Software package for mosaic-Hankel structured low-rank approximation

Konstantin Usevich and **Ivan Markovsky**

[*] *CRAN, Université de Lorraine, CNRS*
Campus Sciences, BP 70239
54506 Vandœuvre-lès-Nancy, France
Konstantin.Usevich@univ-lorraine.fr

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

Abstract: This paper presents the SLRA package (http://slra.github.io)—C software with interface to MATLAB, Octave, and R for solving low-rank approximation problems with the following features: mosaic Hankel structured approximating matrix, weighted 2-norm approximation criterion, and fixed and missing elements in the approximating matrix. The package has applications in system identification, machine learning, and computer algebra. The paper gives an overview of the features of the package, including the wrapper functions for system identification (IDENT package) and approximate greatest common divisor (AGCD) computations. The addendum to the paper, available from http://homepages.vub.ac.be/~imarkovs/slra-demo, includes examples that demonstrate the usage, versatility, and efficiency of the software.

Keywords: System identification; low-rank approximation; mosaic Hankel matrix; missing data; approximate greatest common factor; software.

1. INTRODUCTION

Structured low-rank approximation is a core problem in system identification, machine learning, and computer algebra. Despite of the huge diversity of problem formulations, the following common in these application areas problem can be extracted.

> *Given* measured data, prior knowledge about the data generating system, and approximation criterion, *find* optimal in the specified sense approximation of the measured data that is consistent with the given prior knowledge.

A key observation (see Markovsky (2008, 2012, 2014)) is that in system identification, machine learning, and computer algebra the prior knowledge about the data can be expressed as a rank constraint on a structured matrix constructed from the data. Then, the common problem stated informally above becomes the following structured low-rank approximation problem.

- *The measured data* is an n_p-dimensional real vector p.
- *The prior knowledge* is
$$\operatorname{rank}\big(\mathscr{S}(p)\big) \leq r, \quad \text{where } \mathscr{S} : \mathbb{R}^{n_p} \to \mathbb{R}^{m\times n},$$
is the matrix structure.
- *The approximation criterion* is a weighted 2-norm
$$\|\Delta p\|_W := \sqrt{\Delta p^\top W \Delta p},$$
of the approximation error vector $\Delta p := p - \widehat{p}$, where W is an $n_p \times n_p$ positive semidefinite matrix.
- *The structured low-rank approximation problem* is

$$\begin{array}{ll} \text{minimize} & \text{over } \widehat{p} \quad \|p-\widehat{p}\|_W \\ \text{subject to} & \operatorname{rank}\big(\mathscr{S}(\bar{p})\big) \leq r. \end{array} \tag{SLRA}$$

Due to the rank constraint, problem (SLRA) is nonconvex. Except for a few notable exceptions (unstructured approximation in the spectral or Frobenius norm, rank-1 weight matrix W, and circulant structure) there are no polynomial time global solution methods for (SLRA). State-of-the-art methods split into convex relaxation and local optimization heuristics. In the special case of (generalized) Hankel and Toeplitz structured matrices, the class of subspace methods Van Overschee and De Moor (1996); Verhaegen and Verdult (2007) provides also an effective and efficient heuristic for structured low-rank approximation.

This paper presents a local optimization-based software package for mosaic Hankel structured low-rank approximation, hosted at

https://github.com/slra/slra/

The SLRA package is built upon an implementation of the following methods for structured low-rank approximation: (i) variable projection (kernel representation) Markovsky (2014); Markovsky and Usevich (2014, 2013); Usevich and Markovsky (2014b) and (ii) penalisation (image representation) Ishteva et al. (2014). In the variable projection approach, the problem is recast as optimization on a Grassmann manifold Usevich and Markovsky (2014a). The package includes wrapper functions for (a) system identification (the IDENT package) described in Markovsky (2013) and (b) AGCD computations Usevich and Markovsky (2017).

We give a comprehensive overview of the features implemented in the package. The previous publication Markovsky and Usevich (2014) is outdated, as it takes into account only the methods of Markovsky and Usevich (2014, 2013); Usevich and Markovsky (2014b)). Here, we describe, in addition, the methods of Usevich and Markovsky (2014b); Ishteva et al. (2014); Usevich and Markovsky (2014a); Markovsky (2013); Usevich and Markovsky (2017). Thus, the paper gathers the description of the parameters and options of the methods scattered in the literature.

The paper is organized as follows. Section 2 defines the problem solved by the package. Section 3 describes the solvers included in the package and their options. Section 4 presents a MATLAB wrapper function for using the SLRA package for linear time-invariant system identification. Section 5 describes the wrapper functions for AGCD computations. An addendum to the paper, available from `http://homepages.vub.ac.be/~imarkovs/slra-demo`, contains numerical examples of using the software.

2. PROBLEM FORMULATION

The SLRA package solves problem (SLRA). Typical code is

```
[ph, info] = slra(p, s, r, opt)
```

where `p` is the data vector, `s` encodes the structure and the weighted norm, and `r` is the rank of the approximation.

In the following subsection, we give more details on the structure specification and the weighted 2-norm (contained in the parameter `s`). The parameter `opt` determines the solver being used and contains the parameters of the solvers (see Section 3 for more details).

The output parameter `ph` contains the computed approximation $\widehat{p}$. The structure `info` contains additional output information from the optimization, such as:

- `info.fmin`: the value of the cost function $\|p-\widehat{p}\|_W^2$;
- `info.iter`: number of iterations;
- `info.time`: execution time;
- `info.Rh`: low-rank certificate—a full row rank matrix $\widehat{R} \in \mathbb{R}^{(m-r)\times m}$, such that

$$\widehat{R}\mathscr{S}(\widehat{p}) = 0. \qquad \text{(KER)}$$

Additional information may be returned in `info` for some of the solvers.

2.1 Structure specification

The SLRA package deals with affine structures, *i.e.*, structured matrices that form an affine subset of $\mathbb{R}^{m\times n}$.

The package allows for the following matrix structures:

- *General affine structures.*

$$\mathscr{S}(p) = \Phi(S_0 + \sum_{k=1}^{n_p} p_k S_k), \qquad (\mathscr{S})$$

where $S_k \in \mathbb{R}^{m'\times n}$ are basis matrices and $\Phi \in \mathbb{R}^{m\times m'}$ is a full row rank matrix (an identity matrix by default). Moreover, S_k must be orthogonal (*i.e.*, $\langle S_k, S_j\rangle = \operatorname{trace}(S_k S_j^\top) = 0$ for $k \neq j$) and consist only of zeros and ones. The latter condition ensures that the linear part (*i.e.*, $\sum_{k=1}^{n_p} p_k S_k$) contains in each entry either zero, or an element of the vector p, so it can be compactly represented as an integer matrix of the indices of the elements of p.

For example, the matrix structure

$$\mathscr{S}(p) = \begin{bmatrix} 4 & 5 & 6 & p_1 \\ 5 & 6 & p_1 & p_2 \\ 6 & p_1 & p_2 & p_3 \end{bmatrix}.$$

can be specified as follows:

```
s.S0  = [4 5 6 0; ...
         5 6 0 0; ...
         6 0 0 0];
s.tts = [0 0 0 1; ...
         0 0 1 2; ...
         0 1 2 3];
```

An example for the matrix Φ will be given next.

- *Mosaic Hankel structures*

The mosaic Hankel Heinig (1995) structure

$$\mathscr{H}_{\mathbf{m},\mathbf{n}}(p) := \begin{bmatrix} \mathscr{H}_{m_1,n_1}(p^{(1,1)}) & \cdots & \mathscr{H}_{m_1,n_N}(p^{(1,N)}) \\ \vdots & & \vdots \\ \mathscr{H}_{m_q,n_1}(p^{(q,1)}) & \cdots & \mathscr{H}_{m_q,n_N}(p^{(q,N)}) \end{bmatrix},$$

is a generalization of the classical Hankel structure

$$\mathscr{H}_{m,n}(p) := \begin{bmatrix} p_1 & p_2 & p_3 & \cdots & p_n \\ p_2 & p_3 & \iddots & & p_{n+1} \\ p_3 & \iddots & & & \vdots \\ \vdots & & & & \\ p_m & p_{m+1} & \cdots & & p_{m+n-1} \end{bmatrix}.$$

A mosaic Hankel matrix $\mathscr{H}_{m,n}(p)$ is a block matrix with scalar Hankel blocks. The vectors

$$\mathbf{m} := [m_1 \ \cdots \ m_q] \quad \text{and} \quad \mathbf{n} := [n_1 \ \cdots \ n_N]$$

define the sizes of the scalar Hankel blocks and therefore uniquely specify the structure. The parameter vector $p \in \mathbb{R}^{n_p}$ is the concatenation of chunks $p^{(i,j)}$

$$p = (p^{(1,1)}, \ldots, p^{(q,1)}, \ldots, p^{(1,N)}, \ldots, p^{(q,N)}),$$

where $p^{(i,j)} \in \mathbb{R}^{m_i+n_j-1}$.

The software package supports mosaic-Hankel-like structures of the form

$$\mathscr{S}(p) := \Phi\mathscr{H}_{\mathbf{m},\mathbf{n}}(p), \qquad (\Phi\mathscr{H}_{\mathbf{m},\mathbf{n}})$$

where Φ is a full row rank matrix, further extending the class of mosaic Hankel matrices to (mosaic) Hankel-like matrices.

For example, a Toeplitz matrix

$$\begin{bmatrix} p_3 & p_4 & p_5 \\ p_2 & p_3 & p_4 \\ p_1 & p_2 & p_3 \end{bmatrix}.$$

can be specified as follows:

```
s.m = 3; s.n = 3; s.Phi = [0 0 1; ...
                           0 1 0; ...
                           1 0 0];
```

A Toeplitz-plus-Hankel matrix can be defined in a similar way, as shown in Markovsky and Usevich (2014).

2.2 Approximation criterion

The approximation criteria supported by the package are weighted 2-norms $\|\cdot\|_W$, with two types of W.

- A diagonal weight matrix $W = \operatorname{diag}(w)$, where w is a vector with nonnegative real elements from 0 to ∞. Weight $w_i = 0$ corresponds to treating p_i as a missing value. In the other extreme $w_i = \infty$, $\widehat{p}_i$ should match the data p_i (an equality constraint in the optimization problem (SLRA)).

In general, the weights are specified by a given vector `s.w` of length n_p. For mosaic-Hankel-like structure, block-wise weights can be specified in two ways:
  - if `s.w` is of length qN, then the values are the weights for the whole chunks $p^{(i,j)}$ in the vectorisation order;
  - if `s.w` is of length q, the value of the i-th element of `s.w` is a weight for $p^{(i,j)}$ for every j.
- Another option is to use a general symmetric positive semi-definite matrix $W \in \mathbb{R}^{n_p \times n_p}$.

3. SLRA PACKAGE: SOLVERS AND OPTIONS

This section gives an overview of methods implemented in the SLRA package. The package contains:

(1) fast C++ implementation of the variable projection (VARPRO) method for mosaic Hankel matrices Usevich and Markovsky (2014b);
(2) an implementation of the VARPRO method for SLRA with missing data Markovsky and Usevich (2013); this method is also called "experimental Matlab solver" in Markovsky and Usevich (2014).
(3) the factorization approach to SLRA based on a penalty method Ishteva et al. (2014).

In Table 1 the supported structure and weight specification for each methods are listed.

Table 1. Structures supported by solvers.

	1)	2)	3)
`opt.solver`	`'c'`	`'m'`	`'r'`
General affine structure	–	+	+
Mosaic-Hankel-like ($\Phi\mathscr{H}_{\mathbf{m},\mathbf{n}}$)	+	+	+
elementwise weights $w_k \in (0,\infty]$	+	+	+
missing data $w_k = 0$	–	+	+
semi-definite weight matrix W	–	+	+

By default, the efficient VARPRO method is used, if a certain structure/weight specification is not supported, the method 2) is used. The user can select the method by using the field `opt.solver`.

3.1 Variable projection methods

In the variable projection Markovsky (2014) approach, the rank constraint is replaced by the equivalent constraint (KER) and problem (SLRA) is restated as an equivalent bi-level optimisation problem

$$\begin{array}{ll} \text{minimize} & \text{over } R \in \mathbb{R}^{d\times m} \quad f(R) \\ \text{subject to} & R \text{ has full row rank,} \end{array} \qquad (\text{SLRA}_R)$$

where $d = m - r$ and $f(R)$ is a solution of the (SLRA) problem for fixed kernel matrix R

$$f(R) := \min_{\widehat{p}} \|p - \widehat{p}\|^2_W \quad \text{subject to} \quad R\mathscr{S}(\widehat{p}) = 0,$$

see Usevich and Markovsky (2014b); Markovsky and Usevich (2013, 2014) for more details.

The idea of the VARPRO method is that the cost function $f(R)$ and its derivatives can be found in a closed form, and the cost function can be minimized using conventional optimisation methods, see Markovsky (2014); Markovsky and Usevich (2013).

Search space and constraints on the kernel The cost function has an invariance property

$$f(R) = f(UR), \quad \text{for any nonsingular } U \in \mathbb{R}^{d\times d}.$$

Hence, the search space is the Grassmann manifold Usevich and Markovsky (2014a).

In addition, the package allows for linear constraints on the kernel. Two types of constraints can be used.

(1) General linear constraint

$$R = \mathscr{R}'(\theta) := \operatorname{vec}_d^{-1}(\theta\Psi), \quad \text{where } \theta \in \mathbb{R}^{n_\theta},$$

and $\Psi \in \mathbb{R}^{n_\theta \times md}$, *i.e.*, it is assumed that R belongs to a linear subspace of $\mathbb{R}^{d\times m}$.
(2) Matrix-product constraint

$$R = \mathscr{R}(\theta) := \Theta\Psi, \quad \text{where } \Theta \in \mathbb{R}^{d\times m''},$$

and $\Psi \in \mathbb{R}^{m''\times m}$ is a full row rank matrix.

The matrix-product linear constraint is a special case of the general linear constraint since

$$\operatorname{vec}^\top(\Theta\Psi) = \operatorname{vec}^\top(\Theta)(\Psi \otimes I_d).$$

Note 1. Both linear constraints are supported by both variable projection methods (efficient C++ solver and experimental MATLAB solver). Note that in Markovsky and Usevich (2014) it was written that only the MATLAB solver supports the general linear constraint.

Both linear constraints are handled by giving the field `opt.Psi`. The package determines the constraint under consideration based on the size of the matrix Ψ. By default, it is assumed that $\Psi = I_m$

3.2 Efficient C++ solver and its options

The efficient C++ solver (`opt.solver = 'c'`) implements the method described in Usevich and Markovsky (2014b). It can handle mosaic Hankel structures ($\Phi\mathscr{H}_{\mathbf{m},\mathbf{n}}$) and elementwise positive weights, with a linear complexity in the size of data evaluation complexity of cost function and its derivatives (assuming m and d fixed). General structure ($\mathscr{S}$), missing data, or general weight matrices cannot be handled. The solver can use optimization algorithms implemented in the GSL (GNU Scientific Library) library GSL (2017) and own implementation of Levenberg-Marquardt method.

GSL and constraints The methods implemented in the GSL library are not designed for optimization on manifolds, hence these methods work under two assumptions:

(1) matrix-product constraint;
(2) and additional constraint

$$\Theta = [X \ -I_d], \quad \text{where } X \in \mathbb{R}^{d\times(m''-d)}. \qquad (X)$$

Note 2. The latter assumption parametrizes the Grassmann manifold minus a subset of measure zero. The whole Grassmann manifold can be considered by employing permutations (in Ψ), as explained in Usevich and Markovsky (2014a).

The employed methods are:

- `opt.method = 'n...'`: derivative-free (Nelder-Mead) methods;
- `opt.method = 'q...'`: quasi-Newton methods;
- `opt.method = 'l...'`: Levenberg-Marquardt (see the next subsection).

The second letter of the string `opt.method` determines the submethod being used. The possible submethods can be found in the documentation of the package, see,

```
cpp/OptimizationOptions.h.
```

Levenberg-Marquardt methods The Levenberg-Marquardt methods use sum of squares representation of the cost function

$$f(R) = \|g(R)\|_2^2, \tag{1}$$

where $g : \mathbb{R}^{d\times m} \to \mathbb{R}^{n_g}$ is a vector-valued map. By default, the Cholesky factorisation is used as g (the function g_s in (Usevich and Markovsky, 2014b, Alg. 6)). Another option (`opt.ls_correction = 1`) is to use the function $g = \Delta p^*(R)$ (see (Usevich and Markovsky, 2014b, p. 435)).

The LM method from the GSL library (`opt.method = 'l...'`) is applicable only to the matrix-product constraint on the kernel. The general linear constraint can be handled by an own implementation of the 'data-driven local coordinates' modification of the Levenberg-Marquardt method, described in (Usevich and Markovsky, 2014a, §3.2). This method (chosen by `opt.method = 'p...'`) automatically takes care of the linear dependencies imposed by the Grassmann manifold and of the general or matrix-product linear constraint.

Note 3. For `opt.method = 'p...'`, in the case of a matrix-product constraint, still the additional constraint (X) is used. If the constraint (X) needs to be avoided, one should specify `opt.avoid_xi = 1`.

Additional parameters The available stopping criteria are:

- `opt.maxiter`: maximum number of iterations;
- `opt.epsgrad`: lower bound on $\|\nabla f\|$
- `opt.epsabs`, `opt.epsrel`: stops if
$$\|X_i - X_{i+1}\| < \text{epsabs} + \text{epsrel}\cdot\|X_i\|.$$
- `opt.maxx`: when parameterisation (X) is used, stops if $\|X\|_\infty$ exceeds `opt.maxx`.

3.3 Experimental (MATLAB) solver

The MATLAB solver (`opt.solver = 'm'`), based on the results of Markovsky and Usevich (2013), uses general purpose optimisation routines such as `fmincon`/`fminunc`.

The constraint on R is imposed in one of the following ways:

- hard constraint (default): $RR^\top = I_d$. In that case, `fmincon` is used. Note that, in this case, the cost function is minimised on the Stiefel manifold.
- "penalty" term (if `opt.method = 'reg'`):
$$f(R) + \gamma\|RR^T - I_d\|_F^2,$$
together with the `fminunc`. In fact, as shown in Markovsky and Usevich (2013), this penalty term is exact (enforces the hard constraint $RR^\top = I_d$).

Note that:

- If the linear constraints on the kernel are present, they are substituted in $RR^\top = I_d$.
- Standard optimisation options (such as `opt.maxiter`, `opt.tolx`, etc.) can be passed in the structure `opt`.

3.4 Factorization (alternating least squares) methods

The method `opt.solver = 'r'` is an implementation of the method in Ishteva et al. (2014). (SLRA) is reformulated as

$$\min_{P\in\mathbb{R}^{m\times r}, L\in\mathbb{R}^{r\times n}} \|p - \Pi_{\mathscr{S}}\operatorname{vec}(PL)\|_W^2 + \lambda_k\|PL - P_{\mathscr{S}}(PL)\|_F^2, \tag{2}$$

where λ_k is the penalty parameter controlling how close the approximation is to the set of structured matrices.

The method proceeds by alternately minimising the cost function for fixed P and L. The hyper parameter λ_k depends on the iteration and is gradually increasing. The additional input parameters for this method are:

- `opt.P_init`: initial approximation for P (default: based on the SVD);
- `opt.lamba_init`: initial value of λ_k;
- `opt.lambda_max`: maximal value of λ_k;
- `opt.max_inner_iter`: maximal number of inner iterations (the maximal number of iterations when λ_k does not change);

Additional output parameters include:

- `info.P`, `info.L`: computed P and L matrices;
- `info.lambda`: last value of lambda;
- `info.outer_iterations`: number of of outer iterations (changes of λ_k);
- `info.inner_iterations`: number of inner iterations for each outer iteration;

4. WRAPPER FUNCTIONS FOR SYSTEM IDENTIFICATION

This section describes the functions of the IDENT package Markovsky (2013) for linear time-invariant system identification. System identification problems can be solved via mosaic Hankel structured low-rank approximation, but a wrapper function is needed in order to convert the system identification problem to the input representation for the SLRA solver.

4.1 The identification problem

We use the behavioral language Polderman and Willems (1998), where a dynamical system is viewed as a collection of trajectories. Let $\mathscr{L}^q_{\mathtt{m},\ell}$ be the model class of linear-time invariant systems with q manifest variables (inputs and outputs) of bounded complexity (at most $\mathtt{m}$ inputs and lag at most ℓ). Given a system $\mathscr{B}\in\mathscr{L}^q_{\mathtt{m},\ell}$, time series $w_{\mathrm{d}} = \{w_{\mathrm{d}}^1,\dots,w_{\mathrm{d}}^N\}$, and weights $v = \{v^1,\dots,v^N\}$, the lack of fit (misfit) between the model $\mathscr{B}$ and the data w_{d} is defined as

$$M(w_{\mathrm{d}},\mathscr{B}) := \min_{\widehat{w}^1,\dots,\widehat{w}^N\in\mathscr{B}} \sqrt{\sum_{k=1}^N \|w_{\mathrm{d}}^k - \widehat{w}^k\|_{v^k}^2},$$

Note that $\widehat{w}^i$ is the projection of w_{d}^i on $\mathscr{B}$.

The identification problem solved by the IDENT package is: given a (set of) time series w_{d}, complexity specification $(\mathtt{m},\ell)$, and weights v, find the model that minimizes the misfit criterion over all models with bounded complexity:

$$\widehat{\mathscr{B}} := \arg\min_{\mathscr{B}\in\mathscr{L}^q_{\mathtt{m},\ell}} M(w_{\mathrm{d}},\mathscr{B}). \tag{SYSID}$$

As shown in (Markovsky, 2008, Section 3.1) (SYSID) yields the maximum likelihood estimator in the errors-in-variables setting Söderström (2007)

$$w_{\mathrm{d}}^k = \bar{w}^k + \tilde{w}^k,$$

where the true data $\bar{w}^k$ is a trajectory of a true model $\bar{\mathscr{B}}\in\mathscr{L}^q_{\mathtt{m},\ell}$ and the measurement noise $\tilde{w}^k$ is zero mean normally

distributed measurement noise with variance $\sigma^2(v^k)^{-1}$, *i.e.*, up to a scaling factor, the weights v are the inverse of the noise covariance. Under additional mild assumptions, the estimator $\widehat{\mathscr{B}}^*$ is consistent and the parameters have asymptotically normal joint distribution Pintelon and Schoukens (2001); Kukush et al. (2005).

4.2 Functions of the package

The function `ident` solves the approximate identification problem (SYSID), and `misfit` computes the misfit $M(w_{\mathtt{d}}, \mathscr{B})$. They implement the following mappings:

`ident`: $(w_{\mathtt{d}}, \mathtt{m}, \ell) \mapsto \widehat{\mathscr{B}}$, where $\widehat{\mathscr{B}}$ (specified by either input/state/output or kernel representation) is a locally optimal solution of (SYSID)

```
[sysh, info, wh, xini] = ...
                  ident(w, m, ell, opt)
```

`misfit`: $\left(w_{\mathtt{d}}, \mathscr{B}\right) \mapsto (M, \widehat{w})$, where M is the misfit between $\mathscr{B}$ (specified by either input/state/output or kernel representation) and $w_{\mathtt{d}}$, and $\widehat{w}$ is the optimal approximation of $w_{\mathtt{d}}$ within $\mathscr{B}$ (the smoothed trajectory)

```
[M, wh, xini] = misfit(w, sysh, opt)
```

Main parameters:

- `w` is the given set of time series $w_{\mathtt{d}}$ — a real MATLAB array of dimension $T \times q \times N$, where T is the number of samples, q is the number of variables, and N is the number of time series. In case of multiple experiments of different duration, `w` should be specified as a cell array with N cells, each one of which is a $T_i \times q$ matrix containing the ith time series, *i.e.*,
$$\mathtt{w}(\mathtt{t}, :, \mathtt{k}) = \left(w^k(t)\right)^\top \quad \text{or} \quad \mathtt{w}\{\mathtt{k}\}(\mathtt{t}, :) = \left(w^k(t)\right)^\top.$$
- (`m`, `ell`) is the complexity specification (input dimension `m` and lag ℓ).
- `sysh` is an input/state/output representation of the identified or validated system $\widehat{\mathscr{B}}$, given by an `ss` object, or a parameter $\widehat{R}$ of a kernel representation(KER) if `opt.ss` is set to zero.
- `info` is a structure, containing output information from the structured low-rank approximation solver: `info.M` is the misfit $M(w_{\mathtt{d}}, \widehat{\mathscr{B}})$, `info.time` is the execution time, and `info.iter` is the number of iterations.
- `M` is the misfit $M(w_{\mathtt{d}}, \widehat{\mathscr{B}})$.
- `wh` is the optimal approximating set of time series $\widehat{w}$.
- `xini` is the initial condition for `wh`.

4.3 Optional parameters

`opt` is an optional argument specifying exact variables, exact initial conditions, and options for the optimization solver, used by the `ident` function. The options are passed to the functions `ident` and `misfit` as fields of a structure.

- `'exct'` (default value `[]`) — q-dimensional vector or N-dimensional cell array with q-dimensional vector elements, specifying the indices of the exact variables.
- `'wini'` — specifies exact initial conditions. If `wini = 0`, exact zero initial conditions are specified, *i.e.*, $\mathrm{col}(0, \widehat{w}^k) \in \widehat{\mathscr{B}}$, More generally, $\mathtt{wini} = w_{\mathrm{ini}}$ is an ℓ samples long trajectory (specified by an $\ell \times q \times N$ array or a N-dimensional cell array of $\ell \times q$ matrices), defining initial conditions for the time series $\widehat{w}$, *i.e.*, $\mathrm{col}(w^k_{\mathrm{ini}}, \widehat{w}^k) \in \widehat{\mathscr{B}}$.
- `'sys0'` — initial approximation: an input/state/output representation of a system, given as an `ss` object, with `m` inputs, $\mathtt{p} := q - \mathtt{m}$ outputs, and order $\mathtt{n} := \ell\mathtt{p}$; or specified by a minimal kernel parameter, *i.e.*, a $\mathtt{p} \times (\ell+1)q$ matrix Default value is computed by the `slra` function, using unstructured low-rank approximation.
- Arguments allowing the user to specify different optimization algorithms (`'solver'` and `'method'`), control the displayed information (`'disp'`), and change the convergence criteria, see Section 3.
- `'ss'` set to zero disables the conversion of $\widehat{R}$ to the state-space representation $(\widehat{A}, \widehat{B}, \widehat{C}, \widehat{D})$.
- `'n'` — order of the identified model. For multiple output system ($\mathtt{p} > 1$), the order of the identified model $\widehat{\mathscr{B}} \in \mathscr{L}_{\mathtt{m},\ell}$ is a multiple of the number of outputs $\mathtt{n} = \mathtt{p}\ell$. A model of order $\mathtt{p}(\ell-1) < \mathtt{n} < \mathtt{p}\ell$ can be specified by the optinal parameter `opt.n`. In this case, the IDENT package does balanced model order reduction of the identified model $\widehat{\mathscr{B}}$ to obtained a model of order `n`.

5. APPROXIMATE GREATEST COMMON DIVISOR COMPUTATIONS

The problem of finding the AGCD of several polynomials appears, for example, in blind deblurring and distance to controllability problems. As shown in Usevich and Markovsky (2017), the AGCD problem can be reformulated as a mosaic-Hankel structured low-rank approximation.

Let $\mathscr{P}_n$ denote the space of (real or complex) polynomials of degree n. The AGCD problem solved by the package is the following. Given N polynomials $p_1 \in \mathscr{P}_{n_1}, \ldots, p_N \in \mathscr{P}_{n_N}$ of and degree d, find the approximating polynomials, which have a common divisor of degree at least d:

$$\text{minimize} \quad \text{over } \widehat{p}_1 \in \mathscr{P}_{n_1}, \ldots, \widehat{p}_N \in \mathscr{P}_{n_N} \quad \sum_{k=1}^{N} \|p_k - \widehat{p}_k\|^2_{v_k}$$
$$\text{subject to} \quad \deg \gcd(\widehat{p}_1, \ldots, \widehat{p}_N) \geq d,$$

where $\|\cdot\|_v$ is a weighted norm on the coefficients of the polynomial. Note that $\deg\gcd(\widehat{p}_1, \ldots, \widehat{p}_N) \geq d$ if there exists a polynomial of $\widehat{h}$ degree d (common divisor) such that

$$\widehat{p}_1 = \widehat{g}_1\widehat{h}, \quad \ldots, \quad \widehat{p}_N = \widehat{g}_N\widehat{h}.$$

The solvers are contained in a subdirectory `agcd` of the SLRA package. The common interface for calling these functions is

```
[ph, info] = gcd_xxx(p, v, d, opt)
```

where the input arguments are:

- `p` is a cell array containing vectors of coefficients of polynomials p_k; the coefficients are given from lowest to highest degree of the monomials;
- `v` is a cell array of the weight vectors of corresponding length;
- `d` is the degree of the GCD;
- `opt` is the structure containing options, which are passed to the `slra` solver; also the starting point for local optimization can be given by either:
  - `opt.hini`: the initial value for $\widehat{h}$,
  - `opt.gini`: the cell array of initial values for $\widehat{g}_k$.

and the output arguments are:

- `ph` is a cell array of approximating polynomials;
- `info` as for the `slra` solver.

The implemented optimization methods are:

- `gcd_nls`: uses variable projection representation with respect to $\widehat{h}$, handles real polynomials;
- `gcd_nls_complex`: uses variable projection representation with respect to $\widehat{h}$, handles complex polynomials;
- `gcd_cofe`: uses variable projection representation with respect to $\widehat{g}_k$, handles real polynomials;
- `gcd_syl`: uses Sylvester subresultant low-rank approximation, handles real polynomials.

The default initial approximation is computed from a kernel of a Sylvester subresultant matrix. There is alternative way to compute the initial approximation using the matrix pencil, which is invoked by the function `h_ini_mp`. For more details on these methods, see Usevich and Markovsky (2017).

ACKNOWLEDGEMENTS

The authors would like to thank Mariya Ishteva, who contributed to the development of the factorization-based solver for structured low-rank approximation. 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", Fund for Scientific Research (FWO-Vlaanderen), FWO projects G028015N "Decoupling multivariate polynomials in nonlinear system identification" and G090117N "Block-oriented nonlinear identification using Volterra series"; and the FWO/F.R.S.-FNRS Excellence of Science project number 30468160 "Structured low-rank matrix / tensor approximation: numerical optimization-based algorithms and applications".

REFERENCES

(2017). GSL — GNU Scientific Library. URL `www.gnu.org/software/gsl/`.

Absil, P.A., Mahony, R., and Sepulchre, R. (2008). *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, NJ.

Absil, P.A., Mahony, R., Sepulchre, R., and Dooren, P.V. (2002). A Grassmann–Rayleigh quotient iteration for computing invariant subspaces. *SIAM Review*, 44(1), 57–73.

Buckheit, J. and Donoho, D. (1995). *Wavelets and statistics*, chapter Wavelab and reproducible research. Springer-Verlag.

De Moor, B., Gersem, P.D., Schutter, B.D., and Favoreel, W. (1997). DAISY: A database for identification of systems. *Journal A*, 38(3), 4–5.

Dominik, C. (2010). *The org mode 7 reference manual*. Network theory ltd. URL `http://orgmode.org/`.

Golub, G. and Pereyra, V. (2003). Separable nonlinear least squares: the variable projection method and its applications. *Institute of Physics, Inverse Problems*, 19, 1–26.

Heinig, G. (1995). Generalized inverses of Hankel and Toeplitz mosaic matrices. *Linear Algebra Appl.*, 216(0), 43–59.

Ishteva, M., Usevich, K., and Markovsky, I. (2014). Factorization approach to structured low-rank approximation with applications. *SIAM J. Matrix Anal. Appl.*, 35(3), 1180–1204.

Knuth, D. (1992). *Literate programming*. Cambridge University Press.

Kukush, A., Markovsky, I., and Van Huffel, S. (2005). Consistency of the structured total least squares estimator in a multivariate errors-in-variables model. *J. Statist. Plann. Inference*, 133(2), 315–358.

Ljung, L. (2013). *System identification toolbox: User's guide*. The MathWorks.

Markovsky, I. (2008). Structured low-rank approximation and its applications. *Automatica*, 44(4), 891–909.

Markovsky, I. (2012). *Low Rank Approximation: Algorithms, Implementation, Applications*. Springer.

Markovsky, I. (2013). A software package for system identification in the behavioral setting. *Control Eng. Practice*, 21(10), 1422–1436.

Markovsky, I. (2014). Recent progress on variable projection methods for structured low-rank approximation. *Signal Processing*, 96PB, 406–419.

Markovsky, I. and Usevich, K. (2013). Structured low-rank approximation with missing data. *SIAM J. Matrix Anal. Appl.*, 34(2), 814–830.

Markovsky, I. and Usevich, K. (2014). Software for weighted structured low-rank approximation. *J. Comput. Appl. Math.*, 256, 278–292.

Marmorat, J.P. and Olivi, M. (2012). RARL2 software (realizations and rational approximation in L_2 norm).

Nelder, J.A. and Mead, R. (1965). A simplex method for function minimization. *Computer J.*, 7, 308–313.

Pintelon, R. and Schoukens, J. (2001). *System Identification: A Frequency Domain Approach*. IEEE Press, Piscataway, NJ.

Polderman, J. and Willems, J.C. (1998). *Introduction to Mathematical Systems Theory*. Springer-Verlag, New York.

Ramsey, N. (1994). Literate programming simplified. *IEEE Software*, 11, 97–105.

Söderström, T. (2007). Errors-in-variables methods in system identification. *Automatica*, 43, 939–958.

Taylor, C., Pedregal, D., Young, P., and Tych, W. (2007). Environmental time series analysis and forecasting with the CAPTAIN toolbox. *Env. Modelling&Software*, 22, 797–814.

Usevich, K. and Markovsky, I. (2014a). Optimization on a Grassmann manifold with application to system identification. *Automatica*, 50, 1656–1662.

Usevich, K. and Markovsky, I. (2014b). Variable projection for affinely structured low-rank approximation in weighted 2-norms. *J. Comput. Appl. Math.*, 272, 430–448.

Usevich, K. and Markovsky, I. (2017). Variable projection methods for approximate (greatest) common divisor computations. *Theoretical Computer Science*.

Van Huffel, S., Sima, V., Varga, A., Hammarling, S., and Delebecque, F. (2004). High-performance numerical software for control. *IEEE Control Systems Magazine*, 24, 60–76.

Van Overschee, P. and De Moor, B. (1996). *Subspace identification for linear systems: Theory, implementation, applications*. Kluwer, Boston.

Vandewalle, P., Kovacevic, J., and Vetterli, M. (2009). Reproducible research in signal processing - what, why, and how. *IEEE Signal Proc. Magazine*, 26(3), 37–47.

Verhaegen, M. and Verdult, V. (2007). *Filtering and system identification: a least squares approach*. Cambridge university press.