IMPERIAL COLLEGE LONDON, DEPARTMENT OF MATHEMATICS

Scientific Computing (M3SC)

Peter J. Schmid

February 28, 2017

1 RECOMMENDER SYSTEMS (MATRIX COMPLETION ALGORITHM)

In many of today's consumer businesses, it is common practice to provide suggestions for possible purchases based on previous orders. This type of problem is referred to as a **recommender problem**, or more popularly, as the Netflix problem, after the online movie-rental company. The Netflix problem is concerned with computing recommendations for customers, when only a few previous movie rentals and their ratings are known. From a mathematical point of view, we represent the Netflix data-base as a rectangular matrix (see Figure 1.1 for a sketch), where each row represents one customer and each column accounts for a particular movie in the Netflix collection. For each customer (row) we record an entry in the matrix that describes the rating of the rented movie (in the respective column), from 0 for the lowest rating to 5 for the highest. The entire Netflix data-base matrix is then a sparsely filled rectangular matrix of positive values. The question is then on how to fill the remaining entries of the matrix, so that Netflix can make intelligent suggestions for new rentals based on a customer's preferences in movies.

The underlying principle is to complete a matrix based on only a few entries. In the mathematical literature, this is known as the **matrix completion problem**. In general, it is an ill-determined problem, unless additional conditions are imposed on the full matrix to be constructed. For example, if we are asked to fill two missing elements (indicated by $\times$) in the matrix D

$$\mathsf{D} = \begin{bmatrix} 1 & 2 \\ \times & 6 \\ 2 & \times \end{bmatrix} \tag{1.1}$$

we would conclude that this is an ill-posed problem, i.e., we are not given sufficient information to complete the task. However, if we are asked to come up with a rank-one solution, the problem becomes trivial. The matrix D can have a maximal rank of two (two linearly independent columns). By asking for a rank-one matrix, we ask for a matrix with only one independent column; the second column is simply a multiple of the first. Looking at the first row, it becomes clear that the second column is just the double of the first column; we therefore can complete the matrix as follows.

$$\mathsf{D} = \begin{bmatrix} 1 & 2 \\ \times & 6 \\ 2 & \times \end{bmatrix} \quad \longrightarrow \quad \begin{bmatrix} 1 & 2 \\ 3 & 6 \\ 2 & 4 \end{bmatrix} \tag{1.2}$$

The additional constraint of looking for a low-rank (or rank-deficient) matrix made the problem well-posed.

We will apply a similar regularization approach to the Netflix problem. Again, we are using the rank of the recovered matrix, i.e., the number of linearly independent columns of the data matrix, as an additional condition that needs to be satisfied. In different terms, we postulate that the taste in movies can be expressed in simple terms with only a few column vectors (far less than the number of movies in the Netflix collection).
We introduce the notation Ω which denotes the set of (i, j) indices where we have data (all the ratings in Figure 1.1). The ratings themselves are given by N_{ij}.
We can then formulate the recommender problem as follows: we wish to recover a matrix A by solving the following optimization problem:

$$\min \operatorname{rank}(\mathsf{A}), \tag{1.3a}$$
$$\text{subject to} \quad \mathsf{A}_{ij} = \mathsf{D}_{ij} \qquad \{i, j\} \in \Omega, \tag{1.3b}$$

i.e., we minimize the rank of the recovered matrix A while respecting the set Ω of known entries D_{ij}.
The above optimization problem is difficult due to the fact that the rank of a matrix (the number of linearly independent columns or rows) takes on integer values, making the objective in the optimization problem discontinuous, and the optimization problem non-convex. It is advantageous to replace the rank condition by a continuous analog that still measures the rank; this condition is known as the nuclear norm of A, denoted by $\|\mathsf{A}\|_*$. The nuclear norm of a matrix A is defined as the *sum* of its singular values, while the rank is the *number* of non-zero singular values.
The new (convexified) optimization problem is then

$$\min \|\mathsf{A}\|_*, \tag{1.4a}$$
$$\text{subject to} \quad \mathsf{A}_{ij} = \mathsf{D}_{ij} \qquad \{i, j\} \in \Omega, \tag{1.4b}$$

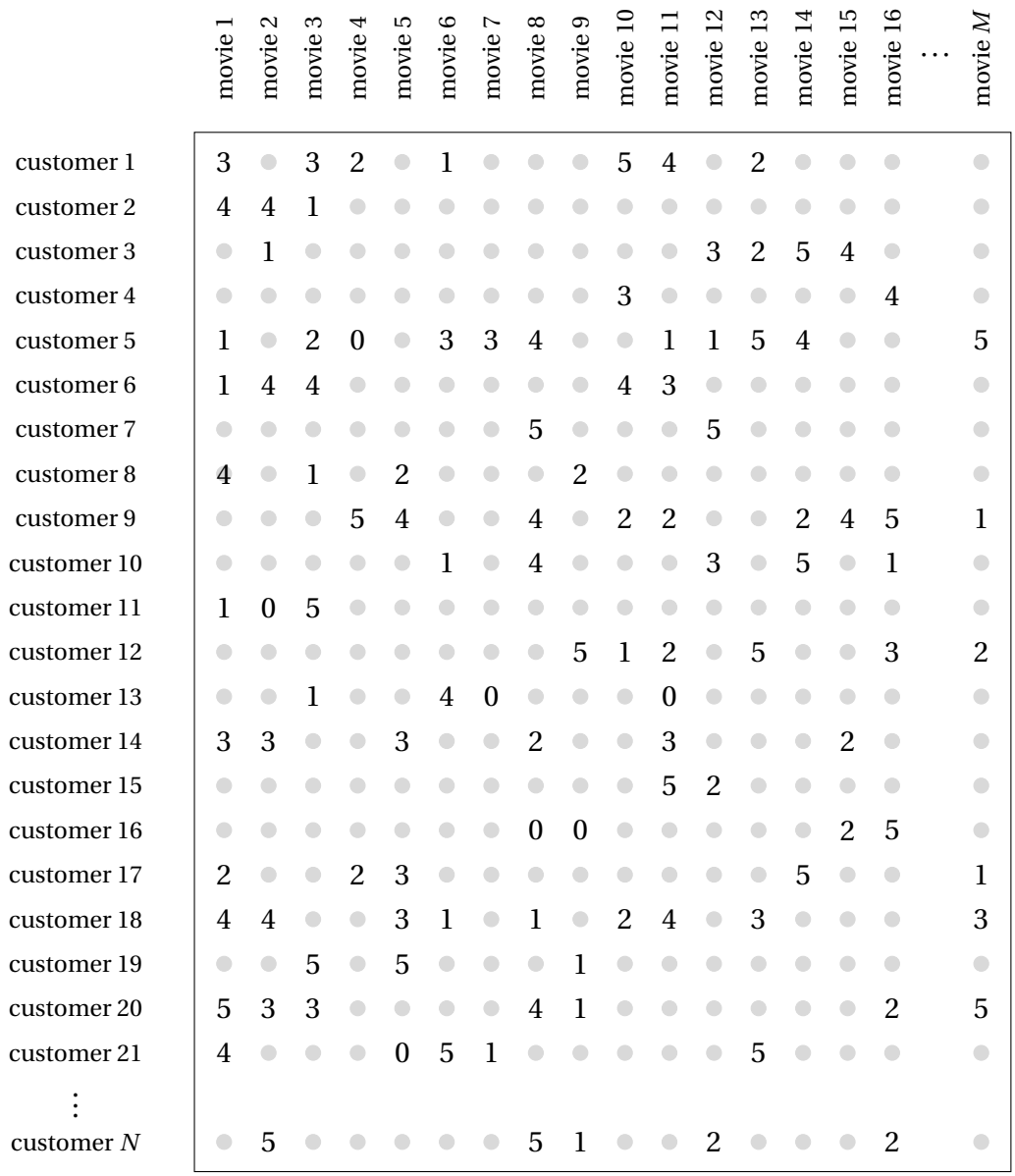

Figure 1.1: Example of a data matrix D for a recommender system.

i.e., we minimize the nuclear norm of the recovered matrix A while, as before, observing the set Ω of known entries D_{ij}.

It seems obvious that rank, singular values, nuclear norm, etc. will feature prominently in the algorithm. For this reason, we will present a brief review of the singular value decomposition, which underlies these concepts.

1.1 INTERLUDE: THE SINGULAR VALUE DECOMPOSITION

The singular value decomposition, or SVD for short, is an essential decomposition of a general (rectangular) matrix. Another, maybe more familiar decompositions, is the eigenvalue decomposition of a square matrix. In general, decompositions are very useful tools to categorize, quantify or analyze matrices. They are based on the following question: how can a matrix be described by its action on a set of vectors? In this sense, a decomposition constitutes an input-output analysis for a matrix. We know that acting with a matrix on a vector will, in general, rotate and stretch the vector. Decompositions are a way of isolating the rotation from the stretching part. Stretching a vector is simply a multiplication by a scalar; and stretching a collection of column vectors can be expressed as a multiplication by a diagonal matrix (with the individual stretching factors along the diagonal).

The singular value decomposition proposes the following split: we transform the vectors of an orthogonal input basis to an orthogonal output basis. This transformation contains additional stretching of the individual vectors that we express as a multiplication by a diagonal matrix. The rotational part of the transformation is contained in the fact that the input basis is not the same as the output basis, but rather the output basis is rotated with respect to the input basis. A sketch of the decomposition is given in Figure 1.2.

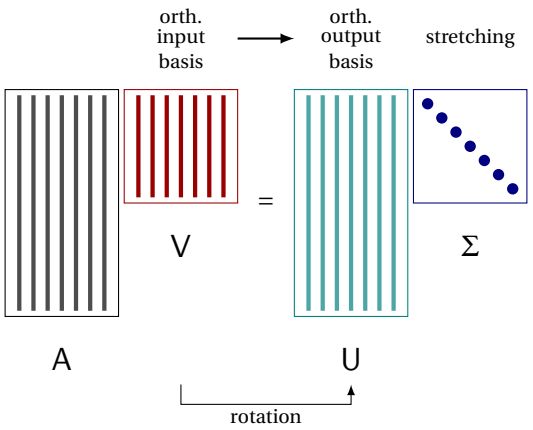

Figure 1.2: The singular value decomposition of a matrix A is expressed as the matrix' action on an orthogonal input matrix V. This actions (right-hand side) is broken into a rotated, orthogonal output basis U and a diagonal matrix Σ that accounts for the stretching of the individual basis vectors.

In two dimensions and for a 2×2-matrix A, the sketch in Figure 1.3 illustrates the action of the matrix A : we transform the orthogonal (and normalized) input basis (the vectors $\mathbf{v}_1$ and $\mathbf{v}_2$) into the orthogonal basis (the vectors $\mathbf{u}_1$ and $\mathbf{u}_2$) with additional stretching factors σ_1 and σ_2. Under this mapping and representation, circles in the $\mathbf{v}_1$-$\mathbf{v}_2$-coordinate system turn into ellipses in the $\mathbf{u}_1$-$\mathbf{u}_2$-coordinate system. The stretching factors (singular values $\sigma_{1,2}$) determine the aspect ratio (eccentricity) of the ellipse. In the special case of $\sigma_2 = 0$, the ellipse degenerates to a line.

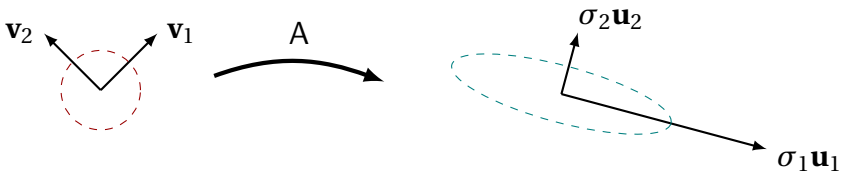

Figure 1.3: Mapping of a 2-dimensional orthogonal basis under A onto a rotated, orthogonal, stretched basis. Circles in the input basis map to ellipses in the output basis.

The stretching factors are referred to as **singular values**, are real, non-negative numbers and – by convention – are presented in descending order. Singular values can be zero; in this case, certain vectors from the input space get mapped to zero in the output space, i.e., they vanish. In other words, zero singular values are associated with a loss of "dimensionality" of the matrix A. The number of nonzero singular values is referred to as the **rank** of a matrix.

A simple reordering of the singular value decomposition allows us to decompose *any* matrix into its input-output formulation. For this, we transfer the input-matrix V to the right-hand side. More mathematically, we have

$$\mathsf{AV} = \mathsf{U}\Sigma \qquad \rightarrow \qquad \mathsf{A} = \mathsf{U}\Sigma\mathsf{V}^H \tag{1.5}$$

where U and V are orthogonal (or unitary) matrices containing the output and input basis vectors as columns, and Σ is a diagonal matrix with the stretching factors (ordered singular values) along the diagonal. The second expression above uses the fact that the inverse of an orthogonal (unitary) matrix is simply given by its conjugate transpose (indicated by ^{H}). Breaking the resulting product of three matrices $\mathsf{U}\Sigma\mathsf{V}^H$ into its individual vector components, we arrive at an interesting representation: it says that any matrix A can be written as the **sum of rank-one matrices**. See Figure 1.4.

We have

$$\mathsf{A} = \mathbf{u}_1\sigma_1\mathbf{v}_1^H + \mathbf{u}_2\sigma_2\mathbf{v}_2^H + \cdots + \mathbf{u}_m\sigma_m\mathbf{v}_m^H \tag{1.6}$$

with m as the number of columns in A (which we assume to be smaller than the number of rows, i.e., A is a tall-and-skinny matrix). Each one of these components on the right-hand side is a rank-one matrix, since the same column vector $\mathbf{u}_i$ is multiplied by the elements in $\mathbf{v}_i$ and σ_i to form a matrix; in the end, each resulting matrix $\mathbf{u}_i\sigma_i\mathbf{v}_i^H$ contains just stretched copies of the first column.

With the singular value decomposition we can thus determine the rank of the matrix A by looking at the singular values σ_i. The rank is given by the number of non-zero singular values. For example, a matrix with $\Sigma = \mathrm{diag}\{\sigma_1, \sigma_2, \sigma_3, 0, 0, \ldots, 0\}$ with $\sigma_{1,2,3} \neq 0$ has rank three. In this case, the above rank-one expansion (1.6) terminates after three terms.

Even if the singular values do not exactly drop to zero, we can still give an optimal rank-three approximation to a matrix A using the singular value decomposition. We get

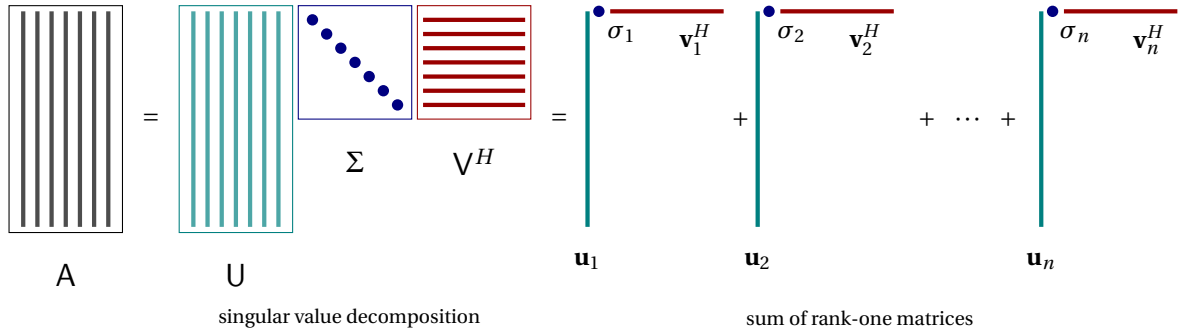

Figure 1.4: Singular value decomposition of a rectangular matrix A resulting into a product of three matrices: the output basis U, the stretching factors Σ and the output basis V. On the right: a representation of the decomposition as a sum of rank-one matrices $\mathbf{u}_i \sigma_i \mathbf{v}_i^H$.

$$\mathsf{A} \approx \mathbf{u}_1 \sigma_1 \mathbf{v}_1^H + \mathbf{u}_2 \sigma_2 \mathbf{v}_2^H + \mathbf{u}_3 \sigma_3 \mathbf{v}_3^H = \mathsf{U}_{(1:3)} \Sigma_{(1:3)} (\mathsf{V}_{(1:3)})^H. \tag{1.7}$$

We will take advantage of this low-rank truncation, based on the SVD, in the Netflix or matrix-completion algorithm below. From the expressions above, we conclude that the rank (given by the singular values) is a proxy for the "information content" of a matrix. Even though a low-rank matrix can consist of many entries (degrees of freedom), its action can be described rather low-dimensionally.

1.2 The mathematics of the optimization problem

Continuing with the matrix completion problem, we introduce an operator $\pi_\Omega(\cdot)$ which, when operating on a matrix, extracts the known entries $\{i, j\} \in \Omega$ from the matrix. We then have

$$\min \|\mathsf{A}\|_*, \tag{1.8a}$$
$$\text{subject to} \quad \pi_\Omega(\mathsf{A}) = \pi_\Omega(\mathsf{D}) \tag{1.8b}$$

or, introducing the matrix E which recovers the missing entries while leaving the true entries untouched, we have

$$\min \|\mathsf{A}\|_*, \tag{1.9a}$$
$$\text{subject to} \quad \mathsf{A} + \mathsf{E} = \mathsf{D} \quad \text{and} \quad \pi_\Omega(\mathsf{E}) = 0. \tag{1.9b}$$

Since the matrix E will recover the missing entries of D, these entries of D are simply set to zeros.

The above problem is a constrained optimization problem: we try to minimize the nuclear norm (a continuous proxy for rank) subject to matching the known entries in D. We transform the constrained problem to an unconstrained problem by adding the constraints to the cost functional using Lagrange multipliers. We then seek to minimize the augmented expression

$$\mathscr{L}(\mathsf{A},\mathsf{E},\mathsf{Y},\mu) = \|\mathsf{A}\|_* \; + \; \langle \mathsf{Y}, \mathsf{D}-\mathsf{A}-\mathsf{E}\rangle \; + \; \mu\|\mathsf{D}-\mathsf{A}-\mathsf{E}\|_F^2. \tag{1.10}$$

We see that we have added the constraint $\mathsf{D}-\mathsf{A}-\mathsf{E}=0$ via the Lagrange multiplier Y and the scalar product $\langle\cdot,\cdot\rangle$ to the objective $\|\mathsf{A}\|_*$. The last expression is introduced to turn the optimization problem into a convex optimization problem; this will have advantageous for the algorithm. This procedure is also referred to as a relaxation of the optimization problem. The parameter μ is very small and adaptive and is introduced as an unknown variable (to be optimized). The subscript $_F$ on the last expression denotes the Frobenius norm. Ideally, $\mathsf{D}-\mathsf{A}-\mathsf{E}$ should be very small; so the last expression is negligible near the optimum and does not significantly influence the final solution. The last constraint $\pi_\Omega(\mathsf{E})=0$ is not explicitly enforced, but rather needs to be imposed at every step of the algorithm.

The above unconstrained optimization problem needs to be solved by finding a set of independent variables $\mathsf{A},\mathsf{E},\mathsf{Y}$ and μ that minimize the augmented expression. Gradients with respect to all four variables, together with a standard gradient-based algorithm, would direct the optimization process. However, we choose a different strategy which employs the idea of alternating directions. To this end, we solve a sequence of simpler optimization problems by freezing three of the four variables and optimizing with respect to the last one (see Figure 1.5). By cyclically rotating through all four variables (always freezing three and optimizing with respect to the other), we iteratively approach the final solution that minimizes the augmented (composite) expression.

With four independent variables, each iteration of the algorithm involves four local optimizations. First we optimize A, while keeping E, Y and μ constant.

$$\mathsf{A}_{k+1} = \arg\min_{\mathsf{A}} \; \mathscr{L}\left(\mathsf{A},\mathsf{E}_k,\mathsf{Y}_k,\mu_k\right) \qquad \text{[opt 1]} \tag{1.11}$$

where we introduced the subscript $_k$ to indicate the iteration count. Next, we optimize with respect to the variable E. We get

$$\mathsf{E}_{k+1} = \arg\min_{\mathsf{E}} \; \mathscr{L}\left(\mathsf{A}_{k+1},\mathsf{E},\mathsf{Y}_k,\mu_k\right) \qquad \text{[opt 2]} \tag{1.12}$$

where we took the updated iterate A_{k+1} from the previous step. The third step is a simple update: we determine the mismatch between the new $\mathsf{A}_{k+1}+\mathsf{E}_{k+1}$ and D for the elements in the complementary set of indices $\overline{\Omega}$ to update Y_k to Y_{k+1}. We notice that only indices of the complementary set are non-zero in Y, since only these indices have to be adjusted. We have

$$\mathsf{Y}_{k+1} = \mathsf{Y}_k + \mu_k\left(\mathsf{D}-\mathsf{A}_{k+1}-\mathsf{E}_{k+1}\right). \qquad \text{[opt 3]} \tag{1.13}$$

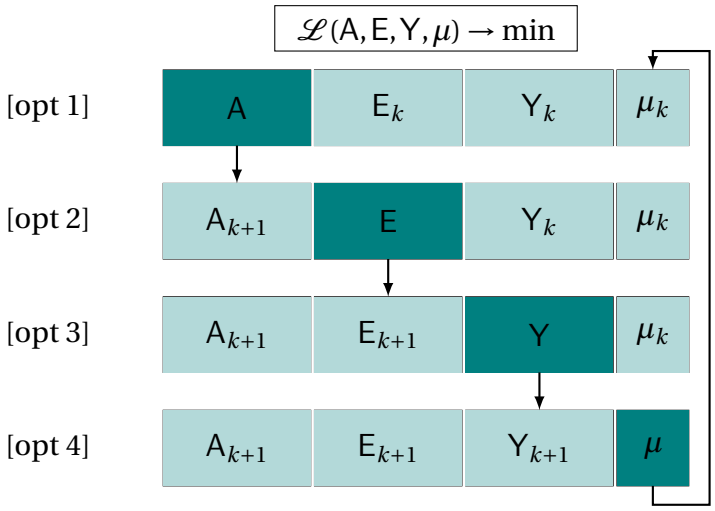

Figure 1.5: Sketch of iterative scheme of the incomplete alternating Lagrangian method (IALM).

The fourth and final step will update the parameter μ_k to μ_{k+1}. A heuristic dependence on the iteration counter will be used to approach a steady (and small) value of the regularization parameter μ. We then return to the first step.

The first optimization problem, where we determine a new A matrix, is given in generic form

$$\mathsf{X} = \arg\min_{\mathsf{X}} \ \epsilon\|\mathsf{X}\|_* + \frac{1}{2}\|\mathsf{X}-\mathsf{W}\|_F^2 \tag{1.14}$$

which is a mixed-norm optimization problem with one term covering the nuclear norm (the low-rank objective) and another term implementing the Frobenius-norm regularization. The solution to this optimization problem can be obtained using the singular value decomposition (SVD). Without proof, we give the solution of the above optimization as

$$\mathsf{X} = \mathsf{U}\ \mathrm{shrink}_\epsilon(\Sigma)\ \mathsf{V}^H \tag{1.15}$$

where $\mathsf{U}, \Sigma, \mathsf{V}$ are the three components of an SVD of W. The shrink-function (see Figure 1.6) is given as

$$\mathrm{shrink}_\epsilon(x) = \begin{cases} x-\epsilon, & \text{if } x > \epsilon \\ x+\epsilon, & \text{if } x < -\epsilon \\ 0, & \text{otherwise} \end{cases} \tag{1.16}$$

The solution to the (regularized) optimization problem, i.e. matching a matrix X of minimal rank to a given matrix W, is thus given by a thresholded singular value decomposition. Applied to our recommender system problem, we have to determine the A_{k+1} in the first subproblem as follows,

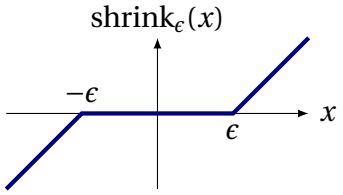

Figure 1.6: Sketch of the shrink-function for soft thresholding of the singular values.

$$\begin{aligned} \operatorname{svd}\left(\mathsf{D}-\mathsf{E}_k+\mu_k^{-1}\mathsf{Y}_k\right) &= \mathsf{U}\Sigma\mathsf{V}^H && (1.17a)\\ \mathsf{A}_{k+1} &= \mathsf{U}\,\operatorname{shrink}_{\mu_k^{-1}}(\Sigma)\,\mathsf{V}^H \qquad [\text{opt 1}] && (1.17b)\end{aligned}$$

i.e., we take a singular value decomposition, apply the shrink-function to the singular values with a cut-off threshold of μ_k^{-1}, and then reconstitute the three-part decomposition with a replaced diagonal matrix $\operatorname{shrink}_{\mu_k^{-1}}(\Sigma)$. This part of the algorithm will aim for a low-rank matrix A.

With a new matrix A_{k+1} we solve the second optimization problem (1.12). This is simply accomplished by a projection using $\pi_{\overline{\Omega}}$ onto the

$$\mathsf{E}_{k+1}=\pi_{\overline{\Omega}}\left(\mathsf{D}-\mathsf{A}_{k+1}+\mu_k^{-1}\mathsf{Y}_k\right). \qquad [\text{opt 2}] \qquad (1.18)$$

For this reason, the new matrix E_{k+1} has only non-zero entries at indices where we need to recover missing data (and zeros at indices where we do have data).

The third optimization step, which yields a new value for the Lagrange multiplier Y_{k+1}, is already given by (1.13). There is no need to project again, since E_{k+1} has non-zero entries for indices that need to be recovered.

Finally, the threshold and regularization value μ_k has to be updated. This last step concludes one iteration. We return to the first subproblem or terminate if a user-specified criterion is satisfied.

Two important remarks are in order:

1. Sparsity. For each $k \geq 0$, Y_k vanishes outside of Ω and is, therefore, sparse, a fact which can be used to evaluate the shrink-function rapidly.

2. Low-rank property. The matrix A_k turns out to have low rank, and hence the algorithm has minimum storage requirement, since we only need to keep principal factors in memory.

1.3 Algorithm

We formulate an augmented Lagrange multiplier algorithm which we will solve by alternating between four optimization sub-problems until convergence is reached.

For our matrix completion problem, we try to recover missing entries in the matrix by using the assumption of a low-rank property of the underlying (complete) matrix. Problems of this type not only appear in recommender systems, but also in machine learning, control applications and computer vision.
The full algorithm loops through the following steps:

1. initialize all matrices and parameters.
2. compute the singular value decomposition of $D - E_k + \mu_k^{-1} Y_k$ and apply the shrink-function to the singular values with a threshold of μ_k^{-1}. The solution of the first optimization subproblem is the reconstituted singular value decomposition with the thresholded singular values.
3. compute the update for E_{k+1} according to $\pi_{\overline{\Omega}}(D - A_{k+1} - Y_k/\mu_k)$, i.e. a projection onto the complementary index set.
4. update the Lagrange multiplier matrix Y_{k+1} according to $Y_k + \mu_k(D - A_{k+1} - E_{k+1})$.
5. adjust the parameter μ_k.
6. go back to step 1 or terminate if proper convergence conditions are met.

1.4 Implementation

The implementation of the above algorithmic steps is given below.

```
1  import numpy as np
2  import math as ma
3
4  def IALM(D,mu,rho):
5      # incomplete alternating Lagrangian
6      # method (IALM) for solving the
7      # matrix completion problem
8
9      # thresholds
10     ep1 = 1.e-7
11     ep2 = 1.e-6
12     Dn  = np.linalg.norm(D,'fro')
13     # projector matrix (complementary)
14     PP = (D == 0)
15     P  = PP.astype(np.float)
16     # initialization
17     m,n  = np.shape(D)
18     Y    = np.zeros((m,n))
19     Eold = np.zeros((m,n))
20     # iteration
```

```
    for i in range(1,1000):
        # compute SVD
        tmp   = D - Eold + Y/mu
        U,S,V = np.linalg.svd(tmp,full_matrices=False)
        # threshold and patch matrix back together
        ss = S-(1/mu)
        s2 = np.clip(ss,0,max(ss))
        A  = np.dot(U,np.dot(np.diag(s2),V))
        # project
        Enew = P*(D - A + Y/mu)
        DAE  = D - A - Enew
        Y   += mu*DAE
        # check residual and (maybe) exit
        r1   = np.linalg.norm(DAE,'fro')
        resi = r1/Dn
        print i,' residual ',resi
        if (resi < ep1):
            break

        # adjust mu-factor (heuristic)
        muf  = np.linalg.norm((Enew-Eold),'fro')
        fac  = min(mu,ma.sqrt(mu))*(muf/Dn)
        if (fac < ep2):
            mu *= rho

        # update E and go back
        Eold = np.copy(Enew)

    E = np.copy(Enew)
    return A,E
```

The main code is given as

```
if __name__ == '__main__':

    # matrix completion problem (Netflix problem)

    # size of problem
    m = 500                     # rows
    n = 150                     # columns
    r = int(round(min(m,n)/3))  # rank
    # construct low-rank matrix
    U = np.random.random((m,r))
    V = np.random.random((r,n))
    A = np.dot(U,V)
```

```
13      # sampling matrix
14      PP = (np.random.random((m,n)) > 0.433)
15      P  = PP.astype(np.float)
16      # number of non-zero elements
17      Omega = np.count_nonzero(P)
18      # data matrix
19      D = P*A
20      fratio = float(Omega)/(m*n)
21      print 'fill ratio ', fratio
22      # initialize parameters
23      mu  = 1./np.linalg.norm(D,2)
24      rho = 1.2172 + 1.8588*fratio
25      # call IALM-algorithm
26      AA,EE = IALM(D,mu,rho)
27      # compare
28      print('\n')
29      print('Data matrix')
30      print D[0:5,0:5]
31      print('\n')
32      print('Recovered matrix')
33      print AA[0:5,0:5]
34      print('\n')
35      print('Original matrix')
36      print A[0:5,0:5]
```

1.5 Example

The recovery of the missing data entries of a low-rank matrix depends on the number of known entries. Theory tells us that the number of known entries p must follow the inequality $p \geq Cr\,n^{6/5}\ln n$ where n denotes the number of columns, r stands for the rank and C represents a positive constant.