Chapter 3
Algorithms

Writing a book is a little more difficult than writing a technical paper, but writing software is a lot more difficult than writing a book.

D. Knuth

Summary: A few special structured low rank approximation problems have analytic solutions, however in general the structured low rank approximation problem is NP-hard. There are three conceptually different approaches for solving it: convex relaxations, local optimization, and global optimization. System realization methods and methods based on local optimization and convex relaxation using the nuclear norm heuristic are presented. The local optimization and convex relaxation method can deal with general affine structure and approximation norm, and allow regularization and inequality constraints on the approximation.

The latest version of the code presented is available from the book's web page.

3.1 Subspace methods

The singular value decomposition is at the core of many algorithms for approximate modeling, most notably the methods based on balanced model reduction, the subspace identification methods, and the MUSIC and ESPRIT methods in signal processing. The reason for this is that the singular value decomposition is a robust and efficient way of computing unstructured low rank approximation of a matrix in the Frobenius norm. In system identification, signal processing, and computer algebra, however, the low rank approximation is restricted to the class of matrices with specific (Hankel, Toeplitz, Sylvester) structure. Ignoring the structure constraint renders the singular value decomposition-based methods suboptimal with respect to a desired optimality criterion.

Except for the few special cases described in Section 2.5 there are no global solution methods for general structured and weighted low rank approximation problems. The singular value decomposition based methods can be seen as relaxations of the original NP-hard structured weighted low rank approximation problem, obtained by

removing the structure constraint and using the Frobenius norm in the approximation criterion. Another approach is taken in Section 3.3, where convex relaxations of the related rank minimization problem are proposed. Convex relaxation methods give polynomial time suboptimal solutions and are shown to provide globally optimal solutions in certain cases.

Presently there is no uniformly best method for computing suboptimal structured low rank approximation. In the context of system identification (*i.e.*, block-Hankel structured low rank approximation) subspace and local optimization based methods have been compared on practical data sets. In general, the heuristic methods are faster but less accurate than the methods based on local optimization. It is a common practice to use a suboptimal solution obtained by a heuristic method as an initial approximation for an optimization based method. Therefore, the two approaches complement each other.

Realization algorithms

The aim of the realization algorithms is to compute a state space representation $\mathscr{B}_{\text{i/s/o}}(A,B,C,D)$ of the minimal realization $\mathscr{B}_{\text{mpum}}(H)$ of H, see Section 2.2. Finding the model parameters A,B,C,D can be done by computing a rank revealing factorization of the Hankel matrix.

Let

$$\mathscr{H}_{\mathtt{n}_{\max}+1,\mathtt{n}_{\max}+1}(\sigma H) = \Gamma\Delta, \quad \text{where} \quad \Gamma \in \mathbb{R}^{\mathtt{p}(\mathtt{n}_{\max}+1)\times \mathtt{n}} \text{ and } \Delta \in \mathbb{R}^{\mathtt{n}\times \mathtt{m}(\mathtt{n}_{\max}+1)}$$

be a rank revealing factorization of the finite Hankel matrix $\mathscr{H}_{\mathtt{n}_{\max}+1,\mathtt{n}_{\max}+1}(\sigma H)$. The Hankel structure implies that the factors Γ and Δ are observability and controllability matrices, *i.e.*, there are matrices $A \in \mathbb{R}^{\mathtt{n}\times\mathtt{n}}$, $C \in \mathbb{R}^{\mathtt{p}\times\mathtt{n}}$, and $B \in \mathbb{R}^{\mathtt{n}\times\mathtt{m}}$, such that

$$\Gamma = \mathscr{O}_{\mathtt{n}_{\max}+1}(A,C) \qquad \text{and} \qquad \Delta = \mathscr{C}_{\mathtt{n}_{\max}+1}(A,B).$$

Then $\mathscr{B}_{\text{i/s/o}}(A,B,C,D)$ is the minimal realization of H.

A rank revealing factorization is not unique. For any $\mathtt{n}\times\mathtt{n}$ nonsingular matrix T, we obtain a new factorization

$$\mathscr{H}_{\mathtt{n}_{\max}+1,\mathtt{n}_{\max}+1}(\sigma H) = \Gamma\Delta = \underbrace{\Gamma T}_{\Gamma'}\underbrace{T^{-1}\Delta}_{\Delta'}$$

with the same inner dimension. The nonuniqueness of the factorization corresponds to the nonuniqueness of the input/state/output representation of the minimal realization due to a change of the state space bases:

$$\mathscr{B}_{\text{i/s/o}}(A,B,C,D) = \mathscr{B}_{\text{i/s/o}}(T^{-1}AT, T^{-1}B, CT, D).$$

The structure of the observability and controllability matrices is referred to as the *shift structure*. The parameters B and C of an input/state/output representation

of the realization are directly available from the first block elements of Γ and Δ, respectively. The parameter A is computed from the overdetermined system of linear equations

$$\sigma^{-1}\Gamma A = \sigma\Gamma \qquad \text{or} \qquad A\sigma^{-1}\Delta = \sigma\Delta. \tag{$*$}$$

Acting on a block matrix, σ and σ^{-1} remove the first and last block elements, respectively.

77a ⟨$\Gamma,\Delta) \mapsto (A,B,C)$ 77a⟩≡
```
a = O(1:end - p, :) \ O((p + 1):end, :);
b = C(:, 1:m); c = O(1:p, :);
```

Implementation

As square as possible Hankel matrix $\mathscr{H}_{i,j}(\sigma H)$ is formed, using all data points, *i.e.*,

$$i = \lceil T/2 \rceil \qquad \text{and} \qquad j = T - i + 1.$$

The key computational step of the realization algorithm is the factorization of the Hankel matrix. In particular, this step involves rank determination. In finite precision arithmetic, however, rank determination is a nontrivial problem. A numerically reliable way of computing rank is the singular value decomposition

$$\mathscr{H}_{\lceil T/2 \rceil}(\sigma H) = U\Sigma V^\top.$$

77b ⟨*singular value decomposition of* $\mathscr{H}_{\lceil T/2 \rceil}$ 77b⟩≡
```
[U, S, V] = svd(blkhank(h(:, :, 2:end), ceil(T / 2)), 0);
```
Uses `blkhank` 25b.

The order `n` of the realization is theoretically equal to the number of nonzero singular values and in practice is estimated as the number of singular values above a user specified tolerance.

77c ⟨*order selection* 77c⟩≡
```
s = diag(S); if ~exist('n', 'var'), n = sum(s > tol); end
```

Defining the partitioning

$$U =: \overset{\mathtt{n}}{\begin{bmatrix} U_1 & U_2 \end{bmatrix}}, \quad \Sigma =: \overset{\mathtt{n}}{\begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix}}_{\mathtt{n}}, \quad \text{and} \quad V =: \overset{\mathtt{n}}{\begin{bmatrix} V_1 & V_2 \end{bmatrix}},$$

the factors Γ and Δ of the rank revealing factorization are chosen as follows

$$\Gamma := U_1\sqrt{\Sigma_1} \qquad \text{and} \qquad \Delta := \sqrt{\Sigma_1}V_1^\top.$$

77d ⟨*rank revealing factorization of* $\mathscr{H}_{\lceil T/2 \rceil}$ 77d⟩≡
```
sqrt_s = sqrt(s(1:n))';
O =  sqrt_s(ones(size(U, 1), 1), :) .* U(:, 1:n);
C = (sqrt_s(ones(size(V, 1), 1), :) .* V(:, 1:n))';
```


This choice leads to a *finite-time balanced* realization of $\mathscr{B}_{\text{i/s/o}}(A,B,C,D)$, *i.e.*, the finite time controllability and observability Gramians

$$\mathscr{O}_t^\top(A,C)\mathscr{O}_t(A,C) = \Gamma^\top\Gamma \qquad \text{and} \qquad \mathscr{C}_t(A,B)\mathscr{C}_t^\top(A,B) = \Delta\Delta^\top$$

are equal,

$$\Gamma^\top\Gamma = \Delta\Delta^\top = \Sigma.$$

Note 3.1 (Kung's algorithm). The combination of the described realization algorithm with the singular value decomposition based rank revealing factorization, *i.e.*, unstructured low rank approximation, is referred to as *Kung's algorithm*.

78a ⟨$H \mapsto \mathscr{B}_{\text{i/s/o}}(A,B,C,D)$ 78a⟩≡
```
function [sys, hh] = h2ss(h, n, tol)
⟨default tolerance tol 40b⟩
⟨reshape H and define m, p, T 78b⟩
⟨singular value decomposition of H_⌈T/2⌉ 77b⟩
⟨order selection 77c⟩
⟨rank revealing factorization of H_⌈T/2⌉ 77d⟩
⟨Γ,Δ) ↦ (A,B,C) 77a⟩
d = h(:, :, 1);  sys = ss(a, b, c, d, -1);
if nargout > 1, hh = shiftdim(impulse(sys, T), 1); end
```
Defines:
`h2ss`, used in chunks 78, 81, 78a, 100, 78a, 109c, 78a, 110a, 114a, 78a, 125a, and 78a.

78b ⟨*reshape H and define* `m`, `p`, T 78b⟩≡
```
if length(size(h)) == 2
  [p, T] = size(h); if p > T, h = h'; [p, T] = size(h); end
  h = reshape(h, p, 1, T);
end
[p, m, T] = size(h);
```

When H is not realizable by a linear time-invariant system of order less than $\mathtt{n}_{\max}$, *i.e.*, $\mathscr{H}_{\mathtt{n}_{\max}+1}(\sigma H)$ is full rank, `h2ss` finds an approximate realization of order $\mathtt{n}_{\max}$. The link between the realization problem and Hankel structured low rank approximation implies that

> Kung's algorithm, implemented in the function `h2ss`, is a suboptimal method for Hankel structured low rank approximation.

The structured low rank approximation of $\mathscr{H}_{\mathtt{n}_{\max}+1}(\sigma H_{\text{d}})$ is $\mathscr{H}_{\mathtt{n}_{\max}+1}(\sigma\widehat{H})$, where $\widehat{H}$ is the impulse response of the approximate realization $\mathscr{B}_{\text{i/s/o}}(A,B,C,D)$ computed by Kung's algorithm.

The reason for the suboptimality of Kung's method is that unstructured instead of Hankel structured low rank approximation low rank approximation is used. Unless the data is exact, the factors Γ and Δ in the unstructured low rank factorization are not extended obervability and controllability matrices, respectively. As a result, the shift equation ($*$) has no solution. Then Kung's algorithm is computing an approximate solution in the least squares sense. Note that when a unique solution exists, the

code in chunk **??** computes it, and when a solution does not exist, the code computes a least squares approximate solution.

The two levels of approximation:

1. approximation of the Hankel matrix of the data by unstructured low rank matrix, and
2. computation of an approximate solution for the parameter estimate by the ordinary least squares method

are common to all subspace methods. In contrast, methods based on Hankel structured low rank approximation do not involve approximatation on the second stage for model parameter computation because by construction an exact solution (the parameter estimate) exists.

Computation of the impulse response from general trajectory

In the realization problem the given data is a special trajectory of the to-be-realized system—its impulse response. Thus, the realization problem is a special exact identification problem. In this section, the general exact identification problem

$$w_{\mathrm{d}} \xrightarrow{\text{exact identification}} \mathscr{B}_{\mathrm{mpum}}(w_{\mathrm{d}})$$

is solved by reducing it to realization:

$$w_{\mathrm{d}} \xrightarrow{\text{impulse response computation}} H_{\mathrm{mpum}} \xrightarrow{\text{realization}} \mathscr{B}_{\mathrm{mpum}}(w_{\mathrm{d}}).$$

First, the impulse response H_{mpum} of the most powerful unfalsified model is computed from the given general trajectory w_{d} by the method described in this section. Then, an input/state/outpu representation of the most powerful unfalsified model is computed from H_{mpum} by a realization algorithm.

The key observation in finding an algorithm for the computation of the impulse response from a trajectory w_{d} is that the image of the Hankel matrix $\mathscr{H}_t(w_{\mathrm{d}})$ constructed from the data w_{d} is the restriction $\mathscr{B}_{\mathrm{mpum}}(w_{\mathrm{d}})|_{[1,t]}$ of the most powerful unfalsified model on the interval $[1,t]$, *i.e.*,

$$\mathscr{B}_{\mathrm{mpum}}(w_{\mathrm{d}})|_{[1,t]} = \operatorname{span}\big(\mathscr{H}_t(w_{\mathrm{d}})\big). \tag{DD}$$

Therefore, any t-samples long trajectory w of $\mathscr{B}_{\mathrm{mpum}}(w_{\mathrm{d}})$ can be constructed as a linear combitation of the columns of the Hankel matrix, *i.e.*,

$$w = \mathscr{H}_t(w_{\mathrm{d}})g$$

for some vector g.

As in the realization problem, the default input/output partitioning $w = \operatorname{col}(u,y)$ is assumed. The impulse response is a matrix valued trajectory, the columns of

which are the $\mathtt{m}$ trajectories corresponding to zero initial conditions and inputs $e_1\delta$, $\ldots$, $e_{\mathtt{m}}\delta$, where e_i is the ith column of the $\mathtt{m}\times\mathtt{m}$ identity matrix and δ is the delta function. Therefore, the problem of computing the impulse response is reduced to the problem of finding a vector g_i, such that $\mathscr{H}_t(w_{\mathrm{d}})g_i$ is of the form $(e_i\delta, h_i)$, where $h_i(\tau) = 0$, for all $\tau < 0$. The identically zero input/output trajectory for negative time (in the past) implies that the response from time zero on (in the future) is the impulse response.

In order to describe the construction of a vector g_i that achieves the impulse response, define the "past" Hankel matrix

$$\mathbf{H}_{\mathrm{p}} := \mathscr{H}_{1,j}(w_{\mathrm{d}}), \qquad \text{where} \quad j := T - \mathtt{l} - t.$$

and the "future" input and output Hankel matrices

$$\mathbf{H}_{\mathrm{f},u} := \mathscr{H}_{t,j}(\sigma^{\mathtt{l}}u_{\mathrm{d}}) \qquad \text{and} \qquad \mathbf{H}_{\mathrm{f},y} := \mathscr{H}_{t,j}(\sigma^{\mathtt{l}}y_{\mathrm{d}}).$$

80a ⟨*define* $\mathbf{H}_{\mathrm{p}}$, $\mathbf{H}_{\mathrm{f},u}$, *and* $\mathbf{H}_{\mathrm{f},y}$ 80a⟩≡

```
j = T - (l + t);
Hp  = blkhank(w, l, j);
Hfu = blkhank(u(:, (l + 1):end), t, j);
Hfy = blkhank(y(:, (l + 1):end), t, j);
```

Uses `blkhank` 25b.

With these definitions, a vector g_i that achieves the impulse response h_i must satisfy the linear system of equations

$$\begin{bmatrix}\mathbf{H}_{\mathrm{p}}\\ \mathbf{H}_{\mathrm{f},u}\end{bmatrix} g_i = \begin{bmatrix} 0_{\mathtt{q}\mathtt{l}\times 1} \\ \begin{bmatrix} e_i \\ 0_{(t-1)\mathtt{m}\times 1}\end{bmatrix}\end{bmatrix}. \tag{$*$}$$

By construction, any solution g_i of the system of equations $(*)$ is such that

$$\mathbf{H}_{\mathrm{f},y}g_i = h_i.$$

Choosing the least norm solution as a particular solution and using matrix notation the first t samples of the impulse response are given by

$$H = \mathbf{H}_{\mathrm{f},y}\begin{bmatrix}\mathbf{H}_{\mathrm{p}}\\ \mathbf{H}_{\mathrm{f},u}\end{bmatrix}^{\top}\left(\begin{bmatrix}\mathbf{H}_{\mathrm{p}}\\ \mathbf{H}_{\mathrm{f},u}\end{bmatrix}\begin{bmatrix}\mathbf{H}_{\mathrm{p}}\\ \mathbf{H}_{\mathrm{f},u}\end{bmatrix}^{\top}\right)^{-1}\begin{bmatrix} 0_{\mathtt{q}\mathtt{l}\times\mathtt{m}} \\ \begin{bmatrix} I_{\mathtt{m}} \\ 0_{(t-1)\mathtt{m}\times\mathtt{m}}\end{bmatrix}\end{bmatrix}. \tag{$**$}$$

80b ⟨*data driven computation of the impulse response* 80b⟩≡

```
wini_uf = [zeros(l * ttw, m); eye(m); zeros((t - 1) * m, m)];
h_ = Hfy * pinv([Hp; Hfu]) * wini_uf;
```

80c ⟨$w \mapsto H$ 80c⟩≡

```
function h = w2h(w, m, n, t)
[ttw, T] = size(w); if ttw > T, w = w'; [ttw, T] = size(w); end
p = ttw - m; l = ceil(n / p);
u = w(1:m, :); y = w((m + 1):ttw, :);
⟨define Hp, Hf,u, and Hf,y 80a⟩
```

⟨*data driven computation of the impulse response* 80b⟩
```
for i = 1:t, h(:, :, i) = h_(((i - 1) * p + 1):(i * p), :); end
```
Defines:
w2h, used in chunks 80c and 81.

Based on the functions `w2h` and `h2ss`, we have the following method for exact identification.

81 ⟨*Most powerful unfalsified model in* $\mathscr{L}_{\mathtt{m}}^{\mathtt{q,n}}$ 81⟩≡
```
function sys = w2h2ss(w, m, n)
sys = h2ss(w2h(w, m, n, 2 * n + 2), n);
```
Defines:
w2h2ss, used in chunk 116a.
Uses h2ss 78a and w2h 80c.

3.2 Algorithms based on local optimization

Consider the structured low rank approximation problem

$$\text{minimize} \quad \text{over } \widehat{p} \quad \|p-\widehat{p}\|_W \quad \text{subject to} \quad \operatorname{rank}\bigl(\mathscr{S}(\widehat{p})\bigr) \leq r. \qquad \text{(SLRA)}$$

As discussed in Section 1.4, different methods for solving (SLRA) are obtained by choosing different combinations of

- rank parametrization, and
- optimization method.

In this section, we choose the kernel representation for the rank constraint

$$\operatorname{rank}\bigl(\mathscr{S}(\widehat{p})\bigr) \leq r \quad \Longleftrightarrow \quad \text{there is } R \in \mathbb{R}^{(m-r)\times m}, \text{ such that } R\mathscr{S}(\widehat{p}) = 0 \text{ and } RR^\top = I_{m-r}, \quad (\text{rank}_R)$$

and the variable projections approach (in combination with standard methods for nonlinear least squares optimization) for solving the resulting parameter optimization problem.

The developed method is applicable for the general affinely structured and weighted low rank approximation problem (SLRA). The price paid for the generality, however, is lack of efficiency compared to specialized methods exploiting the structure (when present) of the data matrix $\mathscr{S}(p)$ and the weight matrix W. In two special cases—single input single output linear time-invariant system identification and computation of approximate greatest common divisor—more efficient methods are developed later in the chapter. The notes and references section links the presented material to state-of-the art methods for structured low rank approximation.

Representing the constraint of (SLRA) in the kernel form (rank_R), leads to the double minimization problem

$$\begin{aligned} &\text{minimize} \quad \text{over } R \quad f(R) \quad \text{subject to} \quad RR^\top = I_{m-r}, \qquad (\text{SLRA}_R)\\ &\text{where} \quad f(R) := \min_{\widehat{p}} \|p-\widehat{p}\| \quad \text{subject to} \quad R\mathscr{S}(\widehat{p}) = 0. \end{aligned}$$

The inner minimization (computation of $f(R)$) is over the correction $\widehat{p}$ and the outer minimization is over the model parameter $R \in \mathbb{R}^{(m-r)\times m}$. The inner minimization problem can be given the interpretation of projecting the columns of $\mathscr{S}(p)$ onto the model $\mathscr{B} := \ker(R)$, for a given matrix R. However, the projection depends on the parameter R, which is the variable in the outer optimization problem.

With affine structure $\mathscr{S}$, the constraint $R\mathscr{S}(\widehat{p}) = 0$ is bilinear in the optimization variables R and $\widehat{p}$. Then, the evaluation of the cost function f for the outer minimization problem is a linear least norm problem. Direct solution has computational complexity $O(n_p^3)$, where n_p is the number of structure parameters. Exploiting the structure of the problem (inherited from $\mathscr{S}$), results in computational methods with cost $O(n_p^2)$ or $O(n_p)$, depending on the type of structure. For a class of structures including block Hankel, block Toeplitz, and block Sylvester efficient $O(n_p)$ cost function evaluation can be done by Cholesky factorization of a block-Toeplitz banded matrix.

A method for affinely structured problems

Structure specification

The general affine structure

$$\mathscr{S}(\widehat{p}) = S_0 + \sum_{k=1}^{n_p} S_k \widehat{p}_k \qquad (\mathscr{S}(\widehat{p}))$$

is specified by the matrices $S_0, S_1, \ldots, S_{n_p} \in \mathbb{R}^{m\times n}$. This data is represented in the MATLAB code by an $m \times n$ matrix variable `s0`, corresponding to the matrix S_0, and an $mn \times n_p$ matrix variable `bfs`, corresponding to the matrix

$$\mathbf{S} := \bigl[\operatorname{vec}(S_1) \cdots \operatorname{vec}(S_{n_p})\bigr] \in \mathbb{R}^{mn\times n_p}.$$

With this representation, the sum in $(\mathscr{S}(\widehat{p}))$ is implemented as a matrix–vector product:

$$\begin{aligned} &\operatorname{vec}\bigl(\mathscr{S}(\widehat{p})\bigr) = \operatorname{vec}(S_0) + \mathbf{S}\widehat{p}, \qquad (\mathbf{S})\\ &\text{or} \quad \mathscr{S}(\widehat{p}) = S_0 + \operatorname{vec}^{-1}(A\widehat{p}). \end{aligned}$$

82 ⟨$(S_0, \mathbf{S}, \widehat{p}) \mapsto \widehat{D} = \mathscr{S}(\widehat{p})$ 82⟩≡
```
dh = s0 + reshape(bfs * ph, m, n);
```

Note 3.2. In many applications the matrices S_k are sparse, so that, for efficiency, they can be stored and manipulated as sparse matrices.

A commonly encountered special case of an affine structure is

$$\big[\mathscr{S}(\widehat{p})\big]_{ij} = \begin{cases} S_{0,ij}, & \text{if } \mathtt{S}_{ij} = 0 \\ \widehat{p}_{\mathtt{S}_{ij}} & \text{otherwise} \end{cases} \qquad \text{for some} \quad \mathtt{S}_{ij} \in \{0,1,\ldots,n_p\}^{m\times n}, \qquad (\mathtt{S})$$

or, written more compactly,

$$\big[\mathscr{S}(\widehat{p})\big]_{ij} = S_{0,ij} + \widehat{p}_{\mathrm{ext},\mathtt{S}_{ij}}, \qquad \text{where} \quad \widehat{p}_{\mathrm{ext}} := \begin{bmatrix} 0 \\ \widehat{p} \end{bmatrix}.$$

In (S), each element of the structured matrix $\mathscr{S}(p)$ is equal to the corresponding element of the matrix S_0 or to the $\mathtt{S}_{ij}$th element of the parameter vector p. The structure is then specified by the matrices S_0 and S. Although (S) is a special case of the general affine structure ($\mathscr{S}(\widehat{p})$), it covers all linear modeling problems considered in this book and will therefore be used in the implementation of the solution method.

In the implementation of the algorithm, the matrix S corresponds to a variable `tts` and the extended parameter vector p_{ext} corresponds to a variable `pext`. Since in MATLAB, indeces are positive integers (zero index is not allowed), in all indexing operations of `pext`, the index is incremented by one. Given the matrices S_0 and S, specifying the structure, and a structure parameter vector $\widehat{p}$, the structured matrix $\mathscr{S}(\widehat{p})$ is constructed by

83a ⟨$(S_0, \mathtt{S}, \widehat{p}) \mapsto \widehat{D} = \mathscr{S}(\widehat{p})$ 83a⟩≡
```
phext = [0; ph(:)]; dh = s0 + phext(tts + 1);
```

The matrix dimensions `m`, `n`, and the number of parameters `np` are obtained from S as follows:

83b ⟨$\mathtt{S} \mapsto (m,n,n_p)$ 83b⟩≡
```
[m, n] = size(tts); np = max(max(tts));
```

Finally, the transition from the specification of (S) to the specification in general affine case ($\mathscr{S}(\widehat{p})$) is done by

83c ⟨$\mathtt{S} \mapsto \mathbf{S}$ 83c⟩≡
```
bfs = zeros(m * n, np); for k = 1:np, bfs(:, k) = vec(tts == k); end
```

Conversely, for a linear structure, defined by $\mathbf{S}$ (and m, n) which is of the type (S), the matrix S is constructed by

83d ⟨$\mathbf{S} \mapsto \mathtt{S}$ 83d⟩≡
```
tts = reshape(bfs * (1:np)', m, n);
```

In most applications that we consider, the structure $\mathscr{S}$ is linear, so that `s0` is an optional input argument to the solovers with default value the zero matrix.

83e ⟨*default* `s0` 83e⟩≡
```
if ~exist('s0', 'var') | isempty(s0), s0 = zeros(m, n); end
```

The default weight matrix W in the approximation criterion is the identity matrix

83f ⟨*default weight matrix* 83f⟩≡
```
if ~exist('w', 'var') | isempty(w), w = eye(np); end
```


Minimization over $\widehat{p}$

In order to solve the optimization problem (SLRA) we change variables

$$\widehat{p} \quad \mapsto \quad \Delta p = p - \widehat{p}.$$

Then, the constraint is written as a standard linear system of equations with unknown Δp:

$$\begin{aligned} R\mathscr{S}(\widehat{p}) = 0 \quad &\Longleftrightarrow \quad R\mathscr{S}(p - \Delta p) = 0 \\ &\Longleftrightarrow \quad R\mathscr{S}(p) - R\mathscr{S}(\Delta p) + RS_0 = 0 \\ &\Longleftrightarrow \quad \operatorname{vec}\big(R\mathscr{S}(\Delta p)\big) = \operatorname{vec}\big(R\mathscr{S}(p)\big) + \operatorname{vec}(RS_0) \\ &\Longleftrightarrow \quad \underbrace{\big[\operatorname{vec}(RS_1) \ \cdots \ \operatorname{vec}(RS_{n_p})\big]}_{G(R)} \Delta p = \underbrace{G(R)p + \operatorname{vec}(RS_0)}_{h(R)} \\ &\Longleftrightarrow \quad G(R)\Delta p = h(R). \end{aligned}$$

84a ⟨*form $G(R)$ and $h(R)$* 84a⟩≡
```
g = reshape(R * reshape(bfs, m, n * np), size(R, 1) * n, np);
h = g * p + vec(R * s0);
```

The inner minimimization in (SLRA_R) with respect to the new variable Δp is a linear least norm problem

$$\text{minimize} \quad \text{over } \Delta p \quad \|\Delta p\|_W \quad \text{subject to} \quad G(R)\Delta p = h(R) \qquad (\mathrm{LNP})$$

and has the analytic solution

$$\Delta p^*(R) = W^{-1}G^\top(R)\big(G(R)W^{-1}G^\top(R)\big)^{-1}h(R).$$

84b ⟨*solve the least-norm problem* 84b⟩≡
```
dp = inv_w * g' * (pinv(g * inv_w * g') * h);
```

Finally, the cost function to be minimized over the parameter R is

$$f(R) = \|\Delta p^*(R)\|_W = \sqrt{\Delta p^{*\top}(R) W \Delta p^*(R)}.$$

The function f corresponds to the data–model misfit function in data modeling problems and will be refered to as the structured low rank approximation misfit.

84c ⟨*Structured low rank approximation misfit* 84c⟩≡
```
function [M, ph] = misfit_slra(R, tts, p, w, s0, bfs, inv_w)
⟨S ↦ (m,n,n_p) 83b⟩
⟨default s0 83e⟩
⟨default weight matrix 83f⟩
if ~exist('bfs'), ⟨S ↦ S 83c⟩, end
⟨form G(R) and h(R) 84a⟩
if ~exist('inv_w'), inv_w = inv(w); end
⟨solve the least-norm problem 84b⟩
M = sqrt(dp' * w * dp); ph = p - dp;
```

Defines:
`misfit_slra`, used in chunk 85.

Minimization over R

General purpose constrained optimization methods are used for the outer minimization problem in (SLRA$_R$), *i.e.*, the minimization of f over R, subject to the constraint $RR^\top = I$. This is a nonconvex optimization problem, so that there is no guarantee that a globally optimal solution is found.

85a ⟨*set optimization solver and options* 85a⟩≡
```
prob = optimset();
prob.solver = 'fmincon';
prob.options = optimset('disp', 'off');
```

85b ⟨*nonlinear optimization over R* 85b⟩≡
⟨*set optimization solver and options* 85a⟩
```
prob.x0 = Rini; inv_w = inv(w);
prob.objective = ...
          @(R) misfit_slra(R, tts, p, w, s0, bfs, inv_w);
prob.nonlcon = @(R) deal([], [R * R' - eye(size(R, 1))]);
[R, fval, flag, info] = fmincon(prob); info.M = fval;
```
Uses `misfit_slra` 84c.

If not specified, the initial approximation is computed from a heuristic that ignores the structure and replaces the weighted norm by the Frobenius norm, so that the resulting problem can be solved by the singular value decomposition (function `lra`).

85c ⟨*default initial approximation* 85c⟩≡
```
if ~exist('Rini') | isempty(Rini)
  ph = p; ⟨(S0, S, p̂) ↦ D̂ = 𝒮(p̂) 83a⟩, Rini = lra(dh, r);
end
```
Uses `lra` 66.

The resulting function is:

85d ⟨*Structured low rank approximation* 85d⟩≡
```
function [R, ph, info] = slra(tts, p, r, w, s0, Rini)
⟨S ↦ (m, n, n_p) 83b⟩, ⟨S ↦ S 83c⟩
⟨default s0 83e⟩, ⟨default weight matrix 83f⟩
⟨default initial approximation 85c⟩
⟨nonlinear optimization over R 85b⟩
if nargout > 1,
  [M, ph] = misfit_slra(R, tts, p, w, s0, bfs, inv_w);
end
```
Defines:
`slra`, used in chunks 85, 100a, 85d, 108, and 119.
Uses `misfit_slra` 84c.

Exercise 3.3. Use `slra` and `r2x` to solve approximately an overdetermined linear system of equations $AX \approx B$ in the least squares sense. Check the accuracy of the answer by using the analytical expression. □

Exercise 3.4. Use `slra` to solve the basic low rank approximation problem

$$\text{minimize} \quad \text{over } \widehat{D} \quad \|D - \widehat{D}\|_{\mathrm{F}} \quad \text{subject to} \quad \operatorname{rank}(\widehat{D}) \leq \mathtt{m}.$$

(unstructured approximation in the Frobenius norm). Check the accuracy of the answer by the Eckart–Young–Mirsky theorem (`lra`). □

Exercise 3.5. Use `slra` to solve the weighted low rank approximation problem (unstructured approximation in the weighted norm ($\|\cdot\|_W$), defined on page 62). Check the accuracy of the answer in the special case of two-sided weighted low rank approximation, using Theorem 2.29. □

Algorithms for linear system identification

Approximate linear system identification problems can be solved as equivalent Hankel structured low rank approximation problems. Therefore, the function `slra`, implemented in the previous section can be used for linear time-invariant system identification. This approach is developed in Section 4.3.

In this section an alternative approach for approximate system identification that is motivated from a system theoretic view of the problem is used. The Hankel structure in the problem is exploited, which results in efficient computational methods.

Misfit computation

Consider the misfit between the data w_{d} and a model $\mathscr{B}$

$$\text{misfit}(w_{\mathrm{d}}, \mathscr{B}) := \min_{\widehat{w}} \|w_{\mathrm{d}} - \widehat{w}\|_2 \quad \text{subject to} \quad \widehat{w} \in \mathscr{B}. \qquad (\text{misfit } \mathscr{L}_{\mathtt{m},\mathtt{l}})$$

Geometrically, misfit$(w_{\mathrm{d}}, \mathscr{B})$ is the orthogonal projection of w_{d} on $\mathscr{B}$. Assuming that $\mathscr{B}$ is controllable, $\mathscr{B}$ has a minimal image representation image$\big(P(\sigma)\big)$. In terms of the parameter P, the constraint $\widehat{w} \in \mathscr{B}$ becomes $\widehat{w} = P(\sigma)\ell$, for some latent variable ℓ. In a matrix form,

$$\widehat{w} = \mathscr{T}_T(P)\ell,$$

where

$$\mathscr{T}_T(P) := \begin{bmatrix} P_0 & P_1 & \cdots & P_1 & & \\ & P_0 & P_1 & \cdots & P_1 & \\ & & \ddots & \ddots & & \ddots \\ & & & P_0 & P_1 & \cdots & P_1 \end{bmatrix} \in \mathbb{R}^{\mathtt{q}T \times (T+1)}. \qquad (\mathscr{T})$$

86 ⟨*Toeplitz matrix constructor* 86⟩≡
```
function TP = blktoep(P, T)
[q, l1] = size(P); l = l1 - 1; TP = zeros(T * q, T + l);
ind = 1 + (0:T - 1) * q * (T + 1);
```

```
for i = 1:q
  for j = 1:l1
    TP(ind + (i - 1) + (j - 1) * (T * q)) = P(i, j);
  end
end
```

Defines:
`blktoep`, used in chunk 87a.

The misfit computation problem (misfit $\mathscr{L}_{\mathtt{m},1}$) is equivalent to the standard linear least squares problem

$$\text{minimize} \quad \text{over } \ell \quad \|w_{\mathrm{d}} - \mathscr{T}_T(P)\ell\|, \qquad \text{(misfit}_P\text{)}$$

so that the solution, implemented in the functions `misfit_siso`, is

$$\widehat{w} = \mathscr{T}_T(P)\big(\mathscr{T}_T^\top(P)\mathscr{T}_T(P)\big)^{-1}\mathscr{T}_T^\top(P)w_{\mathrm{d}}.$$

87a ⟨dist($w_{\mathrm{d}}, \mathscr{B}$) 87a⟩≡

```
function [M, wh] = misfit_siso(w, P)
[ttw, T] = size(w); if T < ttw, w = w'; [ttw, T] = size(w); end
TP = blktoep(P, T); wh = reshape(TP * (TP \ w(:)), 2, T);
M  = norm(w - wh, 'fro');
```

Defines:
`misfit_siso`, used in chunks 87a, 88b, and 126c.
Uses `blktoep` 86.

Misfit minimization

Consider the misfit minimization problem

$$\widehat{\mathscr{B}}^* := \arg\min_{\widehat{\mathscr{B}}} \quad \text{misfit}(w_{\mathrm{d}}, \mathscr{B}) \quad \text{subject to} \quad \widehat{\mathscr{B}} \in \mathscr{L}_{\mathtt{m},\mathtt{l}}^{\mathtt{q}}, \qquad \text{(SYSID)}$$

Using the representation $\mathscr{B} = \text{image}(P)$, (SYSID) is equivalent to

$$\begin{aligned} &\text{minimize} \quad \text{over } P \in \mathbb{R}^{\mathtt{q}(\mathtt{l}+1)\times\mathtt{m}} \quad \text{misfit}\left(w_{\mathrm{d}}, \text{image}\left(P(\sigma)\right)\right) \\ &\text{subject to} \quad P^\top P = I_{\mathtt{m}}, \end{aligned} \qquad \text{(SYSID}_P\text{)}$$

which is a constrained nonlinear least squares problem.

87b ⟨*Single input single output system identification* 87b⟩≡

```
function [sysh, M, wh] = ident_siso(w, n, sys)
if ~exist('sys', 'var')
  ⟨suboptimal approximate system identification 88a⟩
else
  ⟨(TF) ↦ P 88d⟩
end
⟨misfit minimization 88b⟩
```

Defines:
`ident_siso`, used in chunks 87, 89b, 87b, and 127a.

The initial approximation is computed from a relaxation ignoring the structure constraint:

88a ⟨*suboptimal approximate system identification* 88a⟩≡

```
R = lra(blkhank(w, n + 1), 2 * n + 1); ⟨R ↦ P 88e⟩
```

Uses `blkhank` 25b and `lra` 66.

MATLAB's Optimization toolbox is used for performing the misfit minimization.

88b ⟨*misfit minimization* 88b⟩≡

```
⟨set optimization solver and options 85a⟩
prob.x0 = P;
prob.objective = @(P) misfit_siso(w, P);
prob.nonlcon = @(P) deal([], [P(1, :) * P(1, :)' - 1]);
[P, M, flag, info] = fmincon(prob);
⟨P ↦ (TF) 88c⟩ sysh = sys;
if nargout > 2, [M, wh] = misfit_siso(w, P); end
```

Uses `misfit_siso` 87a.

The obtained solution is an image representation of a (locally) optimal approximate model $\widehat{\mathscr{B}}^*$. In the function `ident_siso`, the image representation is converted to a transfer function representation as follows:

88c ⟨$P \mapsto$ *(TF)* 88c⟩≡

```
p = fliplr(P(1, :)); q = fliplr(P(2, :)); sys = tf(q, p, -1);
```

The reverse transformation

88d ⟨*(TF)* $\mapsto P$ 88d⟩≡

```
[q, p] = tfdata(tf(sys), 'v'); P = zeros(2, length(p));
P(1, :) = fliplr(p);
P(2, :) = fliplr([q zeros(length(p) - length(q))]);
```

is used when an initial approximation is specified by a transfer function representation. Finally, when no initial approximation is supplied, a default one is computed by unstructured low rank approximation, which produces a kernel representation of the model. Transition from kernel to image representation is done indirectly by passing through a transfer function representation:

88e ⟨$R \mapsto P$ 88e⟩≡
⟨$R \mapsto$ *(TF)* 88f⟩ ⟨*(TF)* $\mapsto P$ 88d⟩

where

88f ⟨$R \mapsto$ *(TF)* 88f⟩≡

```
q = - fliplr(R(1:2:end)); p = fliplr(R(2:2:end));
sys = tf(q, p, -1);
```

For later use, next, we define the reverse mapping:

88g ⟨$P \mapsto R$ 88g⟩≡
⟨$P \mapsto$ *(TF)* 88c⟩ ⟨*(TF)* $\mapsto R$ 88h⟩

88h ⟨*(TF)* $\mapsto R$ 88h⟩≡

```
[q, p] = tfdata(tf(sys), 'v'); R = zeros(1, length(p) * 2);
R(1:2:end) = - fliplr([q zeros(length(p) - length(q))]);
R(2:2:end) = fliplr(p);
```

Numerical example

The function `ident_siso` is applied on data obtained in the errors-in-variables setup (EIV). The true data generating model is a random single input single output linear time-invariant system and the true data w_0 is a random trajectory of that system:

89a ⟨test_ident_siso 89a⟩≡
```
n = 3; sys0 = drss(n); T = 20;
xini0 = rand(n, 1); u0 = rand(T, 1);
y0 = lsim(sys0, u0, 1:T, xini0);
w = [u0 y0] + 0.1 * randn(T, 2);
```

The optimal approximation obtained by `ident_siso`

89b ⟨test_ident_siso 89a⟩+≡
```
[sysh, M] = ident_siso(w, n);
```
Uses ident_siso 87b.

is verified by comparing it with the approximation obtained by the function `slra` (called by the wrapper function `ident_eiv`, see Section 4.3)

89c ⟨test_ident_siso 89a⟩+≡
```
sysh_ = ident_eiv(w, 1, n); norm(sysh - sysh_)
```
Uses ident_eiv 115c.

In a particualr example, the resulting norm of the difference between the two approximations is of the order of magnitude of the convergence tolergance used in the optimization methods, which shows that the two methods have converged to the same locally optimal solution.

Computation of an approximate greatest common divisor

Let $\mathbb{P}_n$ be the set of scalar polynomials of degree less than or equal to n, *i.e.*,

$$\mathbb{P}_n := \{\, p \in \mathbb{R}[z] \mid \text{degree}(p) \leq n \,\}.$$

Associated with a polynomial

$$p(z) := p_0 + p_1 z + \cdots + p_n z^n \in \mathbb{P}_n$$

of degree at most n is an $(n+1)$-dimensional coefficients vector

$$p := \text{col}(p_0, p_1, \ldots, p_n) \in \mathbb{R}^{n+1}$$

and vice verse a vector $p \in \mathbb{R}^{n+1}$ corresponds to a polynomial $p \in \mathbb{P}_n$. The coefficients vector of a polynomial, however, is not unique (scaling of the coefficients vector by a nonzero number result in the same polynomial). In order to remove this nonuniqueness, we scale the coefficients, so that the highest power coefficient is

equal to one (monic polynomial). In what follows, we assume that the coefficients are aways scaled in this way.

The polynomials $p \in \mathbb{P}_n$ and $\widehat{p} \in \mathbb{P}_n$ are "close" to each other if the distance measure

$$\text{dist}\,(p, \widehat{p}) := \|p - \widehat{p}\|_2$$

is "small", *i.e.*, if the norm of the normalized coefficients vector of the error polynomial $\Delta p := p - \widehat{p}$ is small.

Note 3.6. The distance $\text{dist}\,(p, \widehat{p})$, defined above, might not be an appropriate distance measure in applications where the polynomial roots rather than coefficients are of primary interest. Polynomial roots might be sensitive (especially for high order polynomials) to perturbations in the coefficients, so that closeness of coefficients does not necessarily imply closeness of roots. Using the quadratic distance measure in terms of the polynomial coefficients, however, simplifies the solution of the approximate common divisor problem defined next.

Problem 3.7 (Approximate common divisor). Given polynomials p and q, and a natural number $\mathtt{d}$, smaller than the degrees of p and q, find polynomials $\widehat{p}$ and $\widehat{q}$ that have a common divisor c of degree $\mathtt{d}$ and minimize the approximation error

$$\text{dist}\,\big(\text{col}(p,q), \text{col}(\widehat{p},\widehat{q})\big).$$

The polynomial c is an optimal (in the specified sense) approximate common divisor of p and q. □

Note 3.8. The object of interest in solving Problem 3.7 is the approximate common divisor c. The approximating polynomials $\widehat{p}$ and $\widehat{q}$ are auxiliary variables introduced for the purpose of defining c.

Problem 3.7 has the following system theoretic interpretation. Consider the single-input single-output linear time-invariant system $\mathscr{B} = \mathscr{B}_{\text{i/o}}(p,q)$. The system $\mathscr{B}$ is controllable if and only if p and q have no common factor. Therefore, Problem 3.7 finds the nearest uncontrollable system $\widehat{\mathscr{B}} = \mathscr{B}_{\text{i/o}}(\widehat{p},\widehat{q})$ to the given system $\mathscr{B}$. The bigger the approximation error is, the more robust the controllability property of $\mathscr{B}$ is. In particular, with zero approximation error, $\mathscr{B}$ is uncontrollable.

Equivalent optimization problem

By definition, the polynomial c is a common divisor of $\widehat{p}$ and $\widehat{q}$ if there are polynomials u and v, such that

$$\widehat{p} = uc \qquad \text{and} \qquad \widehat{q} = vc. \tag{GCD}$$

With the auxiliary variables u and v, Problem 3.7 becomes the following optimization problem:

$$\begin{aligned} &\text{minimize} \quad \text{over } \widehat{p},\, \widehat{q},\, u,\, v, \text{ and } c \quad \operatorname{dist}\big(\operatorname{col}(p,q), \operatorname{col}(\widehat{p},\widehat{q})\big) \\ &\text{subject to} \quad \widehat{p} = uc, \quad \widehat{q} = vc, \quad \text{and} \quad \operatorname{degree}(c) = \mathtt{d}. \end{aligned} \tag{AGCD}$$

Theorem 3.9. *The optimization problem (AGCD) is equivalent to*

$$\text{minimize} \quad \text{over } c_0, \ldots, c_{\mathtt{d}-1} \in \mathbb{R} \quad f(c), \tag{AGCD'}$$

where

$$f(c) := \operatorname{trace}\Big(\begin{bmatrix} p & q \end{bmatrix}^\top \Big(I - \mathscr{T}_{n+1}(c) \big(\mathscr{T}_{n+1}^\top(c) \mathscr{T}_{n+1}(c) \big)^{-1} \mathscr{T}_{n+1}^\top(c) \Big) \begin{bmatrix} p & q \end{bmatrix} \Big),$$

and $\mathscr{T}_{n+1}(c)$ is an upper triangular Toeplitz matrix, defined in ($\mathscr{T}$) on page 86.

The proof is given in Appendix B.

Compared with the original optimization problem (AGCD), in (AGCD'), the constraint and the auxiliary decision variables $\widehat{p}$, $\widehat{q}$, u, and v are eliminated. This achieves significant simplification from a numerical optimization point of view. The equivalent problem (AGCD') is a nonlinear least squares problem and can be solved by standard local optimization methods, see Algorithm 1.

Algorithm 1 Optimal approximate common divisor computation.

Input: Polynomials p and q and a positive integer $\mathtt{d}$.

1: Compute an initial approximation $c_{\text{ini}} \in \mathbb{R}^{\mathtt{d}+1}$.
2: Execute a standard optimization algorithm for the minimization (AGCD') with initial approximation c_{ini}.
3: **if** $\widehat{p}$ and $\widehat{q}$ have to be displayed **then**
4: Solve for u and v the linear least squares problem in u and v

$$\begin{bmatrix} p & q \end{bmatrix} = \mathscr{T}_{n-\mathtt{d}+1}(c) \begin{bmatrix} u & v \end{bmatrix}.$$

5: Define $\widehat{p} = u \star c$ and $\widehat{q} = v \star c$, where $\star$ denotes discrete convolution.
6: **end if**

Output: The approximation $c \in \mathbb{R}^{\mathtt{d}+1}$ found by the optimization algorithm upon convergence, the value of the cost function $f(c)$ at the optimal solution, and if computed $\widehat{p}$ and $\widehat{q}$.

Since

$$f(c) = \operatorname{dist}\big(\operatorname{col}(p,q), \operatorname{col}(\widehat{p},\widehat{q})\big)$$

the value of the cost function $f(c)$ shows the approximation errors in taking c as an approximate common divisor of p and q. Optionally, Algorithm 1 returns a "certificate" $\widehat{p}$ and $\widehat{q}$ for c being an approximate common divisor of p and q with approximation error $f(c)$.

In order to complete Algorithm 1, next, we specify the computation of the initial approximation c_{ini}. Also, the fact that the analytic expression for $f(c)$ involves the highly structured matrix $\mathscr{T}_{n-\mathtt{d}+1}(c)$ suggests that $f(c)$ (and its derivatives) can be evaluated efficiently.

Efficient cost function evaluation

The most expensive operation in the cost function evaluation is solving the least squares problem

$$\begin{bmatrix} p & q \end{bmatrix} = \mathscr{T}_{n-\mathtt{d}+1}(c) \begin{bmatrix} u & v \end{bmatrix}.$$

Since $\mathscr{T}_{n-\mathtt{d}+1}(c)$ is an upper triangular, banded, Toeplitz matrix, this operation can be done efficiently. One approach is to compute efficiently the QR factorization of $\mathscr{T}_{n-\mathtt{d}+1}(c)$, *e.g.*, via the *generalized Schur algorithm*. Another approach is to solve the normal system of equations

$$\mathscr{T}_{n+1}^\top(c) \begin{bmatrix} p & q \end{bmatrix} = \mathscr{T}_{n-\mathtt{d}+1}^\top(c) \mathscr{T}_{n-\mathtt{d}+1}(c) \begin{bmatrix} u & v \end{bmatrix},$$

exploiting the fact that $\mathscr{T}_{n-\mathtt{d}+1}^\top(c)\mathscr{T}_{n-\mathtt{d}+1}(c)$ is banded and Toeplitz structured. The first approach is implemented in the function `MB02ID` from the SLICOT library.

Once the least squares problem is solved, the product

$$\mathscr{T}_{n-\mathtt{d}+1}(c) \begin{bmatrix} u & v \end{bmatrix} = \begin{bmatrix} c \star u & c \star v \end{bmatrix}$$

is computed efficiently by the *fast Fourier transform*. The resulting algorithm has computational complexity $O(n)$ operations. The first derivative $f'(c)$ can be evaluated also in $O(n)$ operations, so assuming that $\mathtt{d} \ll n$, the overall cost per iteration for Algorithm 1 is $O(n)$.

Initial approximation

Suboptimal initial approximation can be computed by the singular value decomposition of the Sylvester (sub)matrix

$$\begin{aligned} \mathscr{R}_{\mathtt{d}}(p,q) &= \begin{bmatrix} \mathscr{T}_{n-\mathtt{d}+1}^\top(p) & \mathscr{T}_{n-\mathtt{d}+1}^\top(q) \end{bmatrix} \\ &= \begin{bmatrix} p_0 & & & q_0 & & \\ p_1 & p_0 & & q_1 & q_0 & \\ \vdots & p_1 & \ddots & \vdots & q_1 & \ddots \\ p_n & \vdots & \ddots & p_0 & q_n & \vdots & \ddots & q_0 \\ & p_n & & p_1 & & q_n & & q_1 \\ & & \ddots & \vdots & & & \ddots & \vdots \\ & & & p_n & & & & q_n \end{bmatrix} \in \mathbb{R}^{(2n-\mathtt{d}+1)\times(2n-2\mathtt{d}+2)}. \end{aligned}$$

Since, the approximate greatest common divisor problem (AGCD) is a structured low rank approximation problem, ignoring the Sylvester structure constraint results a suboptimal solution method—unstructured low rank approximation. A suboptimal solution is therefore computable by the singular value decomposition.

The polynomial c is a common divisor of $\widehat{p}$ and $\widehat{q}$ if and only if there are polynomials u and v, such that

$$\widehat{p}v = \widehat{q}u. \qquad (*)$$

With $\text{degree}(c) = \mathtt{d}$, the polynomial equation $(*)$ is equivalent to the system of algebraic equations

$$\mathscr{R}_{\mathtt{d}}(\widehat{p},\widehat{q}) \begin{bmatrix} v \\ -u \end{bmatrix} = 0.$$

The degree constraint for c is equivalent to

$$\text{degree}(u) = n - \mathtt{d},$$

or equivalently $u_{n-\mathtt{d}+1} \neq 0$. Since u is defined up to a scaling factor, we impose the normalization $u_{n-\mathtt{d}+1} = 1$. This shows that problem (AGCD) is equivalent to

$$\begin{array}{ll} \text{minimize} & \text{over } \widehat{p},\widehat{q} \in \mathbb{R}^{n+1} \text{ and } u,v \in \mathbb{R}^{n-\mathtt{d}+1} \quad \left\| \begin{bmatrix} p & q \end{bmatrix} - \begin{bmatrix} \widehat{p} & \widehat{q} \end{bmatrix} \right\|_{\mathrm{F}} \\ \text{subject to} & \mathscr{R}_{\mathtt{d}}(\widehat{p},\widehat{q}) \begin{bmatrix} v \\ -u \end{bmatrix} = 0 \quad \text{and} \quad u_{n-\mathtt{d}+1} = 1. \end{array} \qquad \text{(AGCD")}$$

The approximate common factor c is not explicitly computed in (AGCD"). Once the optimal u and v are known, however, c can be found from (GCD). (By construction these equations have unique solution). Alternatively, without using the auxiliary variables $\widehat{p}$ and $\widehat{q}$, c can be computed from the least squares problem

$$\begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} u \\ v \end{bmatrix} c,$$

or in linear algebra notation

$$\begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} \mathscr{T}_{\mathtt{d}+1}^{\top}(u) \\ \mathscr{T}_{\mathtt{d}+1}^{\top}(v) \end{bmatrix} c. \qquad (3.1)$$

Problem (AGCD") is a structured low rank approximation problem: it aims to find a Sylvester rank deficient matrix $\mathscr{R}_{\mathtt{d}}(\widehat{p},\widehat{q})$ as close as possible to a given matrix $\mathscr{R}_{\mathtt{d}}(p,q)$ with the same structure. If p and q have no common divisor of degree $\mathtt{d}$, $\mathscr{R}_{\mathtt{d}}(p,q)$ is full rank so that an approximation is needed.

The (unstructured) low rank approximation problem

$$\begin{array}{ll} \text{minimize} & \text{over } \widehat{D} \text{ and } r \quad \left\| \mathscr{R}_{\mathtt{d}}(p,q) - \widehat{D} \right\|_{\mathrm{F}}^2 \\ \text{subject to} & \widehat{D}r = 0 \quad \text{and} \quad r^{\top}r = 1 \end{array} \qquad \text{(LRA}_r\text{)}$$

has an analytic solution in terms of the singular value decomposition of $\mathscr{R}_{\mathtt{d}}(p,q)$. The vector $r \in \mathbb{R}^{2(n-\mathtt{d}+1)}$ corresponding to the optimal solution of (LRA$_r$) is equal to the right singular vector of $\mathscr{R}_{\mathtt{d}}(p,q)$ corresponding to the smallest singular value. The vector $\text{col}(v,-u)$ composed of the coefficients of the approximate divisors v

and $-u$ is up to a scaling factor (that enforces the normalization constraint $u_{n-\mathtt{d}+1} = 1$) equal to r. This gives Algorithm 2 as a method for computing a suboptimal initial approximation.

Algorithm 2 Suboptimal approximate common divisor computation.

Input: Polynomials p and q and an integer $\mathtt{d}$.
1: Solve the unstructured low rank approximation problem (LRA$_r$).
2: Let $\text{col}(v,-u) := r$, where $u,v \in \mathbb{R}^{n-\mathtt{d}+1}$.
3: Solve the least squares problem (3.1).
Output: The solution c of the least squares problem.

Numerical examples

Implementation of the method for computation of approximate common divisor, described in this section, is available from the book's web page. We verify the results obtained by Algorithm 1 on examples from (Zhi and Yang, 2004) and (Karmarkar and Lakshman, 1998). Up to the number of digits shown our results match the ones reported in the literature.

Example 4.1 from (Zhi and Yang, 2004)

The given polynomials are

$$\begin{aligned} p(z) &= (4+2z+z^2)(5+2z) + 0.05 + 0.03z + 0.04z^2 \\ q(z) &= (4+2z+z^2)(5+\ z) + 0.04 + 0.02z + 0.01z^2 \end{aligned}$$

and an approximate common divisor c of degree $d = 2$ is sought. Algorithm 1 converges in 4 iteration steps with the following answer

$$c(z) = 3.9830 + 1.9998z + 1.0000z^2.$$

To this approximate common divisor correspond approximating polynomials

$$\begin{aligned} \widehat{p}(z) &= 20.0500 + 18.0332z + 9.0337z^2 + 2.0001z^3 \\ \widehat{q}(z) &= 20.0392 + 14.0178z + 7.0176z^2 + 0.9933z^3 \end{aligned}$$

and the approximation error is

$$f(c) = \text{dist}^2\left(p,\widehat{p}\right) + \text{dist}^2\left(q,\widehat{q}\right) = 1.5831 \times 10^{-4}.$$

Example 4.2, case 1, from (Zhi and Yang, 2004) (originally given in (Karmarkar and Lakshman, 1998))

The given polynomials are

$$p(z) = (1-z)(5-z) = 5 - 6z + z^2$$
$$q(z) = (1.1-z)(5.2-z) = 5.72 - 6.3z + z^2$$

and an approximate common divisor c of degree $\mathtt{d} = 1$ (a common root) is sought. Algorithm 1 converges in 6 iteration steps with the following answer

$$c(z) = -5.0989 + 1.0000z.$$

The corresponding approximating polynomials are

$$\widehat{p}(z) = 4.9994 - 6.0029z + 0.9850z^2$$
$$\widehat{q}(z) = 5.7206 - 6.2971z + 1.0150z^2$$

and the approximation error is $f(c) = 4.6630 \times 10^{-4}$.

3.3 Data modeling using the nuclear norm heuristic

The nuclear norm heuristic leads to a semidefinite optimization problem, which can be solved by existing algorithms with provable convergence properties and readily available high quality software packages. Apart from theoretical justification and easy implementation in practice, formulating the problem as a semidefinite program has the advantage of flexibility. For example, adding regularization and affine inequality constraints in the data modeling problem still leads to semidefinite optimization problems that can be solved by the same algorithms and software as the original problem.

A disadvantage of using the nuclear norm heuristic is the fact that the number of optimization variables in the semidefinite optimization problem depends quadratically on the number of data points in the data modeling problem. This makes methods based on the nuclear norm heuristic impractical for problems with more than a few hundreds of data points. Such problems are "small size" data modeling problem.

Nuclear norm heuristics for structured low rank approximation

Regularized nuclear norm minimization

The nuclear norm of a matrix is the sum of the matrix's singular values

$$\|M\|_* = \text{sum of the singular values of } M. \tag{NN}$$

Recall the mapping $\mathscr{S}$, see ($\mathscr{S}(\widehat{p})$), on page 82, from a structure parameter space $\mathbb{R}^{n_p}$ to the set of matrices $\mathbb{R}^{m\times n}$. Regularized nuclear norm minimization

$$\begin{array}{ll} \text{minimize} & \text{over } \widehat{p} \quad \|\mathscr{S}(\widehat{p})\|_* + \gamma\|p-\widehat{p}\| \\ \text{subject to} & G\widehat{p} \leq h \end{array} \tag{NNM}$$

is a convex optimization problem and can be solved globally and efficiently. Using the fact

$$\|\widehat{D}\|_* < \mu \quad \Longleftrightarrow \quad \frac{1}{2}\big(\operatorname{trace}(U) + \operatorname{trace}(V)\big) < \mu \quad \text{and} \quad \begin{bmatrix} U & \widehat{D}^\top \\ \widehat{D} & V \end{bmatrix} \succeq 0,$$

we obtain an equivalent problem

$$\begin{array}{lll} \text{minimize} & \text{over } \widehat{p}, U, V, \text{ and } v & \frac{1}{2}\big(\operatorname{trace}(U) + \operatorname{trace}(V)\big) + \gamma v \\ \text{subject to} & \begin{bmatrix} U & \mathscr{S}(\widehat{p})^\top \\ \mathscr{S}(\widehat{p}) & V \end{bmatrix} \succeq 0, & \\ & \|p-\widehat{p}\| < v, \quad \text{and} \quad G\widehat{p} \leq h, & \end{array} \tag{NNM'}$$

which can further be reduced to a semidefinite programming problem and solved by standard methods.

Structured low rank approximation

Consider the affine structured low rank approximation problem (SLRA). Due to the rank constraint, this problem is non-convex. Replacing the rank constraint by a constraint on the nuclear norm of the affine structured matrix, however, results in a convex relaxation of (SLRA)

$$\text{minimize} \quad \text{over } \widehat{p} \quad \|p-\widehat{p}\| \quad \text{subject to} \quad \|\mathscr{S}(\widehat{p})\|_* \leq \mu. \tag{RLRA}$$

The motivation for this heuristic of solving (SLRA) is that approximation with an appropriately chosen bound on the nuclear norm tends to give solutions $\mathscr{S}(\widehat{p})$ of low (but nonzero) rank. Moreover, the nuclear norm is the tightest relaxation of the rank function, in the same way ℓ_1-norm is the tightest relaxation of the function mapping a vector to the number of its nonzero entries.

Problem (RLRA) can also be written in the equivalent unconstrained form

$$\text{minimize} \quad \text{over } \widehat{p} \quad \|\mathscr{S}(\widehat{p})\|_* + \gamma\|p-\widehat{p}\|. \tag{RLRA'}$$

Here γ is a regularization parameter that is related to the parameter μ in (RLRA). The latter formulation of the relaxed affine structured low rank approximation problem is a regularized nuclear norm minimization problem (NNM').

Literate programs

Regularized nuclear norm minimization

The CVX package is used in order to automatically translate problem (NNM') into a standard convex optimization problem and solve it by existing optimization solvers.

97a ⟨*Regularized nuclear norm minimization* 97a⟩≡
```
function [ph, info] = nucnrm(tts, p, gamma, nrm, w, s0, g, h)
⟨S ↦ (m,n,n_p) 83b⟩, ⟨default s0 83e⟩
```
Defines:
nucnrm, used in chunks 97a, 98a, and 103.

The code consists of definition of the optimization variables:

97b ⟨*Regularized nuclear norm minimization* 97a⟩+≡
```
cvx_begin sdp; cvx_quiet(true);
    variable U(n, n) symmetric;
    variable V(m, m) symmetric;
    variables ph(np) nu;
    ⟨(S_0,S,p̂) ↦ D̂ = 𝒮(p̂) 83a⟩
```

and direct rewriting of the cost function and constraints of (NNM') in MATLAB's syntax:

97c ⟨*Regularized nuclear norm minimization* 97a⟩+≡
```
    minimize( trace(U) / 2 + trace(V) / 2 + gamma * nu );
    subject to
        [U dh'; dh V] > 0;
        norm(w * (p - ph), nrm) < nu;
        if (nargin > 6) & ~isempty(g), g * ph < h; end
cvx_end
```

The w argument specifies the norm ($\|\cdot\|_w$) and is equal to 1, 2, inf, or (in the case of a weighted 2-norm) a $n_p \times n_p$ positive semidefinite matrix. The info output variable is a structure with fields optval (the optimal value) and status (a string indicating the convergence status).

Structured low rank approximation

The following function finds suboptimal solution of the structured low rank approximation problem by solving the relaxation problem (RLRA'). Affine structures of the type (S) are considered.

97d ⟨*Structured low rank approximation using the nuclear norm* 97d⟩≡
```
function [ph, gamma] = slra_nn(tts, p, r, gamma, nrm, w, s0)
⟨S ↦ (m,n,n_p) 83b⟩, ⟨S ↦ S 83c⟩, ⟨default s0 83e⟩, ⟨default weight matrix 83f⟩
```



```
if ~exist('gamma', 'var'), gamma = []; end % default gamma
if ~exist('nrm', 'var'),   nrm = 2;    end % default norm
```

Defines:
slra_nn, used in chunks 97, 99e, 100c, 97d, 124b, and 126b.

If a parameter γ is supplied, the convex relaxation (RLRA') is completely specified and can be solved by a call to nucnrm.

98a ⟨*solve the convex relaxation (RLRA') for given γ parameter* 98a⟩≡
```
ph = nucnrm(tts, p, gamma, nrm, w, s0);
```
Uses nucnrm 97a.

Large values of γ lead to solutions $\widehat{p}$ with small approximation error $\|p-\widehat{p}\|_W$, but potentially high rank. Vice verse, small values of γ lead to solutions $\widehat{p}$ with low rank, but potentially high approximation error $\|p-\widehat{p}\|_W$. If not given as an input argument, a value of γ which gives an approximation matrix $\mathscr{S}(\widehat{p})$ with numerical rank r can be found by bisection on an a priori given interval $[\gamma_{\min}, \gamma_{\max}]$. The interval can be supplied via the input argument gamma, in which case it is a vector $[\gamma_{\min}, \gamma_{\max}]$.

98b ⟨*Structured low rank approximation using the nuclear norm* 97d⟩+≡
```
if ~isempty(gamma) & isscalar(gamma)
    ⟨solve the convex relaxation (RLRA') for given γ parameter 98a⟩
else
    if ~isempty(gamma)
        gamma_min = gamma(1); gamma_max = gamma(2);
    else
        gamma_min = 0; gamma_max = 100;
    end
    ⟨parameters of the bisection algorithm 99a⟩
    ⟨bisection on γ 98c⟩
end
```

On each iteration of the bisection algorithm, the convex relaxation (RLRA') is solved for γ equal to the mid point $(\gamma_{\min}+\gamma_{\max})/2$ of the interval and the numerical rank of the approximation $\mathscr{S}(\widehat{p})$ is checked by computing the singular values of the approximation. If the numerical rank is higher than r, $\gamma_{\max}$ is redefined to the mid point, so that the search continuous on smaller values of γ (which have the potential of decreasing the rank). Otherwise, $\gamma_{\max}$ is redefined to the mid point, so that the search continuous on higher values of γ (which have the potential of increasing the rank). The search continuous till the interval $[\gamma_{\min}, \gamma_{\max}]$ is sufficiently small or a maximum number of iterations is exceeded.

98c ⟨*bisection on γ* 98c⟩≡
```
iter = 0;
while ((gamma_max - gamma_min) / gamma_max > rel_gamma_tol) ...
      & (iter < maxiter)
    gamma = (gamma_min + gamma_max) / 2;
    ⟨solve the convex relaxation (RLRA') for given γ parameter 98a⟩
    sv = svd(ph(tts));
    if (sv(r + 1) / sv(1) > rel_rank_tol) ...
        & (sv(1) > abs_rank_tol)
        gamma_max = gamma;
```

```
    else
        gamma_min = gamma;
    end
    iter = iter + 1;
end
```

The rank test and the interval width test involve a priori set tolerances.

99a ⟨*parameters of the bisection algorithm* 99a⟩≡
```
rel_rank_tol  = 1e-6; abs_rank_tol = 1e-6;
rel_gamma_tol = 1e-5; maxiter      = 20;
```

Examples

Unstructured and Hankel structured problems

The function `slra_nn` for affine structured low rank approximation by the nuclear norm heuristic is tested on randomly generated Hankel structured and unstructured examples. A rank deficient "true" data matrix is constructed, where the rank r_0 is a simulation parameter. In the case of a Hankel structure, the "true" structure parameter vector p_0 is generated as the impulse response (skipping the first sample) of a discrete-time random linear time-invariant system of order r_0. This ensures that the "true" Hankel structured data matrix $\mathscr{S}(p_0)$ has the desired rank r_0.

99b ⟨`test_slra_nn` 99b⟩≡
```
randn('state', 0), rand('state', 0)
if strcmp(structure, 'hankel')
    np = m + n - 1; tts = hankel(1:m, m:np);
    p0 = impulse(drss(r0), np + 1); p0 = p0(2:end);
```

In the unstructured case, the data matrix is generated by multiplication of random $m \times r_0$ and $r_0 \times n$ factors of a rank revealing factorization of the data matrix $\mathscr{S}(p_0)$.

99c ⟨`test_slra_nn` 99b⟩+≡
```
else % unstructured
    np = m * n; tts = reshape(1:np, m, n);
    p0 = rand(m, r0) * rand(r0, n); p0 = p0(:);
end
```

The data parameter p, passed to the low rank approximation function, is a noisy version of the true data parameter p_0, where the additive noise's standard deviation is a simulation parameter.

99d ⟨`test_slra_nn` 99b⟩+≡
```
e = randn(np, 1); p = p0 + nl * e / norm(e) * norm(p0);
```

The results obtained by `slra_nn`

99e ⟨`test_slra_nn` 99b⟩+≡
```
[ph, gamma] = slra_nn(tts, p, r0);
```
Uses `slra_nn` 97d.

are compared with the ones of alternative methods by checking the singular values of $\mathscr{S}(\widehat{p})$, indicating the numerical rank of the approximation, and the fitting error $\|p - \widehat{p}\|$.

In the case of a Hankel structure, the alternative methods, being used, is Kung's method (implemented in the function `h2ss`) and the method based on local optimization (implemented in the function `slra`).

100a ⟨`test_slra_nn` 99b⟩+≡
```
if strcmp(structure, 'hankel')
    sysh = h2ss([0; p], r0);
    ph2 = impulse(sysh, np + 1); ph2 = ph2(2:end);
    tts_ = hankel(1:(r0 + 1), (r0 + 1):np);
    [Rh, ph3] = slra(tts_, p, r0);
    sv   = [svd(p(tts)) svd(ph(tts)) svd(ph2(tts)) svd(ph3(tts))]
    cost = [norm(p - p) norm(p - ph) ...
            norm(p - ph2) norm(p - ph3)]
```
Uses `h2ss` 78a and `slra` 85d.

In the unstructured case, the alternative method (implemented in the function `lra`) uses the singular value decomposition of the data matrix $\mathscr{S}(p)$ and gives globally optimal result.

100b ⟨`test_slra_nn` 99b⟩+≡
```
else % unstructured
    [Rh, Ph, dh] = lra(p(tts)', r0); dh = dh';
    sv = [svd(p(tts)) svd(ph(tts)) svd(dh)]
    cost = [norm(p - p) norm(p - ph) norm(p - dh(:))]
end
⟨trade-off curve 100c⟩
```
Uses `lra` 66.

Finally, we compute and plot the numerical rank vs fitting error trade-off curves for the methods based on nuclear norm minimization and singular value decomposition.

100c ⟨*trade-off curve* 100c⟩≡
```
E = []; E_ = [];
for rr = 1:min(m,n) - 1
    ph = slra_nn(tts, p, rr);
    E  = [E  norm(p - ph)];
    if strcmp(structure, 'hankel')
        sysh = h2ss([0;p], rr);
        ph = impulse(sysh, np + 1); ph = ph(2:end);
    else % unstructured
        if m > n, [Rh, Ph, dh] = lra(p(tts)', rr); dh = dh';
        else      [Rh, Ph, dh] = lra(p(tts), rr);  end
        ph = dh(:);
    end
    E_ = [E_ norm(p - ph)];
end
plot(E , 1:min(m,n)-1, 'bx', 'markersize', 8), hold on
plot(E_, 1:min(m,n)-1, 'ro', 'markersize', 8)
legend('Nuclear norm','SVD')
plot(E , 1:min(m,n)-1, 'b-', 'linewidth', 2, 'markersize', 8)
plot(E_, 1:min(m,n)-1, 'r-.', 'linewidth', 2, 'markersize', 8)
```

```
print_fig(structure)
```

Uses h2ss 78a, lra 66, print_fig 25a, and slra_nn 97d.

The first test example is a 5×5, unstructured matrix, whose true value has rank 3. Note that in this case the method lra, which is based on the singular value decomposition gives an optimal approximation.

101a ⟨*Test* slra_nn *on unstructured problem* 101a⟩≡

```
m = 5; n = 5; r0 = 3; nl = 1; structure = 'unstructured';
test_slra_nn
```

The output of test_slra_nn is given in Table 3.1, left. It shows that the numerical rank (with tolerance 10^{-5}) of both approximations is equal to the specified rank but the approximation error achieved by the nuclear norm heuristic is about two times the approximation error of the optimal approximation. The trade-off curve is shown in Figure 3.1, left plot.

Table 3.1 Output of test_slra_nn in unstructured problem.

```
sv =

    5.2579    4.1950    5.2579
    2.5845    1.5216    2.5845
    1.6169    0.5539    1.6169
    1.0616    0.0000    0.0000
    0.0027    0.0000    0.0000


cost =

         0    2.1252    1.0616
```

The second test example is a 6×6, Hankel structured matrix, whose true value has rank 3. In this case the singular value decomposition-based method h2ss and the local optimization based method slra are also heuristics for solving the Hankel structured low rank approximation problem and give, respectively, suboptimal and locally optimal results.

101b ⟨*Test* slra_nn *on Hankel structured problem* 101b⟩≡

```
m = 6; n = 6; r0 = 3; nl = 0.1; structure = 'hankel';
test_slra_nn
```

The output of the test_slra_nn is given in Table 3.1, right. It again shows that the approximations have numerical rank matching the specification but the nuclear norm heuristic gives more than two times larger approximation error. The corresponding trade-off curve is shown in Figure 3.1, right plot.

Table 3.2 Output of test_slra_nn in example 2.

```
sv =
    0.4423    0.3720    0.4428    0.4419
    0.2981    0.2440    0.2975    0.2977
    0.0620    0.0161    0.0500    0.0568
    0.0311    0.0000    0.0000    0.0000
    0.0216    0.0000    0.0000    0.0000
    0.0059    0.0000    0.0000    0.0000

cost =
         0    0.0713    0.0337    0.0323
```

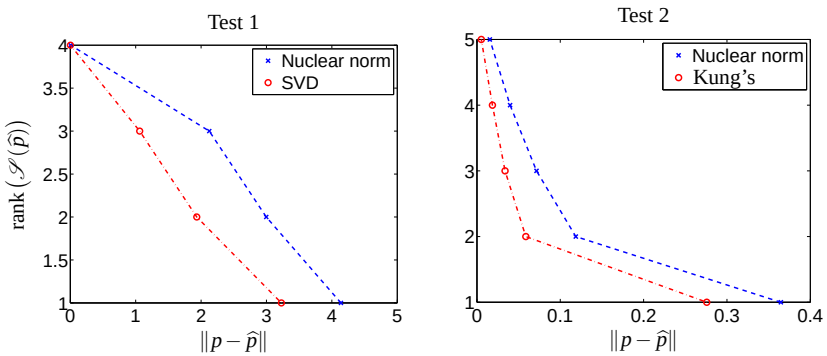

Fig. 3.1 Rank vs approximation error trade-off curve.

Missing data estimation

A random rank deficient "true" data matrix is constructed, where the matrix dimensions $m \times n$ and its rank r_0 are simulation parameters.

102a ⟨test_missing_data 102a⟩≡

```
p0 = rand(m, r0) * rand(r0, n); p0 = p0(:);
```

The true matrix is unstructured:

102b ⟨test_missing_data 102a⟩+≡

```
np = m * n; tts = reshape(1:np, m, n);
```

and is perturbed by a sparse matrix with sparsity level od, where od is a simulation parameter. The perturbation has constant values nl (a simulation parameter) in order to simulate outliers.

102c ⟨test_missing_data 102a⟩+≡

```
pt = zeros(np, 1); no = round(od * np);
I  = randperm(np); I = I(1:no);
pt(I) = nl * ones(no, 1); p = p0 + pt;
```

Under certain conditions, derived in (Candés et al, 2009), the problem of recovering the true values from the perturbed data can be solved *exactly* by the regularized nuclear norm heuristic with 1-norm regularization and regularization parameter

$$\gamma = \sqrt{\frac{1}{\min(m,n)}}.$$

103a ⟨test_missing_data 102a⟩+≡
```
ph1 = nucnrm(tts, p, 1 / sqrt(max(m, n)), 1, eye(np));
```
Uses nucnrm 97a.

In addition, the method can deal with missing data at known locations by setting the corresponding entries of the weight vector w to zero. In order to test this feature, we simulate the first element of the data matrix as missing and recover it by the approximation problem

103b ⟨test_missing_data 102a⟩+≡
```
p(1) = 0;
ph2 = nucnrm(tts, p, 1 / sqrt(max(m, n)), ...
             1, diag([0; ones(np - 1, 1)]));
```
Uses nucnrm 97a.

We check the approximation errors $\|p_0 - \widehat{p}\|_2$, were p_0 is the true value of the data parameter and $\widehat{p}$ is its approximation, obtained by the nuclear norm heuristic method. For comparison, we print also the perturbation size $\|p_0 - p\|_2$.

103c ⟨test_missing_data 102a⟩+≡
```
[norm(p0 - ph1)  norm(p0 - ph2) norm(p0 - p)]
```
Finally, we check the error of the recovered missing value.

103d ⟨test_missing_data 102a⟩+≡
```
[abs(p0(1) - ph1(1)) abs(p0(1) - ph2(1))]
```
The result of a particular test

103e ⟨*Test* slra_nn *on small problem with missing data* 103e⟩≡
```
m = 50; n = 100; r0 = 5; nl = 0.4; od = 0.15; test_missing_data
```
is

```
ans =

    0.5891    0.5891   10.9782


ans =

   1.0e-09 *

    0.2715    0.3814
```

showing that the method indeed recovers the missing data exactly.

3.4 Notes and references

Efficient software for structured low rank approximation

The SLICOT library includes high quality FORTRAN implementation of algorithms for Cholesky factorization of positive definite Toeplitz banded matrices. The library is used in a software package (Markovsky and Van Huffel, 2005; Markovsky et al, 2005) for solving structured low rank approximation problems, based on the variable projections approach (Golub and Pereyra, 2003) and Levenberg–Marquardt's algorithm, implemented in MINPACK. This algorithm is globally convergent with a superlinear convergence rate.

Approximate greatest common divisor

An alternative method for solving structured low rank approximation problems, called structured total least norm, has been modified for Sylvester structured matrices and applied to computation of approximate common divisor in (Zhi and Yang, 2004). The structured total least norm approach is different from the approach present in this chapter because it solves directly problem (AGCD) and does not use the elimination step leading to the equivalent problem (AGCD').

Nuclear norm heuristic

The nuclear norm relaxation for solving rank minimization problems (RM) was proposed in (Fazel, 2002). It is a generalization of the ℓ_1-norm heuristic from sparse vector approximation problems to low rank matrix approximation problems. The CVX package is developed and maintained by Grant and Boyd (2008a), see also (Grant and Boyd, 2008b). A Python version is also available (Dahl and Vandenberghe, 2010).

The computational engines of CVX are SDPT3 and SeDuMi. These solvers can deal with a few tens of parameters ($n_p < 100$). An efficient interior point method for solving (NNM), which can deal with up to 500 parameters, is presented in (Liu and Vandenberghe, 2009). The method is implemented in Python.

References

Candés E, Li X, Ma Y, Wright J (2009) Robust principal component analysis? www-stat.stanford.edu/~candes/papers/RobustPCA.pdf

Dahl J, Vandenberghe L (2010) CVXOPT: Python software for convex optimization. URL abel.ee.ucla.edu/cvxopt

Fazel M (2002) Matrix rank minimization with applications. PhD thesis, Elec. Eng. Dept., Stanford University

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

Grant M, Boyd S (2008a) CVX: Matlab software for disciplined convex programming. `stanford.edu/~boyd/cvx`

Grant M, Boyd S (2008b) Graph implementations for nonsmooth convex programs. In: Blondel V, Boyd S, Kimura H (eds) Recent Advances in Learning and Control, Springer, `stanford.edu/~boyd/graph_dcp.html`, pp 95–110

Karmarkar N, Lakshman Y (1998) On approximate GCDs of univariate polynomials. In: Watt S, Stetter H (eds) Journal of Symbolic Computation, vol 26, pp 653–666, special issue on Symbolic Numeric Algebra for Polynomials

Liu Z, Vandenberghe L (2009) Interior-Point method for nuclear norm approximation with application to system identification. SIAM J Matrix Anal Appl 31(3):1235–1256

Markovsky I, Van Huffel S (2005) High-performance numerical algorithms and software for structured total least squares. J of Comput and Appl Math 180(2):311–331

Markovsky I, Van Huffel S, Pintelon R (2005) Block-Toeplitz/Hankel structured total least squares. SIAM J Matrix Anal Appl 26(4):1083–1099

Zhi L, Yang Z (2004) Computing approximate GCD of univariate polynomials by structure total least norm. In: MM Research Preprints, 24, Academia Sinica, pp 375–387