Algorithms and literate programs for weighted low-rank approximation with missing data

I. Markovsky

School of Electronics and Computer Science, University of Southampton
SO17 1BJ, UK, im@ecs.soton.ac.uk

Summary. Linear models identification from data with missing values is posed as a weighted low-rank approximation problem with weights related to the missing values equal to zero. Alternating projections and variable projections methods for solving the resulting problem are outlined and implemented in a literate programming style, using Matlab/Octave's scripting language. The methods are evaluated on synthetic data and real data from the MovieLens data sets.

1 Introduction

Low-rank approximation

We consider the following low-rank approximation problem: given a real matrix D of dimensions $q \times N$ and an integer $\mathtt{m}$, $0 < \mathtt{m} < \min(q, N)$, find a matrix $\widehat{D}$ of the same dimension as D, with rank at most $\mathtt{m}$, that is as "close" to D as possible, *i.e.*,

$$\text{minimize} \quad \text{over } \widehat{D} \quad \text{dist}(D, \widehat{D}) \quad \text{subject to} \quad \text{rank}(\widehat{D}) \leq \mathtt{m}. \qquad (1)$$

The distance $\text{dist}(D, \widehat{D})$ between the given matrix D and its approximation $\widehat{D}$ can be measured by a norm of the approximation error $\Delta D := D - \widehat{D}$, *i.e.*,

$$\text{dist}(D, \widehat{D}) = \|D - \widehat{D}\|.$$

A typical choice of the norm $\|\cdot\|$ is the Frobenius norm

$$\|\Delta D\|_{\mathrm{F}} = \sqrt{\sum_{i=1}^{q} \sum_{j=1}^{N} \Delta d_{ij}^2},$$

i.e., the square root of the sum of squares of the elements. Assuming that a solution to (1) exists, the minimum value is the distance from D to the

manifold of rank-$\mathtt{m}$ matrices and a minimum point $\widehat{D}^*$ is a "best" (in the sense specified by the distance measure "dist") rank-$\mathtt{m}$ approximation of D.

Apart from being an interesting mathematical problem, low-rank approximation has a large range of applications in diverse areas, *e.g.*, in numerical analysis to find a rank estimate that is robust to "small" perturbations on the matrix. The intrinsic reasons for the widespread appearance of low-rank approximation in applications are 1) low-rank approximation has an interpretation as a data modeling tool and 2) any application area where mathematical methods are used is based on a model. Thus low-rank approximation can provide (approximate) models from data, to be used for analysis, filtering, prediction, control, *etc.*, in the application areas.

In order to make a link between low-rank approximation and data modeling, next we define the notion of a linear static model. Let the observed variables be $d_1, \ldots, d_q$ and let $d := \text{col}(d_1, \ldots, d_q)$ be the column vector of these variables. We say that the variables $d_1, \ldots, d_q$ satisfy a *linear static model* if $d \in \mathcal{L}$, where $\mathcal{L}$, the model, is a subspace of the data space—the q-dimensional real vector space $\mathbb{R}^q$. The *complexity* of a linear model is measured by its dimension. Of interest is data fitting by low complexity models, in which case, generally, the model may only fit approximately the data. Consider a set of data points $\mathcal{D} = \{ d^{(1)}, \ldots, d^{(N)} \} \subset \mathbb{R}^q$ and define the data matrix

$$D := \begin{bmatrix} d^{(1)} & \cdots & d^{(N)} \end{bmatrix} \in \mathbb{R}^{q \times N}.$$

Assuming that there are more measurements than data variables, *i.e.*, $q < N$, it is easy to see that $\text{rank}(D) \leq \mathtt{m}$ if and only if all data points satisfy a linear static model of complexity at most $\mathtt{m}$. This fact is the key link between low-rank approximation and data modeling.

The condition that the data satisfies exactly the model is too strong in practice. For example, if a "true" data $\bar{D}$ satisfies a linear model of low-complexity, *i.e.*, $\text{rank}(\bar{D}) < q$, but is measured subject to noise, *i.e.*, $D = \bar{D} + \widetilde{D}$ ($\widetilde{D}$ being the measurement noise), the noisy measurements D generically do not satisfy a linear model of low-complexity, *i.e.*, almost surely, $\text{rank}(D) = q$. In this case, the modeling goal may be to *estimate* the true but unknown low-complexity model generating $\bar{D}$. Another example showing that the condition $\text{rank}(D) < q$ is too strong is when the data is exact but is generated by a nonlinear phenomenon. In this case, the modeling goal may be to *approximate* the true nonlinear phenomenon by a linear model of bounded complexity. In both cases—estimation and approximation—the data modeling problem leads to low-rank approximation—the rank constraint ensures that the approximation $\widehat{D}$ satisfies exactly a low-complexity linear model. In the estimation example, this takes into account the prior knowledge about the true data generating phenomenon. The approximation criterion "$\min \text{dist}(D, \widehat{D})$" ensures that the obtained model approximates "well" the data. In the estimation case, this corresponds to prior knowledge that the noise is zero mean and "small" in some sense.

Note 1 (Link to the principal component analysis). It can be shown that the well known principal component analysis method is equivalent to low-rank approximation in the Frobenius norm. The number of principal components in the principal component analysis corresponds to the rank constraint in the low-rank approximation problem and the span of the principal components corresponds to the column span of the approximation $\widehat{D}$, *i.e.*, the model. Principal component analysis is typically presented and motivated in a stochastic context, however, the stochastic point of view is not essential and the method is also applicable as a deterministic approximation method.

Note 2 (Link to regression). The classical approach for data fitting involves, in addition to the rank constraint, a priori chosen input/output partition of the variables $\mathrm{col}(a, b) := \Pi d$, where Π is a permutation matrix. Then the low-rank approximation problem reduces to the problem of solving approximately an overdetermined system of equations $AX \approx B$ (from a stochastic point of view—regression), where $\begin{bmatrix} A & B \end{bmatrix} := (\Pi D)^{\top}$. By choosing specific fitting criteria, the classical approach leads to well know optimization problems, *e.g.*, linear least squares, total least squares, robust least squares, and their numerous variations. The total least squares problem [6] is generically equivalent to low-rank approximation in the Frobenius norm.

Missing data

A more general approximation criterion than $\|\Delta D\|_{\mathrm{F}}$ is the element-wise weighted norm of the error matrix

$$\|\Delta D\|_{\Sigma} := \|\Sigma \odot \Delta D\|_{\mathrm{F}}, \qquad \text{where } \odot \text{ denotes element-wise product}$$

and $\Sigma \in \mathbb{R}^{q \times N}$ has positive elements. The low-rank approximation problem (1) with $\mathrm{dist}(D, \widehat{D}) = \|D - \widehat{D}\|_{\Sigma}$, $\Sigma > 0$, is called (regular) weighted low-rank approximation [3, 19, 11, 13, 14]. The weights σ_{ij} allow us to emphasise or de-emphasise the importance of the individual elements of the data matrix D. If σ_{ij} is small, relative to the other weights, then the influence of d_{ij} on the approximation $\widehat{D}$ is small and vice verse.

In the extreme case of a zero weight, *e.g.*, $\sigma_{ij} = 0$, the corresponding element d_{ij} of D is not taken into account in the approximation and therefore it may be missing. In this case, however, $\|\cdot\|_{\Sigma}$ is no longer a norm and the approximation problem is called singular. The above cited work on the weighted low-rank approximation problem treats the regular case and the methods fail in the singular case. The purpose of this paper is to extend the solution methods, derived for the regular weighted low-rank approximation problem to the singular case, so that these algorithms can treat missing data.

Note 3 (Missing rows and columns). The case of missing rows and/or columns of the data matrix is easy to take into account. It reduces the original singular problem to a smaller dimensional regular problem. The same reduction, however, is not possible when the missing elements have no simple pattern.

Low-rank approximation with missing data occurs in

- factor analysis of data from questioners due to questions left answered,
- computer vision due to occlusions,
- signal processing due to irregular measurements in time/space, and
- control due to malfunction of measurement devices.

An iterative solution method (called criss-cross multiple regression) for factor analysis with missing data was developed by Gabriel and Zamir [4]. Their method, however, does not necessarily converge to a minimum point (see the discussion in Section 6, page 491 of [4]). Grung and Manne proposed an alternating projections algorithm for the case of unweighted approximation with missing values, *i.e.*, $\sigma_{ij} \in \{0, 1\}$. Their method was further generalized by Srebro [18] for arbitrary weights.

In this paper, apart from the alternating projections algorithm, we consider an algorithm for weighted low-rank approximation with missing data, based on the variable projections method [5]. The former has linear local convergence rate while the latter has super linear convergence rate which suggests that it may be faster. In addition, we present an implementation of the two algorithms in a literate programming style. A literate program is a combination of computer executable code and human readable description of this code [9]. From the source file, the user extracts both the computer code and its documentation. We use Matlab/Octave's scripting language for the computer code, LaTeX for its documentation, and `noweb` [16] for their combination.

2 Low-rank approximation with uniform weights

Low-rank approximation in the Frobenius norm (equivalently weighted low-rank approximation uniform weights $\sigma_{ij} = \sigma$ for all i, j) can be solved analytically in terms of the singular value decomposition (SVD) of the data matrix D.

Lemma 1 (Matrix approximation lemma). *Let*

$$D = U\Sigma V^{\top}, \qquad \Sigma =: \mathrm{diag}(\sigma_1, \ldots, \sigma_q)$$

be the SVD of $D \in \mathbb{R}^{q \times N}$ and partition the matrices U, Σ, and V as follows:

$$U =: \overset{\mathtt{m} \quad \mathtt{p}}{\begin{bmatrix} U_1 & U_2 \end{bmatrix}} \, q, \quad \Sigma =: \overset{\mathtt{m} \quad \mathtt{p}}{\begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix}} \begin{matrix} \mathtt{m} \\ \mathtt{p} \end{matrix} \quad and \quad V =: \overset{\mathtt{m} \quad \mathtt{p}}{\begin{bmatrix} V_1 & V_2 \end{bmatrix}} \, N, \qquad (2)$$

where $\mathtt{m} \in \mathbb{N}$, $0 \leq \mathtt{m} \leq \min(q, N)$, and $\mathtt{p} := q - \mathtt{m}$. Then

$$\widehat{D}^* = U_1 \Sigma_1 V_1^{\top}$$

is an optimal in the Frobenius norm rank-$\mathtt{m}$ approximation of D, i.e.,

$$\|D - \widehat{D}^*\|_{\mathrm{F}} = \sqrt{\sigma_{\mathtt{m}+1}^2 + \cdots + \sigma_q^2} = \min_{\mathrm{rank}(\widehat{D}) \leq \mathtt{m}} \|D - \widehat{D}\|_{\mathrm{F}}.$$

The solution $\widehat{D}^*$ is unique if and only if $\sigma_{\mathtt{m}+1} \neq \sigma_{\mathtt{m}}$.

From a data modeling point of view of primary interest is the subspace

$$\mathrm{col\,span}(\widehat{D}^*) = \mathrm{col\,span}(U_1) \qquad (3)$$

rather than the approximation $\widehat{D}$.

5a ⟨*Matrix approximation (SVD)* 5a⟩≡ (5c)
```
[u,s,v] = svds(d,m);
p = u(:,1:m); % basis for the optimal model for D
```

Note that the subspace (3) depends only on the left singular vectors of D. Therefore, the model (3) is optimal for the data DQ, where Q is any orthogonal matrix. Let

$$D = \begin{bmatrix} R_1 & 0 \end{bmatrix} Q^\top \qquad (4)$$

be the QR factorization of D (R_1 is lower triangular.) By the above argument, we can model R_1 instead of D. For $N \gg q$, computing the QR factorization (4) and the SVD of R_1 is a more efficient alternative for finding an image representation of the optimal subspace than computing the SVD of D.

5b ⟨*Data compression (QR)* 5b⟩≡ (5c)
```
if nargout == 1
  d = triu(qr(d'))'; % = R,  where D = QR
  d = d(:,1:q);      % = R1, where R = [R1 0]
end
```

Putting the matrix approximation and data compression code together, we have the following function for low-rank approximation.

5c ⟨`lra` 5c⟩≡ (9)
```
⟨lra header 16⟩
function [p,l] = lra(d,m)

d(isnan(d)) = 0; % Convert missing elements (NaNs) to 0s
[q,N] = size(d); % matrix dimension

⟨Data compression (QR) 5b⟩
⟨Matrix approximation (SVD) 5a⟩
if nargout == 2
  s = diag(s); % column vector
  l = s(1:m,ones(1,N)) .* v(:,1:m)'; % diag(S) * V'
end
```

3 Algorithms

In this section, we consider the weighted low-rank approximation problem

$$\text{minimize} \quad \text{over } \widehat{D} \quad \|D - \widehat{D}\|_{\Sigma}^2 \quad \text{subject to} \quad \mathrm{rank}(\widehat{D}) \leq \mathtt{m}, \qquad (5)$$

where the weight matrix $\Sigma \in \mathbb{R}^{q \times N}$ has *nonnegative* elements. The rank constraint can be represented as follows:

$$\mathrm{rank}(\widehat{D}) \leq \mathtt{m} \quad \Longleftrightarrow \quad \text{there are } P \in \mathbb{R}^{q \times \mathtt{m}} \text{ and } L \in \mathbb{R}^{\mathtt{m} \times N},$$
$$\text{such that } \widehat{D} = PL, \quad (6)$$

which turns problem (5) into the following parameter optimization problem

$$\text{minimize} \quad \text{over } P \in \mathbb{R}^{q \times \mathtt{m}} \text{ and } L \in \mathbb{R}^{\mathtt{m} \times N} \quad \|D - PL\|_{\Sigma}^2. \qquad (7)$$

Unfortunately the problem is nonconvex and there are no efficient methods to solve it. Next we present two local optimization approaches for finding locally optimal solutions, starting from a given initial approximation.

3.1 Alternating projections

The first solution method is motivated by the fact that (7) is linear separately in either P or L. Indeed, by fixing either P or L in (7) the minimization over the free parameter is a (singular) weighted least squares problem, which can be solved globally and efficiently. This suggests an iterative solution method that alternates between the solution of the two weighted least squares problems. The solution of the weighted least squares problems can be interpreted as weighted projections, thus the name of the method—alternating projections.

The alternating projections method is started from an initial guess of one of the parameters P or L. An initial guess is a possibly suboptimal solution of (7), computed by a direct method. Such a solution can be obtained for example by solving the unweighted low-rank approximation problem where all missing elements are filled in by zeros. On each iteration step of the alternating projections algorithm, the cost function value is guaranteed to be non-increasing and is typically decreasing. It can be shown that the iteration converges [10, 8] and that the local convergence rate is linear.

A summary of the alternating projections method is given in Algorithm 1. We use the following Matlab-like notation for indexing a matrix. For a $q \times N$ matrix D and subsets $\mathcal{I}$ and $\mathcal{J}$ of the sets of, respectively, row and column indexes, $D_{\mathcal{I},\mathcal{J}}$ denotes the submatrix of D with elements whose indexes are in $\mathcal{I}$ and $\mathcal{J}$. Either of $\mathcal{I}$ and $\mathcal{J}$ can be replaced by ":" in which case all rows/columns are indexed.

The quantity $e^{(k)}$, computed on step 9 of the algorithm is the squared approximation error

$$e^{(k)} = \|D - D^{(k)}\|_\Sigma^2$$

on the kth iteration step. Convergence of the iteration is judged on the basis of the relative decrease of $e^{(k)}$ after an update step. This corresponds to choosing a tolerance on the relative decrease of the cost function value. More expensive alternatives are to check the convergence of the approximation $\widehat{D}^{(k)}$ or the size of the gradient of the cost function with respect to the model parameters.

3.2 Variable projections

In the second solution methods, we view (7) as a double minimization problem

$$\text{minimize} \quad \text{over } P \in \mathbb{R}^{q \times \mathtt{m}} \quad \underbrace{\min_{L \in \mathbb{R}^{\mathtt{m} \times N}} \|D - PL\|_\Sigma^2}_{f(P)}. \tag{8}$$

The inner minimization is a weighted least squares problem and therefore can be solved in closed form. Using Matlab's set indexing notation, the solution is

$$f(P) = \sum_{j=1}^N D_{\mathcal{J},j}^\top \operatorname{diag}(\Sigma_{\mathcal{J},j}^2) P_{\mathcal{J},:} \left(P_{\mathcal{J},:}^\top \operatorname{diag}(\Sigma_{\mathcal{J},j}^2) P_{\mathcal{J},:} \right)^{-1} P_{\mathcal{J},:}^\top \operatorname{diag}(\Sigma_{\mathcal{J},j}^2) D_{\mathcal{J},j},$$

$$\tag{9}$$

where $\mathcal{J}$ is the set of indexes of the non-missing elements in the jth column of D.

The outer minimization is a nonlinear least squares problem and can be solved by general purpose local optimization methods. There are local optimization methods, *e.g.*, the Levenberg–Marquardt method [15], that guarantee global convergence (to a locally optimal solution) with super-linear convergence rate. Thus, if implemented by such a method and started "close" to a locally optimal solution, the variable projections method requires fewer iterations than the alternating projections method.

The inner minimization can be viewed as a weighted projection on the subspace spanned by the columns of P. Consequently $f(P)$ has the geometric interpretation of the sum of squared distances from the data points to the subspace. Since the parameter P is modified by the outer minimization, the projections are on a varying subspace, hence the name of the method—variable projections.

Note 4 (Gradient and Hessian of f). In the implementation of the method in this version of the paper, we are using finite difference numerical computation of the gradient and Hessian of f. (These approximations are computed by the optimization method.) More efficient alternative, however, is to supply to the method analytical expressions for the gradient and the Hessian. This will be done in later versions of the paper. Please refer to [12] for the latest version.

Algorithm 1 Alternating projections algorithm for weighted low-rank approximation with missing data.

Input: Data matrix $D \in \mathbb{R}^{q \times N}$, rank constraint $\mathtt{m}$, elementwise nonnegative weight matrix $\Sigma \in \mathbb{R}^{q \times N}$, and relative convergence tolerance ε.

1: Initial approximation: compute the Frobenius norm low-rank approximation of D with missing elements filled in with zeros

$$P^{(0)} := \mathtt{lra}(D, \mathtt{m}).$$

2: Let $k := 0$.
3: **repeat**
4:     Let $e^{(k)} := 0$.
5:     **for** $j = 1, \ldots, N$ **do**
6:         Let $\mathcal{J}$ be the set of indexes of the non-missing elements in $D_{:,j}$.
7:         Define

$$c := \operatorname{diag}(\Sigma_{\mathcal{J},j}) D_{\mathcal{J},j} = \Sigma_{\mathcal{J},j} \odot D_{\mathcal{J},j}$$
$$P := \operatorname{diag}(\Sigma_{\mathcal{J},j}) P_{\mathcal{J},:}^{(k)} = (\Sigma_{\mathcal{J},j} \mathbf{1}_{\mathtt{m}}^\top) \odot P_{\mathcal{J},:}^{(k)}.$$

8:         Compute

$$l_j^{(k)} := \left(P^\top P \right)^{-1} P^\top c.$$

9:         Let

$$e^{(k)} := e^{(k)} + \|c - P l_j^{(k)}\|^2.$$

10:     **end for**
11:     Define

$$L^{(k)} = \begin{bmatrix} l_1^{(k)} & \cdots & l_N^{(k)} \end{bmatrix}.$$

12:     Let $e^{(k+1)} := 0$.
13:     **for** $i = 1, \ldots, q$ **do**
14:         Let $\mathcal{I}$ be the set of indexes of the non-missing elements in the ith row $D_{i,:}$.
15:         Define

$$r := D_{i,\mathcal{I}} \operatorname{diag}(\Sigma_{i,\mathcal{I}}) = D_{i,\mathcal{I}} \odot \Sigma_{i,\mathcal{I}}$$
$$L := L_{:,\mathcal{I}}^{(k+1)} \operatorname{diag}(\Sigma_{i,\mathcal{I}}) = L_{:,\mathcal{I}}^{(k+1)} \odot (\mathbf{1}_{\mathtt{m}} \Sigma_{i,\mathcal{I}}).$$

16:         Compute

$$p_i^{(k+1)} := r L^\top \left(L L^\top \right)^{-1}.$$

17:         Let

$$e^{(k+1)} := e^{(k+1)} + \|r - p_i^{(k+1)} L\|^2.$$

18:     **end for**
19:     Define

$$P^{(k+1)} = \begin{bmatrix} p_1^{(k+1)} \\ \vdots \\ p_q^{(k+1)} \end{bmatrix}.$$

20:     $k = k + 1$.
21: **until** $|e^{(k)} - e^{(k-1)}|/e^{(k)} < \varepsilon$.
Output: Locally optimal solution $\widehat{D} = D^{(k)} := P^{(k)} L^{(k)}$ of (7).

---

4 Implementation

Both the alternating projections and the variable projections methods for solving weighted low-rank approximation problems with missing data are callable through the function `wlra`.

9   ⟨`wlra` 9⟩≡

```
⟨wlra header 17b⟩
function [p,l,info] = wlra(d,m,s,opt)

tic % measure the execution time
⟨Default parameters opt 17a⟩
switch lower(opt.Method)
case {'altpro','ap'}
  ⟨Alternating projections method 10a⟩
case {'varpro','vp'}
  ⟨Variable projections method 11c⟩
otherwise
  error('Unknown method %s',opt.Method)
end
info.time = toc; % execution time

⟨lra 5c⟩ % needed for the initial approximation
⟨Cost function 11d⟩ % needed for the variable projections method
```

The output parameter `info` gives the approximation error $\|D - \widehat{D}\|^2_\Sigma$ (`info.err`), the number of iterations (`info.iter`), and the execution time (`info.time`) for computing the local approximation $\widehat{D}$. The optional parameter `opt` specifies which method and (in the case of the variable projections) which algorithm is to be used (`opt.Method` and `opt.Algorithm`), the initial approximation (`opt.P`), the convergence tolerance ε (`opt.TolFun`), an upper bound on the number of iterations (`opt.MaxIter`), and the level of printed information (`opt.Display`).

The initial approximation `opt.P` is a $q \times$ `m` matrix, such that the columns of $P^{(0)})$ form a basis for the span of the columns of $D^{(0)}$, where $D^{(0)}$ is the initial approximation of D, see step 1 in Algorithm 1. If it is not provided via the parameter `opt`, the default initial approximation is chosen to be the unweighted low-rank approximation of the data matrix with all missing elements filled in with zeros.

Note 5 (Large scale, sparse data). In an application of (5) to building recommender systems [17], the data matrix D is large (q and N are several hundreds of thousands) but only a small fraction of the elements (*e.g.*, one percent) are given. Such problems can be handled efficiently, encoding D and Σ as sparse matrices. The convention in this case is that missing elements are zeros. Of course, the S matrix indicates that they should be treated as missing. Thus the convention is a hack allowing us to use the powerful tool of sparse matrix representation and linear algebra available in Matlab/Octave.

4.1 Alternating projections

The iteration loop for the alternating projections algorithm is:

10a   ⟨Alternating projections method 10a⟩≡   (9)

```
[q,N] = size(d); % define q and N
switch lower(opt.Display)
  case {'iter'}, sd = norm(s.*d,'fro')^2; % size of D
end

% Main iteration loop
k     = 0; % iteration counter
cont = 1;
while (cont)
    ⟨Compute L, given P 10b⟩
    ⟨Compute P, given L 10c⟩
    ⟨Check exit condition 11a⟩
    ⟨Print progress information 11b⟩
end
info.err  = el; % approximation error
info.iter = k;  % number of iterations
```

The main computational steps on each iteration of the algorithm are the two weighted least squares problems.

10b   ⟨Compute L, given P 10b⟩≡   (10 11)

```
dd = []; % vec(D - DH)
for j = 1:N
    J   = find(s(:,j));
    sJj = full(s(J,j));
    c   = sJj .* full(d(J,j));
    P   = sJj(:,ones(1,m)) .* p(J,:); % = diag(sJj) * p(J,:)
    l(:,j) = P \ c;
    dd  = [dd; c - P*l(:,j)];
end
ep = norm(dd)^2;
```

10c   ⟨Compute P, given L 10c⟩≡   (10a)

```
dd = []; % vec(D - DH)
for i = 1:q
    I   = find(s(i,:));
    sIi = full(s(i,I));
    r   = sIi .* full(d(i,I));
    L   = sIi(ones(m,1),:) .* l(:,I); % = l(:,I) * diag(sIi)
    p(i,:) = r / L;
    dd  = [dd, r - p(i,:)*L];
end
el = norm(dd)^2;
```

The convergence is checked by the size of the relative decrease in the approximation error $e^{(k)}$ after one update step.

11a   ⟨*Check exit condition* 11a⟩≡   (10a)

```
    k    = k + 1;
    re   = abs(el - ep) / el;
    cont = (k < opt.MaxIter) & (re > opt.TolFun) & (el > eps);
```

If the optimal parameter `opt.Display` is set to `'iter'`, `wlra` prints on each iteration step the relative approximation error.

11b   ⟨*Print progress information* 11b⟩≡   (10a)

```
    switch lower(opt.Display)
      case 'iter', fprintf('%2d : relative error = %18.8f\n', k, el/sd)
    end
```

4.2 Variable projections

We use Matlab's Optimization Toolbox for performing the outer minimization in (8), *i.e.*, the nonlinear minimization over the P parameter. The parameter `opt.Algorithm` specifies the algorithm to be used. The available options are `fminunc` — a quasi-Newton type algorithm, and `fminunc` — a nonlinear least squares algorithm. Both algorithm allow for numerical approximation of the gradient and Hessian/Jacobian through finite difference computations. In the current version of the code, we use the numerical approximation.

11c   ⟨*Variable projections method* 11c⟩≡   (9)

```
    switch lower(opt.Algorithm)
      case {'fminunc'}
        [p,err,f,info] = fminunc(@(p)wlra_err(p,d,s),p,opt);
      case {'lsqnonlin'}
        [p,rn,r,f,info] = lsqnonlin(@(p)wlra_err_mat(p,d,s),p,[],[]);
      otherwise
        error('Unknown algorithm %s.',opt.Algorithm)
    end
    [info.err,l] = wlra_err(p,d,s); % in order to obtain the L parameter
```

The inner minimization in (8) has an analytical solution (9). The implementation of (9) is actually the chunk of code for computing the L parameter, given the P parameter, already used in the alternating projections algorithm.

11d   ⟨*Cost function* 11d⟩≡   (9) 11e▷

```
    function [ep,l] = wlra_err(p,d,s)
    N = size(d,2); m = size(p,2);
    ⟨Compute L, given P 10b⟩
```

In the case of using a nonlinear least squares type algorithm, the cost function is not the sum of squares of the errors but the vector of the errors `dd`.

11e   ⟨*Cost function* 11d⟩+≡   (9) ◁11d

```
    function dd = wlra_err_mat(p,d,s)
    N = size(d,2); m = size(p,2);
    ⟨Compute L, given P 10b⟩
```

5 Test on simulated data

A "true" random rank-m matrix $\bar{D}$ is selected by generating randomly its factors $\bar{P}$ and $\bar{L}$ in a rank revealing factorization $\bar{D} = \bar{P}\bar{L}$, where $\bar{P} \in \mathbb{R}^{q \times m}$ and $\bar{L} \in \mathbb{R}^{m \times N}$.

12a   ⟨test 12a⟩≡   12b▷

```
    randn('state',0); rand('state',0);
    p0 = rand(q,m); l0 = rand(m,N);    % true data matrix
```

The location of the given elements is chosen randomly row by row. The number of given elements is such that the sparsity of the resulting matrix, defined as the ratio of the number of missing elements to the total number qN of elements, matches the specification `r`.

12b   ⟨test 12a⟩+≡   ◁12a 12c▷

```
    ne  = round((1-r)*q*N);  % number of given elements
    ner = round(ne/q);        % number of given elements per row
    I = []; J = [];           % row/column indeces of the given elements
    for i = 1:q
      I = [I i*ones(1,ner)]; % all selecteded elements are in the ith row
      rp = randperm(N);
      J = [J rp(1:ner)];      % and have random column indexes
    end
    ne = length(I);
```

By construction there are `ner` given elements in each row of the data matrix, however, there may be columns with a few (or even zero) given elements. Columns with less than `m` given elements can not be recovered from the given observations, even when the data is noise-free. Therefore, we remove such columns from the data matrix.

12c   ⟨test 12a⟩+≡   ◁12b 12d▷

```
    % Find indexes of columns with less than M given elements
    tmp = (1:N)';
    J_del = find(sum(J(ones(N,1),:) == tmp(:,ones(1,ne)),2) < m);
    % Remove them
    l0(:,J_del) = [];
    % Redefine I and J
    tmp = sparse(I,J,ones(ne,1),q,N); tmp(:,J_del) = [];
    [I,J] = find(tmp); N = size(l0,2);
```

Next, we construct a noisy data matrix with missing elements by adding to the true values of the given data elements independent, identically, distributed, zero mean, Gaussian noise, with a specified standard deviation `s`. The weight matrix Σ is binary: $\sigma_{ij} = 1$ if d_{ij} is given and $\sigma_{ij} = 1$ if d_{ij} is missing.

12d   ⟨test 12a⟩+≡   ◁12c 13a▷

```
    d0 = p0 * l0;        % full true data matrix
    Ie = I + q * (J-1); % indexes of the given elements from d0(:)
    d  = zeros(q*N,1);  d(Ie) = d0(Ie) + sigma*randn(size(d0(Ie)));
    d  = reshape(d,q,N);
    s  = zeros(q,N); s(Ie) = 1;
```

We apply the methods implemented in `lra` and `wlra` on the noisy data matrix D with missing elements and validate the results against the complete true data matrix $\bar{D}$.

13a   ⟨test 12a⟩+≡   ◁12d 13b▷

```
tic, [p0,l0] = lra(d,m); t0 = toc;
err0 = norm(s.*(d - p0*l0),'fro')^2; e0 = norm(d0 - p0*l0,'fro')^2;
[ph1,lh1,info1] = wlra(d,m,s);     e1 = norm(d0 - ph1*lh1,'fro')^2;
opt.Method = 'vp'; opt.Algorithm = 'fminunc';
[ph2,lh2,info2] = wlra(d,m,s,opt); e2 = norm(d0 - ph2*lh2,'fro')^2;
opt.Method = 'vp'; opt.Algorithm = 'lsqnonlin';
[ph3,lh3,info3] = wlra(d,m,s,opt); e3 = norm(d0 - ph3*lh3,'fro')^2;
```

For comparison, we use also a method for low-rank matrix completion, called singular value thresholding (SVT) [1]. Low-rank matrix completion is a low-rank approximation problem with missing data for exact data, *i.e.*, data of a low-rank matrix. Although the SVT method is initially designed for the exact case, it is demonstrated to cope with noisy data as well, *i.e.*, solve low-rank approximation problems with missing data. The method is based on convex relaxation of the rank constraint and does not require an initial approximation. A Matlab implementation of the SVT method is available at `http://svt.caltech.edu/`

13b   ⟨test 12a⟩+≡   ◁13a 13c▷

```
tau = 5*sqrt(q*N); delta = 1.2/(ne/q/N); % SVT calling parameters
try
    tic, [U,S,V] = SVT([q N],Ie,d(Ie),tau,delta); t4 = toc;
    dh4  = U(:,1:m)*S(1:m,1:m)*V(:,1:m)'; % approximation
catch
    dh4 = NaN; t4 = NaN; % SVT not installed
end
err4 = norm(s.*(d - dh4),'fro')^2; e4 = norm(d0 - dh4,'fro')^2;
```

The final result shows the relative approximation error $\|D - \widehat{D}\|_{\Sigma}^2/\|D\|_{\Sigma}^2$, the estimation error $\|\bar{D} - \widehat{D}\|_{\mathrm{F}}^2/\|\bar{D}\|_{\mathrm{F}}^2$, and the computation time for the five methods.

13c   ⟨test 12a⟩+≡   ◁13b

```
nd = norm(s.*d,'fro')^2; nd0 = norm(d0,'fro')^2;
format long
res = [err0/nd info1.err/nd  info2.err/nd  info3.err/nd err4/nd;
       e0/nd0   e1/nd0        e2/nd0        e3/nd0        e4/nd0;
       t0       info1.time    info2.time    info3.time    t4]
```

First, we call the test script with exact (noise free) data.

13d   ⟨*Experiment 1: small sparsity, exact data* 13d⟩≡

```
q = 10; N = 100; m = 2; r = 0.1; sigma = 0; test
```

Table 1. Results for Experiment 1.

	lra	ap	vp + fminunc	vp + lsqnonlin	SVT
$\|D - \widehat{D}\|_{\Sigma}^2/\|D\|_{\Sigma}^2$	0.02	10^{-19}	10^{-12}	10^{-17}	10^{-8}
$\|\bar{D} - \widehat{D}\|_{\mathrm{F}}^2/\|D\|_{\mathrm{F}}^2$	0.03	10^{-20}	10^{-12}	10^{-17}	10^{-8}
Execution time (sec)	0.01	0.05	2	3	0.37

The experiment corresponds to a matrix completion problem [2]. The results, summarized in Tables 1, show that all methods, except for `lra`, complete correctly the missing elements. As proved by Candés in [2], exact matrix completion is indeed possible in the case of Experiment 1.

The second experiment is with noisy data.

14a   ⟨*Experiment 2: small sparsity, noisy data* 14a⟩≡

```
q = 10; N = 100; m = 2; r = 0.1; sigma = 0.1; test
```

The results, shown in Tables 2, indicate that the methods implemented in `wlra` converge to the same (locally) optimal solution. The relative prediction error is less than 6%. The alternating projections method is about 100 times faster than the variable projections methods, using the Optimization Toolbox functions `fminunc` and `lsqnonlin`. The solution produces by the SVT method is close to being optimal. The SVT method is about 10 times slower than the alternating projections method. The solution of `lra` is far from optimal.

Table 2. Results for Experiment 2.

	lra	ap	vp + fminunc	vp + lsqnonlin	SVT
$\|D - \widehat{D}\|_{\Sigma}^2/\|D\|_{\Sigma}^2$	0.037	0.0149	0.0149	0.0149	0.0151
$\|\bar{D} - \widehat{D}\|_{\mathrm{F}}^2/\|D\|_{\mathrm{F}}^2$	0.037	0.0054	0.0054	0.0055	0.0056
Execution time (sec)	0.01	0.03	2	3	0.39

In the third experiment we keep the noise standard deviation the same as in Experiment 2 but increase the sparsity.

14b   ⟨*Experiment 3: bigger sparsity, noisy data* 14b⟩≡

```
q = 10; N = 100; m = 2; r = 0.4; sigma = 0.1; test
```

The results, shown in Tables 3, again indicate that the methods implemented in `wlra` converge to the same (locally) optimal solutions. In this case, the SVT method is further away from being (locally) optimal, but is still much better than the solution of `lra` — 1% vs 25% prediction error.

Table 3. Results for Experiment 3

	lra	ap	vp + fminunc	vp + lsqnonlin	SVT
$\|D - \widehat{D}\|_{\Sigma}^2/\|D\|_{\Sigma}^2$	0.16	0.0133	0.0133	0.0133	0.0157
$\|\bar{D} - \widehat{D}\|_{\mathrm{F}}^2/\|D\|_{\mathrm{F}}^2$	0.25	0.0095	0.0095	0.0095	0.0106
Execution time (sec)	0.01	0.04	3	5	0.56

6 Test on the MoviLens data

The MoviLens date sets [7] were collected and published by the GroupLens Research Project at the University of Minnesota about during in 1998. Currently, they are recognized as a benchmark for predicting missing data in recommender systems. The "100K data set" consists of 100000 ratings of $q = 943$ users' on $N = 1682$ movies and demographic information for the users. (The ratings are encoded by integers in the range from 1 to 5.) In this paper, we use only the ratings, which constitute a $q \times N$ matrix with missing elements. The task is to fill in the missing elements.

Assuming that the true complete data matrix is rank deficient, building a recommender system is a problem of low-rank approximation with missing elements. The assumption that the true data matrix is low-rank is reasonable in practice because user ratings are influences by a few factors. Thus, we can identify typical users (related to different combinations of factors) and reconstruct the ratings of any user as a linear combination of the ratings of the typical users. As long as the typical users are fewer than the number of users, the data matrix is low-rank. In reality, the number of factors is not small but there are a few dominant ones, so that the true data matrix is only approximately low-rank.

It turns out that two factors allow us to reconstruct the missing elements with 7.1% average error. The reconstruction results are validated by cross validation with 80% identification data and 20% validation data. Five such partitionings of the data are as given on the MoviLens web site. The matrix $\Sigma_{\mathrm{idt}}^{(k)} \in \{0,1\}^{q \times N}$ indicates the positions of the given elements in the kth partition ($\Sigma_{\mathrm{idt},ij}^{(k)} = 1$ means that the element D_{ij} is used for identification and $\Sigma_{\mathrm{idt},ij}^{(k)} = 0$ means that D_{ij} is missing). Similarly, $\Sigma_{\mathrm{val}}^{(k)}$ indicates the validation elements in the kth partition.

Table 4 shows the mean relative identification and validation errors

$$
e_{\mathrm{idt}} := \frac{1}{5} \sum_{k=1}^{5} \frac{\|D - \widehat{D}^{(k)}\|_{\Sigma_{\mathrm{idt}}^{(k)}}^2}{\|D\|_{\Sigma_{\mathrm{idt}}^{(k)}}^2} \qquad \text{and} \qquad e_{\mathrm{val}} := \frac{1}{5} \sum_{k=1}^{5} \frac{\|D - \widehat{D}^{(k)}\|_{\Sigma_{\mathrm{val}}^{(k)}}^2}{\|D\|_{\Sigma_{\mathrm{val}}^{(k)}}^2},
$$

where $\widehat{D}^{(k)}$ is the reconstructed matrix in the kth partitioning of the data. The SVT method issues a message "Divergence!", which explains the poor results obtained by this method.

Table 4. Results on the MoviLens data.

	lra	ap	SVT
Mean identification error e_{idt}	0.100	0.060	0.298
Mean prediction error e_{val}	0.104	0.071	0.307
Mean execution time (sec)	1.4	156	651

7 Conclusions

Alternating projections and variable projections methods for element-wise weighted low-rank approximation were presented and implemented in a literate programming style. Some of the weights may be set to zero, which corresponds to missing or ignored elements in the data matrix. The problem is of interest for static linear modeling of data with missing elements. The simulation examples suggest that in the current version of the implementation overall most efficient is the alternating projections method, which is applicable to data with a few tens of thousands of rows and columns, provided the sparsity of the given elements is high. In the case of exact data with missing elements, the methods solve a matrix completion problem.

Acknowledgments

A Appendix

16    $\langle$ `lra` *header* 16$\rangle\equiv$ (5c)

```
% LRA - Low-Rank Approximation.
% [PH,LH] = LRA(D,M)
% Finads optimal solution to the problem:
% Minimize over DH norm(D - DH, 'fro')
% Subject to rank(DH) <= M
%
% D  - data matrix of dimension qxN, q < N
% M  - rank constraint, M < q
% PH, LH - PH*LH is the rank-m approximation DH of D
```

17a   ⟨*Default parameters* opt 17a⟩≡   (9)

```
try opt.MaxIter;   catch opt.MaxIter   = 100;   end
try opt.TolFun;    catch opt.TolFun    = 1e-5; end
try opt.Display;   catch opt.Display   = 'off'; end
try opt.Method;    catch opt.Method    = 'ap'; end
try opt.Algorithm; catch opt.Algorithm = 'lsqnonlin'; end
try p = opt.P;     catch
  switch lower(opt.Display)
    case 'iter', fprintf('Computing an initial approximation ...\n')
  end
    p = lra(d,m); % low-rank approximation
end
```

17b   ⟨wlra *header* 17b⟩≡   (9)

```
% WLRA - Weighted Low-Rank Approximation.
% [PH,LH,INFO] = WLRA(D,M,S,OPT)
% Finads locally optimal solution to the problem:
% Minimize over DH norm(S .* (D - DH), 'fro')
% Subject to rank(DH) <= M
%
% D - data matrix of dimension qxN, q < N
% M - rank constraint, M < q
% S - element-wise nonnegative weight matrix of dimension qxN
% OPT  - options for the optimization algorithm
%   OPT.Method has possible values
%     'ap' - alternating projections (default) and
%     'vp' - variable projections (requires Optimization Toolbox)
%   OPT.Algorithm - algorithm for the variable projections
%     'fminunc' - quasi-Newton type method
%     'lsqnonlin' - Levenberg-Marquardt method (default)
%   OPT.P       - initial approximation (default computed via svds)
%   OPT.TolFun  - convergence tolerance for the function value
%   OPT.MaxIter - maximum number of iterations
%   OPT.Display - level of printed information
%     'iter' - prints the cost function value per iteration
% PH, LH - PH*LH is the rank-m approximation DH of D
% INFO - exit information:
%   INFO.err   - approximation error
%   INFO.time  - execution time
%   INFO.iter  - number of iterations performed
%   Note: INFO.iter = OPT.MaxIter indicates lack of convergence
%
% Note: S(i,j) = 0 implies that D(i,j) is missing. Missing elements
% are ignored and in particular can be set to 0. This convention is
% convenient for large sparse data sets when SPARSE repr. is used.
```

References

1. J.-F. Cai, E. Candés, and Z. Shen. A singular value thresholding algorithm for matrix completion. *http://www-stat.stanford.edu/~candes/papers/SVT.pdf*, 2009.
2. E. Candés and B. Recht. Exact matrix completion via convex optimization. *Found. of Comput. Math.*, 2009.
3. B. De Moor. Structured total least squares and L_2 approximation problems. *Linear Algebra Appl.*, 188–189:163–207, 1993.
4. K. Gabriel and S. Zamir. Lower rank approximation of matrices by least squares with any choice of weights. *Technometrics*, 21:489–498, 1979.
5. G. Golub and V. Pereyra. Separable nonlinear least squares: the variable projection method and its applications. *Institute of Physics, Inverse Problems*, 19:1–26, 2003.
6. G. Golub and C. Van Loan. An analysis of the total least squares problem. *SIAM J. Numer. Anal.*, 17:883–893, 1980.
7. GroupLens. Movielens data sets. http://www.grouplens.org/node/73.
8. H. Kiers. Setting up alternating least squares and iterative majorization algorithms for solving various matrix optimization problems. *Comput. Statist. Data Anal.*, 41:157–170, 2002.
9. D. Knuth. *Literate Programming*. Cambridge University Press, 1992.
10. W. Krijnen. Convergence of the sequence of parameters generated by alternating least squares algorithms. *Comput. Statist. Data Anal.*, 51:481–489, 2006.
11. J. Manton, R. Mahony, and Y. Hua. The geometry of weighted low-rank approximations. *IEEE Trans. Signal Process.*, 51(2):500–514, 2003.
12. I. Markovsky. Algorithms and literate programs for weighted low-rank approximation with missing data. Technical Report 18296, ECS, Univ. of Southampton, http://eprints.ecs.soton.ac.uk/18296/, 2009.
13. I. Markovsky, M.-L. Rastello, A. Premoli, A. Kukush, and S. Van Huffel. The element-wise weighted total least squares problem. *Comput. Statist. Data Anal.*, 50(1):181–209, 2005.
14. I. Markovsky and S. Van Huffel. Left vs right representations for solving weighted low rank approximation problems. *Linear Algebra Appl.*, 422:540–552, 2007.
15. D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM J. Appl. Math.*, 11:431–441, 1963.
16. N. Ramsey. Literate programming simplified. *IEEE Software*, 11:97–105, 1994.
17. T. Segaran. *Programming Collective Intelligence: Building Smart Web 2.0 Applications*. O'Reilly Media, 2007.
18. N. Srebro. *Learning with Matrix Factorizations*. PhD thesis, MIT, 2004.
19. P. Wentzell, D. Andrews, D. Hamilton, K. Faber, and B. Kowalski. Maximum likelihood principal component analysis. *J. Chemometrics*, 11:339–366, 1997.