Low-Rank Matrix Approximation of 2D Data

Chapter Intended Learning Outcomes:

(i) Realize that many real-world signals can be approximated using their lower-dimensional representation

(ii) Understand singular value decomposition (SVD) and principal component analysis (PCA) and their relationship

(iii) Able to apply SVD and PCA in relevant real-world applications

(iv) Understand the basics of tensor decomposition and application

Real-World Signals as Low-Rank Matrices

Many data in our real-world can be modelled as a low-rank matrix.

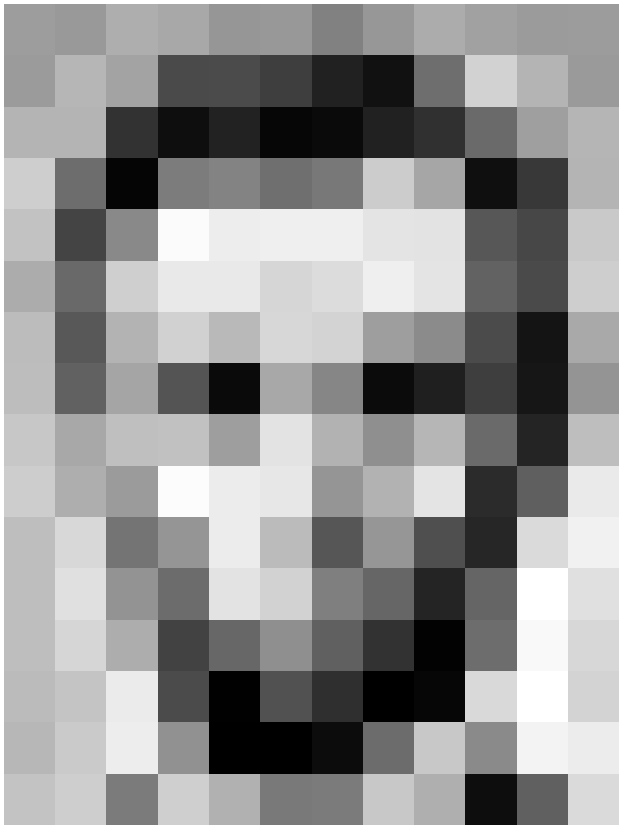
The most direct scenarios correspond to two-dimensional (2D) data such as images.

1D signals such as audio or financial data can be considered as column (or row) vectors of a matrix.

Higher-dimensional or tensor signals can be unfolded to matrices. For example, a 3-D signal or 3rd-order tensor can be unfolded to a matrix of three possible forms.

We may say tensor generalizes vector and matrix.

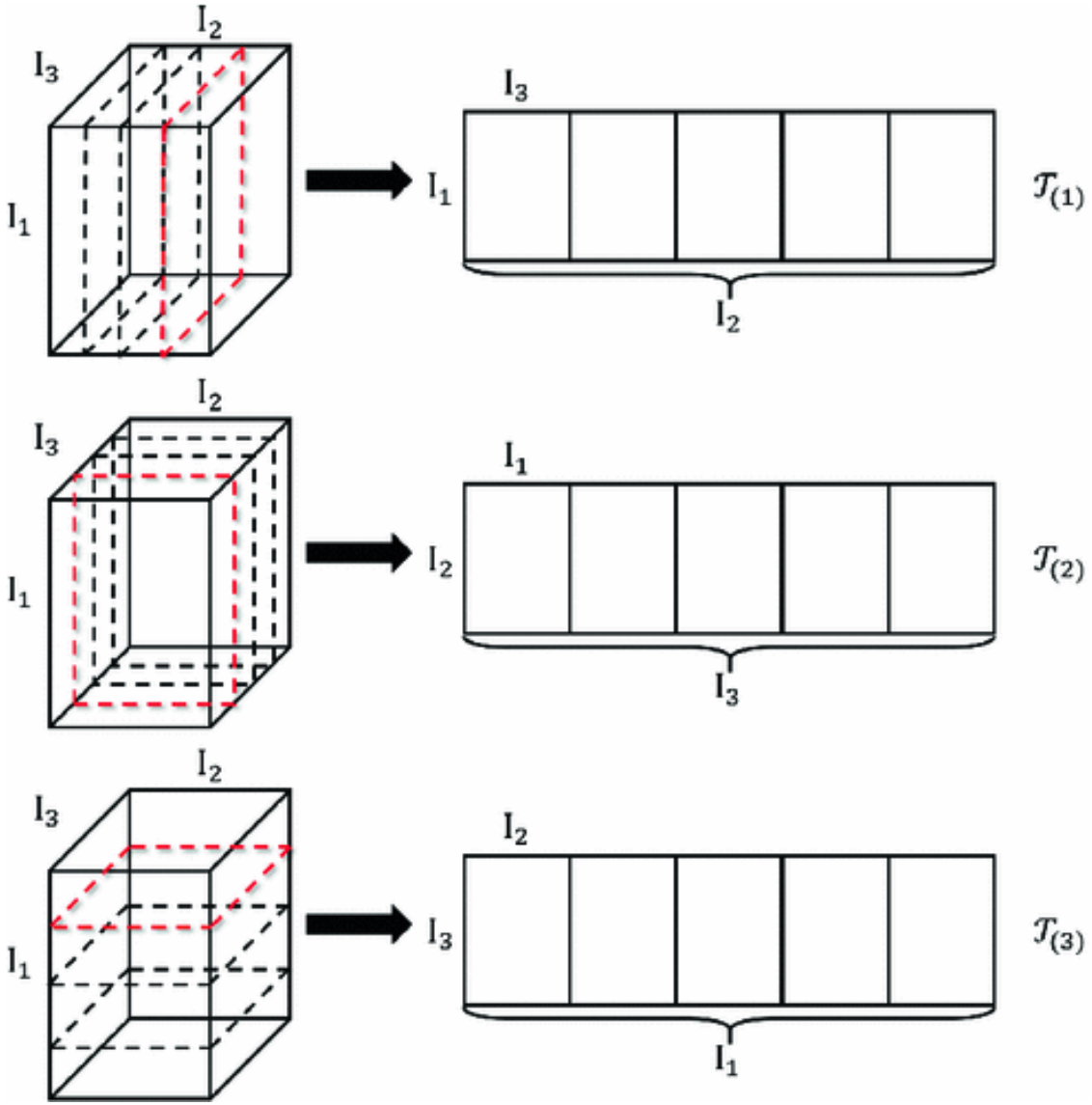
For gray-scale image, the gray level is based on 8-bit representation: 0 (black) to 255 (white):



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Source: https://www.researchgate.net/publication/330902210_The_visual_digital_turn_Using_neural_networks_to_study_historical_images/figures?lo=1



Tensor unfolding illustration

Source: Hong X., Xu Y., Zhao G. (2017) LBP-TOP: A Tensor Unfolding Revisit. In: Chen CS., Lu J., Ma KK. (eds) Computer Vision – ACCV 2016 Workshops. ACCV 2016. Lecture Notes in Computer Science, vol 10116. Springer, Cham

Analogously, vectorization refers to transforming a matrix to vector. For example, we can convert a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ to a column vector of length mn:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \Rightarrow \begin{bmatrix} a_{11} \\ \vdots \\ a_{m1} \\ a_{12} \\ \vdots \\ a_{m2} \\ \vdots \\ a_{1n} \\ \vdots \\ a_{mn} \end{bmatrix} \text{ or } \begin{bmatrix} a_{11} \\ \vdots \\ a_{1n} \\ a_{21} \\ \vdots \\ a_{2n} \\ \vdots \\ a_{m1} \\ \vdots \\ a_{mn} \end{bmatrix}$$

A matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ has full rank if its rank is $r = \min(m, n)$. If $r = n$, this means that all the columns (or rows if $r = m$) are linearly independent.

$\boldsymbol{X} \in \mathbb{R}^{m \times n}$ has low rank if its rank is $r \ll \min(m, n)$. This means that many columns (and rows) are linearly dependent, e.g.,

$$\boldsymbol{X} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 2 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{4 \times 5} \Rightarrow r = \text{rank}(\boldsymbol{X}) = 2$$

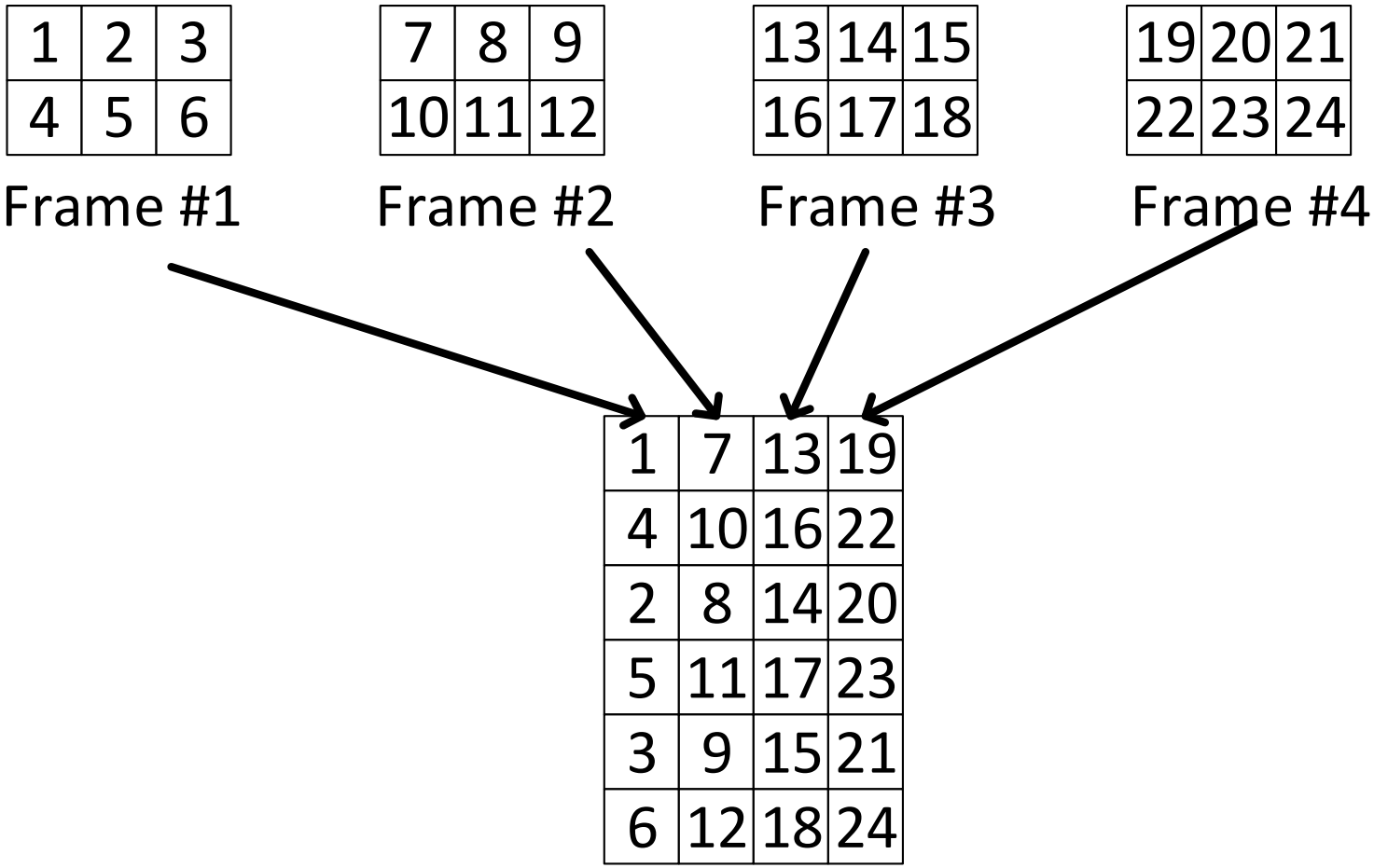
What is the maximum possible rank for a matrix with this dimensions?

Row #1: $\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \end{bmatrix}$

Row #2: $\begin{bmatrix} 1 & 2 & 1 & 2 & 1 \end{bmatrix} = 2 \times \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \end{bmatrix}$

Can you see the linearly dependency among columns?

Video can also be expressed as a matrix by converting each frame (matrix) as a vector, e.g.,



The background component in the video is of low rank.

If the 4 frames are equal, what is the matrix rank?

Singular Value Decomposition (SVD)

SVD is a useful tool to decompose a matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m]\text{diag}\{\sigma_1, \cdots, \sigma_m\}[\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n]^T = \sum_{i=1}^{m} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T \quad (1)$$

where $\boldsymbol{U} \in \mathbb{R}^{m \times m}$ contains the left singular vectors $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m$, $\boldsymbol{V} \in \mathbb{R}^{n \times n}$ contains the right singular vectors $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n$, $\boldsymbol{S} \in \mathbb{R}^{m \times n}$ is diagonal matrix with singular values $\sigma_1, \cdots, \sigma_m$ on the diagonal with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_m$, and $m < n$ is assumed.

The singular vectors are orthogonal such that $\boldsymbol{u}_i^T \boldsymbol{u}_j = 0$, $\boldsymbol{v}_i^T \boldsymbol{v}_j = 0$ for $i \neq j$. To ensure a unique set of $\{\boldsymbol{U}, \boldsymbol{S}, \boldsymbol{V}\}$, $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal, i.e., $\boldsymbol{u}_i^T \boldsymbol{u}_i = \boldsymbol{v}_j^T \boldsymbol{v}_j = 1$.

$\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m$, and $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m$ are unit-norm vectors.

If a 2D signal can be approximated with rank $r < \min(m, n)$, we can write:

$$\boldsymbol{X} \approx \boldsymbol{U}_p \boldsymbol{S}_p \boldsymbol{V}_p^T = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_r] \mathrm{diag}\{\sigma_1, \cdots, \sigma_r\} [\boldsymbol{v}_1, \cdots, \boldsymbol{v}_r]^T = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T \quad (2)$$

which is referred to as truncated SVD.

Example 1

Convert $\boldsymbol{X} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 2 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 2 & 1 & 2 & 1 & 2 \end{bmatrix}$ in the form of SVD.

Using MATLAB:

```
>> X= [1 1 1 1 1; 1 2 1 2 1; 0 1 0 1 0; 1 0 1 0 1; 2 1 2 1 2];
>> [U, S, V] = svd(X)
```

```
U =
   -0.4063   -0.0394   -0.9044    0.1089   -0.0590
   -0.5612   -0.5568    0.2453   -0.4442   -0.3428
   -0.1549   -0.5174    0.0733    0.2728    0.7928
   -0.2515    0.4779   -0.0220   -0.6780    0.4981
   -0.6578    0.4385    0.3406    0.5067   -0.0482

S =
    5.4989         0         0         0         0
         0    2.1823         0         0         0
         0         0    0.0000         0         0
         0         0         0    0.0000         0
         0         0         0         0    0.0000

V =
   -0.4609    0.3477   -0.8139    0.0649         0
   -0.4258   -0.5645    0.0562    0.7049   -0.0000
   -0.4609    0.3477    0.4070   -0.0324   -0.7071
   -0.4258   -0.5645   -0.0562   -0.7049    0.0000
   -0.4609    0.3477    0.4070   -0.0324    0.7071
```

It is clear that $\boldsymbol{X}$ is of rank 2 as there are only two nonzero singular values $\boldsymbol{\sigma}_1 = 5.4989 > \boldsymbol{\sigma}_2 = 2.1823$. As a result, the truncated SVD of $\boldsymbol{X}$ is exactly equal to $\boldsymbol{X}$:

$$\begin{bmatrix} -0.4063 & -0.0394 \\ -0.5612 & -0.5568 \\ -0.4609 & 0.3477 \\ -0.4258 & -0.5645 \\ -0.4609 & 0.3477 \end{bmatrix} \begin{bmatrix} 5.4989 & 0 \\ 0 & 2.1823 \end{bmatrix} \begin{bmatrix} -0.4609 & -0.4258 & -0.4609 & -0.4258 & -0.4609 \\ 0.3477 & -0.5645 & 0.3477 & -0.5645 & 0.3477 \end{bmatrix}$$

and it can be easily verified with

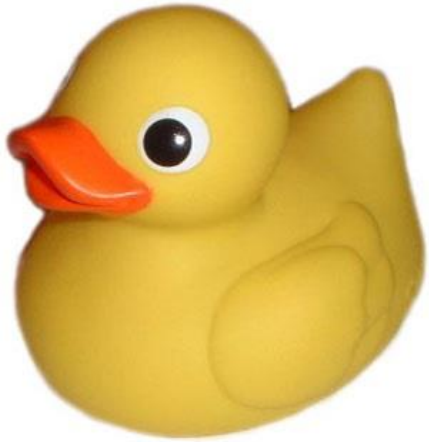
```
>> U(:,1:2)*S(1:2,1:2)*V(:,1:2).'
```

Also:

```
>> (U(:,1)).'*U(:,2)
ans =  -1.1102e-16

>> (U(:,1)).'*U(:,1)
ans = 1
```

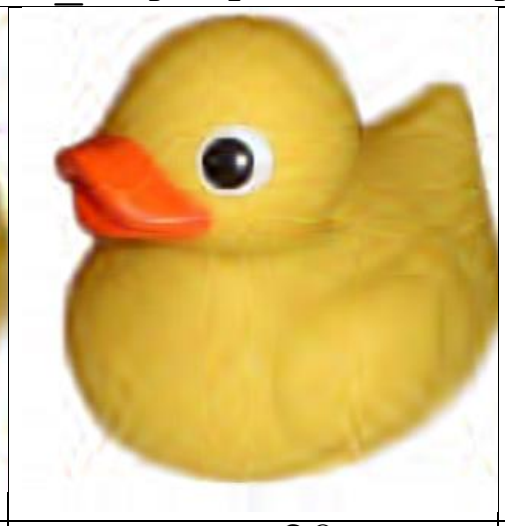
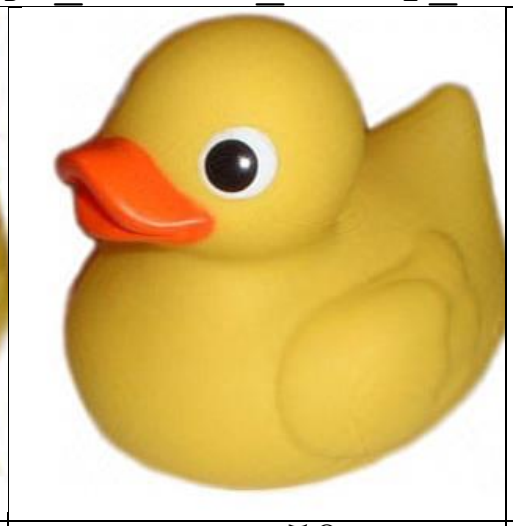
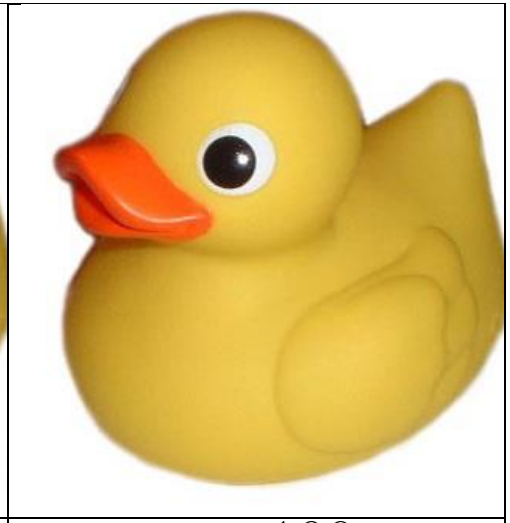
Example 2

Investigate if we can represent an image of rubber ducky of 327 x 327 pixels using a low-rank approximation.

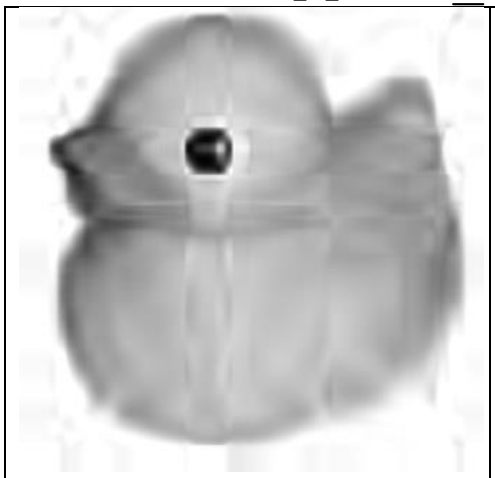
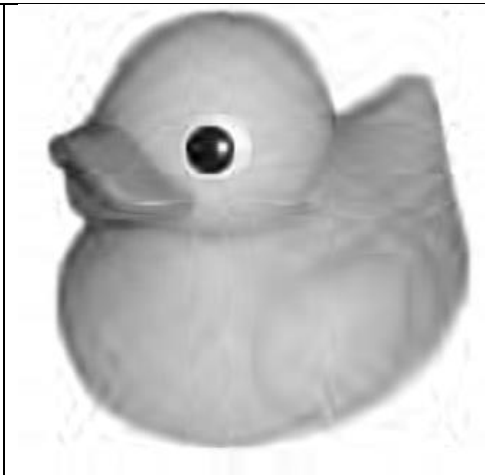
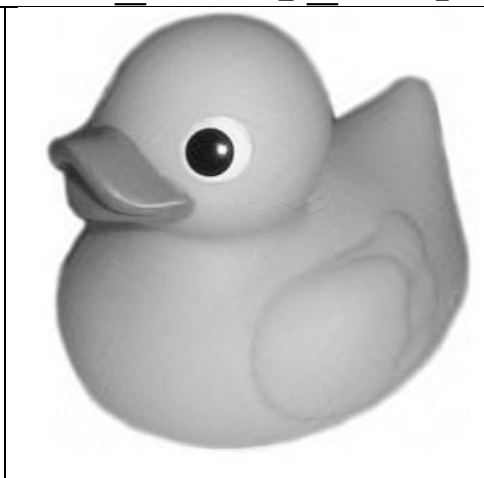
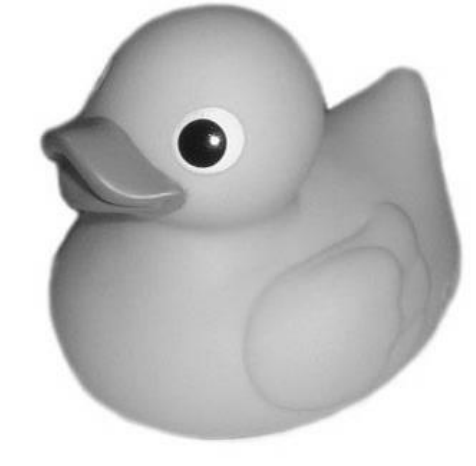


Note that the original RGB color model is a tensor of dimensions 327 x 327 x 3 and we may just perform the SVD for each color separately.

```
>> img = imread('Rubber_Ducky.jpg');
for i = 1:3
    [U(:,:,i),S(:,:,i),V(:,:,i)] = svd(im2double(img(:,:,i)));
end
r=10;
    for i = 1:3
       approx_img(:,:,i) = U(:,1:r,i) * S(1:r,1:r,i) * V(:,1:r,i).';
    end
    imwrite(approx_img,sprintf('rgb_Rubber_Ducky_%d.png', r),'png');
```

			
$r = 10$	$r = 20$	$r = 50$	$r = 100$

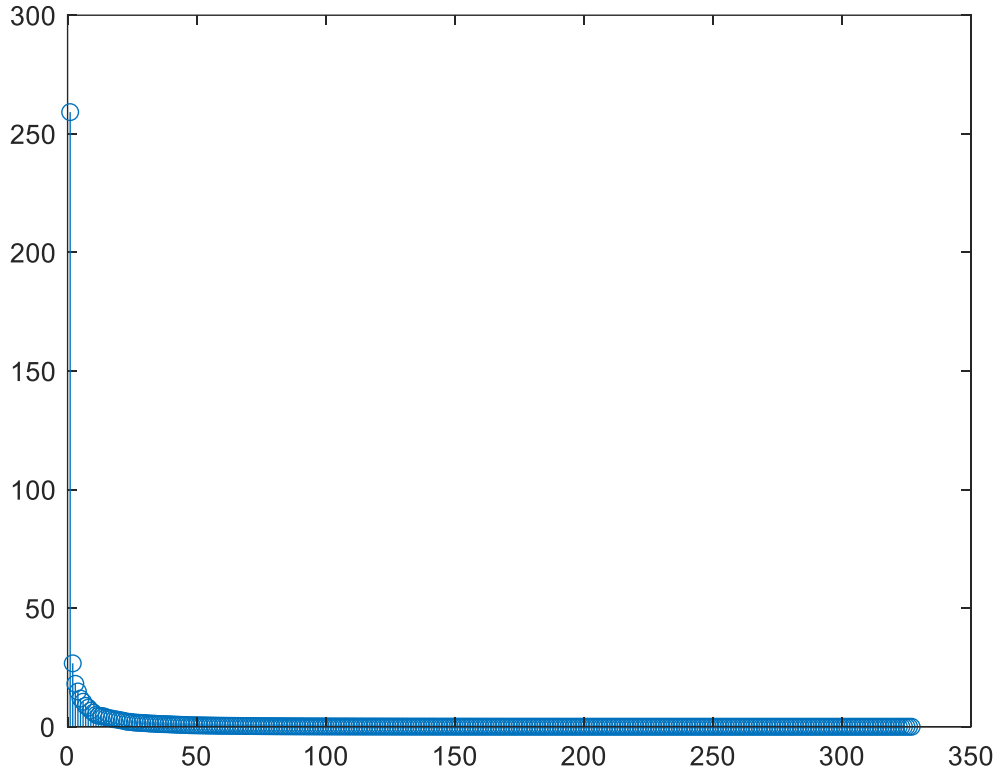
```
>> img = imread('Rubber_Ducky.jpg');
gray = rgb2gray(img);
[U,S,V] = svd(im2double(gray));
r=10;
approx_img = U(:,1:r)*S(1:r,1:r)*V(:,1:r).';
imwrite(approx_img,sprintf('Rubber_Ducky_%d.png', r),'png');
```

			
$r = 10$	$r = 20$	$r = 50$	$r = 100$

Here we see that truncated SVD can be used for data compression. If a gray scale image $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ can be approximated as a matrix of rank r, then the data storage size will be reduced from mn to $mr + r + nr = (m + n + 1)r$.

How to choose an appropriate value of *r*?
Can we get some ideas from the singular values?

```
>> s=diag(S);
>> stem(s);
```



However, SVD may not be an effective compression scheme if the image data are not highly correlated.

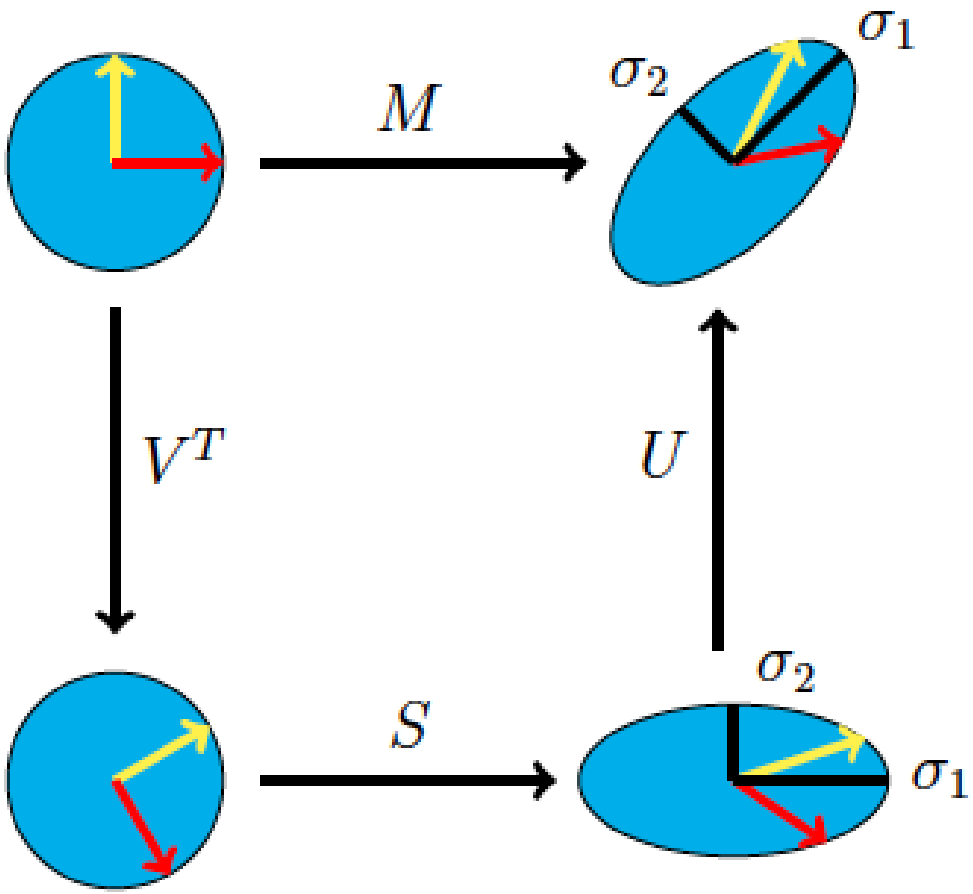


Illustration of SVD as rotation and scaling

Truncated SVD can also be applied for noise reduction.

Example 3

When multiple receivers are used to collect a correlated signal in the presence of noise, SVD can be used for denoising.

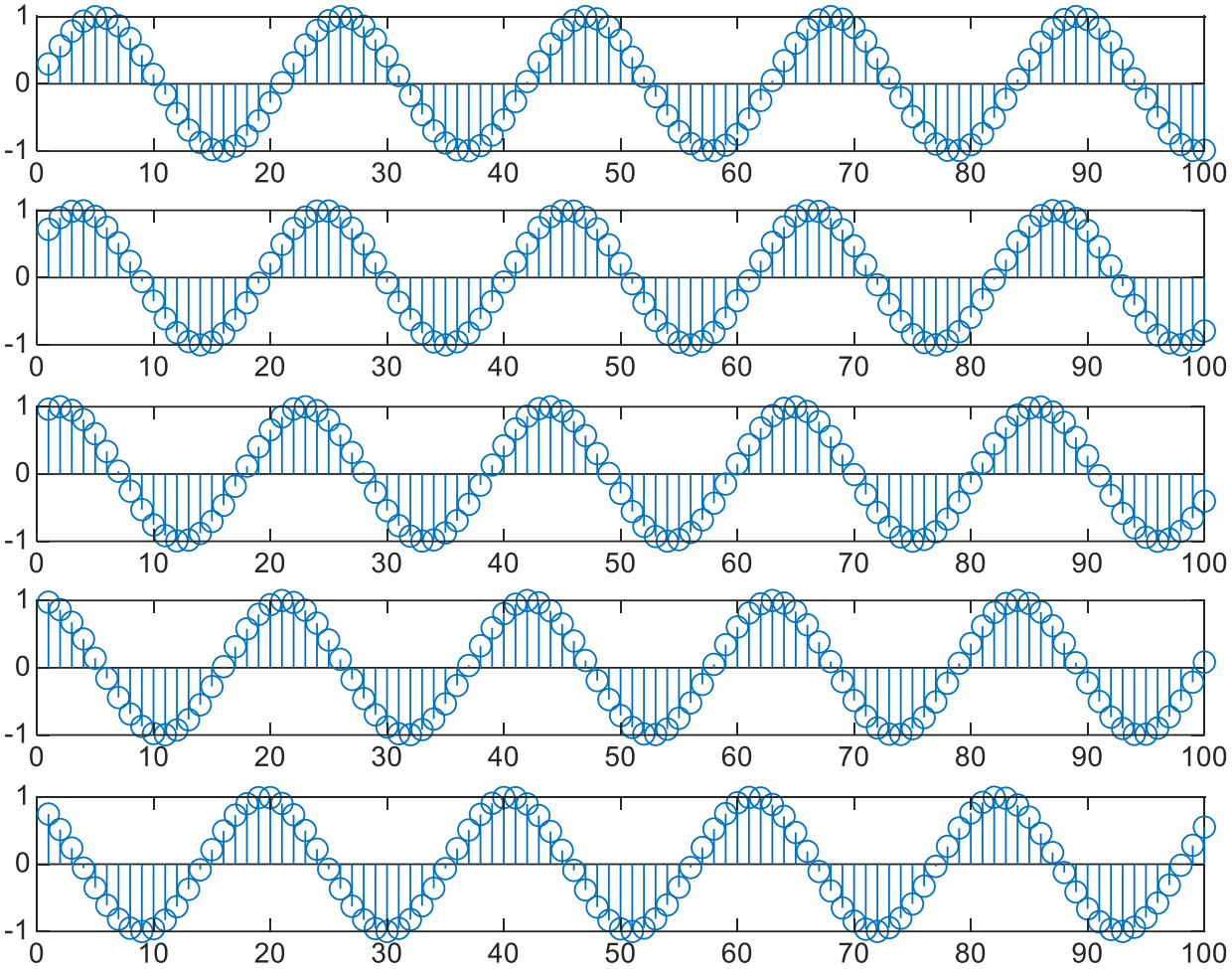
Consider a sinusoid $\cos(\omega t)$ is received by 5 sensors and each sensor obtains 100 samples. As the sensor locations are different, each may receive:

$$x_i(t) = A_i \cos(\omega t + \phi_i) + n_i(t),\ i = 1, \cdots, 5,\ t = 1, \cdots, 100$$

As a sinusoid can be represented as a linear combination of 2 sinusoids of same frequency, e.g., $x_3(t) = \alpha_1 x_1(t) + \alpha_2 x_2(t)$ and $2\cos(\omega)\cos(\omega(t-1)) = \cos(\omega t) + \cos(\omega(t-2))$, the matrix $\boldsymbol{X} \in \mathbb{R}^{5\times 100}$ constructed from sensor output as rows is of rank

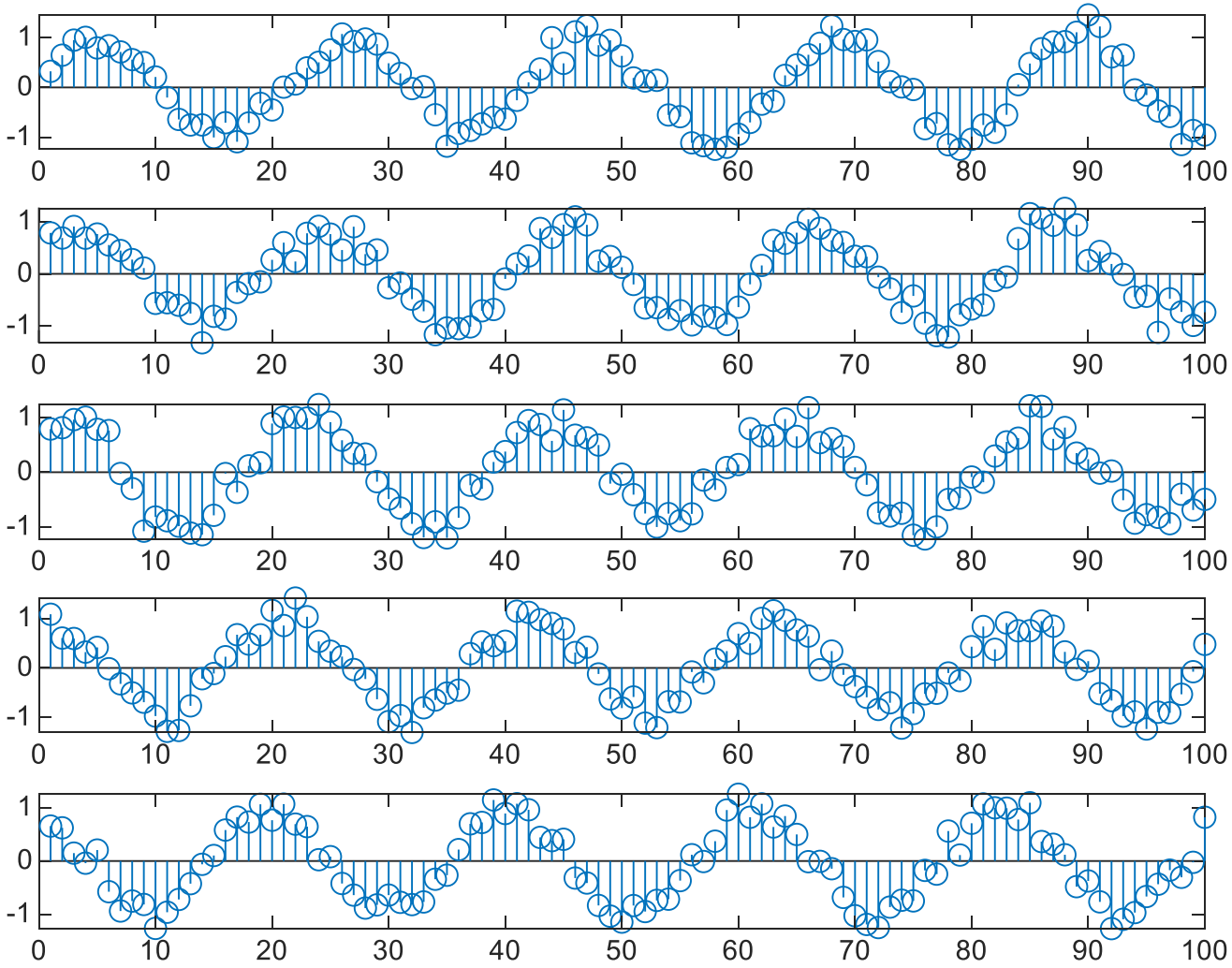
2 in the absence of measurement noise.

```
>> N=100;
w=0.3;
x1=sin(w.*(1:N));
subplot(5,1,1)
stem(x1)
x2=sin(w.*(1:N)+0.5);
subplot(5,1,2)
stem(x2)
x3=sin(w.*(1:N)+1);
subplot(5,1,3)
stem(x3)
x4=sin(w.*(1:N)+1.5);
subplot(5,1,4)
stem(x4)
x5=sin(w.*(1:N)+2);
subplot(5,1,5)
stem(x5)
>> X=[x1; x2; x3; x4; x5;];
>> rank(X)
ans = 2
```

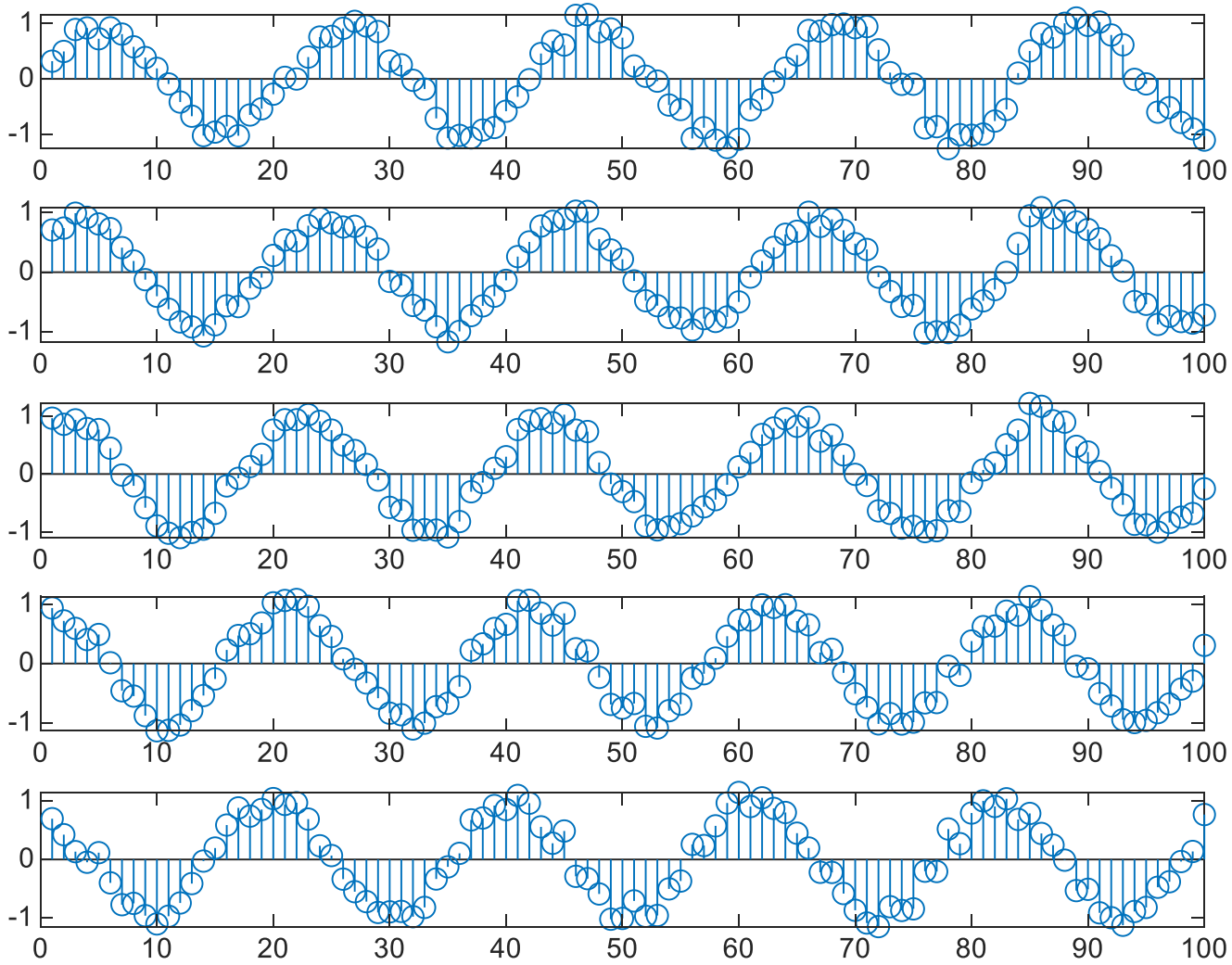


Hence a truncated SVD of $\boldsymbol{X}$ with $r = 2$ can represent the data without approximation.

In practical scenarios, noise $n_i(t)$ is present.



A noise-reduced version is: $[\boldsymbol{u}_1, \boldsymbol{u}_2]\mathrm{diag}\{\sigma_1, \sigma_2\}[\boldsymbol{v}_1, \boldsymbol{v}_2]^T$



Apart from direct observation, a standard performance measure is to use empirical mean square error (MSE):

$$\text{MSE} = \frac{\sum_{n=1}^{N} (x_n - \hat{x}_n)^2}{N}$$

where x_n is the true value and $\hat{x}_n$ is the estimated value of x_n. That is, MSE is a measure of average squared error.

Consider the noisy raw data as the estimates, e.g.,

```
x1=sin(w.*(1:N))+0.2*randn(1,N);%noise standard deviation 0.2
```

which means that noise is zero-mean Gaussian distributed with power $\mathbb{E}\{n_i^2(t)\} = 0.2^2 = 0.04$, we can compute the MSE:

$$\text{MSE} = \frac{\sum_{i=1}^{5} \sum_{t=1}^{100} (x_i(t) - \hat{x}_i(t))^2}{500} = 0.0403$$

which aligns with the noise power value.

We repeat the MSE computation using the denoised data and obtain:

$$\text{MSE} = 0.0149$$

From the MSE, the denoising performance is clearly demonstrated and we know how much noise is reduced in terms of MSE.

Because of the random noise, slightly different numerical results will be obtained in each simulation run.

Note also that we cannot obtain a zero MSE by denoising because the truncated SVD components

$$[\boldsymbol{u}_1, \boldsymbol{u}_2]\text{diag}\{\sigma_1, \sigma_2\}[\boldsymbol{v}_1, \boldsymbol{v}_2]^T$$

also contain noise.

Can you suggest how to get a smaller MSE?

This application may be referred to as subspace estimation. Here, $\boldsymbol{u}_1, \boldsymbol{u}_2, \sigma_1, \sigma_2, \boldsymbol{v}_1, \boldsymbol{v}_2$ correspond to the signal subspace while $\boldsymbol{u}_3, \boldsymbol{u}_4, \boldsymbol{u}_5, \sigma_3, \sigma_4, \sigma_5, \boldsymbol{v}_3, \boldsymbol{v}_4, \boldsymbol{v}_5$ correspond to the noise subspace.

Hence the signal subspace can also be obtained from the truncated SVD with an appropriate value of r:

$$\boldsymbol{X}_s = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$$

Note also that truncated SVD is the best low-rank matrix approximation in the least squares (LS) sense:

$$\boldsymbol{X}_s = \arg\min_{\tilde{\boldsymbol{X}}} \left\| \boldsymbol{X} - \tilde{\boldsymbol{X}} \right\|_F^2, \quad \text{s.t.} \quad \text{rank}(\tilde{\boldsymbol{X}}) = r$$

Principal Component Analysis (PCA)

PCA is similar to truncated SVD but it is based on the eigenvalue analysis of the covariance of the observed data.

Example 4

Suppose there are 4 observed data vectors. We group them as a data matrix as:

$$\boldsymbol{X} = \begin{bmatrix} 2 & 2 & 1 & 2 \\ 1 & 3 & 0 & 3 \\ 0 & 1 & 3 & 1 \\ 3 & 2 & 3 & 0 \\ 1 & 3 & 1 & 3 \\ 1 & 0 & 1 & 2 \end{bmatrix}$$

Compute the covariance matrix $\boldsymbol{C}$ for $\boldsymbol{X}$.

To compute the covariance matrix, we need to make the mean of each row 0. Note that each row can be considered as the same type of information or feature. The mean vector μ is computed as:

```
>> mean(X,2)
     1.7500
     1.7500
     1.2500
     2.0000
     2.0000
      1.0000

>> bX=X-mean(X,2)
     0.2500    0.2500   -0.7500    0.2500
    -0.7500    1.2500   -1.7500    1.2500
    -1.2500   -0.2500    1.7500   -0.2500
     1.0000         0    1.0000   -2.0000
    -1.0000    1.0000   -1.0000    1.0000
          0   -1.0000         0    1.0000
```

Let the data matrix with zero-mean rows be $\overline{X}$. The covariance matrix C is computed as:

$$\frac{1}{4}\overline{X} \cdot \overline{X}^T$$

```
>> C=bX*bX.'/4
    0.1875    0.4375   -0.4375   -0.2500    0.2500         0
    0.4375    1.6875   -0.6875   -1.2500    1.2500         0
   -0.4375   -0.6875    1.1875    0.2500   -0.2500         0
   -0.2500   -1.2500    0.2500    1.5000   -1.0000   -0.5000
    0.2500    1.2500   -0.2500   -1.0000    1.0000         0
         0         0         0   -0.5000         0    0.5000
```

Each diagonal element of C is the variance within a measurement type.

Off-diagonal elements are covariances between all pairs of different types. If covariance is nonzero, there is correlation or redundancy while the two types are uncorrelated for zero covariance.

In general, suppose for each measurement vector, there are m types and we collect a total of n measurements. A data matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ is formed and then its zero-mean row version $\overline{\boldsymbol{X}} \in \mathbb{R}^{m \times n}$ is computed. Then $\boldsymbol{C} \in \mathbb{R}^{m \times m}$ is:

$$\boldsymbol{C} = \frac{1}{n} \overline{\boldsymbol{X}} \cdot \overline{\boldsymbol{X}}^T \tag{3}$$

Since $\boldsymbol{C}$ is square and symmetric such that $\boldsymbol{C} = \boldsymbol{C}^T$, its eigenvalue decomposition (EVD) is:

$$\boldsymbol{C} = \boldsymbol{E}\boldsymbol{D}\boldsymbol{E}^{-1} = \boldsymbol{E}\boldsymbol{D}\boldsymbol{E}^T = \sum_{i=1}^{m} \lambda_i \boldsymbol{e}_i \boldsymbol{e}_i^T \tag{4}$$

where $\boldsymbol{E} \in \mathbb{R}^{m \times m}$ is orthonormal and its column vectors are eigenvectors, while $\boldsymbol{D} = \text{diag}\{\lambda_1, \cdots, \lambda_m\} \in \mathbb{R}^{m \times m}$ with λ_i being the eigenvalue for the ith column.

Suppose $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$, the principal components refer to the first r column vectors of $\boldsymbol{E}$, and the associated $\lambda_1, \cdots, \lambda_r$ are variances or powers of the principal components.

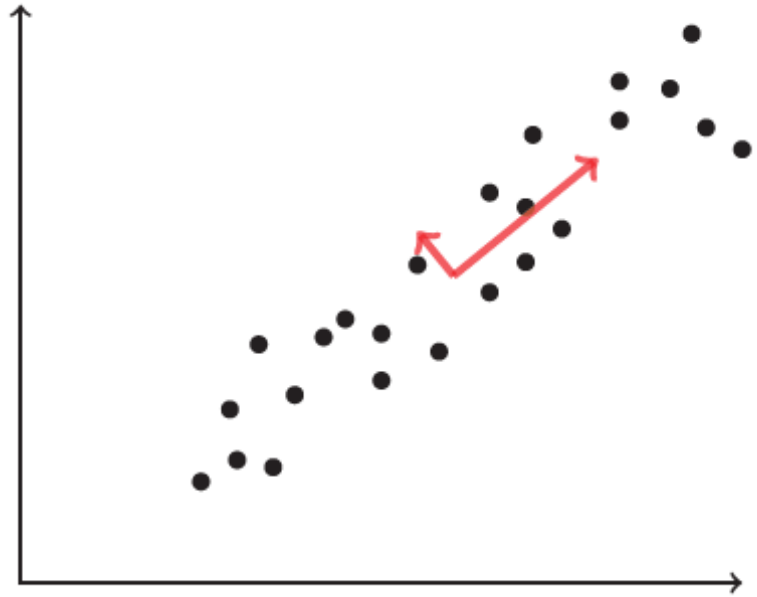
For simplicity but without loss of generality, we may ignore the scalar of $1/n$ in (3) and assume $\boldsymbol{X}$ has zero mean along all the rows such that $\boldsymbol{X} = \overline{\boldsymbol{X}}$. Using (1) and the orthonormality of $\boldsymbol{V}$, it is easy to show that $\boldsymbol{E}$ can be computed from SVD of $\boldsymbol{X}$:

$$\boldsymbol{X}\boldsymbol{X}^T = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T \left(\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T\right)^T = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^T = \boldsymbol{U}\boldsymbol{S}^2\boldsymbol{U}^T \tag{5}$$

That is, $\boldsymbol{E} = \boldsymbol{U}$ and $\boldsymbol{D} = \boldsymbol{S}^2$ or $\lambda_i = \sigma_i^2$ (up to a scalar).

This indicates that SVD operating on the data matrix is same as EVD operating on the covariance matrix.

Using a 2D geometric viewpoint, PCA means to find the best orthogonal basis:



It is clear that the standard basis of $[1, 0]$ and $[0, 1]$ (x- and y-axis) is not good to represent the data points.

Note that a basis is a minimal spanning set to represent all data in that space. For 2D space, $[1, 0]$ and $[0, 1]$ form a basis because any point $[x, y]$ can be written as $x[1, 0] + y[0, 1]$.

The basis in red lines is better in the sense that the direction (cf. eigenvector) is related to the data structure and length (cf. eigenvalue) is related to the importance.

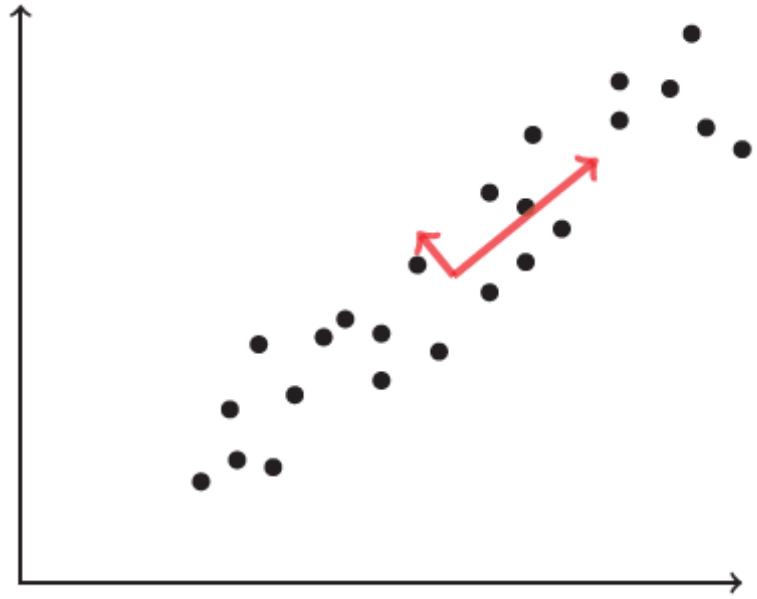
There are three assumptions in PCA:

1. Linearity
   - Change of basis is linear operation.
   - But, some processes are inherently nonlinear.

2. Large variances are most "interesting"
   - Large variance is "signal", small is "noise".
   - May not be valid for some applications.

3. Principal components are orthogonal
   - Makes problem efficiently solvable.
   - But non-orthogonal may be better in some cases.

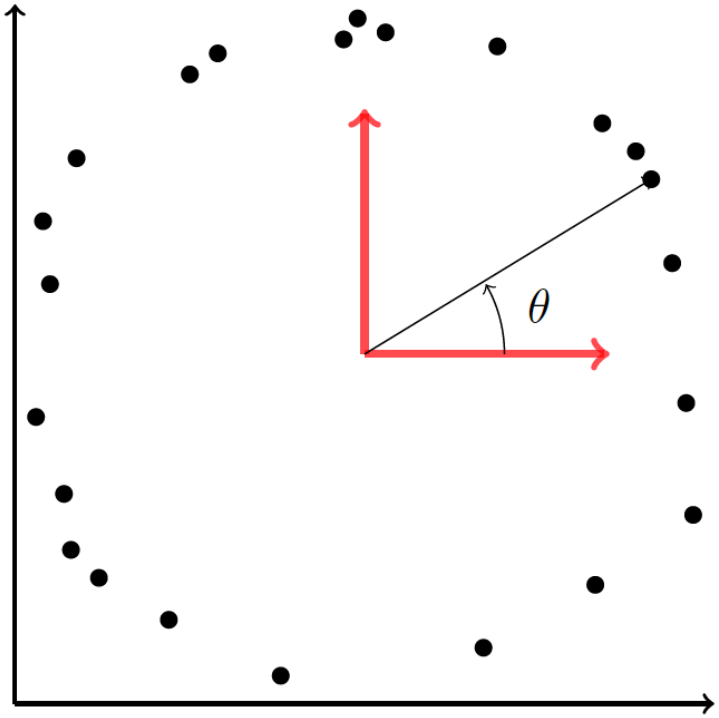
Applying PCA will be perfect when:



The data can be better represented by another orthogonal basis.

Longer red vector is our signal of interest while the shorter vector represents the noise component. Hence we can ignore the short vector and the data can be approximated in 1D, i.e., the first eigenvector is the principal component.

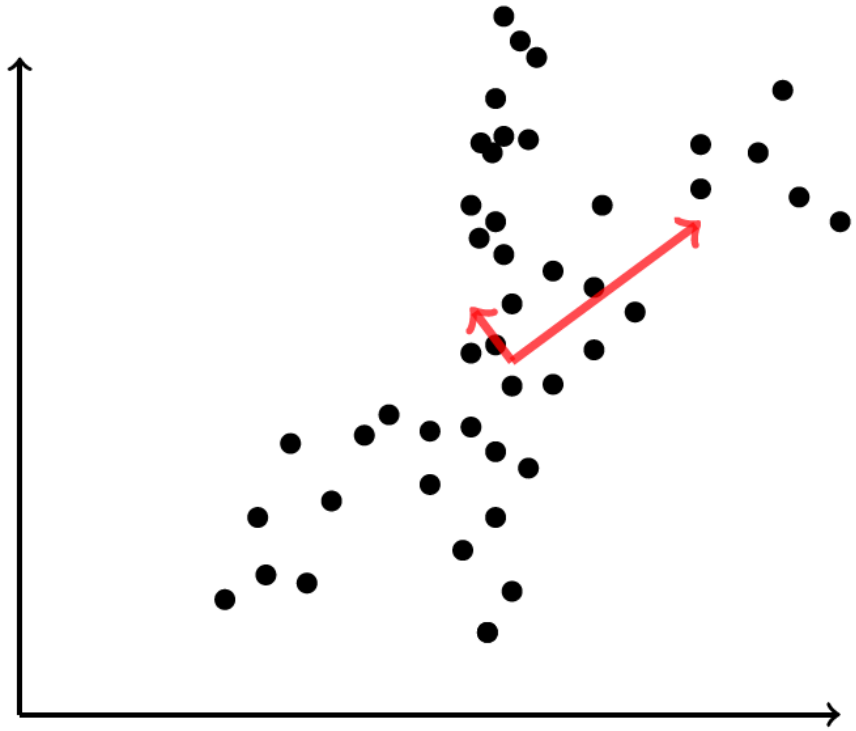
PCA will fail when the data cannot be represented better via another linear transform:



Here, the information is contained in the angle θ.

But θ is nonlinear w.r.t. to (x, y) basis.

PCA may not be suitable tool when the important information is nonorthogonal:



Here, the second dominant vector, which is orthogonal to the first one, does not match the data well.

PCA will fail when we are interested in the non-principal components.

The principal components may correspond to the chessboard.

PCA may not be useful if we are interested in the pieces.

Also, if the data do not have dominant principal components, e.g., random noise, we may not be able to extract useful information using PCA (or other dimensionality reduction schemes):

```
>> svd(randn(1000,10))
   34.2535
   33.5750
   32.6017
   32.1161
   31.8181
   30.9503
   30.2355
   28.9969
   28.5793
    28.4865
```

The singular values are comparable and thus all components are important or interesting.

Training and Scoring with PCA

PCA can be used as classification and two phases are involved: training and scoring.

We assume that there is a set of training data and we perform the evaluation using test data

In training phase, we need to form a scoring matrix from training data. Suppose we have n training vectors $\boldsymbol{x}_1, \cdots \boldsymbol{x}_n$ and each has a length of m.

The steps are:

1. Form $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ and then compute $\overline{\boldsymbol{X}} = [\overline{\boldsymbol{x}}_1, \cdots \overline{\boldsymbol{x}}_n]$, $\overline{\boldsymbol{x}}_i = \boldsymbol{x}_i - \boldsymbol{\mu}$
2. Compute $\boldsymbol{U}$ via EVD of $\boldsymbol{C}$ or SVD of $\overline{\boldsymbol{X}}$
3. Extract r dominant vectors to form $\boldsymbol{U}_r = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_r] \in \mathbb{R}^{m \times r}$
4. Compute the scoring matrix $\Delta = \boldsymbol{U}_r^T \overline{\boldsymbol{X}} \in \mathbb{R}^{r \times n}$

Note that the columns of Δ is formed by projecting the training vectors $\overline{\boldsymbol{x}}_1, \cdots \overline{\boldsymbol{x}}_n$ onto the eigenspace $\boldsymbol{U}_r^T$, e.g., the ith column of Δ is:

$$\Delta_i = \begin{bmatrix} \boldsymbol{u}_1^T \overline{\boldsymbol{x}}_i \\ \boldsymbol{u}_2^T \overline{\boldsymbol{x}}_i \\ \vdots \\ \boldsymbol{u}_r^T \overline{\boldsymbol{x}}_i \end{bmatrix} \tag{6}$$

Hence Δ can be referred to as trained model.

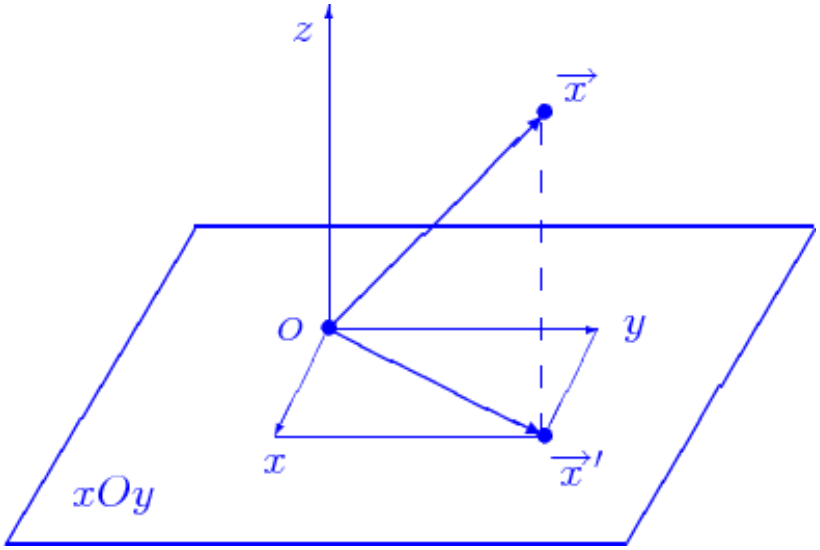


Illustration of projecting a vector on a 2D space

Source: https://www.quora.com/What-is-meant-by-the-projection-of-a-vector

In the scoring phase, the task is to find the best match of a vector $\boldsymbol{y}$ with $\overline{\boldsymbol{x}}_1, \cdots \overline{\boldsymbol{x}}_n$. The main idea is to project the test vector onto $\boldsymbol{U}_r$ and then find the closest Δ_i. That is, we perform the classification in the projected space.

The steps are:

1. Compute $\overline{\boldsymbol{y}} = \boldsymbol{y} - \boldsymbol{\mu}$
2. Compute $\boldsymbol{w} = \boldsymbol{U}_r^T \overline{\boldsymbol{y}}$, i.e., the projection of $\overline{y}$ onto $\boldsymbol{U}_r$
3. Compute the distance between $\boldsymbol{w}$ and each of the Δ_i. The distance is usually represented as the Euclidean distance:

$$\epsilon_i = \sqrt{(w_1 - \Delta_{i,1})^2 + (w_2 - \Delta_{i,2})^2 + \cdots + (w_r - \Delta_{i,r})^2}, \quad i = 1, \cdots, n$$

4. The score is given by:

$$\text{score}(\boldsymbol{y}) = \min_i \epsilon_i$$

The best match of $\boldsymbol{y}$ is $\boldsymbol{x}_{i^\star}$ where $i^\star = \arg\min_i \epsilon_i$

It is seen that training is somewhat involved particularly for large m and/or n, while scoring is simple and fast.

Example 5

We continue with Example 4. We consider the 4 vectors as training data:

$$\boldsymbol{X} = \begin{bmatrix} 2 & 2 & 1 & 2 \\ 1 & 3 & 0 & 3 \\ 0 & 1 & 3 & 1 \\ 3 & 2 & 3 & 0 \\ 1 & 3 & 1 & 3 \\ 1 & 0 & 1 & 2 \end{bmatrix}$$

```
>> [U,S,V]=svd(bX)
```

U

```
    0.1641    0.2443   -0.0710    0.6482    0.6137   -0.3341
    0.6278    0.1070    0.2934    0.4387   -0.4970    0.2624
   -0.2604   -0.8017    0.3952    0.3623    0.0389   -0.0239
   -0.5389    0.4277    0.3439    0.2109    0.1133    0.5925
    0.4637   -0.1373    0.3644   -0.4089    0.5909    0.3420
    0.0752   -0.2904   -0.7083    0.2109    0.1133    0.5925
```

S

```
    4.0414         0         0         0
         0    2.2239         0         0
         0         0    1.7237         0
         0         0         0    0.0000
         0         0         0         0
         0         0         0         0
```

V

```
   -0.2739    0.6961   -0.4364    0.5000
    0.3166    0.2466    0.7674    0.5000
   -0.6631   -0.5434    0.1224    0.5000
    0.6205   -0.3993   -0.4534    0.5000
```

As the number of nonzero singular values is at most $\min(m, n)$, "economic" mode is preferred to save complexity:

```
>> [U,S,V]=svd(bX,'econ')

U =
    0.1641    0.2443   -0.0710    0.6482
    0.6278    0.1070    0.2934    0.4387
   -0.2604   -0.8017    0.3952    0.3623
   -0.5389    0.4277    0.3439    0.2109
    0.4637   -0.1373    0.3644   -0.4089
    0.0752   -0.2904   -0.7083    0.2109
S =
    4.0414         0         0         0
         0    2.2239         0         0
         0         0    1.7237         0
         0         0         0    0.0000
V =
   -0.2739    0.6961   -0.4364    0.5000
    0.3166    0.2466    0.7674    0.5000
   -0.6631   -0.5434    0.1224    0.5000
    0.6205   -0.3993   -0.4534    0.5000
```

Note that $\mathrm{rank}(\overline{\boldsymbol{X}}) = 3$, it can be exactly represented as truncated SVD with components:

```
>> U(:,1:3)
    0.1641    0.2443   -0.0710
    0.6278    0.1070    0.2934
   -0.2604   -0.8017    0.3952
   -0.5389    0.4277    0.3439
    0.4637   -0.1373    0.3644
    0.0752   -0.2904   -0.7083

>> S(1:3,1:3)
    4.0414         0         0
         0    2.2239         0
         0         0    1.7237

>> V(:,1:3)
   -0.2739    0.6961   -0.4364
    0.3166    0.2466    0.7674
   -0.6631   -0.5434    0.1224
    0.6205   -0.3993   -0.4534
```

Note that the singular values are $\sigma_1 = 4.0414$, $\sigma_2 = 2.2239$ and $\sigma_3 = 1.7237$. Then the eigenvalues are $\lambda_1 = 16.3333$, $\lambda_2 = 4.9456$ and $\lambda_3 = 2.9711$ up to a scalar. The exact values should be $\lambda_1 = 4.0833$, $\lambda_2 = 1.2364$ and $\lambda_3 = 0.7428$ because it can be verified that $\lambda_i = \sigma_i^2/n$.

If we use all eigenvectors, that is, we choose $r = 3$ and the scoring matrix $\boldsymbol{\Delta} = \boldsymbol{U}_r^T \overline{\boldsymbol{X}} \in \mathbb{R}^{3\times 4}$ is computed as

```
>> U(:,1:3).'*bX
  -1.1070    1.2794   -2.6801    2.5076
   1.5480    0.5484   -1.2084   -0.8879
  -0.7523    1.3228    0.2110   -0.7815
```

For $r = 2$:

```
>> U(:,1:2).'*bX
   -1.1070    1.2794   -2.6801    2.5076
    1.5480    0.5484   -1.2084   -0.8879
```

For $r = 1$:

```
>> U(:,1:1).'*bX
   -1.1070    1.2794   -2.6801    2.5076
```

Consider matching a vector $\boldsymbol{y} = [2\ 1\ 0\ 3\ 1\ 1]^T$ with the training data set $\boldsymbol{X}$ and using $r = 3$

```
>> y = [2, 1, 0, 3, 1, 1].';
  by=y-mean(X,2);
  U(:,1:3).'*by

    -1.1070
     1.5480
    -0.7523
```

What is the score?
What is the best match of *y*? Why?

Consider another $\boldsymbol{y} = [2\ 3\ 4\ 4\ -3\ 2]^T$ and $r = 3$:

```
>> y = [2, 3, 4, 4, -3, -2].';
by=y-mean(X,2);
w=U(:,1:3).'*by

   -3.5124
    0.4036
    2.4265
```

The distance between $\boldsymbol{w}$ and $\boldsymbol{x}_1$ is then:

$$\begin{aligned}\epsilon_1 &= \sqrt{(-3.5124 + 1.1070)^2 + (0.4036 - 1.5480)^2 + (2.4265 + 0.7523)^2} \\ &= 4.1413\end{aligned}$$

Other distances are:

```
>> D=U(:,1:3).'*bX;
[norm(w-D(:,1)),norm(w-D(:,2)),norm(w-D(:,3)),norm(w-D(:,4))]

    4.1473    4.9193    2.8636    6.9426
```

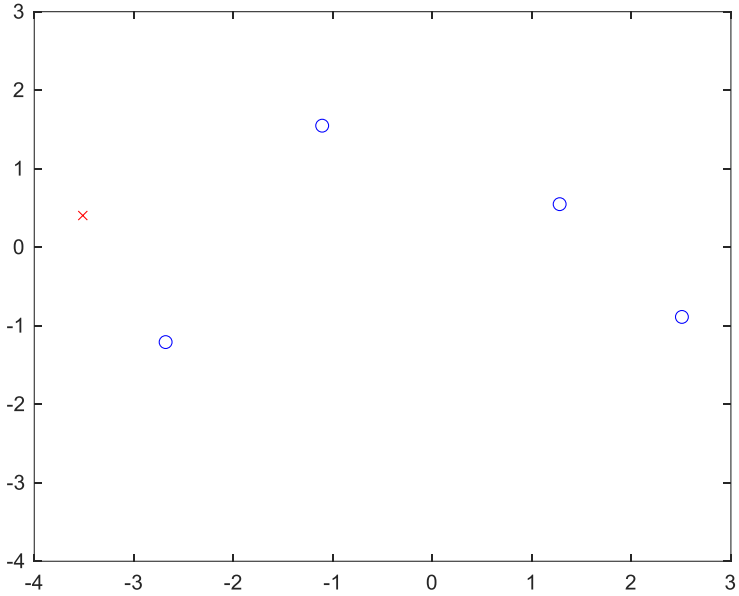
Hence we see that the best match is $\boldsymbol{x}_3$.

Consider another $\boldsymbol{y} = [2\ 3\ 4\ 4\ -3\ 2]^T$ and $r = 2$:

```
>> w=U(:,1:2).'*by;
    -3.5124
     0.4036

>> D=U(:,1:2).'*bX;
  -1.1070    1.2794   -2.6801    2.5076
   1.5480    0.5484   -1.2084   -0.8879

[norm(w-D(:,1)),norm(w-D(:,2)),norm(w-D(:,3)),norm(w-D(:,4))]

    2.6637    4.7939    1.8142    6.1570
```

The best match is still $\boldsymbol{x}_3$.

We will see that for this example, even one principal component can give the consistent classification result.



Consider another $\boldsymbol{y} = [2\ 3\ 4\ 4\ -3\ 2]^T$ and $r = 1$:

```
>> w=U(:,1).'*by
    -3.5124

>> D=U(:,1).'*bX
    -1.1070    1.2794   -2.6801    2.5076

[norm(w-D(:,1)),norm(w-D(:,2)),norm(w-D(:,3)),norm(w-D(:,4))]
    2.4054    4.7918    0.8323    6.0200
```

Again, the best match is still $\boldsymbol{x}_3$.

PCA for Face Recognition

Face recognition is one of the standard applications of PCA. Consider using the ORL face database:

- Composed of 400 images with dimensions 112 x 92.
- There are 40 persons, 10 images per each person.
- The images were taken at different times, lighting and facial expressions.
- The faces are in an upright position in frontal view, with a slight left-right rotation.

The face recognition system can be set up as follows. The 400 images are divided into non-overlapped training data and testing data.

In the training phase, we select 1 to 9 images for each person to create the scoring matrix.

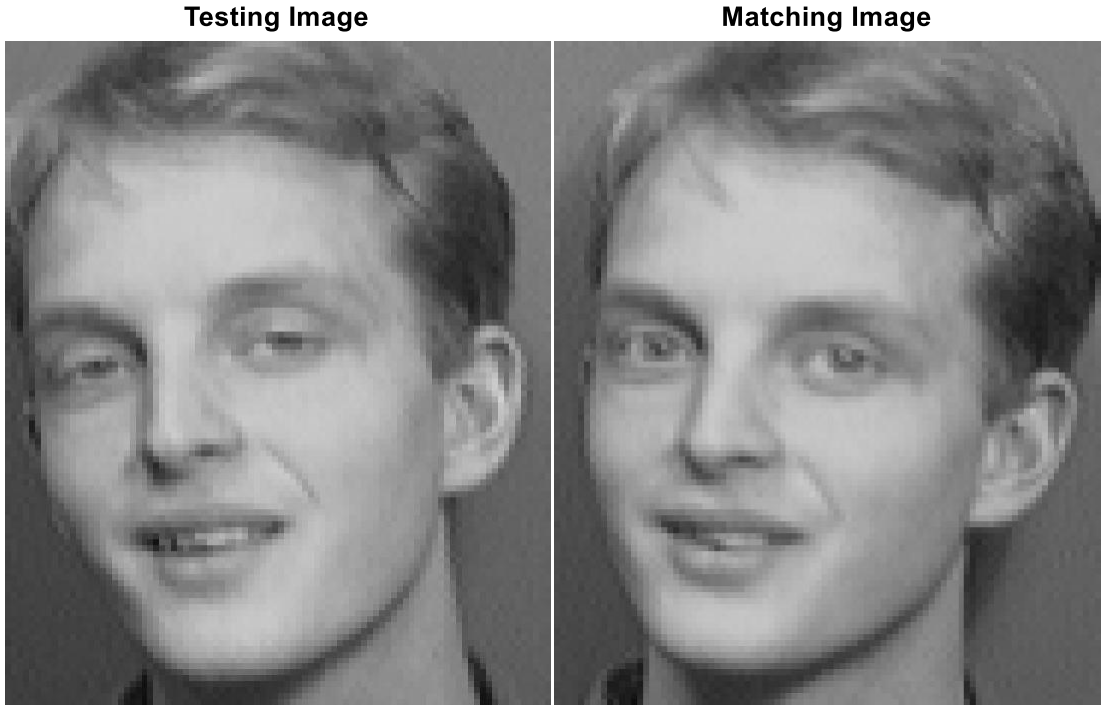
For each image, we convert the matrix to a vector of length 10304.

For example, if 5 images of each person are used, then we have $\boldsymbol{X} \in \mathbb{R}^{10304 \times 200}$. Note that there will be 200 images for testing.

By determining r , we can use the dominant vectors to form $\boldsymbol{U}_r = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_r] \in \mathbb{R}^{m \times r}$, and then obtain the scoring matrix $\boldsymbol{\Delta} = \boldsymbol{U}_r^T \overline{\boldsymbol{X}} \in \mathbb{R}^{r \times 200}$.

In the scoring phase, we pick one vector from the testing database and then project its mean-subtracted version onto U_r and then find the nearest column vector in Δ as the score.

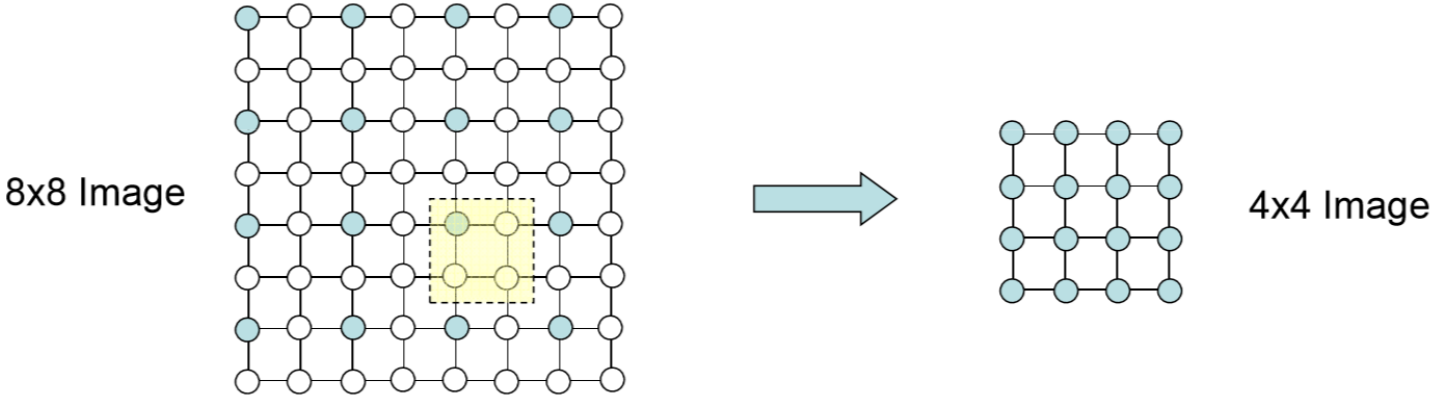
A successful case is shown:



According to the provided source code using the ORL database, the recognition accuracy can be over 90%.

Preprocessing considerations:

- Cropping from the original images is needed
- If the image is large, size reduction is needed, e.g., an image of dimensions $2m \times 2n$ can be reduced to $m \times n$ with a downsampling factor of 2



Source: http://eeweb.poly.edu/~yao/EL5123/lecture8_sampling.pdf

- If all images are not of equal sizes, we may pad zero to make all column vectors having the same length of m

System design and evaluation considerations:

- How many data are used for training? How many data are used for testing? How will this arrangement affect the computational requirement and recognition accuracy?
- How to choose an appropriate value of the number of principal components r? How will r affect the recognition accuracy?

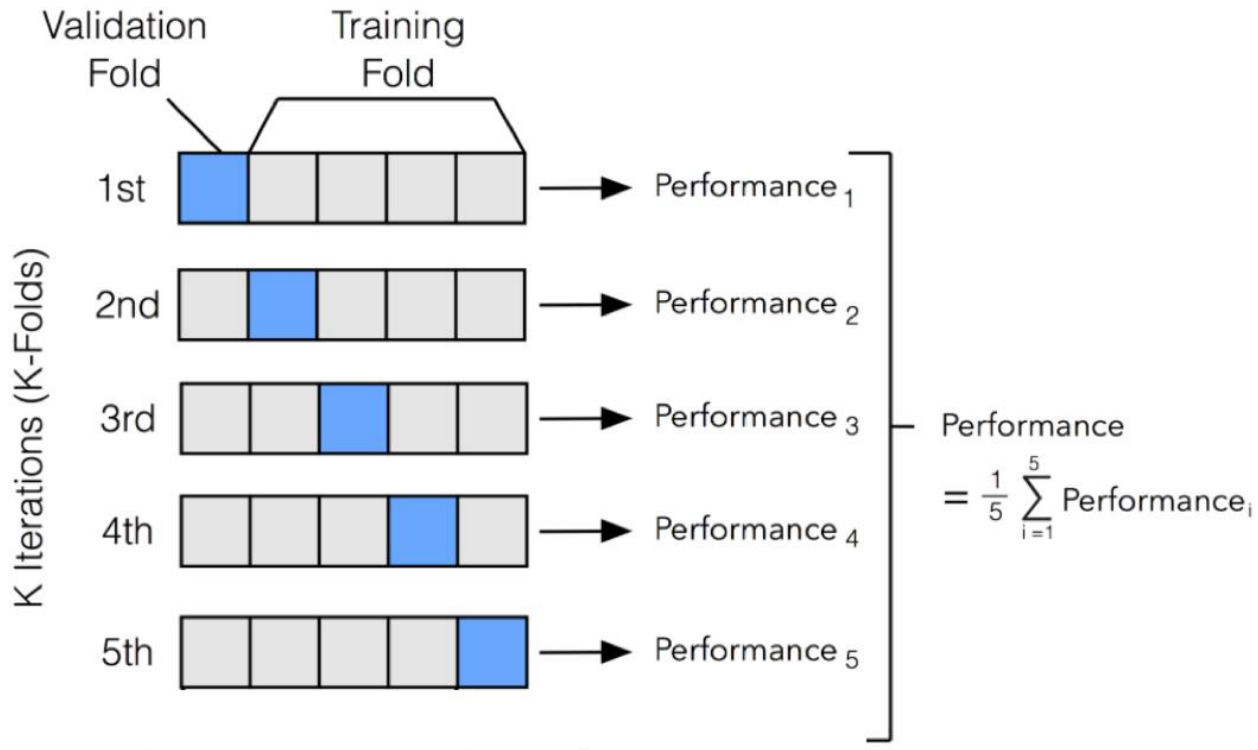
  As the eigenvalue represents the power of each principal component, one possibility is to choose r such that:

$$r = \frac{\arg\min_k \sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{m} \sigma_i^2} > \epsilon$$

  where ϵ is a threshold parameter, e.g., 90%

- Are the results reliable? How can we obtain an average performance?

  Cross validation can be applied to see if the results among different training and testing data partitions are consistent, and obtain the average performance:



Source: https://medium.com/uxai/機器學習馬拉松-034-訓練-測試集切分概念-af08ade0595a

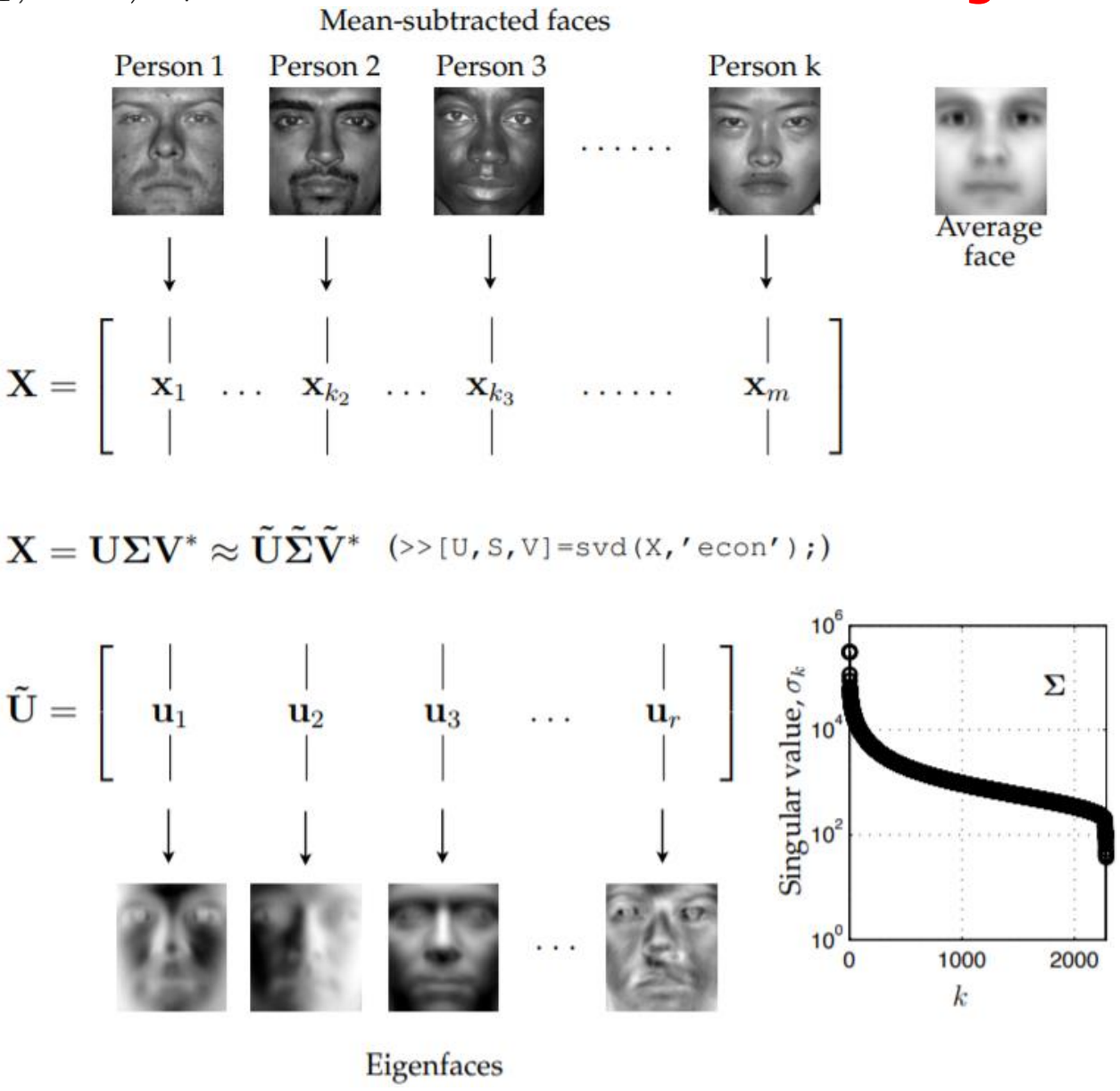
e.g., for the ORL database, we can use k-fold cross validation with $k = 10$:

We randomly divide the 400 images into 10 sets such that each contains 40 different persons.

For each iteration, 9 sets are used for training while the remaining set is reserved for testing.

There will be 10 recognition results and we can check for their consistency and compute the average recognition accuracy.

Note that $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_r$ can be constructed as r eigenfaces:



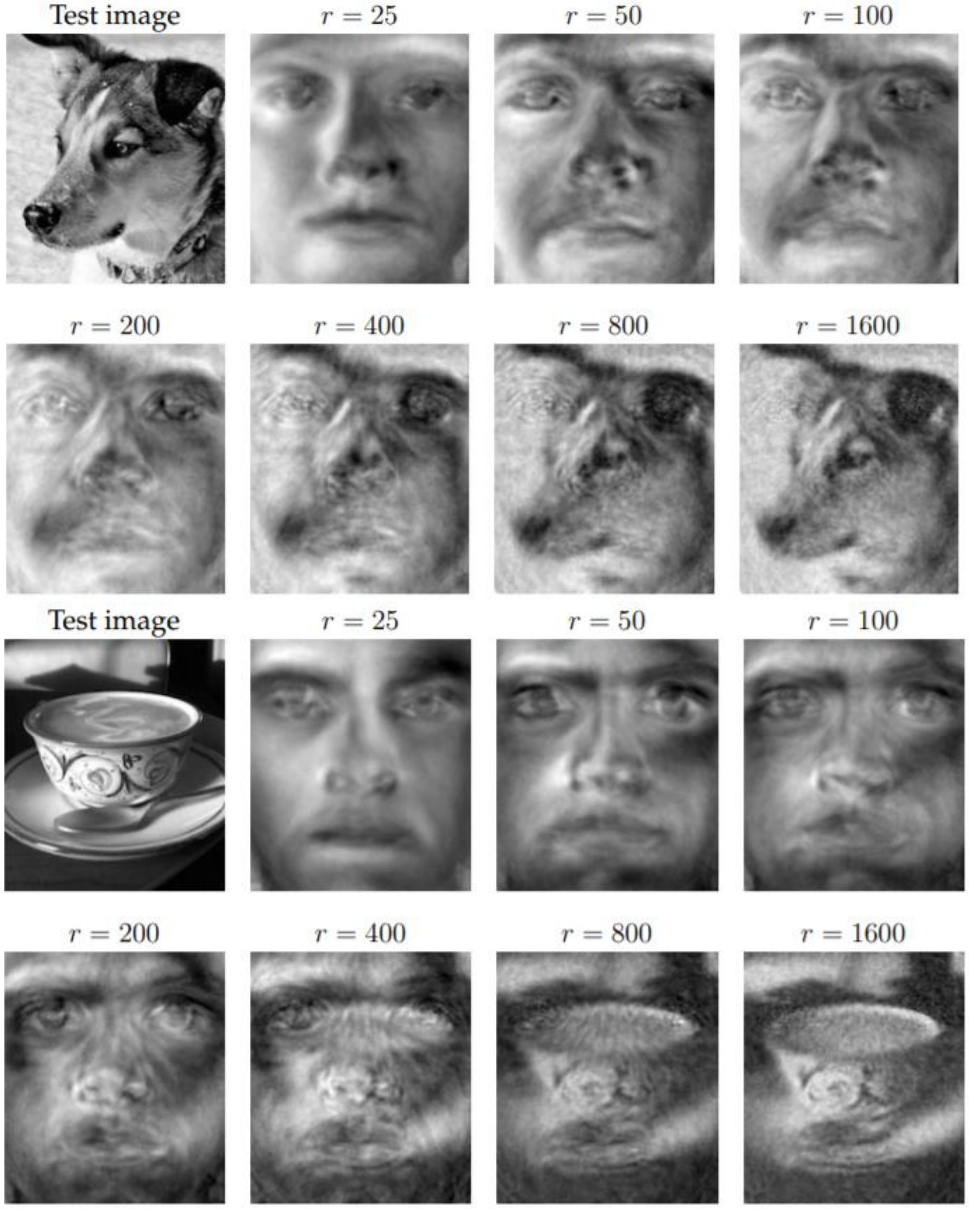
The set of the eigenfaces or U_r can be used to approximate another person not included in the training dataset via the projection:

$$\hat{\boldsymbol{x}}_{\text{new}} = \boldsymbol{U}_r \boldsymbol{U}_r^T \boldsymbol{x}_{\text{new}}$$

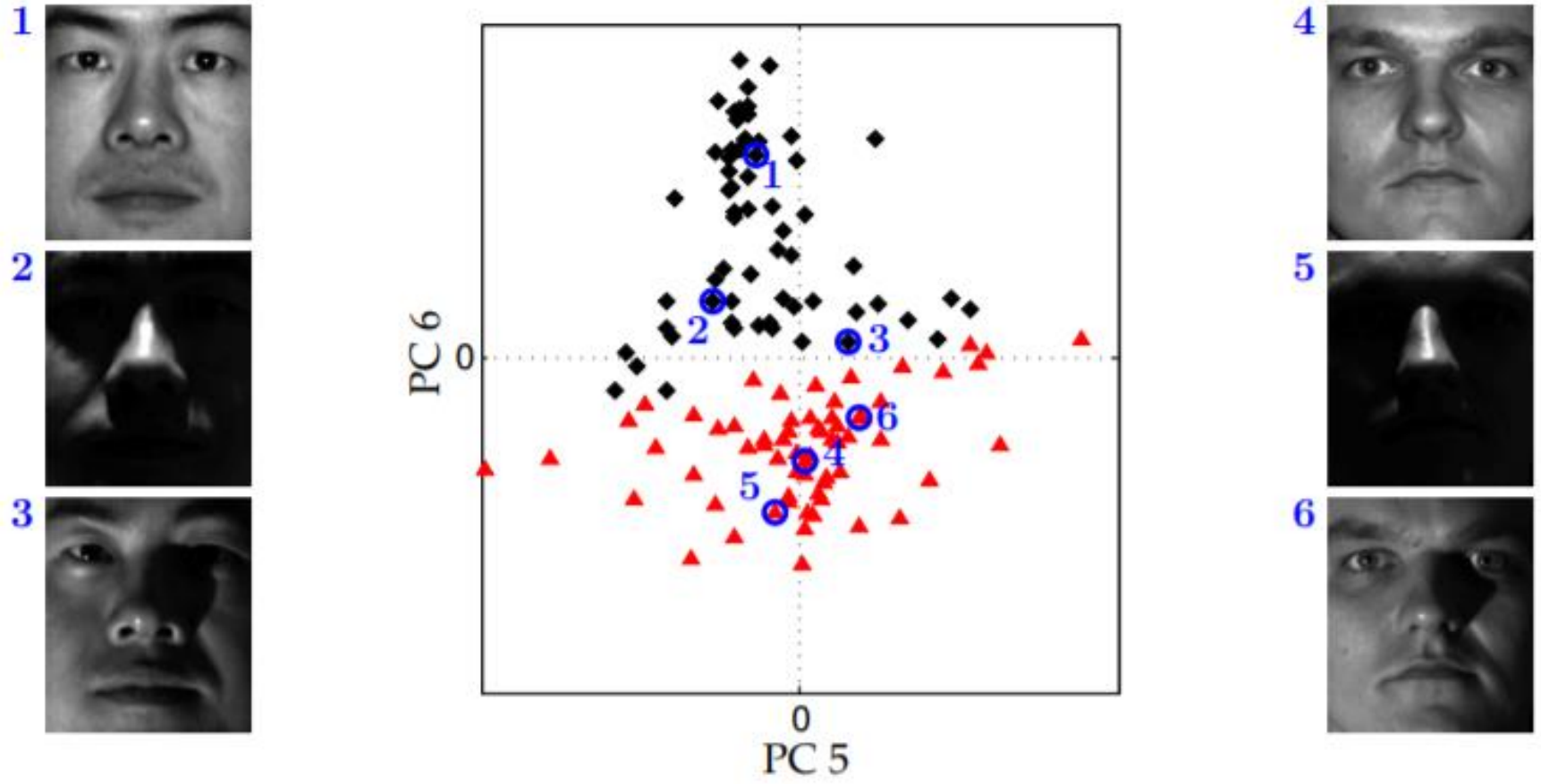
What is the advantage of this approximation?

This is possible because the eigenfaces span a large subspace of image space corresponding to different features such as cheeks, forehead and mouths.

Even dog and cup of coffee:



We can also visualize the use of eigenfaces or $\{\boldsymbol{u}_k\}$ in a 2D coordinate system where each point is $(\boldsymbol{u}_k^T\overline{\boldsymbol{x}}_i, \boldsymbol{u}_l^T\overline{\boldsymbol{x}}_i)$, e.g., projecting two individuals onto the 5th and 6th principal components, and we may still separate them:



Two-Dimensional PCA

2DPCA is a variant of PCA which is a matrix-as-matrix approach, i.e., vectorization of matrices is not needed.

The main idea is to exploit the so-called image covariance matrix.

Given K training matrix data $\boldsymbol{X}_k \in \mathbb{R}^{m\times n}$, $k = 1, 2, \cdots, K$, the image covariance matrix is computed as:

$$\boldsymbol{G} = \frac{1}{K}\sum_{k=1}^{K}\left(\boldsymbol{X}_k - \overline{\boldsymbol{X}}\right)^T\left(\boldsymbol{X}_k - \overline{\boldsymbol{X}}\right) \in \mathbb{R}^{n\times n}, \quad \overline{\boldsymbol{X}} = \frac{1}{K}\sum_{k=1}^{K}\boldsymbol{X}_k$$

The r dominant eigenvectors of $\boldsymbol{G} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T$, grouped as $\boldsymbol{U}_r = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_r] \in \mathbb{R}^{n\times r}$ are utilized.

For each image $\boldsymbol{X}_k$, we compute the projection:

$$\boldsymbol{Y}_k = \boldsymbol{X}_k \boldsymbol{U}_r \in \mathbb{R}^{m \times r}$$

As in the scoring matrix $\boldsymbol{\Delta} = \boldsymbol{U}_r^T \overline{\boldsymbol{X}} \in \mathbb{R}^{r \times n}$ in the PCA, now we have a set of scoring matrices ,$\boldsymbol{Y}_k$, $k = 1, 2, \cdots, K$.

In the scoring phase, suppose there is a testing image $\boldsymbol{X} \in \mathbb{R}^{m \times n}$, we compute the projection:

$$\boldsymbol{Y} = \boldsymbol{X} \boldsymbol{U}_r \in \mathbb{R}^{m \times r}$$

The sum of r distances between the column vectors of $\boldsymbol{Y}$ and $\boldsymbol{Y}_k$ is used for matching:

$$\epsilon_k = ||\boldsymbol{Y}(:,1) - \boldsymbol{Y}_k(:,1)||_2 + \cdots + ||\boldsymbol{Y}(:,r) - \boldsymbol{Y}_k(:,r)||_2, \quad k = 1, \cdots, K$$

As in the PCA, the score is given by:

$$\text{score}(\boldsymbol{X}) = \min_k \epsilon_k$$

The best match of $\boldsymbol{X}$ is $\boldsymbol{X}_{k^\star}$ where $k^\star = \arg\min_k \epsilon_k$

As in SVD or PCA, 2DPCA can be employed to approximate a group of images. To directly store $\boldsymbol{X}_k \in \mathbb{R}^{m\times n}$, $k = 1, 2, \cdots, K$, we need a memory size of mnK.

With the use of 2DPCA, we need to store $\boldsymbol{U}_r \in \mathbb{R}^{n\times r}$ and $\boldsymbol{Y}_k \in \mathbb{R}^{m\times r}$, $k = 1, 2, \cdots, K$, corresponding to $(mK + n)r$. The compression ratio is then:

$$\frac{mnK}{(mK + n)r}$$

Tensor Basics

Tensor is important because it can directly represent data of three or higher dimensions.

Although a tensor can be unfolded as a matrix or even a vector, it is advantageous to process it in the original domain so that the inherent structure is maintained.

A Nth-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is in fact a N-dimensional signal where its nth dimension has length I_n, $n = 1, \cdots, N$, and thus it contains $I_1 \times I_2 \times \cdots \times I_N$ elements.

Tensor generalizes matrix and vector because it corresponds to $N = 2$ and $N = 1$, respectively.

$\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is of rank 1 if it can be written as the outer product of N vectors:

$$\boldsymbol{\mathcal{X}} = \boldsymbol{a}^{(1)} \circ \boldsymbol{a}^{(2)} \circ \cdots \circ \boldsymbol{a}^{(N)}$$

That is, the element of $\boldsymbol{\mathcal{X}}$ is:

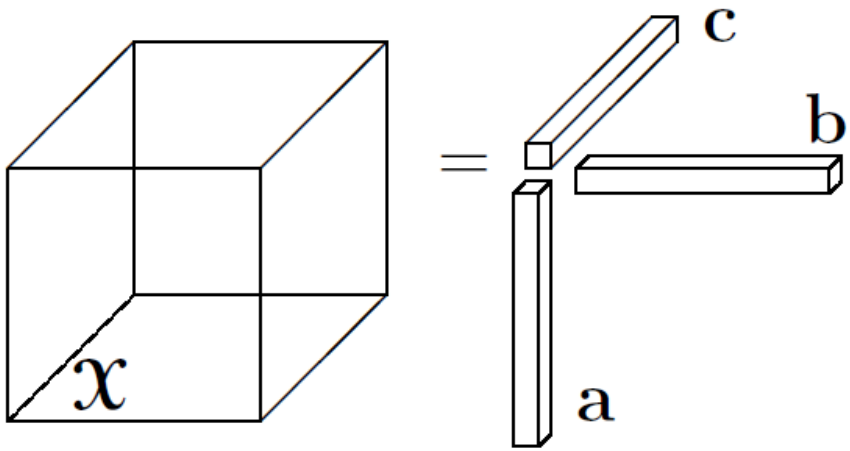
$$x_{i_1 i_2 \cdots i_N} = a^{(1)}_{i_1} a^{(2)}_{i_2} \cdots a^{(N)}_{i_N}$$

For example, if $\boldsymbol{a}^{(1)} = [1\ 2\ 3]^T$ and $\boldsymbol{a}^{(2)} = [4\ 5]^T$, then

$$\boldsymbol{X} = \boldsymbol{a}^{(1)} \circ \boldsymbol{a}^{(2)} = \begin{bmatrix} 4 & 5 \\ 8 & 10 \\ 12 & 15 \end{bmatrix}$$

which is clearly of rank 1.

A rank-1 third-order tensor $\boldsymbol{\mathcal{X}} = \boldsymbol{a} \circ \boldsymbol{b} \circ \boldsymbol{c}$ is:



To approximate a tensor, one approach is to use the CP decomposition, where C stands for canonical decomposition (CANDECOMP) and P stands for parallel factors (PARAFAC).

$\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is approximated as rank-R tensor:

$$\boldsymbol{\mathcal{X}} \approx \sum_{r=1}^{R} \boldsymbol{a}_r^{(1)} \circ \boldsymbol{a}_r^{(2)} \circ \cdots \circ \boldsymbol{a}_r^{(N)} \tag{7}$$

The CP decomposition can be exactly equal to $\boldsymbol{\mathcal{X}}$ when $R \to \infty$.

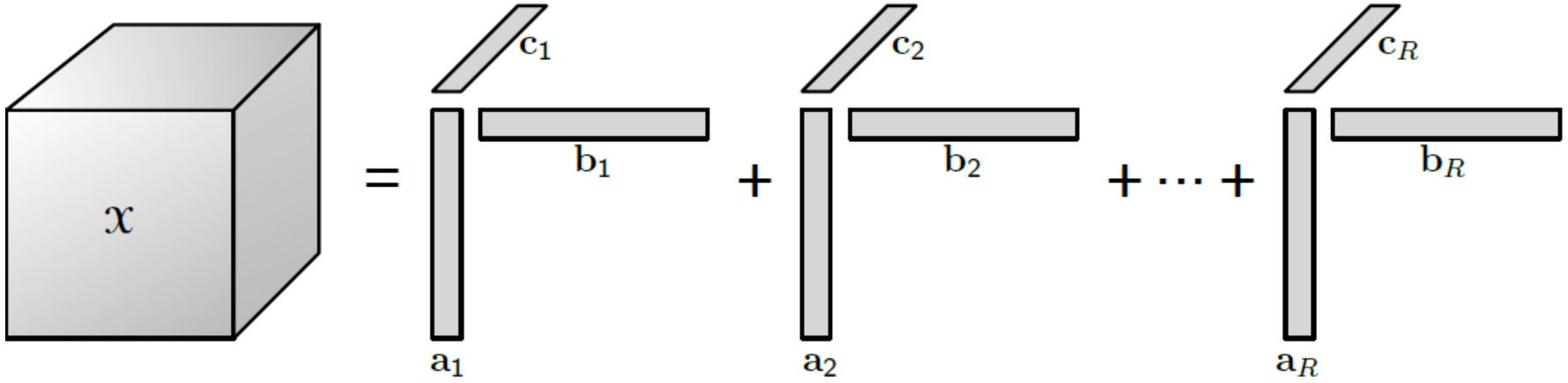
For a third-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$, the CP decomposition is:

$$\boldsymbol{\mathcal{X}} \approx \sum_{r=1}^{R} \boldsymbol{a}_r \circ \boldsymbol{b}_r \circ \boldsymbol{c}_r \tag{8}$$

where $\boldsymbol{a}_r \in \mathbb{R}^I$, $\boldsymbol{b}_r \in \mathbb{R}^J$ and $\boldsymbol{c}_r \in \mathbb{R}^K$.

Its element is expressed as:

$$x_{ijk} \approx \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr}, \quad i = 1, \cdots I, \; j = 1, \cdots, J, \; k = 1, \cdots, K \tag{9}$$



On the other hand, Tucker decomposition or higher-order SVD (HOSVD) can be used.

To proceed, we first define the n-mode product of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and a matrix $\boldsymbol{U} \in \mathbb{R}^{J \times I_n}$, denoted by $\boldsymbol{\mathcal{X}} \times_n \boldsymbol{U} \in \mathbb{R}^{I_1 \times I_2 \times \cdots I_{n-1} J I_{n+1} \cdots \times I_N}$:

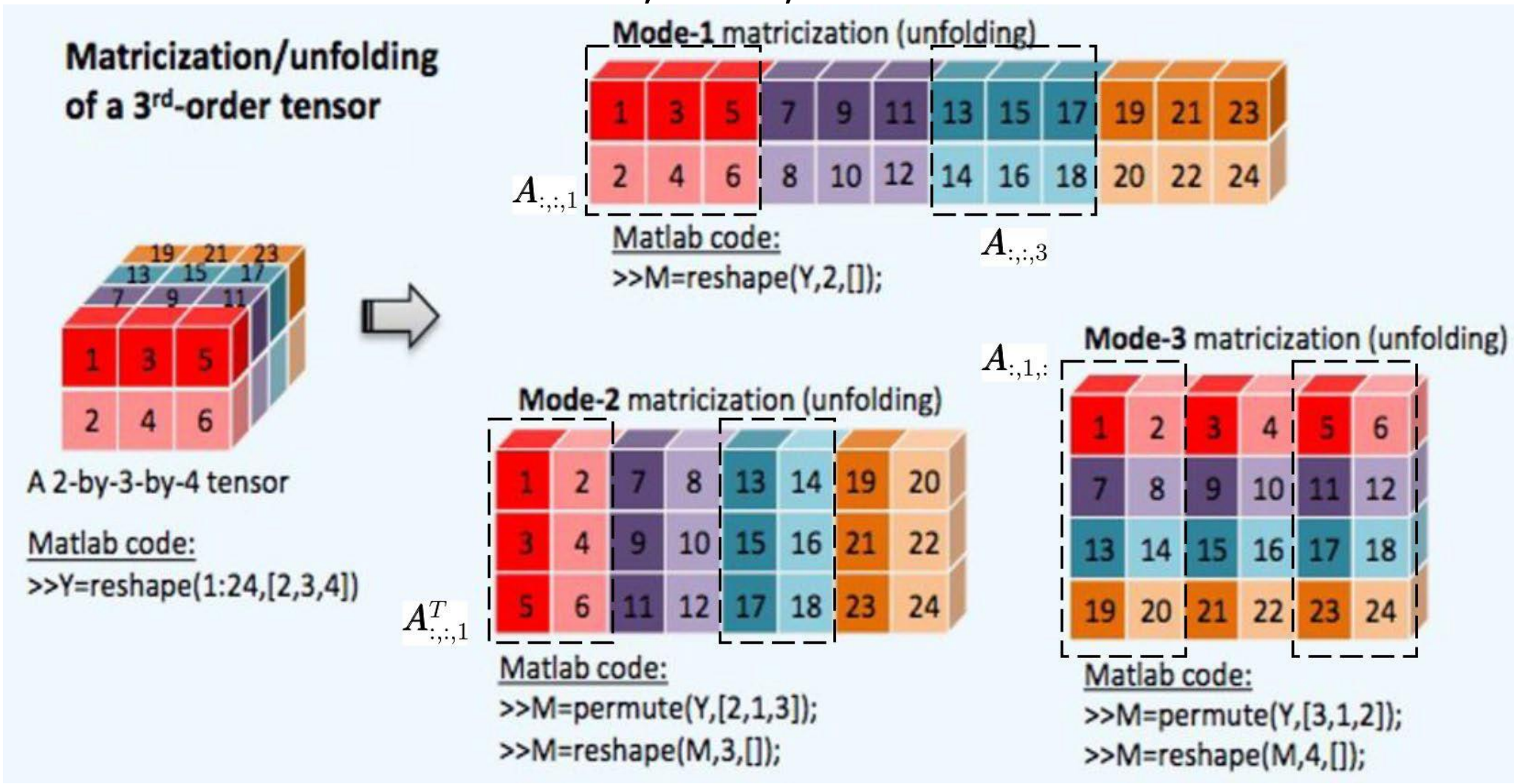
$$(\boldsymbol{\mathcal{X}} \times_n \boldsymbol{U})_{i_1 \cdots i_{n-1} j i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \cdots i_N} u_{j i_n}$$

Let $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} \times_n \boldsymbol{U}$. The idea can be expressed using unfolding:

$$\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} \times_n \boldsymbol{U} \Leftrightarrow \boldsymbol{Y}_{(n)} = \boldsymbol{U} \boldsymbol{X}_{(n)}$$

where $\boldsymbol{Y}_{(n)}$ is the n-mode unfolding of $\boldsymbol{\mathcal{Y}}$.

The n-mode unfolding can be illustrated using a third-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$ with $I = 2$, $J = 3$, and $K = 4$ as follows:



Source: A. Cichocki, R. Zdunek, A. H. Phan, S.-i. Amari, Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation, John Wiley, 2009

Example 6

Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{3\times 4\times 2}$ be sliced at:

$$\boldsymbol{X}_1 = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix} \quad \text{and} \quad \boldsymbol{X}_2 = \begin{bmatrix} 13 & 16 & 19 & 22 \\ 14 & 17 & 20 & 23 \\ 15 & 18 & 21 & 24 \end{bmatrix}$$

Then

$$\boldsymbol{X}_{(1)} = \begin{bmatrix} 1 & 4 & 7 & 10 & 13 & 16 & 19 & 22 \\ 2 & 5 & 8 & 11 & 14 & 17 & 20 & 23 \\ 3 & 6 & 9 & 12 & 15 & 18 & 21 & 24 \end{bmatrix}$$

$$\boldsymbol{X}_{(2)} = \begin{bmatrix} 1 & 2 & 3 & 13 & 14 & 15 \\ 4 & 5 & 6 & 16 & 17 & 18 \\ 7 & 8 & 9 & 19 & 20 & 21 \\ 10 & 11 & 12 & 22 & 23 & 24 \end{bmatrix}$$

$$\boldsymbol{X}_{(3)} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \end{bmatrix}$$

Suppose

$$\boldsymbol{U} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \in \mathbb{R}^{2\times 3}$$

Then $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} \times_1 \boldsymbol{U} \in \mathbb{R}^{2\times 4\times 2}$ consists of slices:

$$\boldsymbol{Y}_1 = \begin{bmatrix} 22 & 49 & 76 & 103 \\ 28 & 64 & 100 & 136 \end{bmatrix} \quad \text{and} \quad \boldsymbol{Y}_2 = \begin{bmatrix} 130 & 157 & 184 & 211 \\ 172 & 208 & 244 & 280 \end{bmatrix}$$

The HOSVD of $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1\times I_2\times \cdots \times I_N}$ is:

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{G}} \times_1 \boldsymbol{A}^{(1)} \times_2 \boldsymbol{A}^{(2)} \cdots \times_N \boldsymbol{A}^{(N)} \tag{10}$$

where $\boldsymbol{A}^{(n)} \in \mathbb{R}^{I_n \times I_n}$ is the matrix containing the left singular vectors of $\boldsymbol{X}_{(n)} \in \mathbb{R}^{I_n \times I_1 I_2 \cdots I_{n-1} I_{n+1} \cdots I_N}$ as in (1).

$\boldsymbol{\mathcal{G}} \in \mathbb{R}^{I_1\times I_2\times \cdots \times I_N}$ is called the core tensor and is computed as:

$$\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}} \times_1 \left(\boldsymbol{A}^{(1)}\right)^T \times_2 \left(\boldsymbol{A}^{(2)}\right)^T \cdots \times_N \left(\boldsymbol{A}^{(N)}\right)^T \tag{11}$$

The element of $\boldsymbol{\mathcal{X}}$ is:

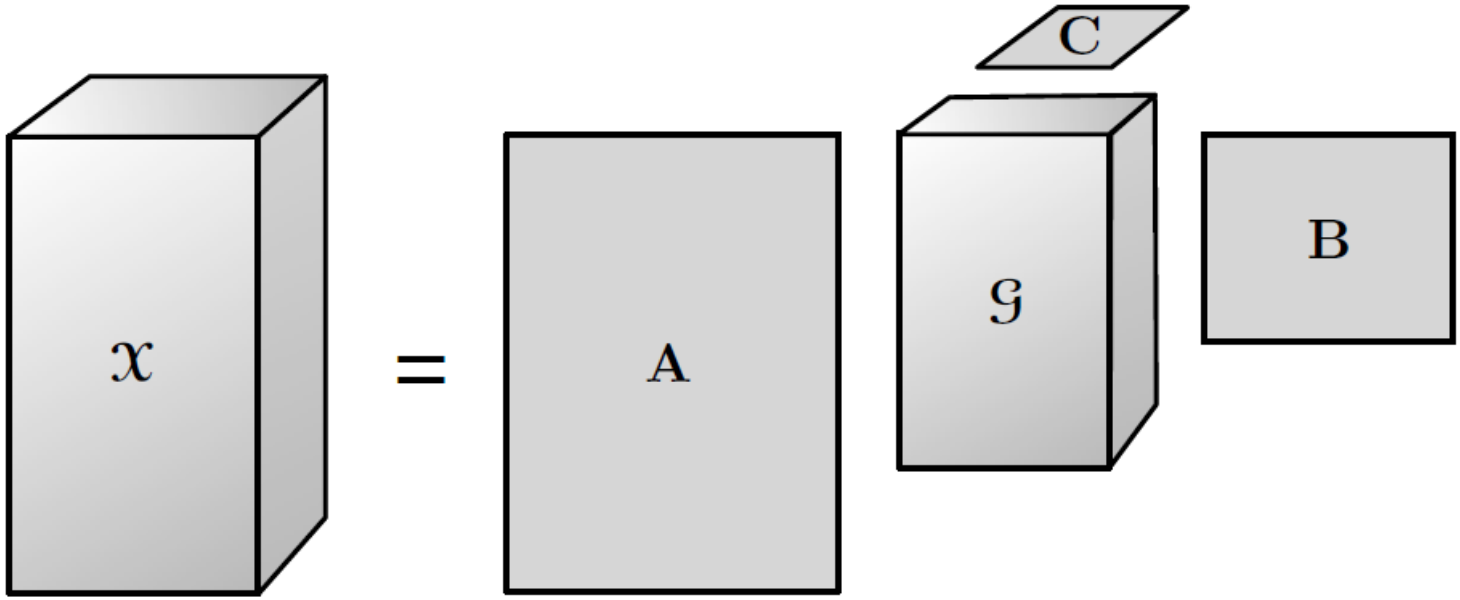
$$x_{i_1 i_2 \cdots i_N} = \sum_{r_1=1}^{I_1} \sum_{r_2=1}^{I_2} \cdots \sum_{r_N=1}^{I_N} g_{r_1 r_2 \cdots r_N} a_{i_1 r_1}^{(1)} a_{i_2 r_2}^{(2)} \cdots a_{i_N r_N}^{(N)}$$

where $i_n = 1, \cdots, I_n,\ n = 1, \cdots, N$. In practice, $\boldsymbol{\mathcal{G}}$ can be of smaller size to achieve data compression.

Taking the third-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$ as illustration:

$$\boldsymbol{\mathcal{X}} \approx \boldsymbol{\mathcal{G}} \times_1 \boldsymbol{A} \times_2 \boldsymbol{B} \times_3 \boldsymbol{C} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} \boldsymbol{a}_p \circ \boldsymbol{b}_q \circ \boldsymbol{c}_r \qquad (12)$$

where $\boldsymbol{A} \in \mathbb{R}^{I \times P}$, $\boldsymbol{B} \in \mathbb{R}^{J \times Q}$ and $\boldsymbol{C} \in \mathbb{R}^{K \times R}$ are factor matrices or principal components in each mode, and $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{P \times Q \times R}$, with $P < I$, $Q < J$ and $R < K$.



Element-wise, (12) means

$$x_{ijk} \approx \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_{ip} b_{jq} c_{kr}, i = 1, \cdots I, j = 1, \cdots, J, k = 1, \cdots, K \quad (13)$$

Comparing (8)-(9) and (12)-(13), CP can be viewed as a special case of HOSVD when $P = Q = R$ and $g_{pqr} = 1$ for $p = q = r$ and 0 otherwise.

Example 7

Perform the HOSVD on the tensor $\mathcal{X} \in \mathbb{R}^{3\times 4\times 2}$ in Example 6.

According to (10), the HOSVD of $\mathcal{X} \in \mathbb{R}^{3\times 4\times 2}$ is:

$$\mathcal{X} = \mathcal{G} \times_1 \boldsymbol{A}^{(1)} \times_2 \boldsymbol{A}^{(2)} \times_3 \boldsymbol{A}^{(3)}$$

where $\boldsymbol{A}^{(1)}$, $\boldsymbol{A}^{(2)}$ and $\boldsymbol{A}^{(3)}$ are matrices containing the left singular vectors of $\boldsymbol{X}_{(1)}$, $\boldsymbol{X}_{(2)}$ and $\boldsymbol{X}_{(3)}$, respectively, which have been determined in Example 6.

Note that the computation is performed from left to right, i.e., $\mathcal{G} \times_1 \boldsymbol{A}^{(1)}$ is first computed.

We make use of tensor toolbox in MATLAB to perform operations including HOSVD and n-mode product. For example, a tensor can be created using `tensor`.

The HOSVD can be computed using:

```
X(:,:,1) = [1 4 7 10; 2 5 8 11; 3 6 9 12];
X(:,:,2) = [13 16 19 22; 14 17 20 23; 15 18 21 24];
X = tensor(X);
Y = hosvd(X,10^-8); G = Y.core;
A1 = Y.U{1};
A2 = Y.U{2};
A3 = Y.U{3};
```

Verification can be done using:

```
GA1 = double(A1)*double(tenmat(G,1));
T = tensor(reshape(GA1,[3,2,2]));
TA2 = double(A2)*double(tenmat(T,2));
T1(:,:,1) = TA2(:,1:3)';
T1(:,:,2) = TA2(:,4:6)';
Final = double(A3)*double(tenmat(T1,3));
R(:,1,:) = Final(:,1:3)';
R(:,2,:) = Final(:,4:6)';
R(:,3,:) = Final(:,7:9)';
R(:,4,:) = Final(:,10:12)';
```

The result based on (10) is

```
>> R
R is a tensor of size 3 x 4 x 2

R(:,:,1) =
 1.0000    4.0000    7.0000   10.0000
 2.0000    5.0000    8.0000   11.0000
 3.0000    6.0000    9.0000   12.0000

R(:,:,2) =
     13.0000   16.0000   19.0000   22.0000
     14.0000   17.0000   20.0000   23.0000
     15.0000   18.0000   21.0000   24.0000
```

which is equal to $\mathcal{X}$.

Approximation using (12) or (13) can be evaluated as well. For example, using $\mathcal{G}$ as a scalar, i.e., g_{111}:

```
S(:,:,1)=zeros(3,4);
S(:,:,2)=zeros(3,4);
S = tensor(S);
for i = 1:3
    for j = 1:4
        for k= 1:2
            S(i,j,k) =G(1,1,1)*A1(i,1)*A2(j,1)*A3(k,1);
        end
    end
end
```

We see $\mathcal{X} \approx \mathcal{S}$:

```
>> S
S(:,:,1) =
    4.5862   5.8637   7.1411   8.4186
    4.8866   6.2477   7.6089   8.9700
    5.1869   6.6318   8.0766   9.5214
S(:,:,2) =
   12.1422  15.5244  18.9066  22.2888
   12.9375  16.5412  20.1450  23.7487
   13.7328  17.5580  21.3833  25.2086
```

Example 8

We demonstrate video compression using HOSVD. Two videos are downloaded from:

http://jacarini.dinf.usherbrooke.ca/dataset2014/

The first is extracted from overpass.zip with dimensions 240 x 320 x 96.

The second is extracted from peopleInshade.zip with dimensions 244 x 380 x 44.

The compression ratio is:

$$\frac{IJK}{IP + JQ + KR + PQR}$$

Original video

P=40, Q=60, R=30

P=80, Q=100, R=40

P=120, Q=160, R=48

Original video

P=40, Q=60, R=20

P=80, Q=100, R=30

P=120, Q=160, R=40

References:

1. http://faculty.washington.edu/sbrunton/me565/pdf/L29secure.pdf
2. http://www.cs.sjsu.edu/~stamp/ML/powerpoint/
3. M. Stamp, *Introduction to Machine Learning with Applications in Information Security*, Chapman & Hall/CRC, 2017
4. T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, 51(3), 455–500, 2009
5. J. Yang, D. Zhang, A. F. Frangi, J.-y. Yang, "Two-dimensional PCA: A new approach to appearance-based face representation and recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Jan. 2004, vol. 26, no. 1, pp. 131-137