

機械学習特論

～ 理論とアルゴリズム ～

(Dimensionality reduction; PCA)

講師：西郷浩人

Matrix Decomposition (行列分解) and Dimensionality reduction (次元削減)

- Unsupervised dimensionality reduction methods

$$X'X = \lambda_1 w_1 w_1' + \lambda_2 w_2 w_2' \dots \lambda_p w_p w_p' \rightarrow \lambda_1 w_1 w_1' + \lambda_2 w_2 w_2' \dots \lambda_k w_k w_k'$$

(k < p)

- PCA(Principal Component Analysis)
- Supervised dimensionality reduction methods

$$X'Y = \lambda_1 w_1 w_1' + \lambda_2 w_2 w_2' \dots \lambda_p w_p w_p' = \lambda_1 w_1 w_1' + \lambda_2 w_2 w_2' \dots \lambda_k w_k w_k'$$

(k < p)

- PLS(Partial Least Squares)
- CCA(Canonical Correlation Analysis)
- Reduced-rank LDA
- In both cases, the number of dimensions k is chosen by a user.

PCA (主成分分析)

- Let $X \in \mathbb{R}^{n \times p}$ be a matrix containing data.
- We consider projecting \mathbf{X} by \mathbf{w} onto a space \mathbf{z} .

$$\mathbf{z} = X \mathbf{w}$$

- The objective is to maximize (squared) variance in the projected space \mathbf{z}

$$\max_{\mathbf{w}} \text{var}(\mathbf{z})^2 = \max_{\mathbf{w}} \mathbf{z}' \mathbf{z} = \max_{\mathbf{w}} \mathbf{w}' X' X \mathbf{w}$$

- This objective can take arbitrary large value (which is useless), so we restrict \mathbf{w} on a unit radius euclidian sphere.

$$\begin{aligned} \max_{\mathbf{w}} \mathbf{w}' X' X \mathbf{w} \\ \text{s.t. } \mathbf{w}' \mathbf{w} = 1 \end{aligned}$$

PCA

$$\begin{aligned} \max_{\mathbf{w}} \mathbf{w}' \mathbf{X}' \mathbf{X} \mathbf{w} \\ \text{s.t. } \mathbf{w}' \mathbf{w} = 1 \end{aligned}$$

- Now we maximize the constrained objective function using Lagrange multipliers. Let

$$L = \mathbf{w}' \mathbf{X}' \mathbf{X} \mathbf{w} - \lambda (\mathbf{w}' \mathbf{w} - 1)$$

- By taking derivative w.r.t. \mathbf{w} and setting it to zero, we get

$$\frac{\partial L}{\partial \mathbf{w}} = 2 \mathbf{X}' \mathbf{X} \mathbf{w} - 2 \lambda \mathbf{w} = 0$$

$$\iff (\mathbf{X}' \mathbf{X} - \lambda \mathbf{I}) \mathbf{w} = 0$$

$$\iff \mathbf{X}' \mathbf{X} \mathbf{w} = \lambda \mathbf{w}$$

- which is an eigenproblem of *covariance matrix* $\mathbf{X}'\mathbf{X}$

Review: eigendecomposition

Eigendecomposition of a symmetric matrix

- $n \times n$ symmetric matrix A has eigenvector x , eigenvalue λ , and satisfies the following condition.

$$A x = \lambda x$$

- Let Σ be a diagonal matrix storing the eigenvalues, and V be a matrix storing eigenvectors in its columns: $V = [x_1, x_2, x_3, \dots, x_n]$

then they can be written as

$$\begin{aligned} V^{-1} A V &= \Sigma \\ \iff A &= V \Sigma V' \end{aligned}$$

$$A = V \Sigma V' = x_1 \times \lambda_1 \times x_1' + x_2 \times \lambda_2 \times x_2' + \dots + x_n \times \lambda_n \times x_n'$$

Interpretation

$$A = V \Sigma V' = x_1 \times \lambda_1 \times x_1' + x_2 \times \lambda_2 \times x_2' + \dots + x_n \times \lambda_n \times x_n'$$


- Outer product $\mathbf{x} \mathbf{x}'$ is a matrix representing a space spanned by an eigenvector.
- Original matrix A is reproduced by weighted sum of spaces spanned by eigenvectors.
 - A weight corresponds to eigenvalue.
- If we reproduce a space using only the top-k eigenvalues and corresponding eigenvectors, we can perform *dimensionality reduction*.

Example in Octave

```
> A=[5 -3; -3 5];
```

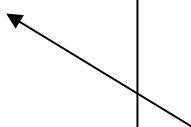
```
> [V S]=eig(A)
```

```
V =
```



-0.70711
-0.70711

2nd eigenvector



0.70711
-0.70711

1st eigenvector

```
S =
```

Diagonal Matrix



2	0
0	8

2nd eigenvalue



1st eigenvalue

Example in Octave

```
> A=[5 -3;-3 5]
ans =
    5.0000   -3.0000
   -3.0000    5.0000
> V(:,1)*S(1,1)*V(:,1)'
ans =
    1.0000    1.0000
    1.0000    1.0000
> V(:,2)*S(2,2)*V(:,2)'
ans =
    4.0000   -4.0000
   -4.0000    4.0000
> V(:,1)*S(1,1)*V(:,1)'+V(:,2)*S(
2,2)*V(:,2)'
ans =
    5.0000   -3.0000
   -3.0000    5.0000
> S(2,2)/trace(S)
ans = 0.80000
```

- A space spanned by the first eigenvector approximates the 80% of the original matrix.

Algorithms for matrix decomposition

- Eigendecomposition for a symmetric matrix
 - LU decomposition
 - QR decomposition
 - Power method
 - Lanczos method
- Singular value decomposition (SVD) for an asymmetric matrix

Power method

- An iterative algorithm for finding the 1st eigenvector of a matrix.
- Lanczos method is an extension of power method that can find the top-k eigenvectors.

%one-liner example

```
> A=[1 2;3 2], x=[1 1]'
```

```
> x = A*x, x = x/norm(x), s = x'*A*x
```

```
> x = A*x, x = x/norm(x), s = x'*A*x
```

```
> x = A*x, x = x/norm(x), s = x'*A*x
```

```
> x = A*x, x = x/norm(x), s = x'*A*x
```

...

power.m

```
function [r] = power(A)
```

```
    r = ones(size(A,1),1);
```

```
    while 1
```

```
        old_r = r;
```

```
        r = A*r
```

```
        r = r./norm(r)
```

```
        if norm(r-old_r) < 0.01
```

```
            break;
```

```
        end
```

```
    end
```

```
endfunction
```

Example: PCA

Amino acid substitution matrix

- 20 by 20 similarity matrix (blosum62.txt)

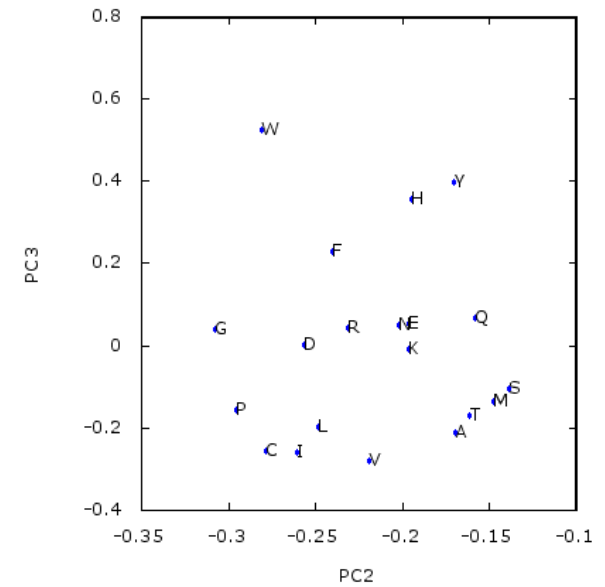
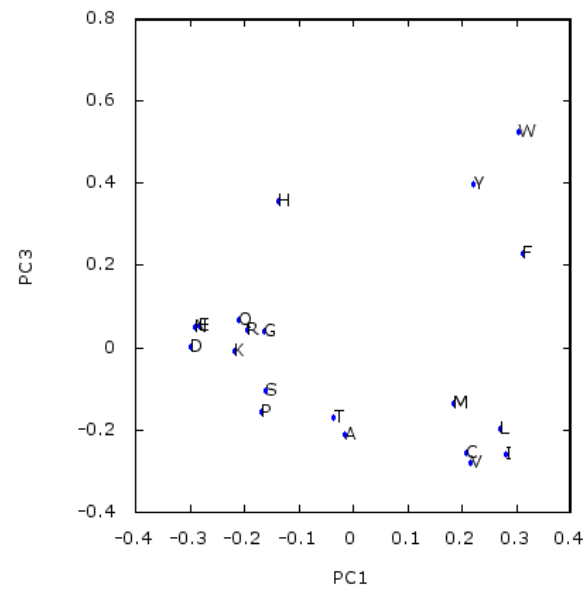
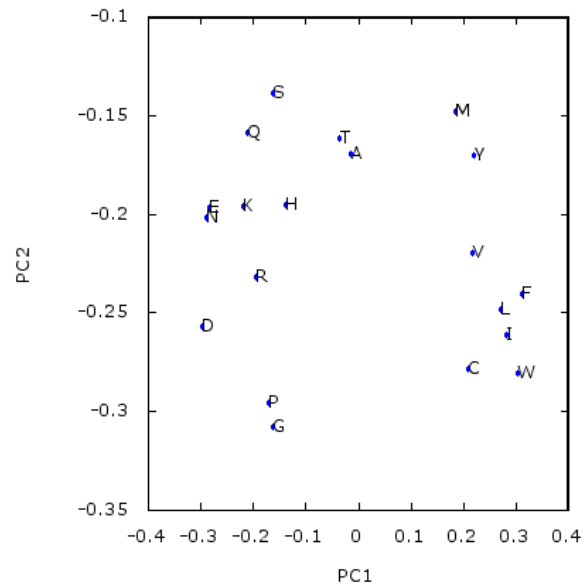
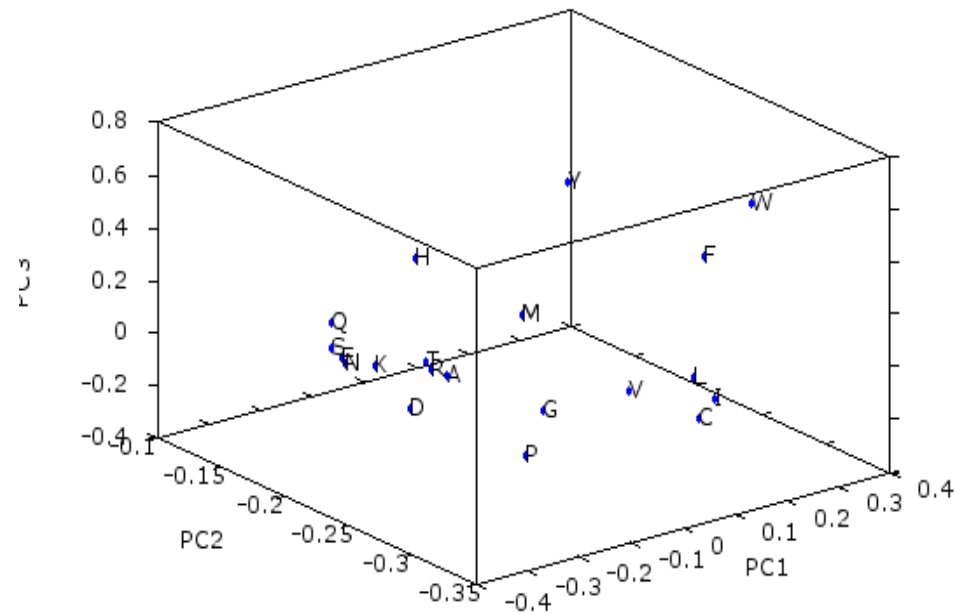
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-2	-1	1	0	-3	-2	0	
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

Eigendecomposition and plot

-

```
M=load('blossum62.txt');
[X S]=eig(M);
shg; clf; axis square
plot3(X(:,20),X(:,1),X(:,19),'.');
text(X(1,20),X(1,1),X(1,19),'A');
text(X(2,20),X(2,1),X(2,19),'R');
text(X(3,20),X(3,1),X(3,19),'N');
text(X(4,20),X(4,1),X(4,19),'D');
text(X(5,20),X(5,1),X(5,19),'C');
text(X(6,20),X(6,1),X(6,19),'Q');
text(X(7,20),X(7,1),X(7,19),'E');
text(X(8,20),X(8,1),X(8,19),'G');
text(X(9,20),X(9,1),X(9,19),'H');
text(X(10,20),X(10,1),X(10,19),'I');
text(X(11,20),X(11,1),X(11,19),'L');
text(X(12,20),X(12,1),X(12,19),'K');
text(X(13,20),X(13,1),X(13,19),'M');
text(X(14,20),X(14,1),X(14,19),'F');
text(X(15,20),X(15,1),X(15,19),'P');
text(X(16,20),X(16,1),X(16,19),'S');
text(X(17,20),X(17,1),X(17,19),'T');
text(X(18,20),X(18,1),X(18,19),'W');
text(X(19,20),X(19,1),X(19,19),'Y');
text(X(20,20),X(20,1),X(20,19),'V');
axis square;
xlabel('PC2');
ylabel('PC3');
print -dgif amino2d3.gif
```

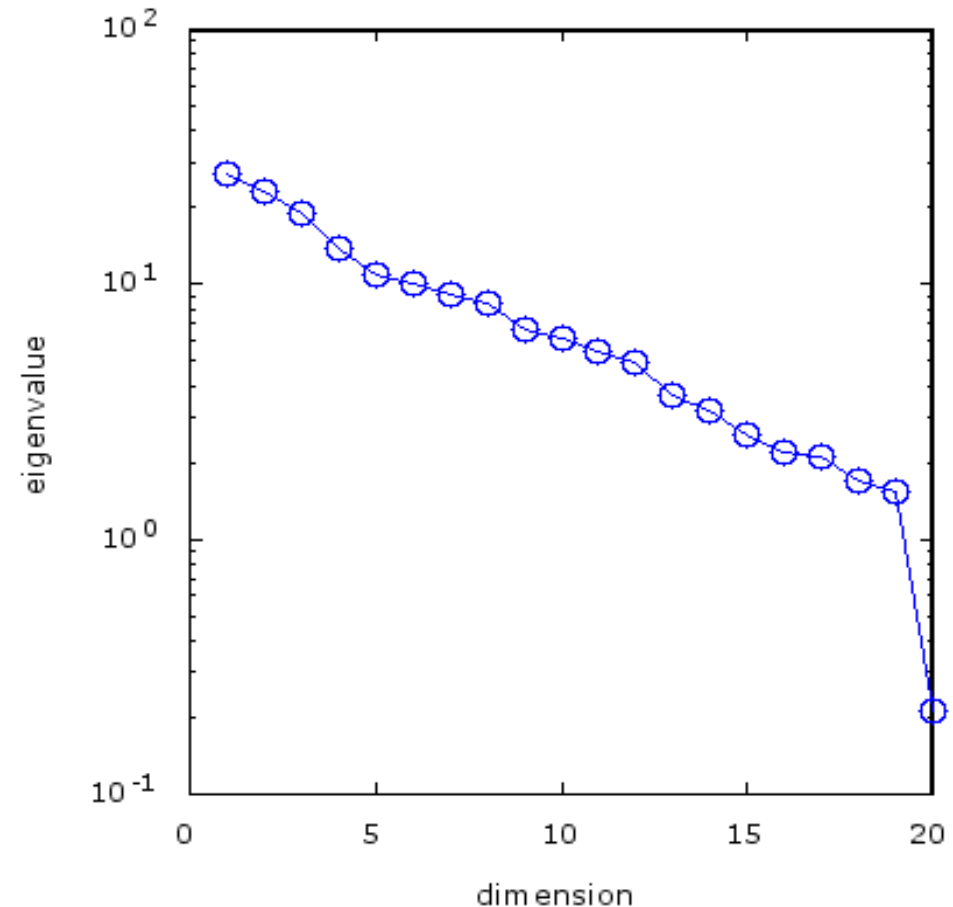
2D and 3D visualization



Distribution of eigenvalues

- Top-3 dimensions cover 42.6% of the variance in data

```
> sigma=abs(diag(S));  
  
> semilogy(sigma,'-o');  
  
> sigma(1)/sum(sigma)  
Ans = 0.16712  
  
> sum(sigma(1:2))/sum(sigma)  
Ans = 0.30873  
  
> sum(sigma(1:3))/sum(sigma)  
Ans = 0.42619  
  
> sum(sigma(1:20))/sum(sigma)  
ans = 1
```



Singular Value Decomposition (SVD)

Eigendecomposition and singular value decomposition (SVD)

- Eigen decomposition handles
 - $n \times n$ symmetric matrices only

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$$

- SVD handles
 - $n \times n$ asymmetric matrix, and $m \times n$ rectangular matrices

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{bmatrix}$$

SVD

- Let A be an $m \times n$ ($m > n$) matrix and its singular value be σ and its pair of singular vectors be \mathbf{u}, \mathbf{v} , they satisfy

$$A \mathbf{v} = \sigma \mathbf{u}, A^H \mathbf{u} = \sigma \mathbf{v}$$

- where A^H is an Hermitian matrix. If elements of A are real numbers, then its Hermitian matrix is A^T equivalent to a transposed matrix of A ;

- Let Σ be a diagonal matrix storing the singular values and U, V be a matrix storing the singular values;
 $U = [\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \dots, \mathbf{u}_n]$ $V = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \dots, \mathbf{v}_n]$

- Then they are called as *singular value decomposition* of A , and written as;

$$A = U \Sigma V' = u_1 \times \sigma_1 \times v_1' + u_2 \times \sigma_2 \times v_2' + \dots + u_n \times \sigma_n \times v_n'$$

Interpretation

$$A = U \Sigma V' = u_1 \times \sigma_1 \times v_1' + u_2 \times \sigma_2 \times v_2' + \dots + u_n \times \sigma_n \times v_n'$$

- If a given matrix A is symmetric, eigendecomposition of A and SVD of A are equivalent up to signs.
 - singular values are square root of eigenvalues.
- Left singular vector \mathbf{U} spans the row space of A , and right singular vector \mathbf{V} spans the column space of \mathbf{A} .
- By using top- k singular values and corresponding pair of singular values, one can perform *dimensionality reduction*.

SVD in Octave

```
A=[1 2 3; 4 5 6; 7 8 9];
```

```
[U S V]=svd(A)
```

```
U =
```

```
-0.21484  0.88723  0.40825  
-0.52059  0.24964 -0.81650  
-0.82634 -0.38794  0.40825
```

```
S =
```

Diagonal Matrix

```
1.6848e+01    0    0  
    0 1.0684e+00    0  
    0    0 3.8388e-16
```

```
V =
```

```
-0.479671 -0.776691 -0.408248  
-0.572368 -0.075686  0.816497  
-0.665064  0.625318 -0.408248
```

Example: SVD

2D black/white image

- Black/white image is represented by 0~255 (8bit) integers. In the following case, lena.gif is a 256 x 256 pixels image; a 256 x 256 matrix.

```
A = imread('lena.gif')
```

```
A =
```

Columns 1 through 16:



```
162 162 159 161 162 159 159 157 156 160 155 155 156 154 154 155
162 162 159 161 162 159 159 157 156 160 155 155 156 154 154 155
163 160 159 160 160 158 157 155 156 157 154 153 156 155 154 156
159 158 159 157 159 159 154 153 155 155 153 152 154 153 151 155
155 157 156 157 158 157 156 155 155 156 156 153 154 153 153 154
156 157 155 153 157 157 156 156 156 155 156 155 152 154 155 152
157 157 156 156 157 156 155 156 157 155 156 155 154 153 154 154
158 157 157 156 155 156 157 155 156 155 154 154 153 153 154 158
156 156 156 153 156 155 156 156 154 157 155 155 152 152 156 162
156 156 157 154 157 156 157 159 158 158 155 157 154 155 157 160
156 156 156 156 156 157 158 158 159 157 154 155 153 155 158 162
157 155 153 157 157 156 158 157 159 157 157 155 153 155 158 163
155 152 154 157 159 159 158 157 158 157 157 157 157 159 161 166
154 155 156 155 157 159 159 158 158 158 157 157 157 162 164 166
158 158 158 157 157 158 159 158 159 159 159 159 160 165 167 169
157 158 158 158 159 159 160 158 159 159 159 162 164 166 167 169
156 158 158 156 158 158 159 160 161 161 159 164 167 169 169 170
159 158 158 158 158 159 162 161 160 162 164 165 167 167 168 168
159 158 159 159 162 164 161 162 163 164 166 168 167 168 167 16
```

SVD of black/white image

```
> A=imread('lena.gif')
> colormap(gray);
> axis square;
> rank(A);
> [U S V]=svd(A);
> for k=1:60; pause(1.0), imagesc(U(:,1:k)*S(1:k,1:k)*V(:,1:k)'), end
```

The last line reconstructs space spanned by the first k eigenvectors, and plot it every 1 second.

One can show the original image by `imagesc(A)`

$$A = U \Sigma V' = \mathbf{u}_1 \times \sigma_1 \times \mathbf{v}_1' + \mathbf{u}_2 \times \sigma_2 \times \mathbf{v}_2' + \dots + \mathbf{u}_n \times \sigma_n \times \mathbf{v}_n'$$

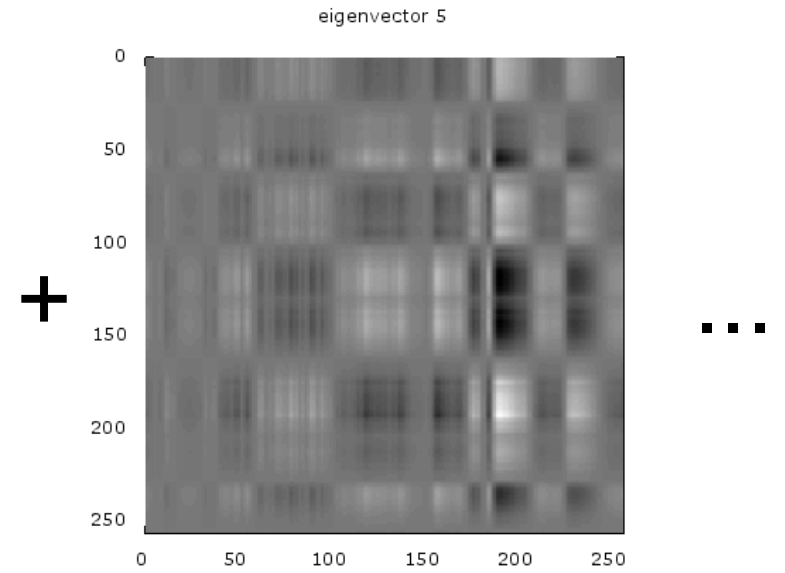
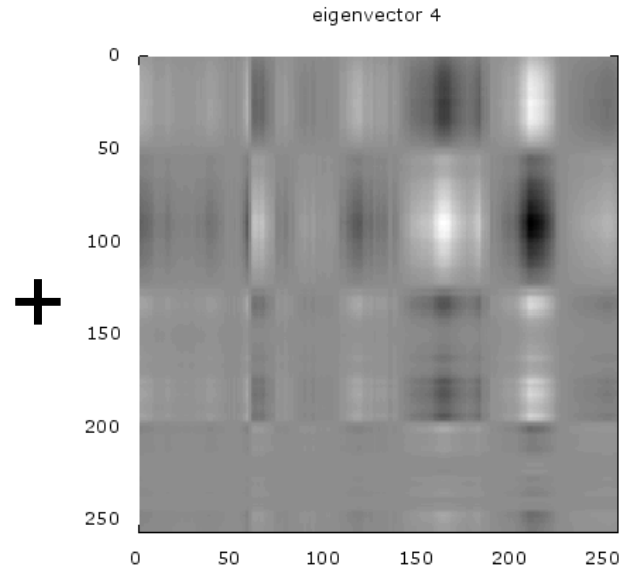
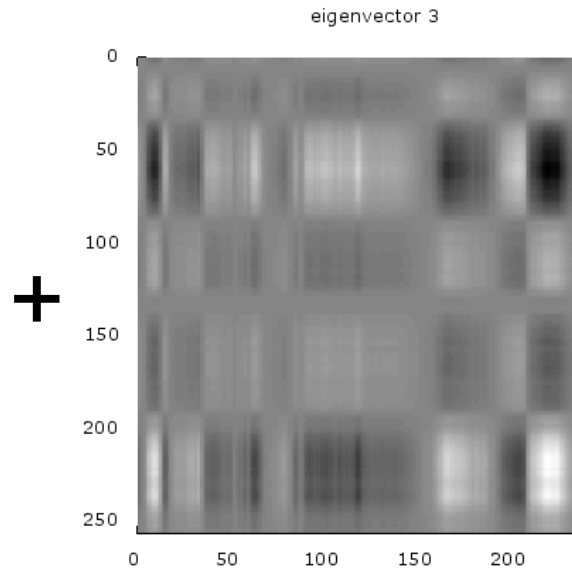
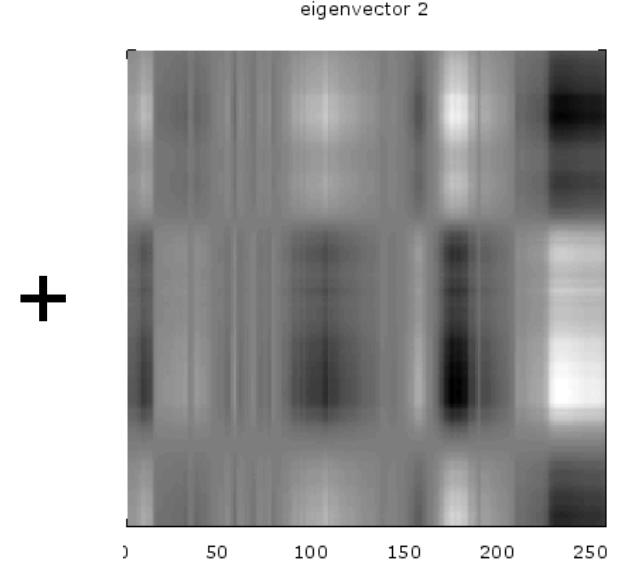
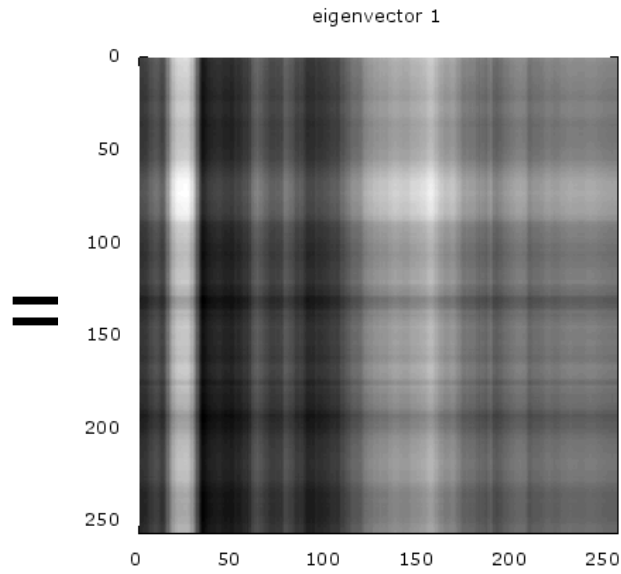
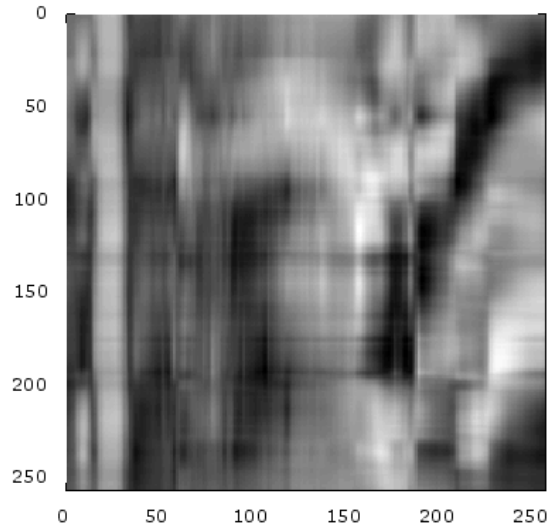
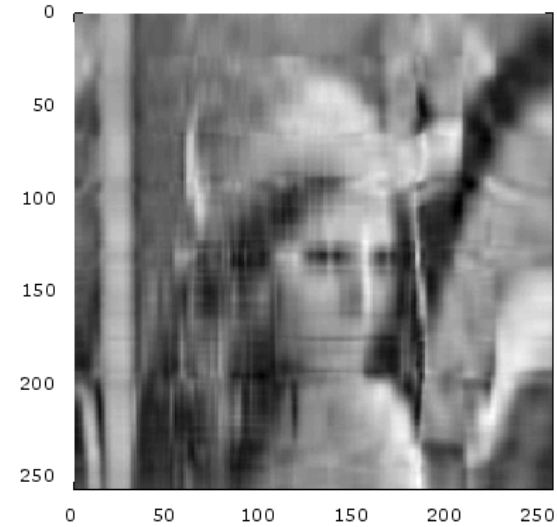


Image reconstruction based on eigenvectors

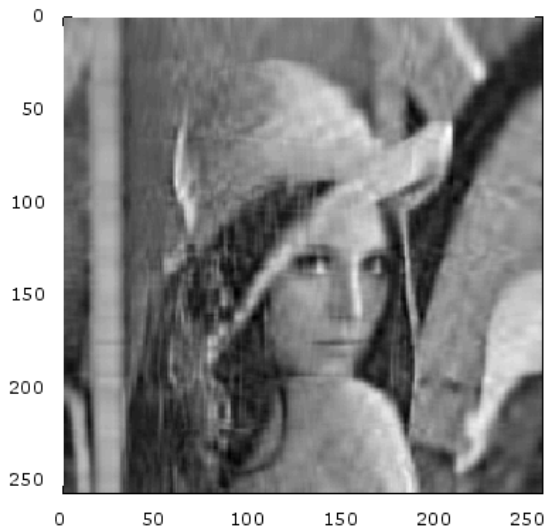
5 eigenvectors



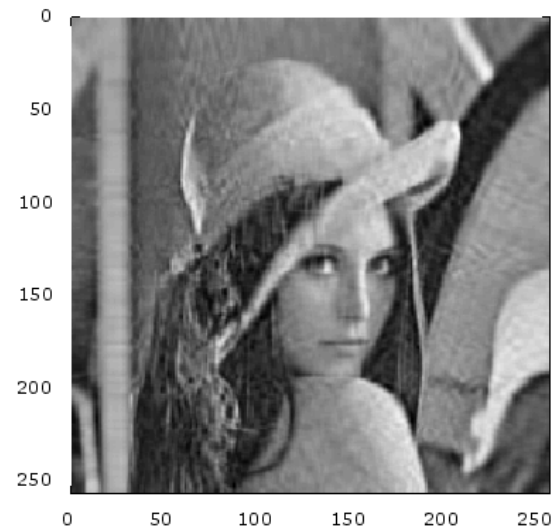
10 eigenvectors



20 eigenvectors



30 eigenvectors



Distribution of eigenvalues

- Top-3 dimensions cover 41.2% of the variance in data

```
> sigma=abs(diag(S));
```

```
> semilogy(sigma,'-o');
```

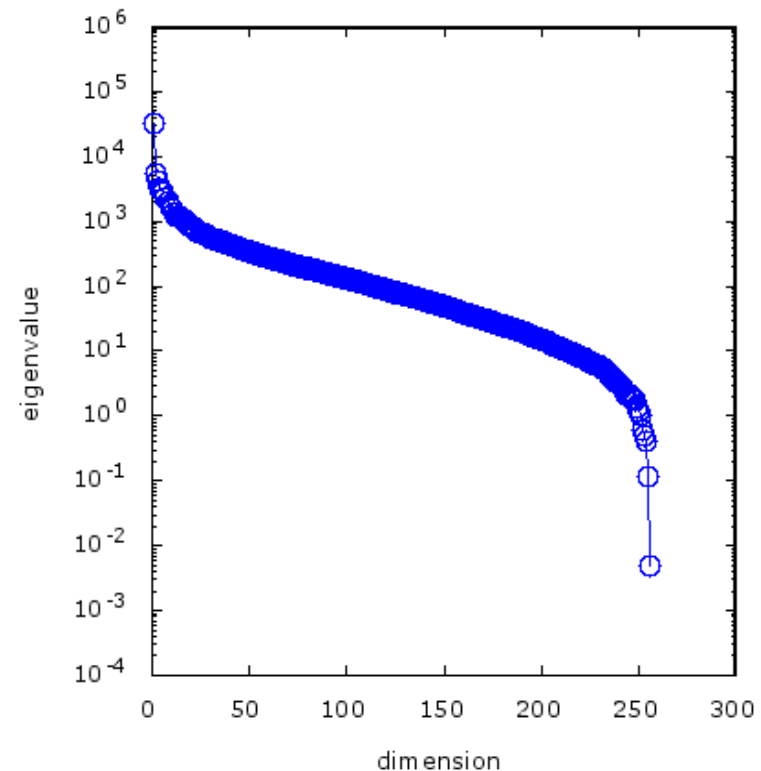
```
> sigma(1)/sum(sigma)  
ans = 0.31909
```

```
> sum(sigma(1:2))/sum(sigma)  
ans = 0.37161
```

```
> sum(sigma(1:3))/sum(sigma)  
ans = 0.41214
```

```
> sum(sigma(1:5))/sum(sigma)  
ans = 0.47340
```

```
> sum(sigma(1:10))/sum(sigma)  
Ans = 0.57651
```



Example: Eigenface

- Input 5000 face images (100 shown below) stored in X (5000×1024 (32×32)).



Face eigenvector (eigenface)

- PCA of $X^T X$, and the resulting top 36 eigenvectors (eigenface)



Original vs recovered face

- Original (5000 x 1024) vs recovered (5000 x 100)
- More than 10 times data compression.

Original faces



Recovered faces



- Try the demo.

```
>eigenface
```

- Look into the code. It is quite simple.
- The code is written by Andrew Ing.
 - <https://www.coursera.org/learn/machine-learning/>

Partial Least Squares (PLS)

PLS (Partial Least Squares)_{PCA}

$$\max_w \text{cov}(Xw, y) = \max_w w' X' y y' X w$$

$$\begin{aligned} \max_w w' X' X w \\ \text{s.t. } w' w = 1 \end{aligned}$$

- Popular in chemometrics for screening chemical compounds.
- The objective function maximizes covariance between Xw and y .
- With the same derivation as in PCA, the solution of PLS is obtained by solving the following eigenproblem.

$$X' y y' X w = \lambda w$$

PCA

$$X' X w = \lambda w$$

- Note that cross-product matrix of X and y is decomposed instead of X .

PLS vs OLS

- PLS performs dimensionality reduction likewise PCA, and the number of dimensions must be specified by a user, too.
 - If one take the number of dimensions to be p , then the obtained solution (coefficients) is equivalent to OLS solution.
- For solving PLS, eigenproblem is not usually used, but simpler iterative algorithm such as NIPALS is used.

NIPLAS algorithm for PLS

- Input
 - $n \times p$ matrix X , length n vector y , dimension k
- Output
 - length p vector β
- Repeat k times
 - $u = y$
 - $w = X' * u$
 - $t = X * w$
 - $c = Y' * t$
 - $p = X' * t$
 - $X = X - t * p'$
- $\beta = w * ((w' * X' * X * w) \oslash (X' * y))$

multi-response PLS

PLS

$$\max_w \text{cov}(X \mathbf{w}, Y_i) = \max_w \mathbf{w}' X' Y_i Y_i' X \mathbf{w}$$

$$\max_w \text{cov}(X \mathbf{w}, \mathbf{y})$$

- When there are several response variables, PLS can be extended to multi-class PLS.
- Notice that response \mathbf{Y} is not a vector anymore, but a matrix. So is \mathbf{W} .
- Multi-class PLS problem can be seen as generalized eigenproblem.

PLS

$$X' Y Y' X W = \lambda W$$

$$X' \mathbf{y} \mathbf{y}' X \mathbf{w} = \lambda \mathbf{w}$$

Example: wine data

- X matrix (predictors) measured by anyone.

Wine	Price	Sugar	Alcohol	Acidity
1	7	7	13	7
2	4	3	14	7
3	10	5	12	5
4	16	7	11	3
5	13	3	10	3

- Y matrix (responses) measured by experts.

Wine	Hedonic	Goes with meat	Goes with dessert
1	14	7	8
2	10	7	6
3	8	5	5
4	2	4	7
5	6	2	4

Interpretation

- Usage example (when # of components = 4)

```
>loadWineData; [beta]=pls(X,Y,4)
```

	Hedonic	Goes with meat	Goes with dessert
Price	1.77	-0.543	-0.179
Sugar	-1.33	0.657	0.759
Alcohol	-4.00	1.00	0.50
Acidity	8.99	-0.971	-0.402

- In the second column, Acidity is the main reason for Hedonic.
- In the third column Alcohol explains the reason for “Goes with meat”.
- In the fourth column, Sugar explains the reason for “Goes with dessert”.

Appendix

Reading colour image

- lena_color.jpg is 512 x 512 pixels. Depth of the basic 3 colors (red, green, blue) are represented by integers ranging from 0~255.
- Colour image is not a matrix, but rather a tensor.

```
> A=imread('lena_color.jpg');  
> [s1 s2 s3] = size(A)  
s1 = 512  
s2 = 512  
s3 = 3
```



SVD of colour image

- Colour image is a 3-d tensor rather than a matrix, so we transform it to matrix using 'reshape' command, then perform SVD. Transforming it back to tensor is required when one wants to visualize it.

```
[U S V]=svd(reshape(A,s1,s2*s3));  
for i=1:60,sleep(0.5),imagesc(reshape(U(:,1:i)*S(1:i,1:i)*V(:,1:i)',s1,s2,s3)),end
```

PCR

(Principal Component Regression)

- Supervised method
- $\text{PCR} = \text{PCA} + \text{OLS}$
- Procedure
 - A set of principal components is extracted using PCA.
 - Regression coefficients are obtained for each principal component by solving OLS.
- PCR is often compare with PLS, but PLS is proven to fit better than PCR.

Connection to matrix decomposition

- In the NIPALS algorithm,
 - The first weight vector \mathbf{w} is the first eigenvector of $\mathbf{X}'\mathbf{Y}\mathbf{Y}'\mathbf{X}$
 - \mathbf{w} is also the first right singular vector of $\mathbf{X}'\mathbf{Y}$
 - The left singular vector of $\mathbf{X}'\mathbf{Y}$ is \mathbf{c}
 - The first score vector \mathbf{t} is the first eigenvector of $\mathbf{X}\mathbf{X}'\mathbf{Y}\mathbf{Y}'$
 - The first \mathbf{c} vector is the first eigenvector of $\mathbf{Y}\mathbf{Y}'\mathbf{X}\mathbf{X}'$