Matrix and Tensor Tools

FOR COMPUTER VISION

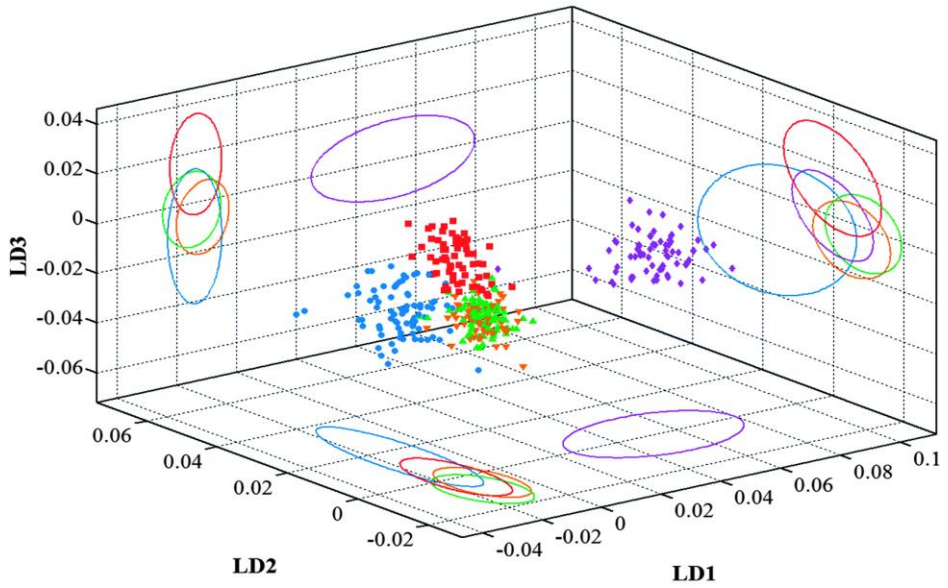
ANDREWS C. SOBRAL

ANDREWSSOBRAL@GMAIL.COM

PH.D. STUDENT, COMPUTER VISION

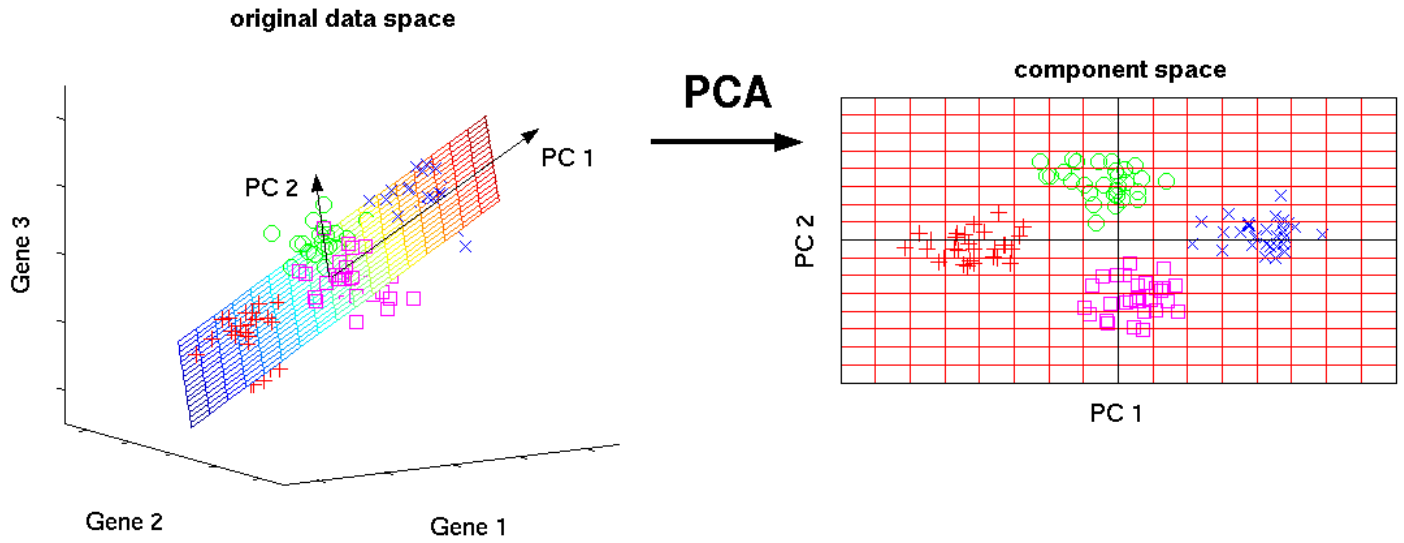
LAB. L3I – UNIV. DE LA ROCHELLE, FRANCE

Principal Component Analysis (PCA)



Principal Component Analysis

PCA is a statistical procedure that uses orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called **principal components**.



Dimensionality reduction

Variants: Multilinear PCA, ICA, LDA, Kernel PCA, Nonlinear PCA, ...

http://www.nlpca.org/pca_principal_component_analysis.html

Principal Component Analysis

1. Estimate the covariance matrix S. Usually the mean value is assumed to be zero, $E[x] = 0$. In this case, the covariance and autocorrelation matrices coincide, $R \equiv E[xx^T] = S$. If this is not the case, we subtract the mean. Recall that, given N feature vectors, $x_i \in \mathcal{R}^l$, $i = 1, 2, \ldots, N$, the autocorrelation matrix estimate is given by

$$R \approx \frac{1}{N} \sum_{i=1}^{N} x_i x_i^T \tag{3.1}$$

2. Perform the eigendecomposition of S and compute the l eigenvalues/eigenvectors, λ_i, $a_i \in \mathcal{R}^l$, $i = 0, 2, \ldots, l-1$.
3. Arrange the eigenvalues in descending order, $\lambda_0 \geq \lambda_1 \geq \cdots \geq \lambda_{l-1}$.
4. Choose the m largest eigenvalues. Usually m is chosen so that the gap between λ_{m-1} and λ_m is large. Eigenvalues $\lambda_0, \lambda_1, \ldots, \lambda_{m-1}$ are known as the *m principal components*.
5. Use the respective (column) eigenvectors a_i, $i = 0, 1, 2, \ldots, m-1$ to form the transformation matrix

$$A = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{m-1} \end{bmatrix}$$

http://store.elsevier.com/Introduction-to-Pattern-Recognition-A-Matlab-Approach/Sergios-Theodoridis/isbn-9780123744869/

Principal Component Analysis

6. Transform each l-dimensional vector x in the original space to an m-dimensional vector y via the transformation $y = A^T x$. In other words, the ith element $y(i)$ of y is the *projection* of x on a_i $\left(y(i) = a_i^T x\right)$.

As pointed out in [Theo 09, Section 6.3], the total variance of the elements of x, $\sum_{i=0}^{l-1} E[x^2(i)]$ (for zero mean), is equal to the sum of the eigenvalues $\sum_{i=0}^{l-1} \lambda_i$. After the transformation, the variance of the ith element, $E[y^2(i)]$, $i = 0, 2, \ldots, l-1$, is equal to λ_i. Thus, selection of the elements that correspond to the m largest eigenvalues retains the maximum variance.

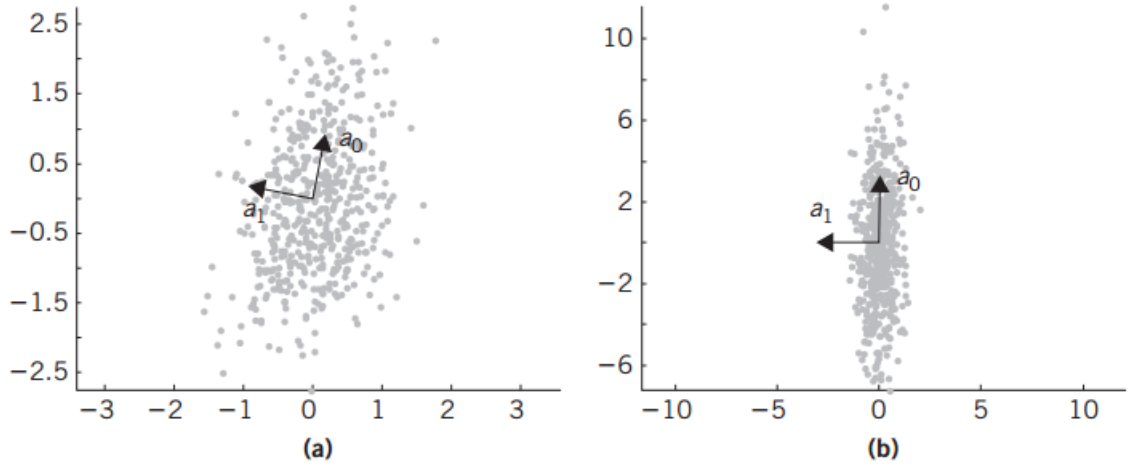
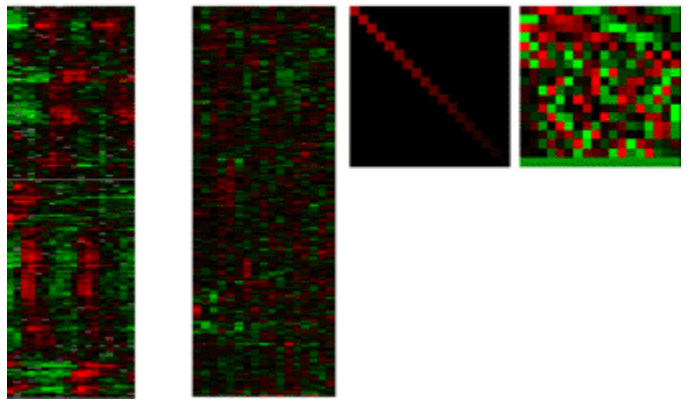


FIGURE 3.1

http://store.elsevier.com/Introduction-to-Pattern-Recognition-A-Matlab-Approach/Sergios-Theodoridis/isbn-9780123744869/

Singular Value Decomposition (SVD)

$$A = U \cdot W \cdot V^{T}$$



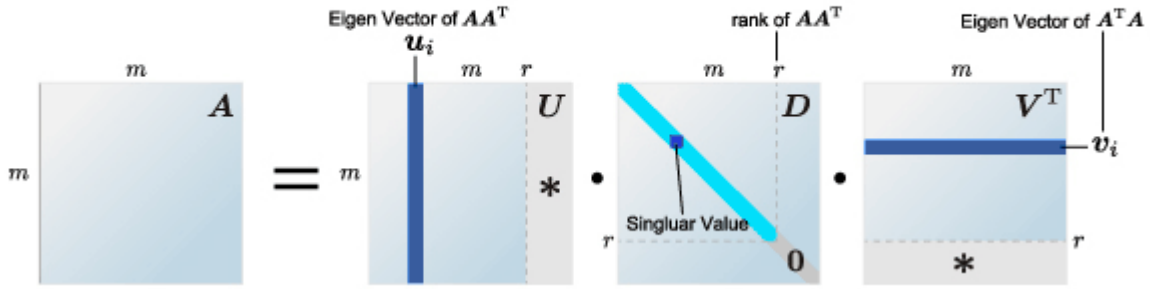
Singular Value Decomposition

Formally, the singular value decomposition of an *m*×*n* real or complex matrix *M* is a factorization of the form:

$$\mathbf{M} = \mathbf{U\Sigma V}^*$$

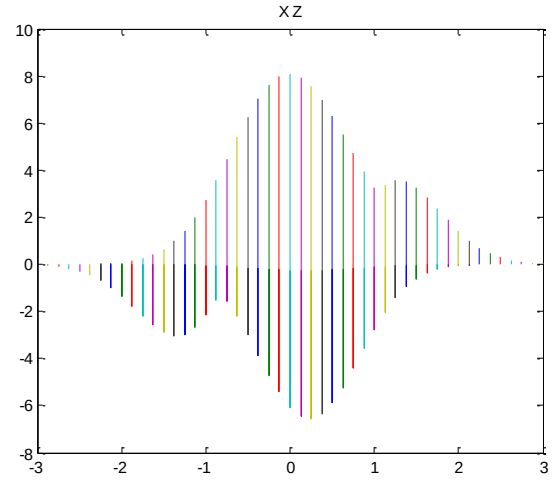
where *U* is a *m*×*m* real or complex unitary matrix, Σ is an *m*×*n* rectangular diagonal matrix with nonnegative real numbers on the diagonal, and *V** (the conjugate transpose of *V*, or simply the transpose of *V* if *V* is real) is an *n*×*n* real or complex unitary matrix. The diagonal entries $\Sigma_{i,i}$ of Σ are known as the **singular values** of *M*. The *m* columns of *U* and the *n* columns of *V* are called the **left-singular vectors** and **right-singular vectors** of *M*, respectively.

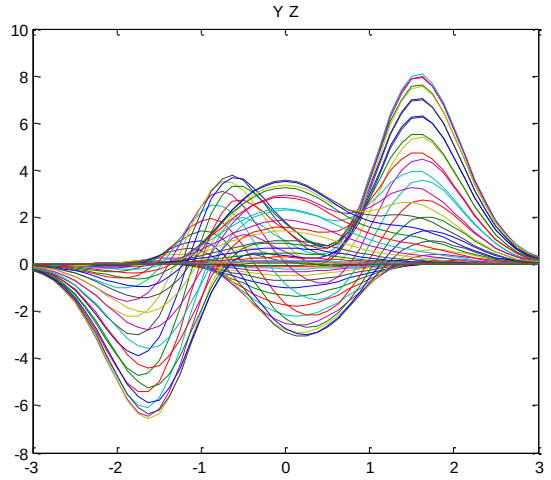
$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{\mathrm{T}}$$

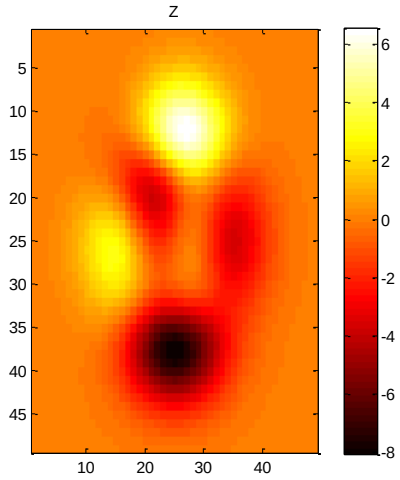


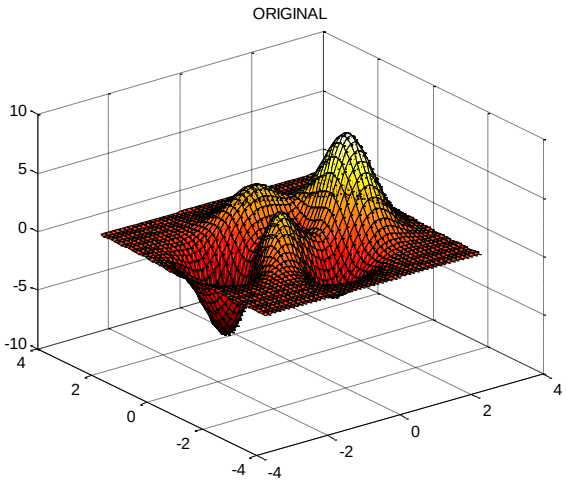
http://www.numtech.com/systems/

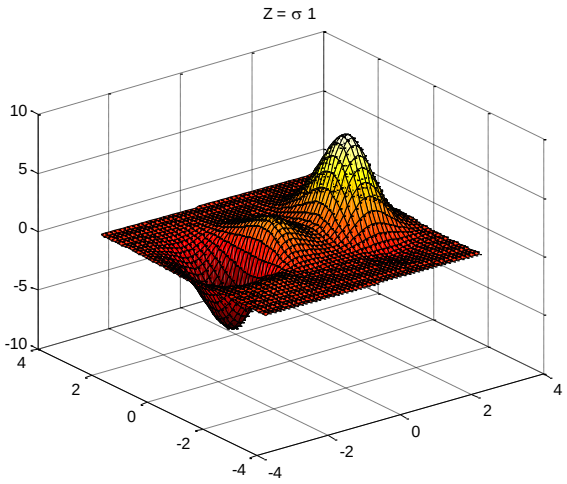
generalization of eigenvalue decomposition

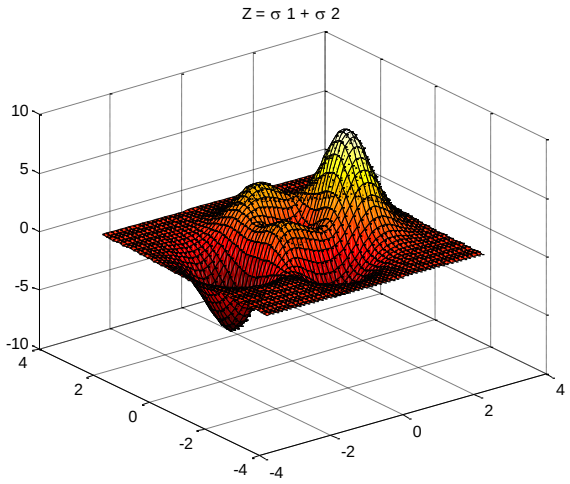

X Z


Y Z


Z

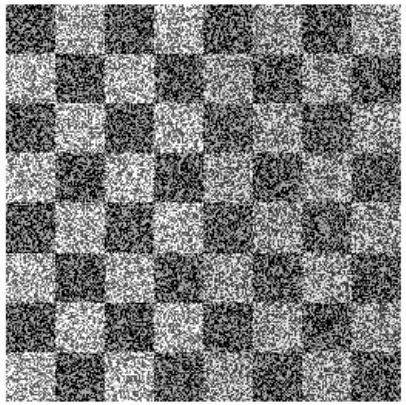

ORIGINAL


Z = σ 1

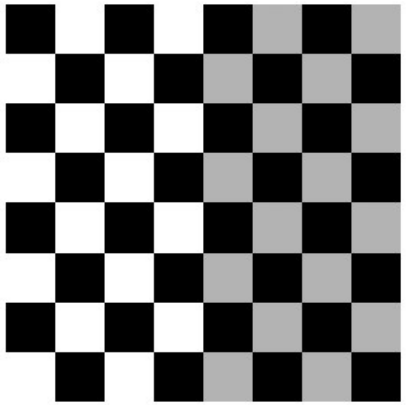

Z = σ 1 + σ 2

Robust PCA (RPCA)

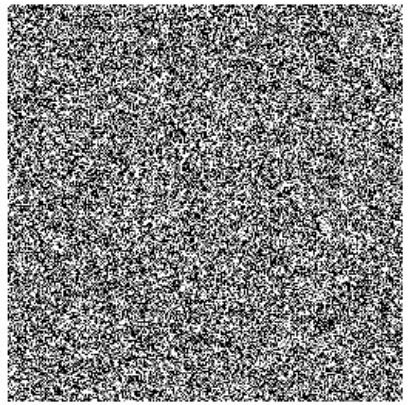
Mixed Image



Low-rank Image

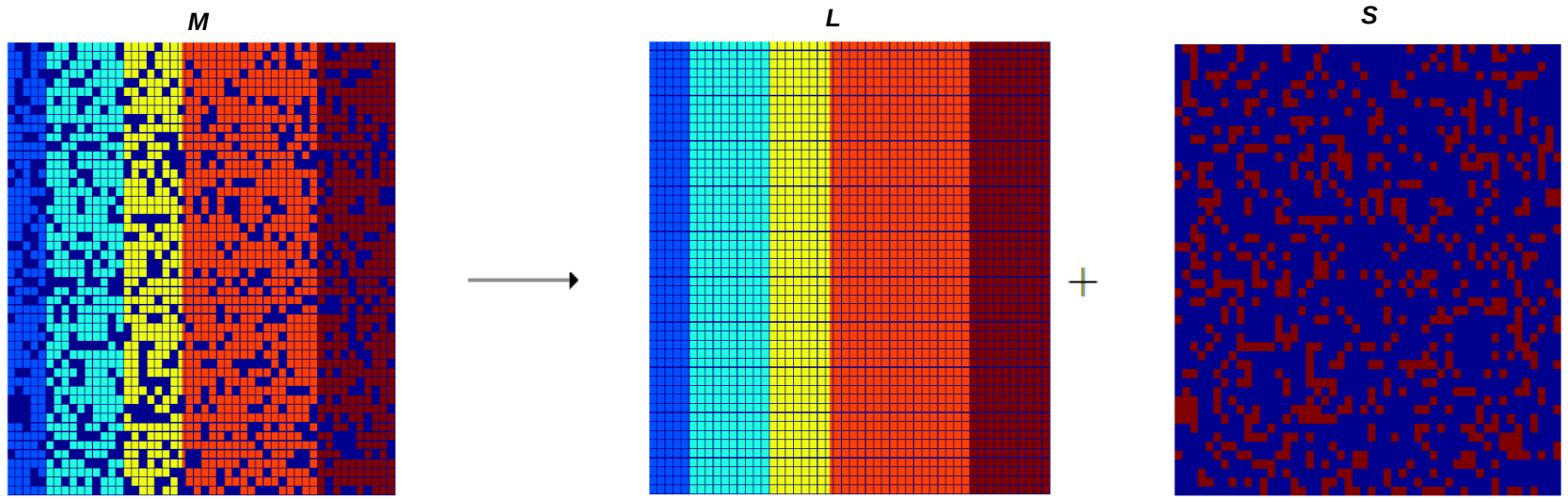


Sparse Image



Robust PCA

- Robust PCA (RPCA) deals with the problem of making PCA robust to outliers and grossly corrupted observations. The RPCA problem can be formulated mathematically as the problem of decomposing a matrix consisting of the sum of a low-rank matrix and a sparse matrix into these two components, without prior knowledge of the low-rank part nor of the sparsity pattern of the sparse part (Pope et al, 2011). The data matrix M consists of a low rank matrix that is corrupted by sparse outliers, i.e.
- $M = L + S$, where L is a low rank matrix having a SVD ($L = UDV^T$) and S is sparse and can have arbitrary large magnitude.



http://perception.csl.illinois.edu/matrix-rank/home.html

Robust PCA

Let $||L||_*$ denotes the nuclear norm of L (i.e. the sum of singular values of L) and $||S||_1$ the ℓ^1-norm (sum of matrix elements magnitude) of the matrix S. It is shown in (Candès et al. 2009) that L and S can be recovered with high probability by solving a convex optimization problem, named as Principal Component Pursuit (PCP),

$$\underset{L,S}{\operatorname{argmin}} \|L\|_* + \lambda \|S\|_1,$$
$$subject\ to\ M = L + S$$

where λ is the weighting parameter (tradeoff between rank and sparsity).

$||.||_*$ enforces low rank in L.
$||.||_1$ enforces sparsity in S.



http://perception.csl.illinois.edu/matrix-rank/home.html

Robust PCA

One effective way to solve PCP for the case of large matrices is to use a standard augmented Lagrangian multiplier method (ALM) (Bertsekas, 1982).

$$\ell(\mathbf{L},\mathbf{S},\mathbf{Y}) \triangleq \|\mathbf{L}\|_* + \lambda\|\mathbf{S}\|_1 + \langle\mathbf{Y},\mathbf{M}-\mathbf{L}-\mathbf{S}\rangle + \mu\|\mathbf{M}-\mathbf{L}-\mathbf{S}\|_{\mathrm{F}}^2 \quad (1)$$

and then minimizing it iteratively by setting

$$(\mathbf{L}^{(k)},\mathbf{S}^{(k)}) = \arg\min_{(\mathbf{L},\mathbf{S})} \ell(\mathbf{L},\mathbf{S},\mathbf{Y}^{(k)}) \quad (2)$$

and updating $\mathbf{Y}^{(k+1)} \leftarrow \mathbf{Y}^{(k)} + \mu(\mathbf{M}-\mathbf{L}^{(k)}-\mathbf{S}^{(k)})$.

Algorithm 1 ALM using alternating directions [2], [3], [5]

1: **input:** $\mathbf{M} \in \mathbb{R}^{n_1\times n_2}$
2: **initialize:** $\mathbf{S}^{(0)} = \mathbf{Y}^{(0)} = \mathbf{0}$, $\lambda = 1/\sqrt{\max\{n_1,n_2\}}$, $\mu = (n_1 n_2)/(4\|\mathbf{M}\|_1)$, $k = 0$
3: **while** not converged **do**
4: $\quad\mathbf{L}^{(k+1)} \leftarrow \mathcal{D}_{\mu^{-1}}(\mathbf{M}-\mathbf{S}^{(k)}+\mu^{-1}\mathbf{Y}^{(k)})$
5: $\quad\mathbf{S}^{(k+1)} \leftarrow \mathcal{S}_{\lambda\mu^{-1}}(\mathbf{M}-\mathbf{L}^{(k+1)}+\mu^{-1}\mathbf{Y}^{(k)})$
6: $\quad\mathbf{Y}^{(k+1)} \leftarrow \mathbf{Y}^{(k)} + \mu(\mathbf{M}-\mathbf{L}^{(k+1)}-\mathbf{S}^{(k+1)})$
7: **end while**
8: **output:** $\mathbf{L}^{(k)}, \mathbf{S}^{(k)}$

Where:

$$\arg\min_{\mathbf{S}} \ell(\mathbf{L},\mathbf{S},\mathbf{Y}) = \mathcal{S}_{\lambda\mu^{-1}}(\mathbf{M}-\mathbf{L}+\mu^{-1}\mathbf{Y}) \quad (3)$$

$$\arg\min_{\mathbf{L}} \ell(\mathbf{L},\mathbf{S},\mathbf{Y}) = \mathcal{D}_{\mu^{-1}}(\mathbf{M}-\mathbf{S}+\mu^{-1}\mathbf{Y}). \quad (4)$$

$\mathcal{S}_\tau(x) \triangleq \mathrm{sgn}(x)\max\{|x|-\tau,0\}$.

$\mathcal{D}_\tau(\mathbf{A}) \triangleq \mathbf{U}\mathcal{S}_\tau(\mathbf{\Sigma})\mathbf{V}^T$

$\langle\mathbf{A},\mathbf{B}\rangle \triangleq \mathrm{tr}(\mathbf{A}^T\mathbf{B})$, where $\mathrm{tr}(\cdot)$ denotes the trace operator

$\lambda = 1/\sqrt{\max\{n_1,n_2\}}$ and $\mu = (n_1 n_2)/(4\|\mathbf{M}\|_1)$

shrinkage operator $D_\tau(.)$

$$\begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}$$

More information:
(Qiu and Vaswani, 2011), (Pope et al. 2011), (Rodríguez and Wohlberg, 2013)

Robust PCA

Algorithm 1 (RPCA via Iterative Thresholding)

Input: Observation matrix $D \in \mathbb{R}^{m \times n}$, weights λ and τ.

while not converged **do**
  $(U, S, V) = \text{svd}(Y_{k-1})$,
  $A_k = U\mathcal{S}_\tau[S]V^T$,
  $E_k = \mathcal{S}_{\lambda\tau}[Y_{k-1}]$,
  $Y_k = Y_{k-1} + \delta_k(D - A_k - E_k)$.
end while

Output: $A \leftarrow A_k$, $E \leftarrow E_k$.

Robust PCA Algorithm Comparison

Algorithm	Rank of estimate	Relative error in estimate of A	Time (s)
Singular Value Thresholding	20	3.4×10^{-4}	877
Accelerated Proximal Gradient	20	2.0×10^{-5}	43
Accelerated Proximal Gradient (with partial SVDs)	20	1.8×10^{-5}	8
Dual Method	20	1.6×10^{-5}	177
Exact ALM	20	7.6×10^{-8}	4
Inexact ALM	20	4.3×10^{-8}	2
Alternating Direction Methods	20	2.2×10^{-5}	5

Algorithm 2 (RPCA via Accelerated Proximal Gradient)

Input: Observation matrix $D \in \mathbb{R}^{m \times n}$, λ.

$A_0 = A_{-1} = 0$; $E_0 = E_{-1} = 0$; $t_0 = t_{-1} = 1$; $\bar{\mu} > 0$; $\eta < 1$.
while not converged **do**
  $Y_k^A = A_k + \frac{t_{k-1}-1}{t_k}(A_k - A_{k-1})$, $Y_k^E = E_k + \frac{t_{k-1}-1}{t_k}(E_k - E_{k-1})$.
  $G_k^A = Y_k^A - \frac{1}{2}\left(Y_k^A + Y_k^E - D\right)$.
  $(U, S, V) = \text{svd}(G_k^A)$, $A_{k+1} = U\mathcal{S}_{\frac{\mu_k}{2}}[S]V^T$.
  $G_k^E = Y_k^E - \frac{1}{2}\left(Y_k^A + Y_k^E - D\right)$.
  $E_{k+1} = \mathcal{S}_{\frac{\lambda\mu_k}{2}}[G_k^E]$.
  $t_{k+1} = \frac{1+\sqrt{4t_k^2+1}}{2}$; $\mu_{k+1} = \max(\eta\,\mu_k, \bar{\mu})$.
  $k \leftarrow k + 1$.
end while

Output: $A \leftarrow A_k$, $E \leftarrow E_k$.

Algorithm 5 (RPCA via the Inexact ALM Method)

Input: Observation matrix $D \in \mathbb{R}^{m \times n}$, λ.

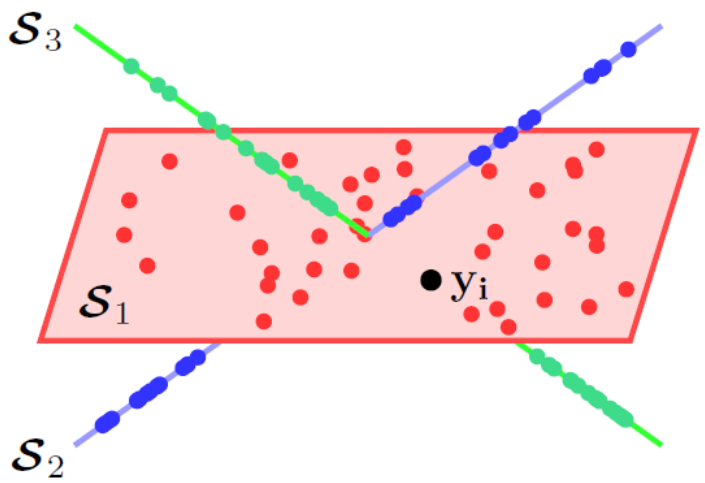
$Y_0 = D/J(D)$; $E_0 = 0$; $\mu_0 > 0$; $\rho > 1$; $k = 0$.
while not converged **do**
  // Lines 4-5 solve $A_{k+1} = \arg\min_A L(A, E_k, Y_k, \mu_k)$.
  $(U, S, V) = \text{svd}(D - E_k + \mu_k^{-1}Y_k)$;
  $A_{k+1} = U\mathcal{S}_{\mu_k^{-1}}[S]V^T$.
  // Line 7 solves $E_{k+1} = \arg\min_E L(A_{k+1}, E, Y_k, \mu_k)$.
  $E_{k+1} = \mathcal{S}_{\lambda\mu_k^{-1}}[D - A_{k+1} + \mu_k^{-1}Y_k]$.
  $Y_{k+1} = Y_k + \mu_k(D - A_{k+1} - E_{k+1})$.
  Update μ_k to μ_{k+1}.
  $k \leftarrow k + 1$.
end while

Output: (A_k, E_k).

For more information see: (Lin et al., 2010)

http://perception.csl.illinois.edu/matrix-rank/sample_code.html

LOW-RANK REPRESENTATION (LRR)



Low-rank Representation (LRR)

- This is another generalization of PCA. We seek a low-rank representation of a data matrix, and not only allow outliers (like the RPCA described above) but also allow simple transformations.
- The problem is:
- $\underset{L,S}{\operatorname{argmin}} \|L\|_* + \lambda \|S\|_1,$
- $subject\ to\ M = AL + S$
- where A is some known transformation (if it is not known, then it can be estimated at a separate step).
- The classical RPCA assumes that the underlying data structure is a single low-rank subspace. So LRR could be regarded as a generalization of RPCA that essentially uses the standard bases as the dictionary. LRR could handle well the data drawn from a union of multiple subspaces (Liu et al. 2012).

Subspace clustering problem!

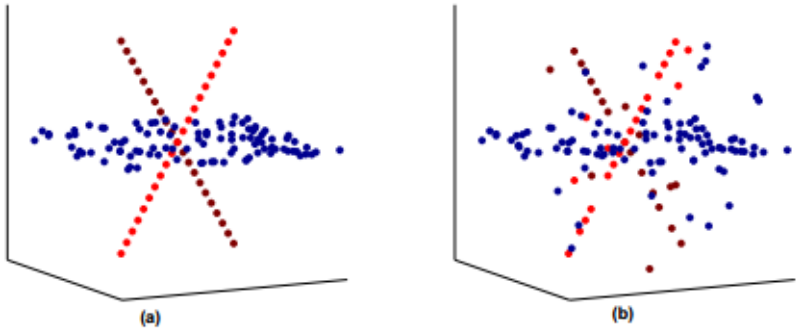
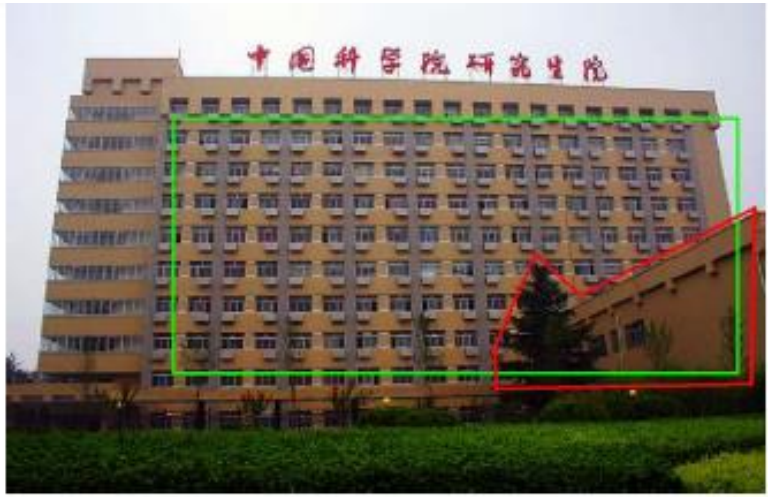


Fig. 1

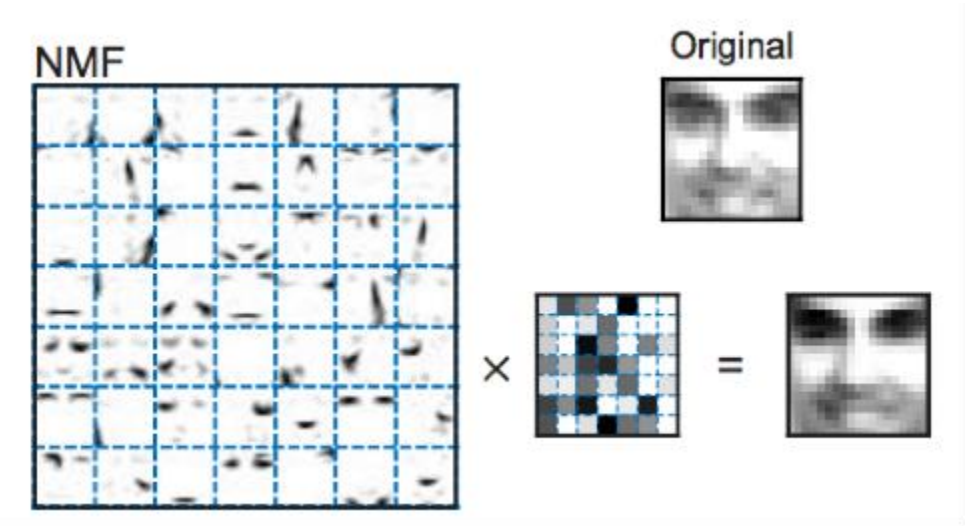
A MIXTURE OF SUBSPACES CONSISTING OF A 2D PLANE AND TWO 1D LINES. (A) THE SAMPLES ARE STRICTLY DRAWN FROM THE UNDERLYING SUBSPACES. (B) THE SAMPLES ARE APPROXIMATELY DRAWN FROM THE UNDERLYING SUBSPACES.

Low-rank Representation (LRR)

Low-rank Representation (LRR)



NON-NEGATIVE MATRIX FACTORIZATION (NMF)



Non-Negative Matrix Factorizations (NMF)

In many applications, data is non-negative, often due to physical considerations.

- images are described by pixel intensities;
- texts are represented by vectors of word counts;

It is desirable to retain the non-negative characteristics of the original data.

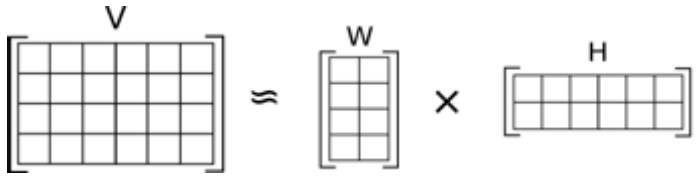
Non-Negative Matrix Factorizations (NMF)

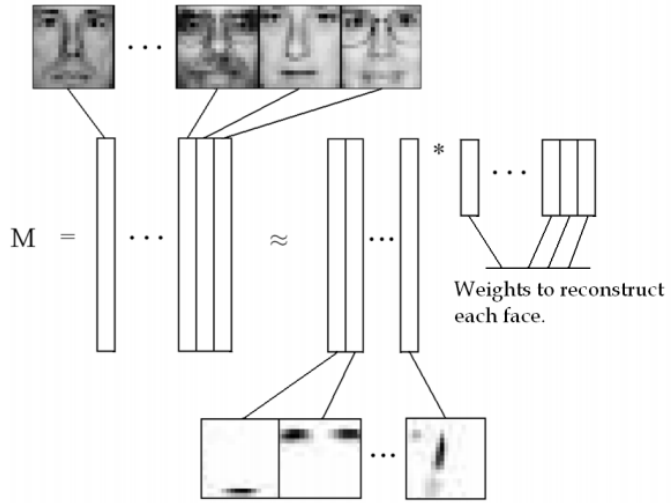
NMF provides an alternative approach to decomposition that assumes that the data and the components are non-negative.

For interpretation purposes, one can think of imposing non-negativity constraints on the factor U so that basis elements belong to the same space as the original data.

H >= 0 constraints the basis elements to be nonnegative. Moreover, in order to force the reconstruction of the basis elements to be additive, one can impose the weights **W** to be nonnegative as well, leading to a part-based representation.

W >= 0 imposes an additive reconstruction.





Given a matrix $M \in \mathbb{R}_{+}^{m \times n}$ and a factorization rank $r \in \mathbb{N}$, find $U \in \mathbb{R}^{m \times r}$ and $V \in \mathbb{R}^{r \times n}$ such that

$$\min_{U \geq 0, V \geq 0} ||M - UV||_F^2 = \sum_{i,j} (M - UV)_{ij}^2 \qquad \text{(NMF)}$$

NMF is an additive linear model for nonnegative data.

References:

The Why and How of Nonnegative Matrix Factorization (Nicolas Gillis, 2014)

Nonnegative Matrix Factorization: Complexity, Algorithms and Applications (Nicolas Gillis, 2011),

Non-Negative Matrix Factorizations (NMF)

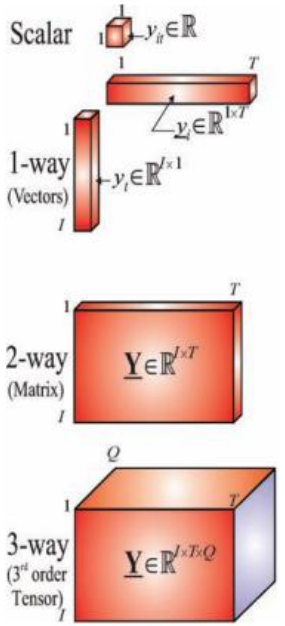
Similar to sparse and low-rank matrix decompositions, e.g. RPCA, MahNMF robustly estimates the low-rank part and the sparse part of a non-negative matrix and thus performs effectively when data are contaminated by outliers.

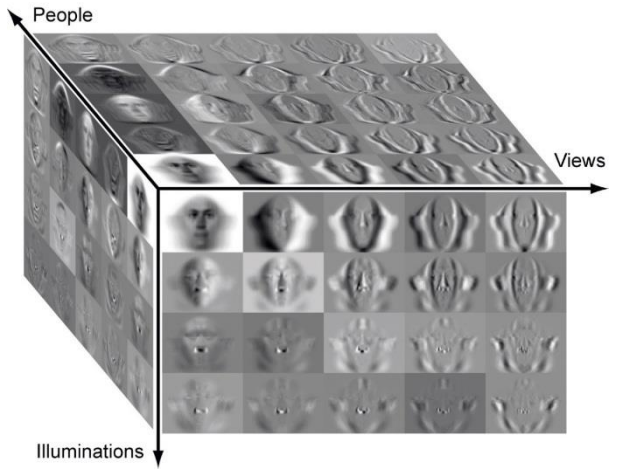
https://sites.google.com/site/nmfsolvers/

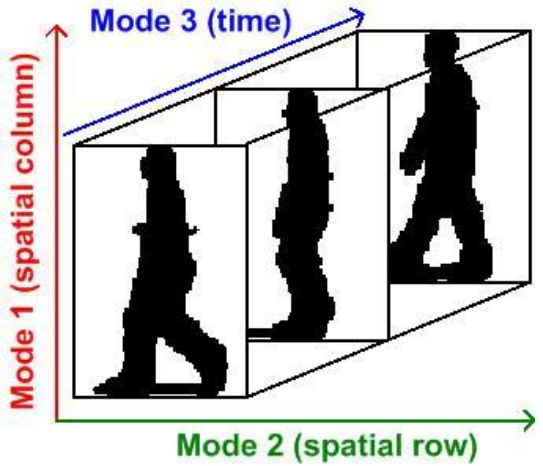
Introduction to tensors

Introduction to tensors

Tensors are simply mathematical objects that can be used to describe physical properties. In fact tensors are merely a generalization of scalars, vectors and matrices; a scalar is a zero rank tensor, a vector is a first rank tensor and a matrix is the second rank tensor.

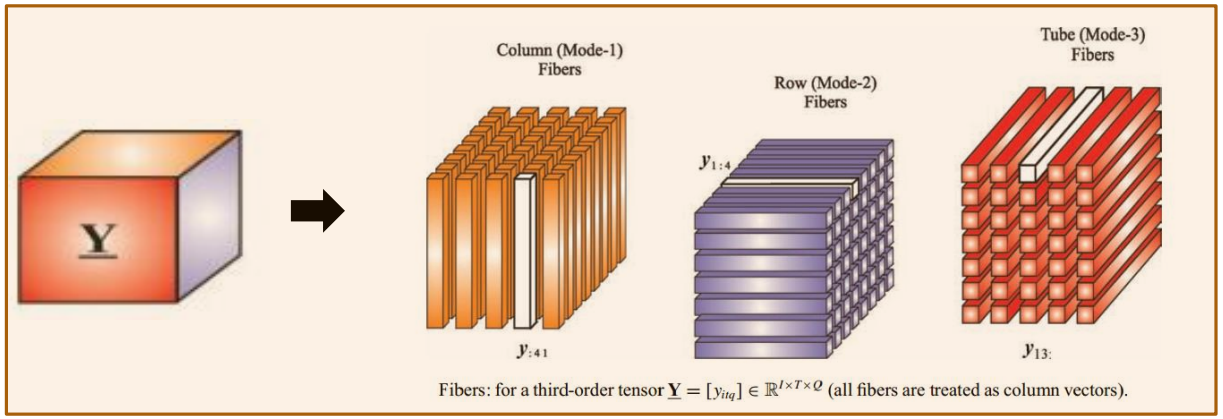






Introduction to tensors

Subarrays, **tubes** and **slices** of a 3rd order tensor.



Fibers: for a third-order tensor $\underline{\mathbf{Y}} = [y_{itq}] \in \mathbb{R}^{I \times T \times Q}$ (all fibers are treated as column vectors).

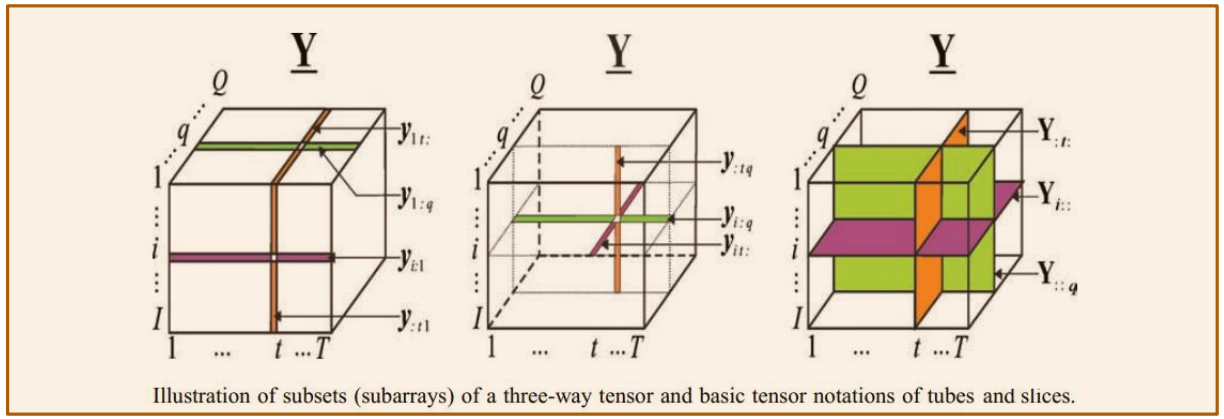
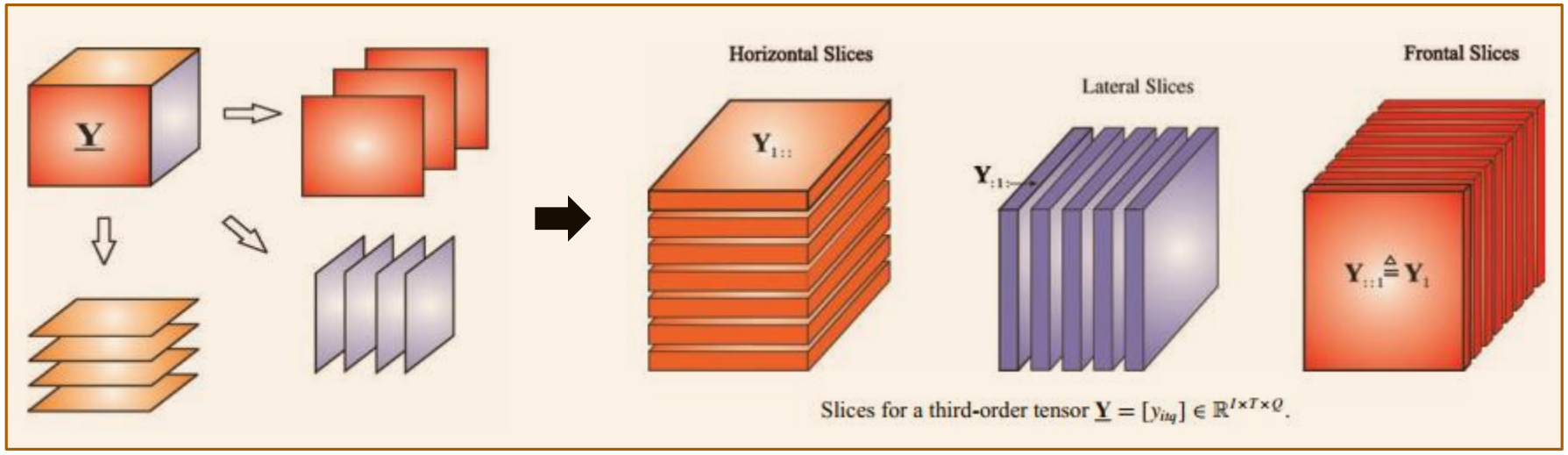


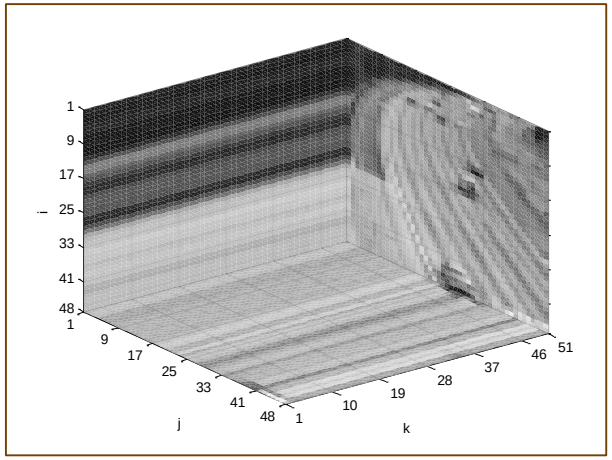
Illustration of subsets (subarrays) of a three-way tensor and basic tensor notations of tubes and slices.

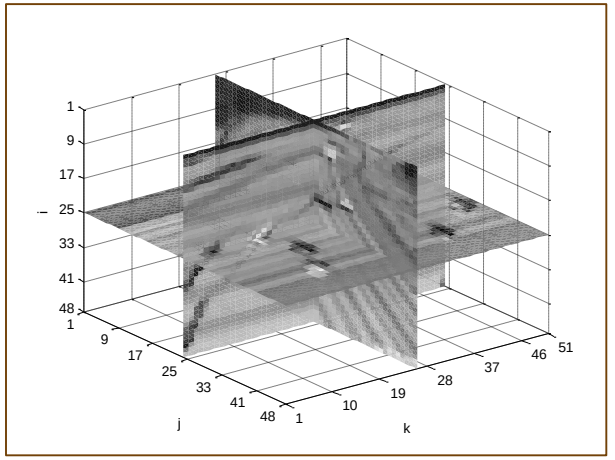
Introduction to tensors

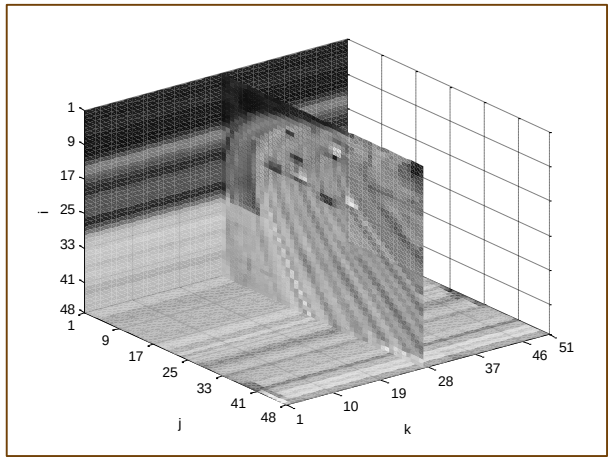
Matricization of a 3rd order tensor.



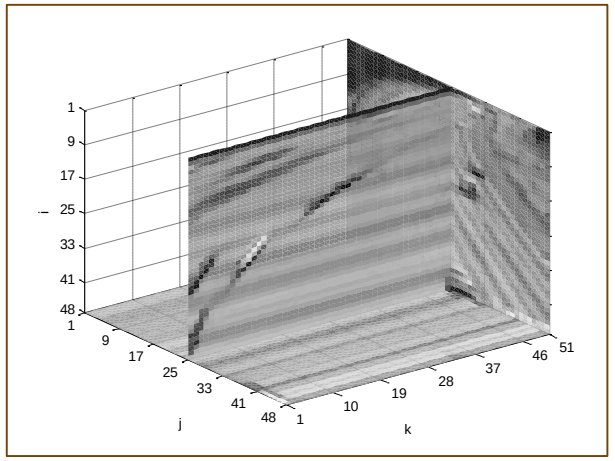
Horizontal, **vertical** and **frontal** slices from a 3rd order tensor



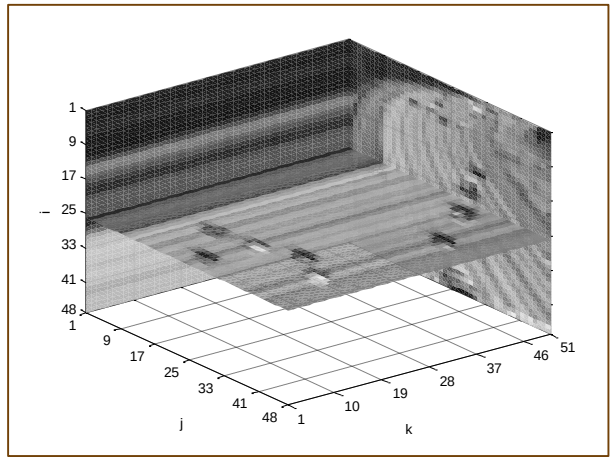




Frontal



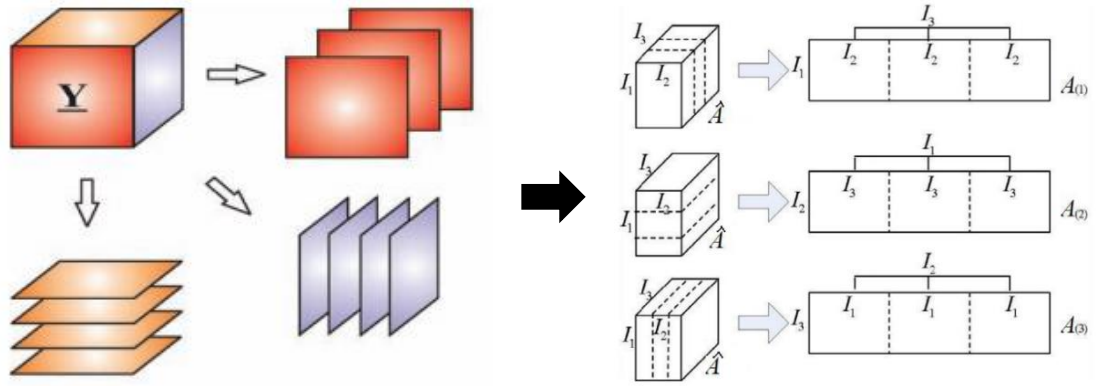
Vertical



Horizontal

Introduction to tensors

Unfolding a 3rd order tensor.



Introduction to tensors

Tensor transposition

- While there is only one way transpose a matrix there are an exponential number of ways to transpose an order-n tensor.

The 3rd order case:

Six possibilities...

If $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, then there are $6 = 3!$ possible transpositions identified by the notation $\mathcal{A}^{<[i\,j\,k]>}$ where $[i\ j\ k]$ is a permutation of $[1\ 2\ 3]$:

$$\mathcal{B} = \left\{ \begin{array}{l} \mathcal{A}^{<[1\,2\,3]>} \\ \mathcal{A}^{<[1\,3\,2]>} \\ \mathcal{A}^{<[2\,1\,3]>} \\ \mathcal{A}^{<[2\,3\,1]>} \\ \mathcal{A}^{<[3\,1\,2]>} \\ \mathcal{A}^{<[3\,2\,1]>} \end{array} \right\} \Longrightarrow \left\{ \begin{array}{l} b_{ijk} \\ b_{ikj} \\ b_{jik} \\ b_{jki} \\ b_{kij} \\ b_{kji} \end{array} \right\} = a_{ijk}$$

for $i = 1{:}n_1,\ j = 1{:}n_2,\ k = 1{:}n_3$.

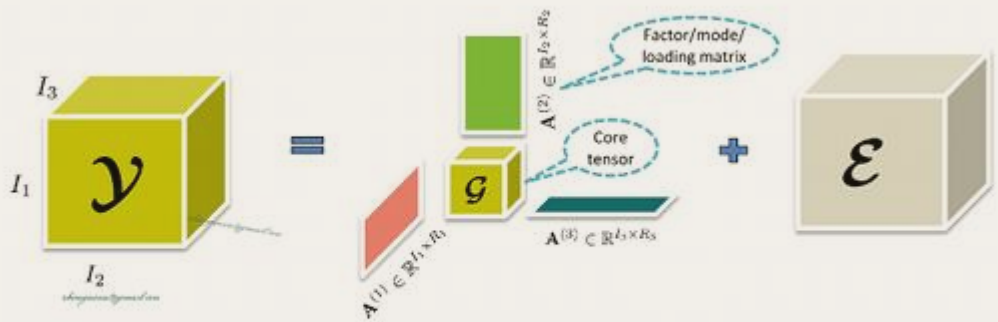
Tensor decomposition methods

Approaches:

- Tucker / HOSVD
- CANDECOMP-PARAFAC (CP)
- Hierarchical Tucker (HT)
- Tensor-Train decomposition (TT)
- NTF (Non-negative Tensor Factorization)
- NTD (Non-negative Tucker Decomposition)
- NCP (Non-negative CP Decomposition)

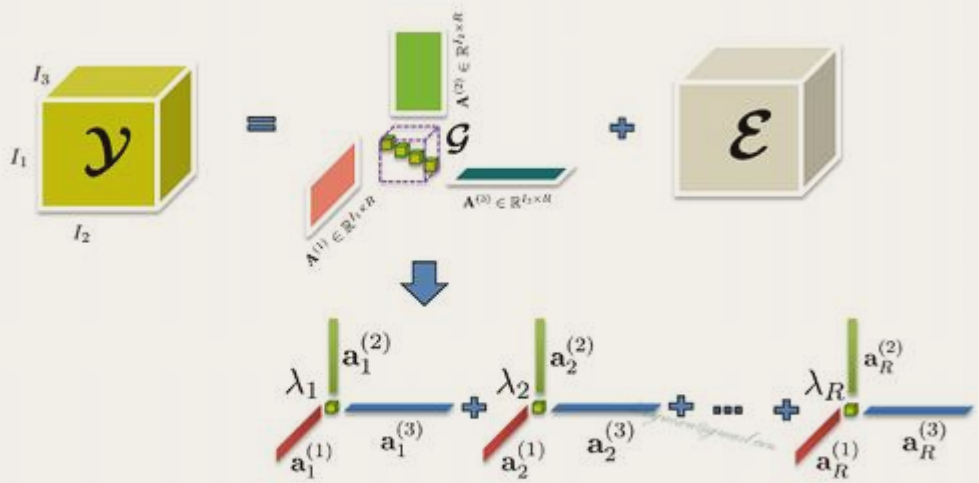
References:
Tensor Decompositions and Applications (Kolda and Bader, 2008)

Tucker Decomposition



$$\mathcal{Y} = \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \mathbf{A}^{(3)} + \mathcal{E}$$

CP Decomposition



- $R_1 = R_2 = R_3 = R$.
- $\mathcal{G}$ is super diagonal.

$$\mathcal{Y} = \sum_r \lambda_r \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \mathbf{a}_r^{(3)} + \mathcal{E}$$

Tucker / HoSVD

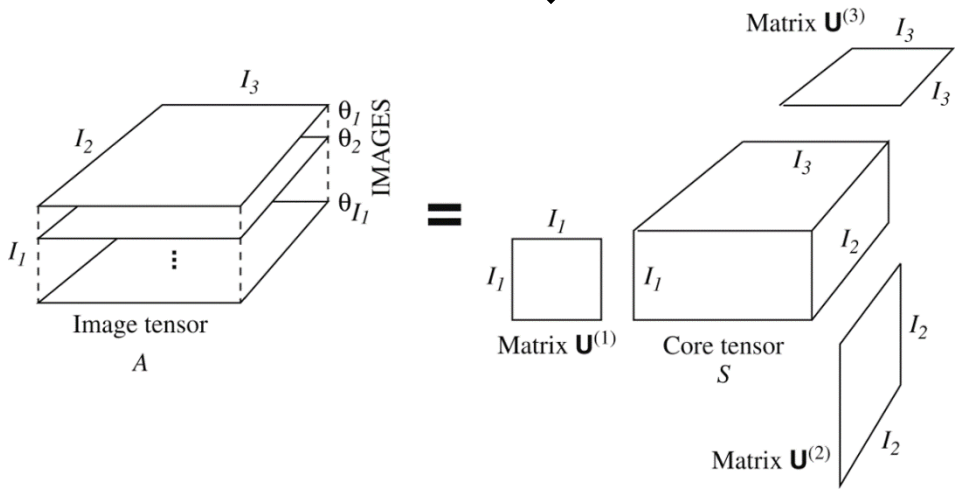
The Higher Order Singular Value Decomposition (HOSVD)

The HOSVD of a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ involves computing the matrix SVDs of its modal unfoldings $\mathcal{A}_{(1)}, \ldots, \mathcal{A}_{(d)}$. This results in a representation of $\mathcal{A}$ as a sum of rank-1 tensors.

```
function [S,U] = HOSVD(A)
% A is an n(1)-by-...-by-n(d) tensor.
% U is a length-d cell array with the
%   property that U{k} is the left singular
%   vector matrix of A's mode-k unfolding.
% S is an n(1)-by-...-by-n(d) tensor given by
%    A x1 U{1} x2 U{2} ... xd U{d}

S = A;
for k=1:length(A.size)
    C = tenmat(A,k);
    [U{k},Sigma,V] = svd(C.data);
    S = ttm(S,U{k}',k);
end
```

$$\mathcal{A} = \mathcal{S} \times_{n=1}^{N} U^{(n)}$$



CP

The CP model is a special case of the Tucker model, where the core tensor is superdiagonal and the number of components in the factor matrices is the same.

Solving by ALS (alternating least squares) framework

```
n = [ 5 6 7 ]; rmax = 35;
% Generate a random tensor...
A = tenrand(n);
for r = 1:rmax
   % Find the closest length-r ktensor...
   X = cp_als(A,r);
   % Display the fit...
   E = double(X)-double(A);
   fit = norm(reshape(E,prod(n),1));
   fprintf('r = %1d, fit = %5.3e\n',r,fit);
end
```

The CP Approximation Problem

Given $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and r, determine $\lambda \in \mathbb{R}^r$, $F \in \mathbb{R}^{n_1 \times r}$, $G \in \mathbb{R}^{n_2 \times r}$, and $H \in \mathbb{R}^{n_3 \times r}$ so that

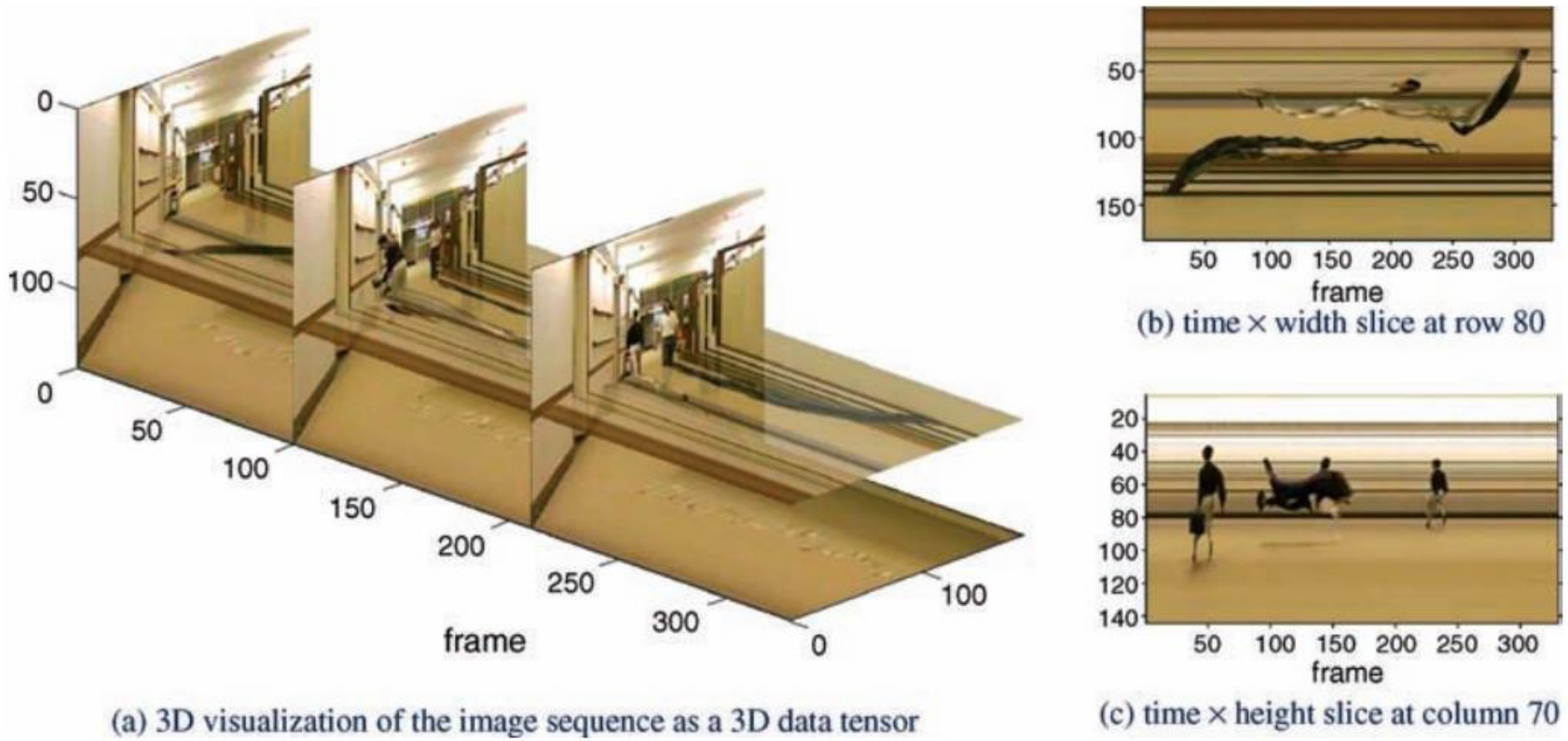
$$\mathcal{A} \approx [[\lambda; F, G, H]] = \mathcal{X}.$$

What About r?

In the CP approximation problem we have assumed that r, the length of the approximating ktensor, is given:

$$\mathcal{A} \approx \mathcal{X} = \sum_{j=1}^{r} \lambda_j U_1(:,j) \circ \cdots \circ U_d(:,j)$$

We can think of $\mathcal{X}$ as a rank-r approximation to $\mathcal{A}$.

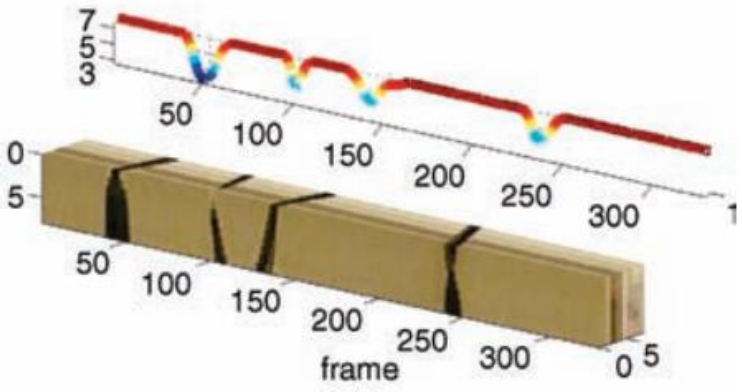


(a) 3D visualization of the image sequence as a 3D data tensor

(b) time × width slice at row 80

(c) time × height slice at column 70

Figure 7.34 Visualization of the Hall Monitor image sequence as a 3D tensor of height × width × time (frame).



(b) A $8 \times 8 \times 3 \times 330$ sub-tensor at pixel (49,65)

(a) Background of the Hall Monitor sequence

Listing 7.13 Background estimation for the `HallMonitor` sequence.

```
% Copyright 2008 by Anh Huy Phan and Andrzej Cichocki
% Load Hall sequence
clear
load Hall4Dtensor;
%% Process full block
sT = size(T);
blksize = 8;
d = zeros(sT(4),prod(sT(1:2))/blksize^2);
kblk = 0;
xy = zeros(1,2);
for xu = 1:8:sT(1)-7
    for yu = 1:8:sT(2)-7
        kblk = kblk + 1;
        Tblk  = T(xu:xu+blksize-1, yu:yu+blksize-1,:,:);

        %% Factorize subtensor with Parafac algorithms R = 1
        Yblk = permute(tensor(Tblk),[4 1 2 3]);
        options = struct('verbose',1,'tol',1e-6,'maxiters',500,...
            'init',2,'nonlinearproj',1);
        [X_hals,Uinit,A_,ldam,iter] = parafac_hals(Yblk,1,options);
        d(:,kblk) = double(A_{1});
        xy(kblk,:) = [xu yu];
    end
end

%% Find stationary blocks and build background image
maxd = max(d); mind = min(d);
thresh = 0.005;
Imbgr = zeros(sT(1:3));
for k = 1:size(d,2);
    edges = [mind(k):thresh:maxd(k) maxd(k)+eps] ;
    [n,bin] = histc(d(:,k),edges);
    m = mode(bin);
    indbgr = find((d(:,k)≥edges(m)) & (d(:,k)≤ edges(m+1)));
    bgrblk  = median(T(xy(k,1):xy(k,1)+blksize-1,...
        xy(k,2):xy(k,2)+blksize-1,:,indbgr),4);
    Imbgr(xy(k,1):xy(k,1)+blksize-1, xy(k,2):xy(k,2)+blksize-1,:) = bgrblk;
end

%% Display the estimated background image
imshow(Imbgr)
```

Tensor decomposition methods

Softwares

- There are several MATLAB toolboxes available for dealing with tensors in CP and Tucker decomposition, including the **Tensor Toolbox**, the **N-way toolbox**, the **PLS Toolbox**, and the **Tensorlab**. The **TT-Toolbox** provides MATLAB classes covering tensors in TT and QTT decomposition, as well as linear operators. There is also a Python implementation of the TT-toolbox called **ttpy**. The **htucker** toolbox provides a MATLAB class representing a tensor in HT decomposition.

Incremental SVD

Incremental SVD

Problem:

- The matrix factorization step in SVD is computationally very expensive.

Solution:

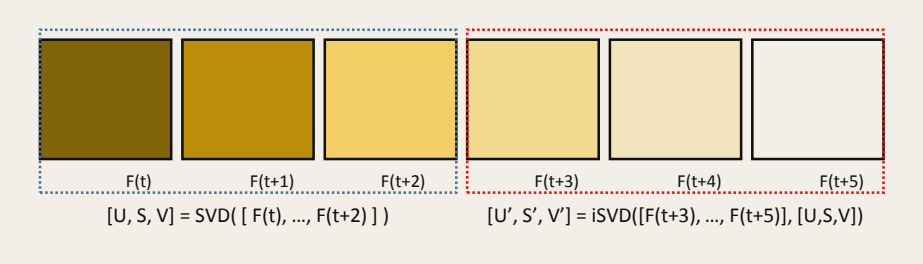
- Have a small pre-computed SVD model, and build upon this model incrementally using inexpensive techniques.

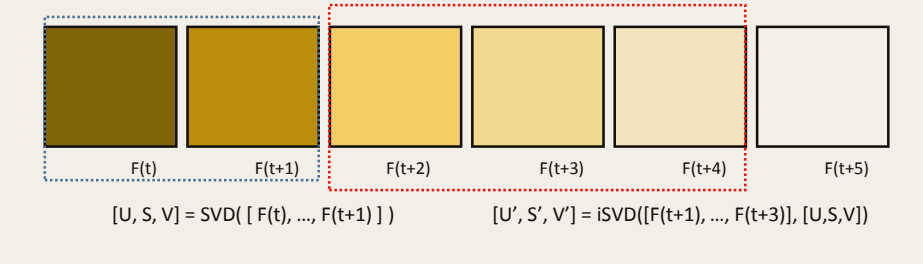
Businger (1970) and Bunch and Nielsen (1978) are the first authors who have proposed to update SVD sequentially with the arrival of more samples, i.e. appending/removing a row/column.

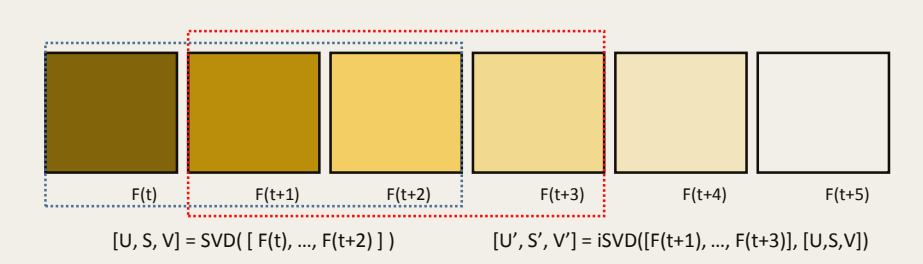
Subsequently various approaches have been proposed to update the SVD more efficiently and supporting new operations.

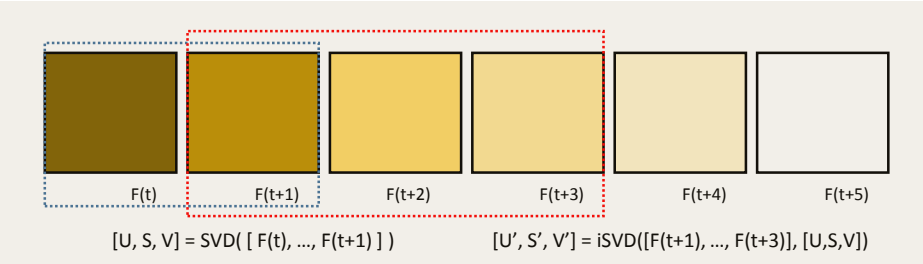
References:
Businger, P.A. Updating a singular value decomposition. 1970
Bunch, J.R.; Nielsen, C.P. Updating the singular value decomposition. 1978

Incremental SVD



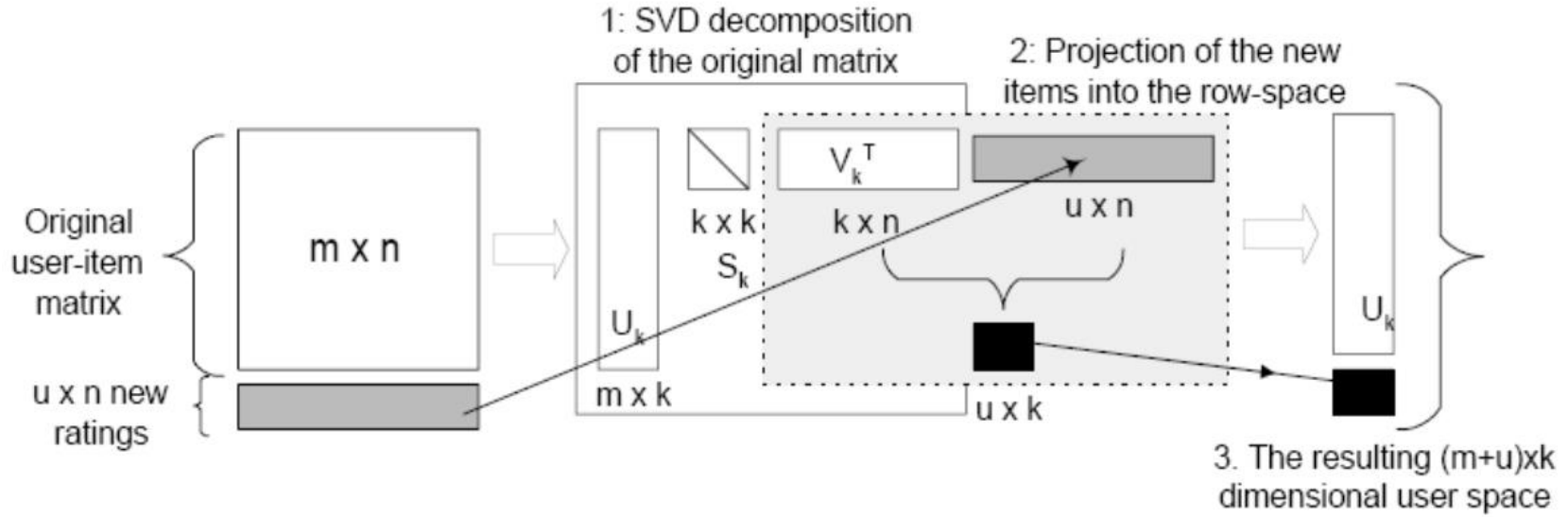






Incremental SVD

Updating operation proposed by Sarwar et al. (2002):



References:
Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems (Sarwar et al., 2002)

Incremental SVD

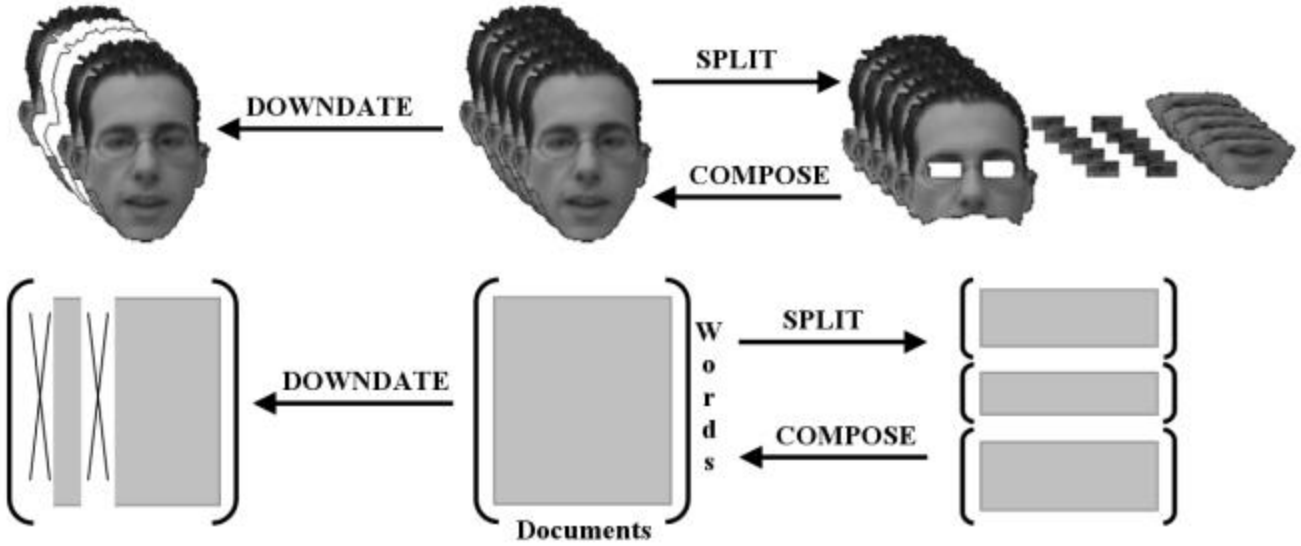
Operations proposed by Matthew Brand (2006):

Operation	Known	Desired	$\mathbf{a}$	$\mathbf{b}^\top$
Update	$\mathbf{US}[\mathbf{V}^\top \quad \mathbf{0}] = [\mathbf{X} \quad \mathbf{0}]$	$\mathbf{U}'\mathbf{S}'\mathbf{V}'^\top = [\mathbf{X} \quad \mathbf{c}]$	$\mathbf{c}$	$[0, \ldots, 0, 1]$
Downdate	$\mathbf{USV}^\top = [\mathbf{X} \quad \mathbf{c}]$	$\mathbf{U}'\mathbf{S}'\mathbf{V}'^\top = \mathbf{X}$	$-\mathbf{c}$	$[0, \ldots, 0, 1]$
Revise	$\mathbf{USV}^\top = [\mathbf{X} \quad \mathbf{c}]$	$\mathbf{U}'\mathbf{S}'\mathbf{V}'^\top = [\mathbf{X} \quad \mathbf{d}]$	$\mathbf{d} - \mathbf{c}$	$[0, \ldots, 0, 1]$
Recenter	$\mathbf{USV}^\top = \mathbf{X}$	$\mathbf{U}'\mathbf{S}'\mathbf{V}'^\top = \mathbf{X}(\mathbf{I} - \frac{1}{q}\mathbf{1}\mathbf{1}^\top)$	$-\frac{1}{q}\mathbf{X}\mathbf{1}$	$\mathbf{1}^\top \doteq [1, \ldots, 1]$

References:
Fast low-rank modifications of the thin singular value decomposition (Matthew Brand, 2006)

Incremental SVD

Operations proposed by Melenchon and Martinez (2007):



References:
Efficiently Downdating, Composing and Splitting Singular Value Decompositions Preserving the Mean Information (Melenchón and Martínez, 2007)

Incremental SVD algorithms in Matlab

By Christopher Baker (Baker et al., 2012)

http://www.math.fsu.edu/~cbaker/IncPACK/

[Up,Sp,Vp] = SEQKL(A,k,t,[U S V])

Original version only supports the Updating operation

Added exponential forgetting factor to support Downdating operation

By David Ross (Ross et al., 2007)

http://www.cs.toronto.edu/~dross/ivt/

[U, D, mu, n] = sklm(data, U0, D0, mu0, n0, ff, K)

Supports mean-update, updating and downdating

By David Wingate (Matthew Brand, 2006)

http://web.mit.edu/~wingated/www/index.html

[Up,Sp,Vp] = svd_update(U,S,V,A,B,force_orth)

update the SVD to be [X + A'*B]=Up*Sp*Vp' (a general matrix update).

[Up,Sp,Vp] = addblock_svd_update(U,S,V,A,force_orth)

update the SVD to be [X A] = Up*Sp*Vp' (add columns [ie, new data points])

size of **Vp** increases

A must be square matrix

[Up,Sp,Vp] = rank_one_svd_update(U,S,V,a,b,force_orth)

update the SVD to be [X + a*b'] = Up*Sp*Vp' (that is, a general rank-one update. This can be used to add columns, zero columns, change columns, recenter the matrix, etc.).

Incremental Tensor Learning

Incremental tensor learning

Proposed by Sun et al. (2008)

- Dynamic Tensor Analysis (DTA)
- Streaming Tensor Analysis (STA)
- Window-based Tensor Analysis (WTA)

References:
Incremental tensor analysis: Theory and applications (Sun et al, 2008)

Incremental tensor learning

Dynamic Tensor Analysis (DTA)

- [T, C] = DTA(Xnew, R, C, alpha)
- Approximately updates tensor PCA, according to new tensor Xnew, old variance matrices in C and he specified dimensions in vector R. The input Xnew is a tensor. The result returned in T is a tucker tensor and C is the cell array of new variance matrices.

Algorithm 2.2: DTA (new tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_M}$, old projection matrices $\mathbf{U}^{(d)}|_{d=1}^{M}$, energy matrices $\mathbf{S}^{(d)}|_{d=1}^{M}$, forgetting factor λ, output ranks $r_d|_{d=1}^{M}$)

Output: projection matrices $\mathbf{U}^{(d)}|_{d=1}^{M} \in \mathbb{R}^{n_d \times r_d}$ and core tensor $\mathcal{Y}_t|_{t=1}^{T} \in \mathbb{R}^{r_1 \times r_2 \times \cdots \times r_M}$

1 **for** $d = 1$ *to* M **do**

  // Reconstruct the old covariance

2 $\mathbf{C}^{(d)} \leftarrow \mathbf{U}^{(d)}\mathbf{S}^{(d)}\mathbf{U}^{(d)\mathsf{T}}$

  // Update the covariance matrix

3 $\mathbf{C}^{(d)} = \lambda\mathbf{C}^{(d)} + \mathbf{X}_{(d)}\mathbf{X}_{(d)}^{\mathsf{T}}$

4 Set $\mathbf{U}^{(d)}$ be the top r_d eigenvectors of $\mathbf{C}^{(d)}$.

5 $\mathcal{Y} = \mathcal{X} \times_1 \mathbf{U}^{(1)\mathsf{T}} \cdots \times_M \mathbf{U}^{(M)\mathsf{T}}$; // Compute the core tensor

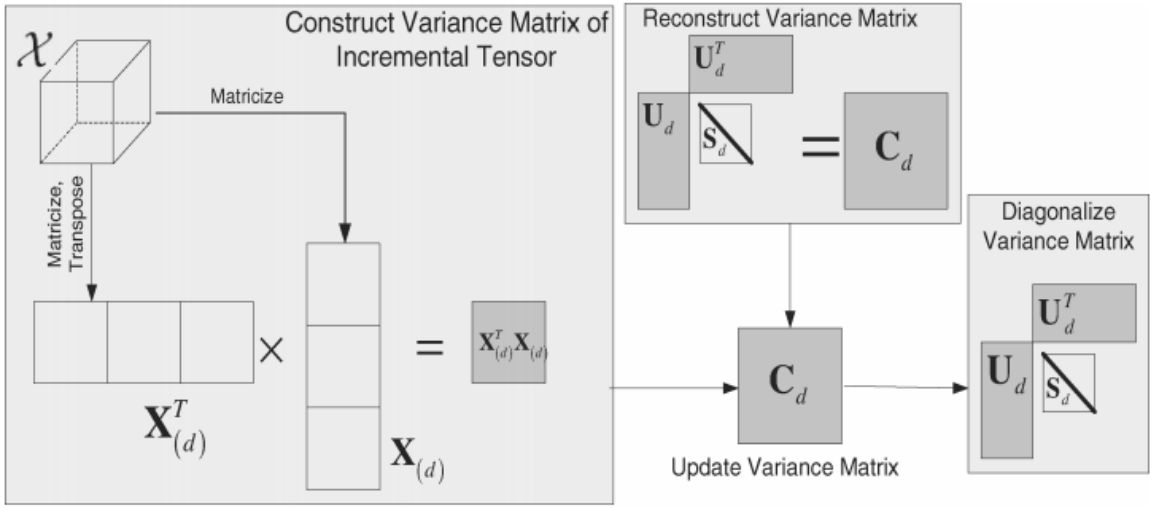


Fig. 7. New tensor $\mathcal{X}$ is matricized along the dth mode. Then covariance matrix $\mathbf{C}_d$ is updated by $\mathbf{X}_{(d)}^T\mathbf{X}_{(d)}$. The projection matrix $\mathbf{U}_d$ is computed by diagonalizing $\mathbf{C}_d$.

Incremental tensor learning

Streaming Tensor Analysis (STA)

- [Tnew, S] = STA(Xnew, R, T, alpha, samplingPercent)
- Approximately updates tensor PCA, according to new tensor Xnew, old tucker decomposition T and he specified dimensions in vector R. The input Xnew is a tensor or sptensor. The result returned in T is a new tucker tensor for Xnew, S has the energy vector along each mode.

Algorithm 2.4: STA(new tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_M}$, old projection matrices $\mathbf{U}^{(d)}|_{d=1}^{M}$, diagonal energy matrices $\mathbf{S}^{(d)} \in \mathbb{R}^{r_d \times r_d}$, forgetting factor λ)

Output: projection matrices $\mathbf{U}^{(d)}|_{d=1}^{M} \in \mathbb{R}^{n_d \times r_d}$ and core tensor $\mathcal{Y}_t|_{t=1}^{T} \in \mathbb{R}^{r_1 \times r_2 \times \cdots \times r_M}$

1 **for** $d = 1$ *to* M **do**
2 **foreach** *column vector* $\mathbf{x}$ *in* $\mathbf{X}_{(d)}$ **do**
3 $(\mathbf{U}^{(d)}, \mathbf{S}^{(d)}) \leftarrow$ TrackU($\mathbf{U}^{(d)}$, $\mathbf{x}$,$\mathbf{S}^{(d)}$, λ) – Algorithm 2.3
4 $\mathcal{Y} = \mathcal{X} \times_1 \mathbf{U}^{(1)\mathsf{T}} \cdots \times_M \mathbf{U}^{(M)}$; // Compute the core tensor

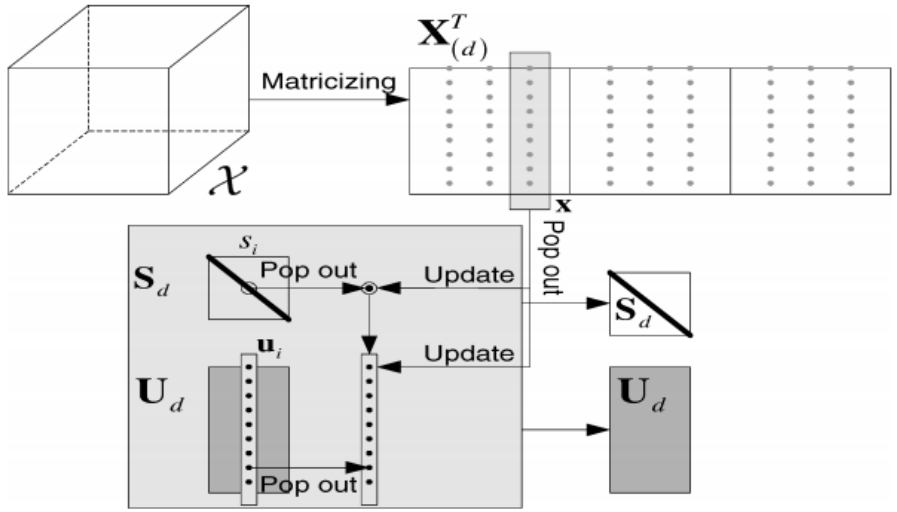


Fig. 8. New tensor $\mathcal{X}$ is matricized along the dth mode. For every row of $\mathbf{X}_d$, we update the projection matrix $\mathbf{U}_d$, and $\mathbf{S}_d$ helps determine the update size.

Incremental tensor learning

Window-based Tensor Analysis (WTA)

- [T, C] = WTA(Xnew, R, Xold, overlap, type, C)
- Compute window-based tensor decomposition, according to Xnew (Xold) the new (old) tensor window, overlap is the number of overlapping tensors and C variance matrices except for the time mode of previous tensor window and the specified dimensions in vector R, type can be 'tucker' (default) or 'parafac'. The input Xnew (Xold) is a tensor,sptensor, where the first mode is time. The result returned in T is a tucker or kruskal tensor depending on type and C is the cell array of new variance matrices.

Algorithm 2.5: WTA (new tensor $\mathcal{D}_n \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_M}$, old tensor $\mathcal{D}_{n-W} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_M}$, old covariance matrices $\mathbf{C}^{(d)}|_{d=1}^{M}$, $\mathbb{R}^{r_0 \times \cdots \times r_M}$)

Output: new covariance matrix $\mathbf{C}^{(d)}|_{d=1}^{M}$, $\mathbb{R}^{r_0 \times \cdots \times r_M}$, projection matrices $\mathbf{U}^{(d)}|_{d=1}^{M} \in \mathbb{R}^{n_d \times r_d}$ and core tensor $\mathcal{Y}_t|_{t=1}^{T} \in \mathbb{R}^{r_1 \times r_2 \times \cdots \times r_M}$

// Initialize every mode except time

1 **for** $d = 1$ *to* M **do**

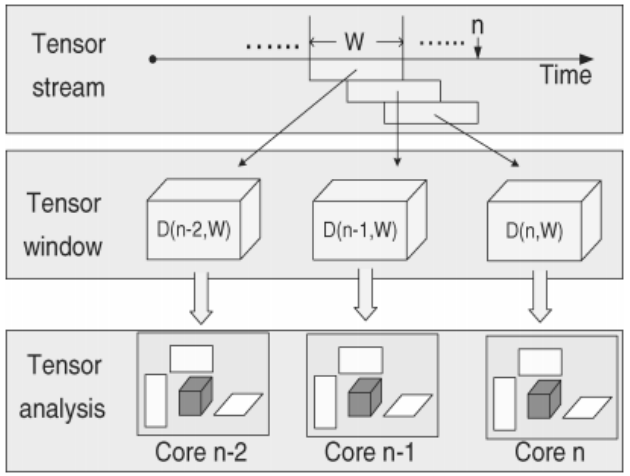
2 Mode-d matricize $\mathcal{D}_{n-W}(\mathcal{D}_n)$ as $\mathbf{D}_{n-W,(d)}(\mathbf{D}_{n,(d)})$

3 Update $\mathbf{C}^{(d)} \leftarrow \mathbf{C}^{(d)} - \mathbf{D}_{n-W,(d)}^{T}\mathbf{D}_{n-W,(d)} + \mathbf{D}_{n,(d)}^{T}\mathbf{D}_{n,(d)}$

4 Diagonalization $\mathbf{C}^{(d)} = \mathbf{U}^{(d)}\mathbf{\Lambda}_d\mathbf{U}^{(d)\mathsf{T}}$

5 Truncate $\mathbf{U}^{(d)}$ to first r_d columns

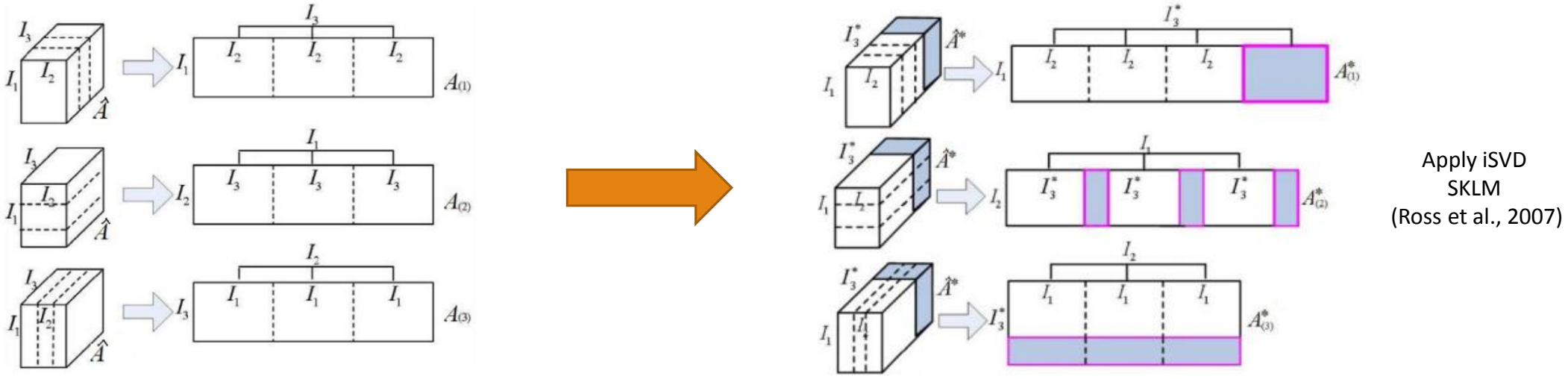
6 Apply the iterative algorithm with the new initialization



Incremental tensor learning

Proposed by Hu et al. (2011)

- Incremental rank-(R1,R2,R3) tensor-based subspace learning
  - IRTSA-GBM (grayscale)
  - IRTSA-CBM (color)

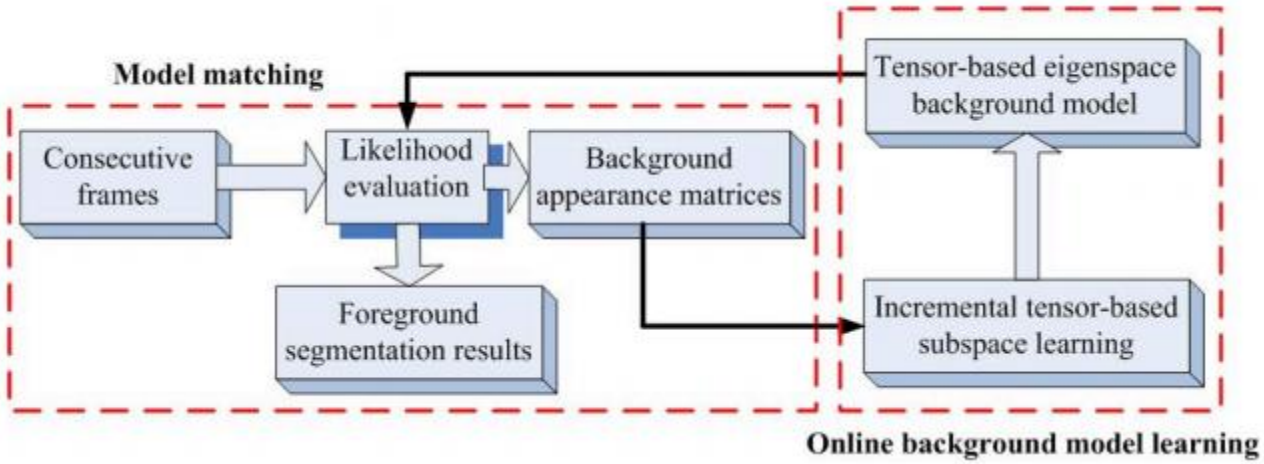


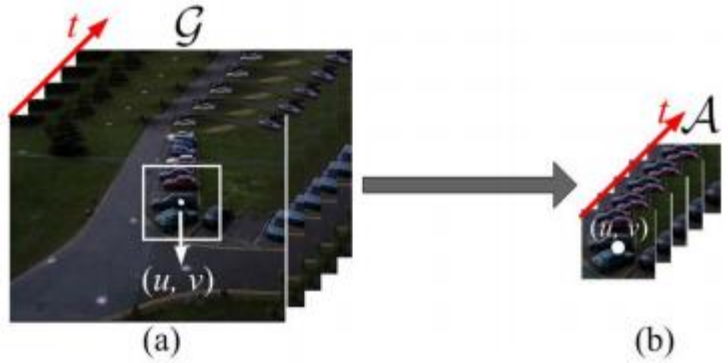
Apply iSVD
SKLM
(Ross et al., 2007)

References:
Incremental Tensor Subspace Learning and Its Applications to Foreground Segmentation and Tracking (Hu et al., 2011)

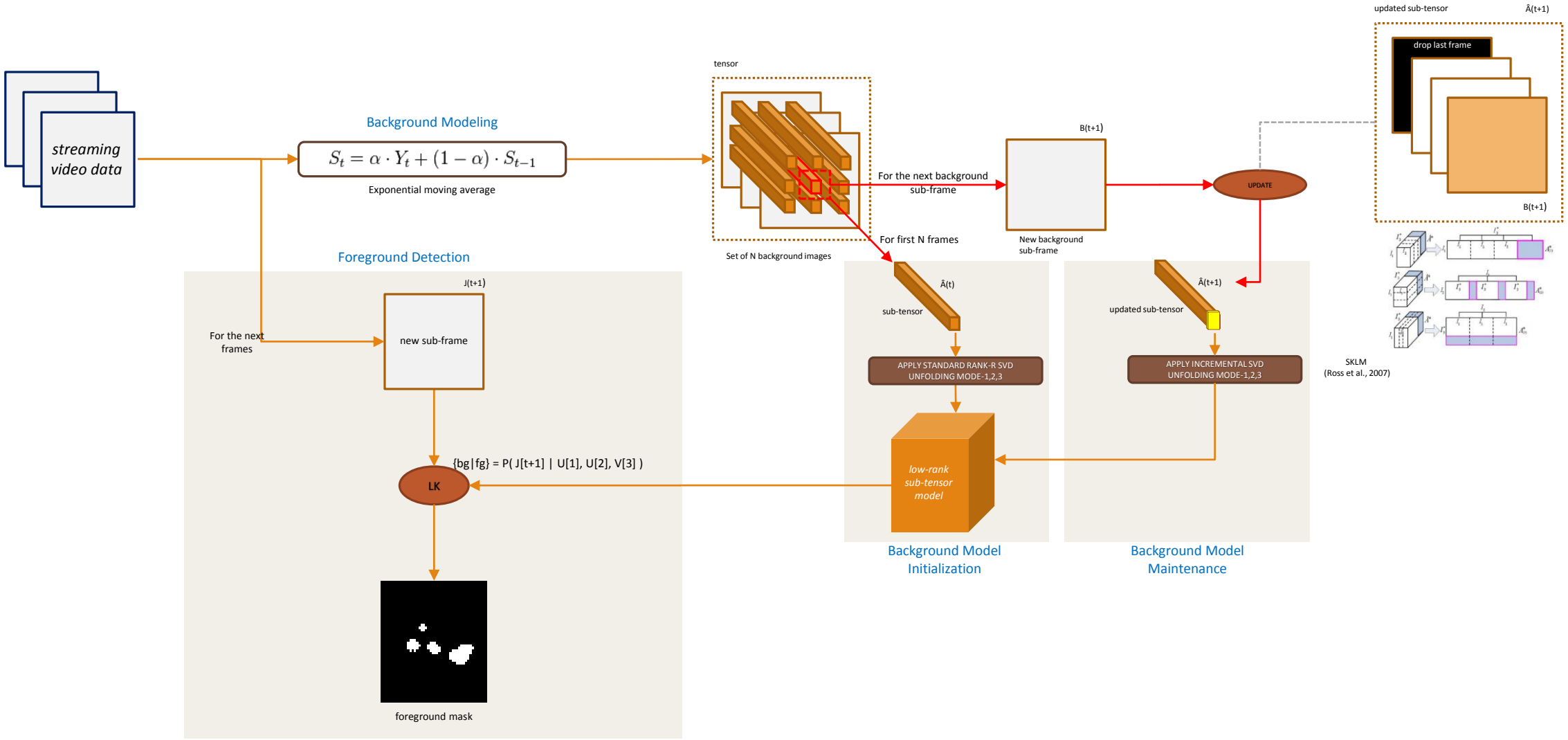
Incremental tensor learning

IRTSA architecture



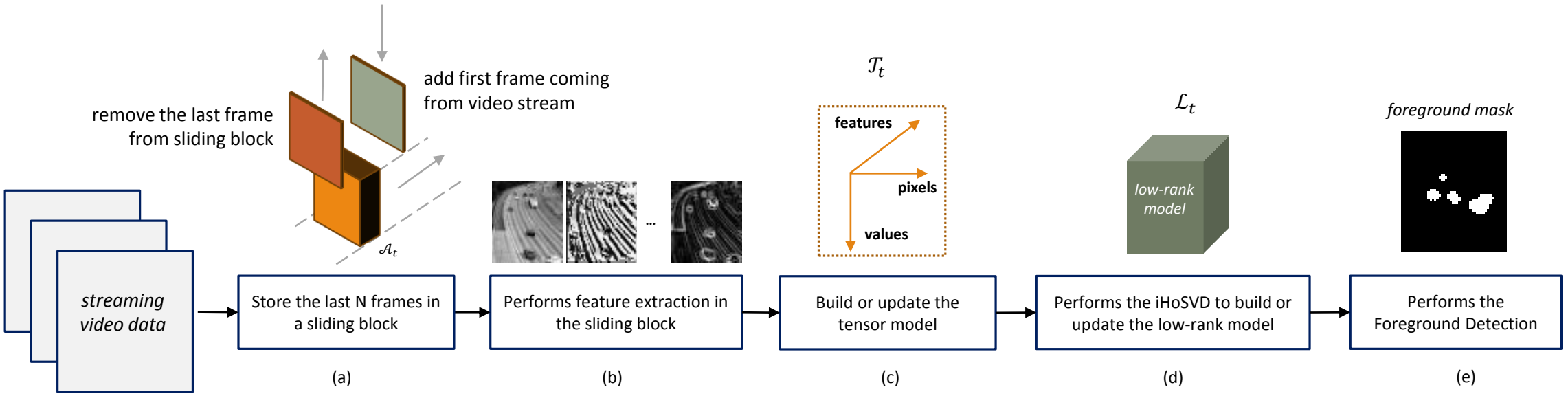


Incremental rank-(R1,R2,R3) tensor-based subspace learning (Hu et al., 2011)



Incremental and Multi-feature Tensor Subspace Learning applied for Background Modeling and Subtraction

Proposed by Sobral et al. (2014)



Highlights:

* Proposes an incremental low-rank HoSVD (iHOSVD) for background modeling and subtraction.
* A unified tensor model to represent the features extracted from the streaming video data.

More info: https://sites.google.com/site/ihosvd/