   | 1     | 2.88           | 4.41           | 35           |
| 10 | Heat flow density through a two layer wall                                   | 1680| 2   | 1   | 2     | 0.09           | 0.22           | 59           |
| 11 | Simulation data of a pH neutralization process                             | 2001| 2   | 1   | 6     | 0.03           | 0.06           | 50           |
| 12 | Data of a CD-player arm                                                      | 2048| 2   | 2   | 1     | 0.09           | 0.23           | 61           |
| 13 | Data from a test setup of an industrial winding process                      | 2500| 5   | 2   | 2     | 0.64           | 0.92           | 30           |
| 14 | Liquid-saturated steam heat exchanger                                       | 4000| 1   | 1   | 2     | 0.03           | 0.07           | 57           |
| 15 | Data from an industrial evaporator                                          | 6305| 3   | 3   | 1     | 1.36           | 2.22           | 39           |
| 16 | Continuous stirred tank reactor                                              | 7500| 1   | 2   | 1     | 0.24           | 0.75           | 68           |
| 17 | Model of a steam generator at Abbott Power Plant                             | 9600| 4   | 4   | 1     | 13.10          | 15.77          | 17           |

Table 3: Performance test on examples from DAISY: $T$—number of data points, $m$—number of inputs, $p$—number of outputs, $\ell$—lag of the identified model, $t_{new}$ and $t_{old}$ execution times in seconds for the new and old versions of the package, respectively, $\Delta t$—percentage speedup, achieved by the new version of the package.

5.2 Approximate greatest common divisor of two polynomials

Consider the polynomials $p^1$ and $p^2$ of degrees $d_1$ and $d_2$, respectively, and a positive integer $\ell < \min(d_1, d_2)$. The Sylvester matrix of $p^1$ and $p^2$ with parameter $\ell$ is a

$$(d_1 + d_2 - 2\ell + 2) \times (d_1 + d_2 - \ell + 1)$$
mosaic Hankel matrix with upper-left and lower-right triangles of the block-elements fixed to zero:

$$S_{\ell}(p^1, p^2) := \begin{bmatrix} S_{d_1 - \ell}(p^1) \\ S_{d_2 - \ell}(p^2) \end{bmatrix}, \quad \text{where} \quad S_d(p) := \begin{bmatrix} p_0 & p_1 & \cdots & p_d \\ \vdots & \ddots & \vdots \\ p_0 & p_1 & \cdots & p_d \end{bmatrix} \in \mathbb{R}^{(d+1) \times (d+d_{p^1})}. \quad \text{(SYL)}$$

The degree of the greatest common divisor of $p^1$ and $p^2$ is equal to

$$\text{degree}(\gcd(p^1, p^2)) = d_1 + d_2 - \text{rank}(S_0(p^1, p^2)). \quad (*)$$

The considered approximate common divisor problem is defined as follows: given polynomials $p^1$ and $p^2$ and a lower bound $\ell > 0$ on the degree of the common divisor, modify $p^1$ and $p^2$, as little as possible, so that the modified polynomials $\hat{p}^1$ and $\hat{p}^2$ have a greatest common divisor of degree at least $\ell$, i.e.,

$$\begin{align*}
\text{minimize} \quad & \hat{p}^1 \in \mathbb{R}^{d_1+1} \text{ and } \hat{p}^2 \in \mathbb{R}^{d_2+1} \\
\text{subject to} \quad & \text{degree}(\gcd(\hat{p}^1, \hat{p}^2)) \geq \ell.
\end{align*} \quad \text{(ACD)}$$

The approximate common divisor for the polynomials $p^1$ and $p^2$ is the exact greatest common divisor of $\hat{p}^1$ and $\hat{p}^2$.

It can be shown [23] that the approximate common divisor problem (ACD) is equivalent to a Sylvester low-rank approximation problem

$$\begin{align*}
\text{minimize} \quad & \hat{p}^1 \in \mathbb{R}^{d_1+1} \text{ and } \hat{p}^2 \in \mathbb{R}^{d_2+1} \\
\text{subject to} \quad & \text{rank}(S_{\ell}(p^1, p^2)) \leq d_1 + d_2 - 2\ell + 1.
\end{align*} \quad \text{slra arguments for approximate GCD of 2 polynomials}$$

Our first example is Example 4.1 from [40]. The given polynomials are

$$p^1(z) = (4 + 2z + z^2)(5 + 2z) + 0.05 + 0.03z + 0.04z^2$$
$$p^2(z) = (4 + 2z + z^2)(5 + z) + 0.04 + 0.02z + 0.01z^2$$

and an approximate common divisor of degree $\ell = 2$ is sought.

\langle example GCD \rangle =

```plaintext
clear all; d1 = 3; d2 = 3; ell = 2;
p1 = conv([4 2 1], [5 2]) + [0.05 0.03 0.04 0];
p2 = conv([4 2 1], [5 1]) + [0.04 0.02 0.01 0];
\langle slra arguments for approximate GCD of 2 polynomials \rangle
[ph, info] = slra(p, s, r);
```

The solution computed by the slra function

```plaintext
\langle example GCD \rangle + =

ph1 = ph(2:5), ph2 = ph(8:11), r_ph1 = roots(ph1), r_ph2 = roots(ph2)
```

Numerical examples

The arguments for approximate GCD of 2 polynomials

```plaintext
s.m = [d2 - ell + 1; d1 - ell + 1];
s.n = d1 + d2 - ell + 1;
z1 = zeros(d2 - ell, 1); z2 = zeros(d1 - ell, 1);
p = [z1; p1(:); z1; z2; p2(:); z2];
s.w = 1 ./ p; s.w(~isinf(s.w)) = 1;
r = d1 + d2 - 2 * ell + 1;
```
coincides (up to errors due to the numerical precision) with the one reported in [40].

The second example is Example 4.2, case 1, from [40] (originally given in [15]). The given polynomials are

\[ p_1(\xi) = (1 - \xi)(5 - \xi) = 5 - 6\xi + \xi^2 \]
\[ p_2(\xi) = (1.1 - \xi)(5.2 - \xi) = 5.72 - 6.3\xi + \xi^2 \]

and an approximate common divisor of degree \( \ell = 1 \) (a common root) is sought.

\[ \langle \text{example GCD} \rangle + \equiv \]
\[ d_1 = 2; d_2 = 2; \quad e_{11} = 1; \]
\[ p_1 = \text{conv}([1 -1], [5 -1]); \]
\[ p_2 = \text{conv}([1.1 -1], [5.2 -1]); \]
\[ \langle \text{slra arguments for approximate GCD of 2 polynomials} \rangle \]
\[ [\text{ph}, \text{info}] = \text{slra}(p, s, r); \]

Again, the solution computed by the \text{slra} function

\[ \langle \text{example GCD} \rangle + \equiv \]
\[ \text{ph1} = \text{ph}(2:4), \text{ph2} = \text{ph}(7:9), \text{r\_ph1} = \text{roots(ph1)}, \text{r\_ph2} = \text{roots(ph2)} \]

coincides (up to numerical computation errors) with the one reported in the literature.

6 Conclusions

The developed software package is a generic tool for data modeling and has numerous applications in system theory and identification, signal processing, machine learning, chemometrics, and computer algebra. Its functionality generalized the one of the software of [24] by allowing specification of element-wise weights, arbitrary fixed and missing elements, linear constraints on the parameter matrix, and by generalizing the structure of the approximating matrix to the class of mosaic Hankel-like matrices. Planned extensions of the package are

- multiple rank constraints,
- mosaic Hankel matrices with repeated Hankel blocks (\textit{i.e.}, structure parameters common to two or more blocks),
- linear equality constraints on the structure parameter vector,
- nonlinear structure of the kernel parameter \( R \).

On the practical side, we plan to experiment with different optimization strategies (methods for optimization on a Grassmannian manifold) and compare the performance of the package with state-of-the-art problem dependent methods for the applications in Table 2.
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”.

References


A Implementation details

The structured low-rank approximation solver is written in C++ language and the implementation uses object-oriented style. The LAPACK and BLAS libraries are used for the matrix computations and the GNU scientific library (GSL) [6] is used for vector-matrix multiplications and for nonlinear optimization. The key computational step—Cholesky factorization of the $\Gamma$ matrix—is done by the LAPACK’s function DPBTRF, which exploits the banded structure of the matrix. In case of blockwise weights (see Section 3), the package can optionally use the MB02GD function from the SLICOT library [38], which exploits both the banded and Toeplitz structure of $\Gamma$. (In our test examples MB02GD gives no advantage in terms of computation time.)

The package is divided into the following sub directories:

- **cpp**: C++ core classes and functions.²
- **mex**: source files for Matlab/Octave interface.
- **Rslra**: source files for R interface.
- **test_m**: demo files for Matlab/Octave interface.
- **test_r**: demo files for R interface.
- **test_c**: demo files for C++ interface.
- **doc**: documentation and examples.
- **ident**: wrapper function for system identification.

A.1 Main function

The solver is called in C via the function slra, defined as follow:

\[
\texttt{slra( const gsl\_vector *p\_in, Structure *s, int d, OptimizationOptions *opt,}
\]
\[
gsl\_matrix *Rini, gsl\_matrix *Phi, gsl\_matrix *Psi,}
\]
\[
gsl\_vector *p\_out, gsl\_matrix *r\_out, gsl\_matrix *v\_out });
\]

- **p_in**: input vector $p$.
- **s**: object of type Structure, see Section A.2.
- **d**: rank reduction.
- **opt**: optimization options and output information.
- **Rini**: transposed initial approximation.
- **Phi**: $\Phi^\top$ matrix (see $S$).
- **Psi**: $\Psi^\top$ matrix (see $\Theta \mapsto R$).
- **p_out**: approximation $\hat{p}$.
- **R_out**: low-rank certificate $\hat{R}^\top$.
- **v_out**: error covariance $(J^\top J)^{-1}$ of vec($X$).

Note 8 (C row-major convention). All matrices are transposed, due to the row-major order convention of C/GSL.

A.2 Structure specification and object-oriented paradigm

The Structure class represents the structure specification $S$ and the weights vector $w$.

\[
\texttt{class Structure \{}
\]
\[
\texttt{public:}
\]
\[
\texttt{virtual ~Structure() {}}
\]
\[
\texttt{virtual int getNp() const = 0;}
\]
\[
\texttt{virtual int getM() const = 0;}
\]

²Type declarations and function prototypes are in slra_xxx.h files; the implementation is in slra_xxx.cpp files.
The function 

\texttt{slra} can deal with an arbitrary \texttt{Structure} object using the following methods:

- \texttt{getNp()}, \texttt{getM()} and \texttt{getN()} return \( n_p, m \) and \( n \), respectively.
- \texttt{fillMatrixFromP()} constructs the matrix \( S(p)^\top \) from the vector of structure parameters \( p \).
- \texttt{correctP()} constructs the correction vector \( \Delta p \) from \( R \) and precomputed \( y_r := \Gamma^{-1}(R)v(R) \).
- \texttt{createCholesky()} and \texttt{createDerivative()} create objects for Cholesky factorization and gradient/Jacobian computation respectively.

The currently implemented structures are:

- \texttt{LayeredHStructure}: class for layered Hankel structure with blockwise weights,
- \texttt{WLayeredHStructure}: class for layered Hankel structure with elementwise weights, and
- \texttt{MosaicHStructure} and \texttt{WMosaicHStructure}: classes for mosaic Hankel structure, that are implemented based on layered Hankel structure.

The object-oriented paradigm facilitates the memory management and software design. In particular, it allows the user to add a new problem specification (for example, a new type of structure) by implementing a new \texttt{Structure} subclass. Further details can be found in the manual for the C++ interface, which can be generated by running \texttt{doxygen} \texttt{[37]} in the \texttt{cpp} directory.

### A.3 Installation instructions

The package is distributed in the form of source code. The interfaces for Matlab/Octave and R can be compiled by calling \texttt{make} in the root directory of the package. The GSL, BLAS, and LAPACK libraries have to be installed in advance. The SLICOT library can be used optionally.

### B Optional input arguments for the optimization method

The following optional input arguments of the \texttt{slra} function are related to the optimization solver, used for the solution of the parameter optimization problem. The options can be specified either as fields of the input argument \texttt{opt} or, when calling via the wrapper function, as pairs (\texttt{‘name’, ‘value’}) of input arguments.

- \texttt{solver} specifies the solver. The options are:
  - \texttt{c} — efficient C++ solver (default), and
  - \texttt{m} — general but inefficient solver, implemented in Matlab.
• method specifies the optimization method to be used with the C++ solver. The available options are all local optimization methods in the GSL library (see the GSL manual [6] for more details):

  - l — Levenberg–Marquardt methods for nonlinear least squares (default),
    * l1 — method lmder (default),
    * ls — method lmsder,
  - q — Quasi-Newton and conjugate gradient methods for nonlinear minimization with derivatives,
    * qb — method bfgs (default),
    * q2 — method bfgs2,
    * qp — method conjugate_pr,
    * qf — method conjugate_fr,
  - n — methods for nonlinear minimization without derivatives,
    * nn — method nmsimplex (default),
    * n2 — method nmsimplex2, and
    * nr — method nmsimplex2rand.

• disp specifies the level of displayed information. The options are:

  - iter — print progress information per iteration,
  - notify — in case of lack of convergence only (default), or
  - off — no display.

• The following arguments control the termination of the optimization:

  - maxiter — maximum number of iterations,
  - tol — tolerance for the change of the cost function value,
  - epsrel and epsabs — relative and absolute tolerance for the element-wise change of the optimization variables,
  - epsgrad — tolerance for the norm of the gradient.