Numerical Linear Algebra
Lecture 8

Nonlinear least squares problems

Suppose that for our data points $(x_i, y_i), i = 1, ..., m$ we want to find the vector of parameters $c = (c_1, ..., c_n)$ which will fit best to the data $y_i, i = 1, ..., m$ of the model function $f(x_i, c), i = 1, ..., m$. We consider the case when the model function $f : R^{n+1} \rightarrow R$ is nonlinear now. Our goal is to find minimum of the residual $r = y - f(x, c)$ in the least squares sense:

$$\min_c \sum_{i=1}^{m} (y_i - f(x_i, c))^2. \tag{1}$$

To solve problem (1) we can still use the linear least squares method if we can transform the nonlinear function $f(x,c)$ to the linear one. This can be done if the function $f(x,c)$ can be represented in the form $f(x,c) = A \exp^{cx}, A = const$. Then taking logarithm of $f(x,c)$ we get: $\ln f = \ln A + cx$, which is already linear function. Then linear least squares problem after this transformation can be written as

$$\min_c \sum_{i=1}^{m} (\ln y_i - \ln f(x_i, c))^2. \tag{2}$$

Another possibility how to deal with nonlinearity is consider the least squares problem as an optimization problem. Let us define the residual $r : R^n \to R^m$ as

$$r_i(c) = y_i - f(x_i, c), \quad i = 1, ..., m. \tag{3}$$

Our goal is now minimize the function

$$F(c) = \frac{1}{2} r(c)^T r(c) = \frac{1}{2} \|r(c)\|_2^2. \tag{4}$$

To find minimum of (4) we should have

$$\nabla F(c) = \frac{\partial F(c)}{\partial c_i} = 0, \quad i = 1, ..., m. \tag{5}$$

Direct computations show that the gradient vector $\nabla F(c)$ is

$$\nabla F(c) = \frac{dF}{dc} = J^T(c) r(c), \tag{6}$$

where J^T is the transposed Jacobian matrix of the residual $r(c)$.

For a sufficiently smooth function $F(c)$ we can write its Taylor expansion as

$$F(c) = F(c_0) + \nabla F(c_0)(c - c_0) + O(h^2), \tag{7}$$

with $|h| = \|c - c_0\|$. Since our goal is to find minimum of $F(c)$, then at a minimum point c^* we should have $\nabla F(c^*) = 0$. Taking derivative with respect to c from (7) we obtain

$$H(F(c_0))(c - c_0) + \underbrace{\nabla F(c_0)}_{compare\ \ with\ (6)} = 0, \tag{8}$$

where H denotes the Hessian matrix of the function $F(c_0)$. Using (6) we also can write

$$\nabla F(c_0) = \frac{dF}{dc}(c_0) = J^T(c_0)r(c_0). \tag{9}$$

Using (9) in (8) we obtain

$$H(F(c_0))(c - c_0) + J^T(c_0)r(c_0) = 0, \tag{10}$$

and from this expression we observe that we have obtained a system of linear equations

$$H(F(c_0))(c - c_0) = -J^T(c_0)r(c_0) \tag{11}$$

which can be solved again using linear least squares method. The Hessian matrix $H(F(c_0))$ can be obtained from (9)

$$\nabla F(c_0) = \frac{dF}{dc}(c_0) = J^T(c_0)r(c_0). \tag{12}$$

as

$$H(F(c_0)) = J^T(c_0)J(c_0) + \sum_{i=1}^{m} r_i(c_0)H(r_i), \tag{13}$$

where $H(r_i)$ denotes the Hessian matrix of the residual function $r_i(c)$. These m matrices $H(r_i)$ are inconvenient to compute, but since they are multiplied to the small residuals $r_i(c_0)$, the second term in (13) is often very small at the solution c_0 and this term can be dropped out.

Then the system (10) is transformed to the following linear system

$$J^T(c_0)J(c_0)(c - c_0) = -J^T(c_0)r(c_0), \tag{14}$$

which actually is a system of normal equations for the $m \times n$ linear least squares problem

$$J(c_0)(c - c_0) = -r(c_0). \tag{15}$$

The system (14) determines the Gauss-Newton method for the solution of the least squares problem as an iterative process

$$c^{k+1} = c^k - [J^T(c_k)J(c_k)]^{-1}J^T(c_k)r(c_k), \tag{16}$$

where k is the number of iteration.

An alternative to the Gauss-Newton method is Levenberg-Marquardt method. This method is based on the finding of minimum of the regularized function

$$F(c) = \frac{1}{2} r(c)^T r(c) + \frac{1}{2}\gamma (c - c_0)^T (c - c_0) = \frac{1}{2}\|r(c)\|_2^2 + \frac{1}{2}\gamma \|c - c_0\|_2^2, \tag{17}$$

where c_0 is a good initial guess for c and γ is a small regularization parameter. Then we repeat all steps which we have performed for the obtaining the Gauss-Newton method, see (6)-(13).

Finally, In the Levenberg-Marquardt method the linear system which should be solved at every iteration k is

$$(J^T(c^k)J(c^k) + \gamma_k I)(c^{k+1} - c^k) = -J^T(c^k)r(c^k), \tag{18}$$

and the corresponding linear least squares problem is

$$\begin{bmatrix} J(c^k) \\ \sqrt{\gamma_k} I \end{bmatrix} \cdot (c^{k+1} - c^k) \approx \begin{bmatrix} -r(c^k) \\ 0 \end{bmatrix}. \tag{19}$$

Example 1

Let us consider the nonlinear model equation

$$A\mathrm{e}^{\mathrm{E/T-T_0}} = \mathrm{y}. \tag{20}$$

Our goal is to determine parameters A, E and T_0 in this equation by knowing y and T. We rewrite (20) as a nonlinear least squares problem in the form

$$\min_{A,E,T_0} \sum_{i=1}^{m} (y_i - A\mathrm{e}^{\mathrm{E/T_i-T_0}})^2. \tag{21}$$

We will show how to obtain from the nonlinear problem (21) the linear one. We take logarithm of (20) to get

$$\ln A + \frac{E}{T - T_0} = \ln y. \tag{22}$$

Now multiply both sides of (22) by $T - T_0$ to obtain:

$$\ln A(T - T_0) + E = \ln y(T - T_0). \tag{23}$$

and rewrite the above equation as

$$T \underbrace{\ln A}_{c_2} \underbrace{- T_0 \ln A + E}_{c_3} + \underbrace{T_0}_{c_1} \ln y = T \ln y. \tag{24}$$

Let now define the vector of parameters $c = (c_1, c_2, c_3)$ with $c_1 = T_0, c_2 = \ln A, c_3 = E - T_0 \ln A$. Now the problem (24) can be written as

$$c_1 \ln y + c_2 T + c_3 = T \ln y, \tag{25}$$

which is already a linear problem. Now we can rewrite (25) denoting by $f(c, y, T) = c_1 \ln y + c_2 T + c_3$ as a linear least squares problem in the form

$$\min_c \sum_{i=1}^{m} (T_i \ln y_i - f(c, y_i, T_i))^2. \tag{26}$$

The system of linear equations which is needed to be solved is

$$\begin{bmatrix} \ln y_1 & T_1 & 1 \\ \ln y_2 & T_2 & 1 \\ \vdots & \vdots & \vdots \\ \ln y_m & T_m & 1 \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} T_1 \ln y_1 \\ T_2 \ln y_2 \\ \vdots \\ T_m \ln y_m \end{bmatrix} \tag{27}$$

Example 2

Suppose that the nonlinear model function is given as

$$f(x,c) = A e^{c_1 x} + \mathrm{B} e^{c_2 x}, \quad \mathrm{A}, \mathrm{B} = \mathrm{const.} > 0, \tag{28}$$

and our goal is to fit this function using Gauss-Newton method. In other words, we will use iterative formula (15) for iterative update of $c = (c_1, c_2)$. The residual function will be

$$r(c) = y - f(x,c) = y - A e^{c_1 x} - \mathrm{B} e^{c_2 x}, \tag{29}$$

where $y = y_i, i = 1, ..., m$ are data points.

First, we compute Jacobian matrix $J(c)$, where two columns in this matrix will be given by

$$\begin{aligned} J(c)_{i,1} &= \frac{\partial r_i}{\partial c_1} = -x_i A \mathrm{e}^{\mathrm{c_1 x_i}}, \quad \mathrm{i} = 1, ..., \mathrm{m}, \\ J(c)_{i,2} &= \frac{\partial r_i}{\partial c_2} = -x_i B \mathrm{e}^{\mathrm{c_2 x_i}}, \quad \mathrm{i} = 1, ..., \mathrm{m}. \end{aligned} \tag{30}$$

If we will take initial guess for the parameters $c^0 = (c_1^0, c_2^0) = (1, 0)$, then we have to solve the following problem at iteration $k = 1$:

$$J(c^0)(c^1 - c^0) = -r(c^0), \tag{31}$$

and the next update for parameters $c^1 = (c_1^1, c_2^1)$ in the Gauss-Newton method can be computed as

$$c^1 = c^0 - [J^T(c_0)J(c_0)]^{-1} J^T(c_0) r(c_0). \tag{32}$$

Here, $r(c^0)$ and $J(c_0)$ can be computed explicitly as follows:

$$r(c^0) = y_i - f(x_i, c^0) = y_i - (A\mathrm{e}^{1\cdot \mathrm{x_i}} + \mathrm{B}\mathrm{e}^{0\cdot \mathrm{x_i}}) = \mathrm{y_i} - \mathrm{A}\mathrm{e}^{\mathrm{x_i}} - \mathrm{B}, \mathrm{i} = 1, ..., \mathrm{m}, \tag{33}$$

and noting that $c^0 = (c_1^0, c_2^0) = (1, 0)$ two columns in the Jacobian matrix $J(c_0)$ will be

$$\begin{aligned} J(c^0)_{i,1} &= -x_i A\mathrm{e}^{1\cdot \mathrm{x_i}} = -\mathrm{x_i}\mathrm{A}\mathrm{e}^{\mathrm{x_i}}, \quad \mathrm{i} = 1, ..., \mathrm{m}, \\ J(c^0)_{i,2} &= -x_i B\mathrm{e}^{0\cdot \mathrm{x_i}} = -\mathrm{x_i}\mathrm{B}, \quad \mathrm{i} = 1, ..., \mathrm{m}. \end{aligned} \tag{34}$$

Substituting (33), (34) into (31) yields following linear system of equations

$$\begin{bmatrix} -x_1 A\mathrm{e}^{\mathrm{x}_1} & -x_1 B \\ -x_2 A\mathrm{e}^{\mathrm{x}_2} & -x_2 B \\ \vdots & \vdots \\ -x_m A\mathrm{e}^{\mathrm{x}_\mathrm{m}} & -x_m B \end{bmatrix} \cdot \begin{bmatrix} c_1^1 - c_1^0 \\ c_2^1 - c_2^0 \end{bmatrix} = - \begin{bmatrix} y_1 - A\mathrm{e}^{\mathrm{x}_1} - \mathrm{B} \\ y_2 - A\mathrm{e}^{\mathrm{x}_2} - \mathrm{B} \\ \vdots \\ y_m - A\mathrm{e}^{\mathrm{x}_\mathrm{m}} - \mathrm{B} \end{bmatrix} \tag{35}$$

which is solved for $c^1 - c^0$ using method of normal equations as

$$\begin{bmatrix} -x_1 A e^{x_1} & -x_1 B \\ -x_2 A e^{x_2} & -x_2 B \\ \vdots & \vdots \\ -x_m A e^{x_m} & -x_m B \end{bmatrix}^T \cdot \begin{bmatrix} -x_1 A e^{x_1} & -x_1 B \\ -x_2 A e^{x_2} & -x_2 B \\ \vdots & \vdots \\ -x_m A e^{x_m} & -x_m B \end{bmatrix} \cdot \begin{bmatrix} c_1^1 - c_1^0 \\ c_2^1 - c_2^0 \end{bmatrix}$$
$$= - \begin{bmatrix} -x_1 A e^{x_1} & -x_1 B \\ -x_2 A e^{x_2} & -x_2 B \\ \vdots & \vdots \\ -x_m A e^{x_m} & -x_m B \end{bmatrix}^T \cdot \begin{bmatrix} y_1 - A e^{x_1} - B \\ y_2 - A e^{x_2} - B \\ \vdots \\ y_m - A e^{x_m} - B \end{bmatrix} \quad (36)$$

This system can be solved for $c^1 - c^0$, and next values c^1 are obtained by using (32).

Outline

- SVD for image compression
- Principal component analysis (PCA) formulation (mean, covariance matrix, computation of eigenvalues and eigenvectors of covariance matrix)
- Principal component analysis (PCA) to find patterns, for data compression and for image orientation

- Principal component analysis (PCA) is a machine learning technique which is widely used for data compression in image processing (data visualization) or in the determination of object orientation.
- PCA problem is closely related to the numerical linear algebra (NLA) problem of finding eigenvalues and eigenvectors for the covariance matrix.
- We will study application of PCA for image compression and object rotation.

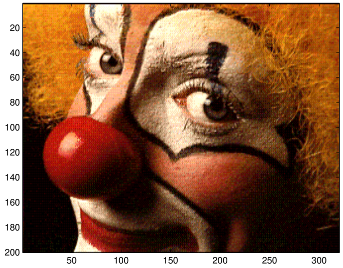
Example of application of linear systems: image compression using SVD

Definition SVD *Let A be an arbitrary m-by-n matrix with $m \geq n$. Then we can write $A = U\Sigma V^T$, where U is m-by-n and satisfies $U^T U = I$, V is n-by-n and satisfies $V^T V = I$, and $\Sigma = diag(\sigma_1, \dots, \sigma_n)$, where $\sigma_1 \geq \dots \geq \sigma_n \geq 0$. The columns $u_1, \dots, u_n$ of U are called* left singular vectors*. The columns $v_1, \dots, v_n$ of V are called* right singular vectors*. The σ_i are called* singular values*. (If $m < n$, the SVD is defined by considering A^T.)*

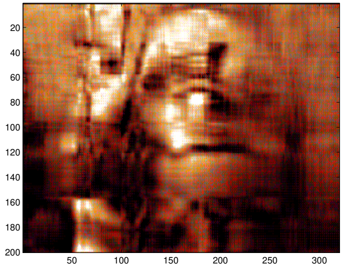
Theorem

Write $V = [\upsilon_1, \upsilon_2, \dots, \upsilon_n]$ and $U = [u_1, u_2, \dots, u_n]$, so $A = U\Sigma V^T = \sum_{i=1}^n \sigma_i u_i \upsilon_i^T$ (a sum of rank-1 matrices). Then a matrix of rank $k < n$ closest to A (measured with $|| \cdot ||_2$ is $A_k = \sum_{i=1}^k \sigma_i u_i \upsilon_i^T$ and $||A - A_k||_2 = \sigma_{k+1}$. We may also write $A_k = U\Sigma_k V^T$ where $\Sigma_k = diag(\sigma_1, \dots, \sigma_k, 0, \dots, 0)$.

Example of application of linear systems: image compression using SVD



a) Original image

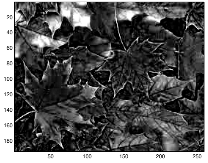


b) Rank k=20 approximation

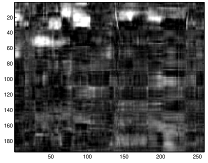
Example of application of linear systems: image compression using SVD in Matlab

See path for other pictures:
/matlab-2012b/toolbox/matlab/demos
load clown.mat;
Size(X) $= m \times n = 320 \times 200$ pixels.
[U,S,V] = svd(X);
colormap(map);
k=20;
image(U(:,1:k)*S(1:k,1:k)*V(:,1:k)');
Now: size(U)= $m \times k$, size(V)= $n \times k$.

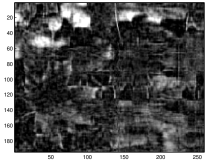
Example of application of linear systems: image compression using SVD in Matlab



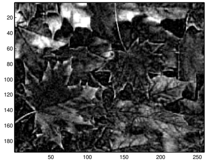
a) Original image



b) Rank k=10 approximation



c) Rank k=20 approximation



d) Rank k=50 approximation

Example of application of linear systems: image compression using SVD for arbitrary image

To get image on the previous slide, I took picture in jpg-format and loaded it in matlab like that:
A = imread('autumn.jpg');
You can not simply apply SVD to A: svd(A) Undefined function 'svd' for input arguments of type 'uint8'.
Apply type "double" to A: DA = double(A), and then perform
[U,S,V] = svd(DA);
colormap('gray');
k=20;
image(U(:,1:k)*S(1:k,1:k)*V(:,1:k)');
Now: size(U)= $m \times k$, size(V)= $n \times k$.

Literature

The proposed books in machine learning:

- Christopher M. Bishop, *Pattern recognition and machine learning*, Springer, 2009.
- Ian Goodfellow, Yoshua Bengio and Aaron Courville, *Deep Learning*, MIT Press, 2016, `http://www.deeplearningbook.org`
- Miroslav Kurbat, *An Introduction to Machine Learning*, Springer, 2017.

PCA formulation

Let $\{x_n\}, n = 1, ..., N$ is a data set of observations, where x_n is a variable of the dimension d. The goal in PCA is to project the data onto a space which has dimension $M < d$ maximizing the variance of the projected data. Let $M = 1$ and thus, we will consider projection onto one-dimensional space. Let the vector u_1 is the direction of this space such that it is a unit vector and $u_1^T u_1 = 1$. Then every point x_n is projected onto a scalar value $u_1^T x_n$. The sample set mean $\bar{x}$ is defined as

$$\bar{x} = \frac{1}{N} \sum_{n=1}^{N} x_n,$$

and the variance of the projected data $u_1^T \bar{x}$ is

$$\frac{1}{N} \sum_{n=1}^{N} (u_1^T x_n - u_1^T \bar{x})^2 = u_1^T S u_1,$$

where S is the data covariance matrix

$$S = \frac{1}{N} \sum (x_n - \bar{x})(x_n - \bar{x})^T.$$

PCA formulation

The next step is maximize the projected variance $u_1^T S u_1$ with respect to u_1 using constrained minimization with the term $u_1^T u_1 = 1$. To enforce this constraint, we use Lagrange multiplier λ to minimize

$$L(u_1) = u_1^T S u_1 + \lambda_1 (1 - u_1^T u_1)$$

Now we minimize $L'(u_1)(\bar{u_1}) = 0$ to get

$$0 = L'(u_1)(\bar{u_1}) = (S u_1 - \lambda_1 u_1)(\bar{u_1}),$$

what means that

$$S u_1 = \lambda_1 u_1 \tag{37}$$

PCA formulation

From the equation (37) follows that (λ_1, u_1) is an eigenpair of S. We observe that

$$u_1^T S u_1 = \lambda_1$$

what means that variance is maximum when we set u_1 to the eigenvector for the largest eigenvalue λ_1 which is called the first principal component. One can obtain all other components in the same way. If we consider M-dimensional projection space then the optimal linear projection for which the variance of the projected data will have maximum, will consists of M eigenvectors $u_1, ..., u_M$ of the covariance matrix S corresponding to the M largest eigenvalues $\lambda_1, ..., \lambda_M$.

PCA: some notions of statistics

Variance is measure of the spread of data in dataspace which is defined as

$$var(X) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{n-1}$$

where mean $\bar{X}$ is defined as

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n}.$$

Standard deviation is defined as

$$s = \sqrt{var(X)} = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{n-1}}$$

PCA: some notions of statistics

Let

X	$(X - \bar{X})$	$(X - \bar{X})^2$
1	-2	4
3	0	0
0	-3	9
5	2	4
6	3	9
Total		26
Divided by $(n-1)$		6.5
Deviation, s		2.5495

Table: *Calculation of variance* $var(X) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{n-1}$. *Here, mean* $\bar{X} = 15/5 = 3$.

PCA: some notions of statistics

Covariance is measure of the spread of data in dataspace which is defined as

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{n-1}$$

Let

X	Y	$(X-\bar{X})$	$(Y-\bar{Y})$	$(X-\bar{X})(Y-\bar{Y})$
1	9	-2	5.4	-10.8
3	1	0	-2.6	0
0	3	-3	-0.6	1.8
5	5	2	1.4	2.8
6	0	3	-3.6	-10.8
Total				-17
Divided by $(n-1)$				-4.25

Table: *Calculation of covariance. Here,* $\bar{X} = 3$, $\bar{Y} = 3.6$.

PCA: some notions of statistics

Below is definition of the covariance matrix for a set of data which have dimension n:

$$C = \begin{bmatrix} cov(Dim_1, Dim_1) & \dots & cov(Dim_1, Dim_n) \\ cov(Dim_2, Dim_1) & \dots & cov(DIm_2, Dim_n) \\ \dots & \dots & \dots \\ cov(Dim_n, Dim_1) & \dots & cov(Dim_n, Dim_n) \end{bmatrix}$$

Here, $dim(C) = n \times n$.
Covariance is always measured between 2 dimensions. For three-dimensional data sets x,y,z one can compute covariance matrix

$$C = \begin{bmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{bmatrix}$$

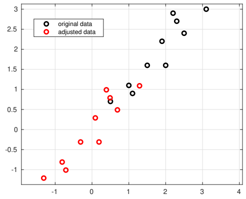
PCA

- PCA can be thought as a way to find patterns in data in order to highlight similarity or difference of data. Thus, it is a powerful tool to analyze data.
- The main advantage of PCA is as soon as patterns in data is found, one can compress the data by reducing number of dimensions without much loss of information. This technique is used in image compression.

PCA: example

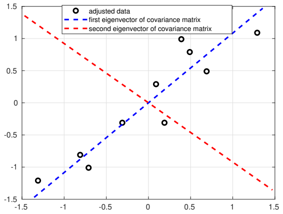
Let us analyze how works PCA on two datasets using Matlab program ExamplePCA.m.

```
  %first create our 2D data
x=[2.5,0.5,2.2,1.9,3.1,2.3,2,1,1.5,1.1];
y=[2.4,0.7,2.9,2.2,3.0,2.7,1.6,1.1,1.6,0.9];
%  compute mean for x
[ax,bx] =size(x);
mean_x= sum(x)/bx;
%  compute mean for y
[ay,by] =size(y);
mean_y = sum(y)/by;
%  compute  adjusted data
adjust_x = x - mean_x;
adjust_y = y - mean_y;
plot(x,y,'o','LineWidth',2,'MarkerEdgeColor','k');  grid('on'); hold on;
 plot(adjust_x,adjust_y,'o','LineWidth',2,'MarkerEdgeColor','r');
legend('original data', 'adjusted data');
```



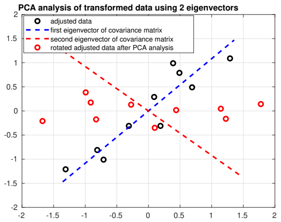
PCA: example

```
  %compute covariance matrix
  C = cov(adjust_x,adjust_y);
% compute eigenvalues and eigenvectors ov the covariance matrix
[eigvec, eigval] = eig(C);
[U,S,V] = svd(C);
% plotting using svd of the covariance matrix ********************************
% first eigenvector via svd: this is the principle component of the data set
% since it's corresponds to the largest eigenvalue.
% We can call it also as feature vector.
u1x=[2*U(1,1),-2*U(1,1)];
u1y=[2*U(2,1),-2*U(2,1)];
plot(u1x,u1y,'-- b', 'LineWidth',2);
% plotting the second eigenvalue vector
u2x=[2*U(2,1),-2*U(2,1)];
u2y=[2*U(2,2),-2*U(2,2)];
plot(u2x,u2y,'-- r', 'LineWidth',2);
```



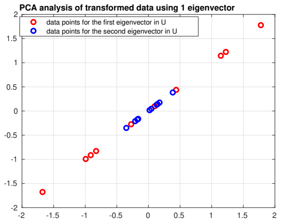
PCA: example

```
  % Now form final rotated data
Data =[adjust_x; adjust_y];
PCA = U*Data;
plot(PCA(1,:),PCA(2,:),'o','LineWidth',2, 'MarkerEdgeColor','r');
axis([-2,2,-2,2])
legend('adjusted data','first eigenvector of covariance matrix','second eigenvector of covariance matrix',
title('PCA analysis of transformed data using 2 eigenvectors')
```



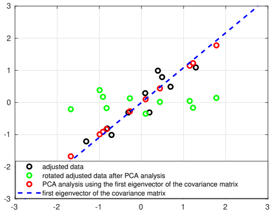
PCA: example

```
% now we plot out vector with the largest eigenvalue
PCAnew1 = U(1,:)*Data;
plot(PCAnew1,PCAnew1,'o','LineWidth',2, 'MarkerEdgeColor','r');
hold on
PCAnew2 = U(2,:)*Data;
plot(PCAnew2,PCAnew2,'o','LineWidth',2, 'MarkerEdgeColor','b');
  grid('on');
legend('data points for the first eigenvector in U','data points for the second eigenvector in U')
title('PCA analysis of transformed data using 1 eigenvector ')
```



PCA: example

```
    plot(adjust_x,adjust_y,'o','LineWidth',2,'MarkerEdgeColor','k');
     grid('on');
hold on
plot(PCA(1,:),PCA(2,:),'o','LineWidth',2, 'MarkerEdgeColor','g');
plot(PCAnew1,PCAnew1,'o','LineWidth',2, 'MarkerEdgeColor','r');
u1x=[5*U(1,1),-5*U(1,1)];
u1y=[5*U(2,1),-5*U(2,1)];
plot(u1x,u1y,'-- b', 'LineWidth',2);
axis([-3,3,-3,3])
legend('adjusted data','rotated adjusted data after PCA analysis',...
'PCA analysis using the first eigenvector of the covariance matrix',...
'first eigenvector of the covariance matrix');
```



Computer exercise 4

Use PCA to find patterns (recognize handwritten numbers) in MNIST Dataset of Handwitten Digits which can be downloaded from the course homepage.

- Use Matlab programs

  ```
  loadmnist_matlab.m
  import_mnist.m
  ```

  on the course homepage to download MNIST Datasets

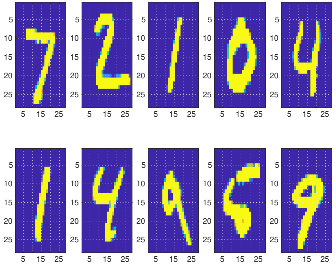
  ```
  mnist_test_10.csv
  mnist_train.csv
  ```

- Perform PCA analysis for the following problem: given an image from the dataset

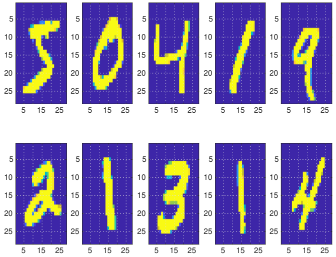
  ```
  mnist_test_10.csv
  ```

  check if there exists the same or similar image in the train dataset

MNIST datasets

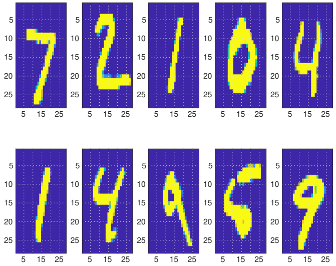


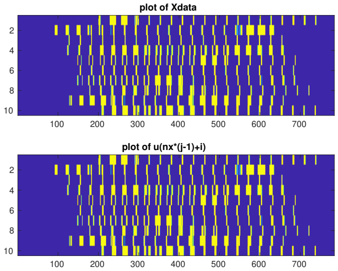
a) Dataset in *mnist_test_10.csv*



b) Images from *mnist_train.csv*

PCA to find patterns for MNIST dataset


a) Dataset in *mnist_test_10.csv*



b) Matrix of images

Dataset

```
mnist_test_10.csv
```

contains 10 images presented in Figure a). This is a test dataset which should be used to recognise handwritten digits in the dataset

```
mnist_train.csv
```

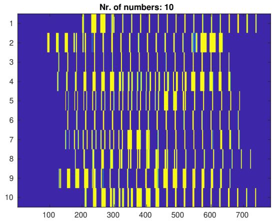
PCA to find patterns for MNIST dataset

Top image of Figure b) represents matrix array Xdata from the program plotmnist.m of all 10 images presented in Figure a). Bottom image of Figure b) is the same array which is read in the another array as

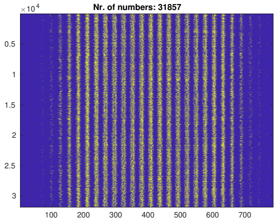
```
[nx,ny] = size(Xdata)
u =   zeros(nx,ny);

for j=1:ny
for i=1:nx
u(nx*(j-1)+i) = Xdata(nx*(j-1)+i);
end
end
```

Every row of this array is a vector of the size $nx \times ny$, where nx is the number of nodes in x direction and ny is the number of nodes in y direction of every image presented in Figure a). In our case, $nx = 28$, $ny = 28$, and this $28 \cdot 28 = 784$ is size of every row of Figure b) for 10 rows. We have 10 rows and 784 columns for matrix of test images.

PCA to find patterns for MNIST dataset



a) Matrix of test images



b) Matrix of train images

We need to perform PCA analysis of matrix of train images and test images. After PCA we will have original data on terms of eigenvectors and eigenvalues of the covariance matrix.
The next step is to measure difference between the new image and the original image, but not along the original axes, but along the new axes which are obtained in PCA analysis.
It was shown [1] that these new axes gives better information for the case of image recognition since the PCA analysis gives the original image in terms of the differences and similarities between data.

[1] J. ZHANG, Y. YAN, M. LADES, Face Recognition: Eigenface, Elastic Matching, and Neural Nets,

Proc. IEEE, 1997.