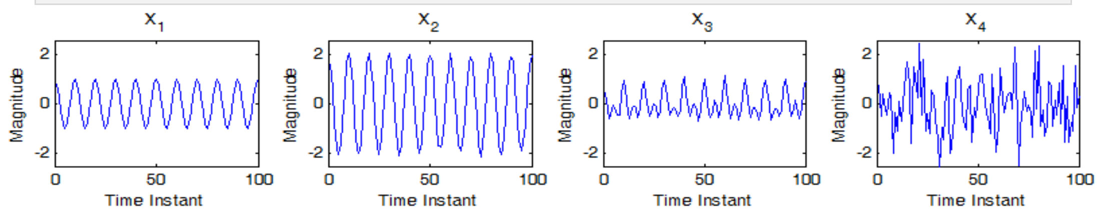
Handout 3: MATLAB Implementation of Multivariate Signal Processing and Pattern Recognition

1. Covariance, Correlation, and Coherence

MATLAB has functions `cov`, `corr`, and `partialcorr` for calculating covariance matrix, correlation matrix, and partial correlation matrix of multivariate signals. Importantly, `corr` and `partialcorr` can compute different types of correlation (Pearson's linear relation coefficient, Kendall's tau, and Spearman's rho) and output p-values testing the hypothesis of no correlation or no partial correlation. These functions are easy to use, but it is important to note that the dimension of input matrix is $N \times P$, where N is the number of samples/observations and P is the number of features/variables.

<u>Example:</u> Compute the covariance, correlation, and partial correlation of the following multivariate signals:

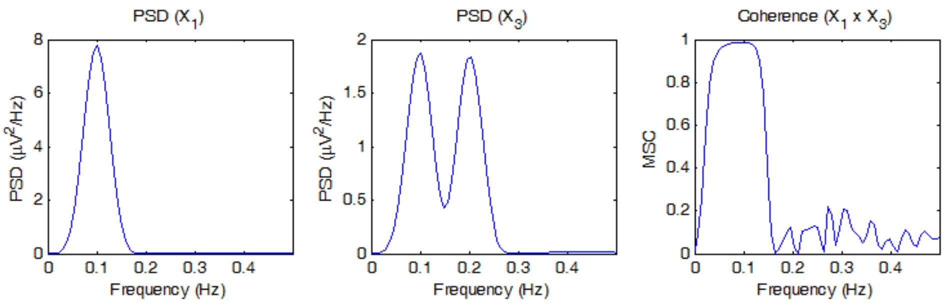
```
N = 100; % the number of samples/observations
P = 4; % the number of features/variables
X = zeros(N,P); % the dimension of the input matrix is NxP
X(:,1) = cos(2*pi*0.1*[1:N]');
X(:,2) = 2*X(:,1) + 0.1*randn(N,1);
X(:,3) = 0.5*cos(2*pi*0.1*[1:N]') + 0.5*cos(2*pi*0.2*[1:N]')+
0.1*randn(N,1);
X(:,4) = randn(N,1);
```



```
%% Compute covariance, correlation, and partial correlation
C_cov = cov(X); % covariance
C_corr = corr(X); % correlation (Pearson's)
C_partialcorr = partialcorr(X); % partial correlation (Pearson's)
```

The PSDs of the 1st and 3rd signals and their coherence can be estimated as follows.

```
%% Compute PSDs (Welch’s method) and coherence
fs = 1; % sampling rate
nfft = 2^nextpow2(N); % number of FFT points
[P1, f] = pwelch(X(:,1),[],[],nfft,fs); % Welch's method
[P3, f] = pwelch(X(:,3),[],[],nfft,fs); % Welch's method
[COH,f] = mscohere(X(:,1),X(:,3),[],[],nfft,fs); % Magnitude
squared coherence
```



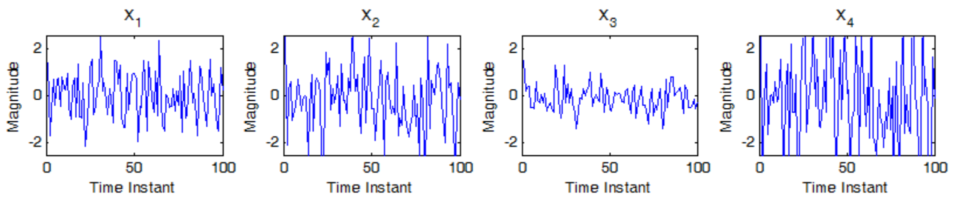
Please refer to the MATLAB help for more details about the functions used in this example.

2. Principle Component Analysis

Principle Component Analysis (PCA) can be implemented using MATLAB function `princomp`. A simple example is given below. Please refer to the MATLAB help for more details about the functions used in this example.

Example: Use PCA to decompose the following multivariate signals, which are generated as linear combinations of two latent components (sources).

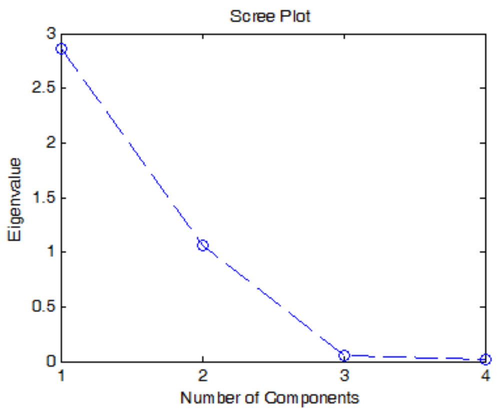
```
N = 100; % the number of samples/observations
P = 4; % the number of features/variables
X = zeros(N,P); % the dimension of the input matrix is NxP
S1 = randn(N,1); % latent component 1
S2 = randn(N,1); % latent component 2
X(:,1) = S1 + 0.2*randn(N,1);
X(:,2) = S1 + S2 + 0.2*randn(N,1);
X(:,3) = 0.5*S2 + 0.2*randn(N,1);
X(:,4) = 2*S1 + S2 + 0.2*randn(N,1);
```



PCA decomposition is implemented as follows.

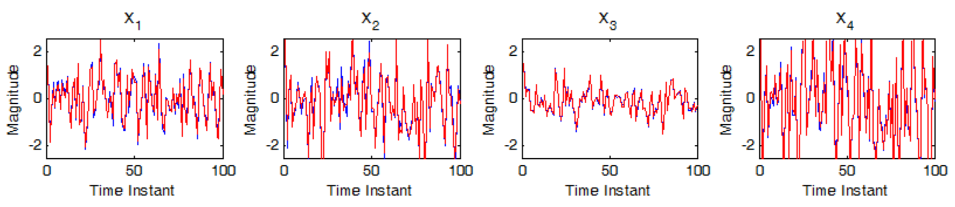
```
%% PCA Decompositon
[Z, mu, sigma] = zscore(X); % standardize the input data to make
it have zero mean and unit variance for each feature/variable
[COEFF,SCORE,latent] = princomp(Z);
% COEFF: a P x P matrix, each column containing coefficients for
one principal component
% SCORE: a N x P matrix, each column being a principal component
% latent: a Px1 vector containing the eigenvalues of the covariance
matrix of X

% Scree plot
figure
plot(latent,'o--')
xlabel('Number of Components')
ylabel('Eigenvalue')
title('Scree Plot')
```

Note that the function `zscore` is used to standardize the input data prior to PCA. Please refer to the MATLAB help for more details about `zscore` and `princomp`.



We can further use two principal components to reconstruct signals.

```
%% Reconstruction
P_new = 2; % the new reduced dimension (P_new < P)
Z_rec = (SCORE(:,1:P_new)*COEFF(:,1:P_new)'); % reconstruct
signals using P_new principal components
X_rec = Z_rec .* (ones(N,1)*sigma) + (ones(N,1)*mu);% rescale the
reconstructed data to the original scale
```



In above graph, blue curves are original signals and red curves are reconstructed signals from two principal components. It can be seen that the signals can be well reconstructed by only using two principal components.

3. Independent Component Analysis

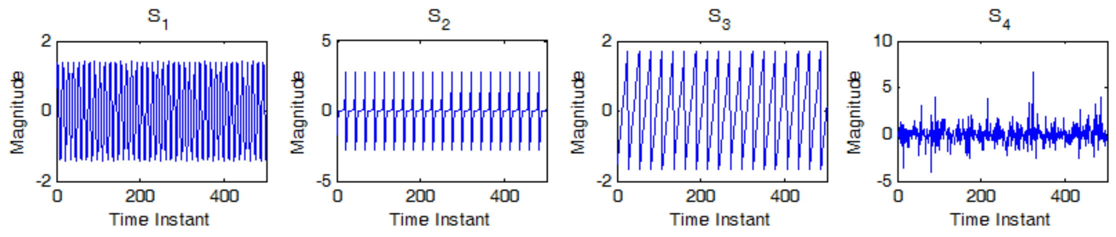
MATLAB does not provide any function to perform independent component analysis (ICA), but there are many well designed MATLAB-based toolkits for ICA. Here, we will use the FastICA toolbox (http://research.ics.aalto.fi/ica/fastica/) to demonstrate the effectiveness of ICA.

Example: The multivariate signals are generated by a linear mixing of four independent sources (one source is the impulsive noise). FastICA can identify four sources from the mixed signals.

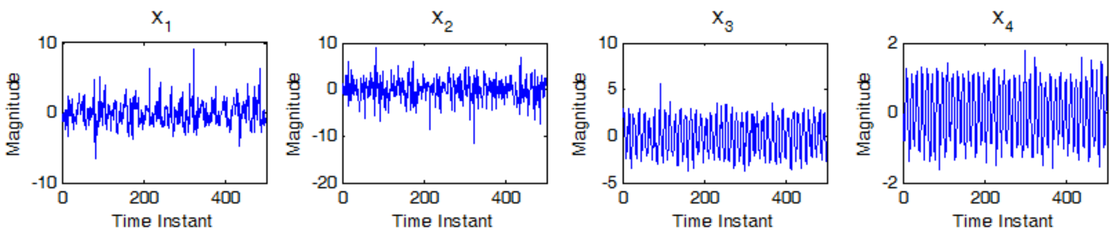
```
N = 500; % the number of samples
P = 4; % the number of sources
S = zeros(P,N); % P sources, each having N time samples
S(1,:)=sin([1:N]/2); % sinusoid
S(2,:)=((rem([1:N],23)-11)/9).^5; % funny curve
S(3,:)=((rem([1:N],27)-13)/9); % saw-tooth
S(4,:)=((rand(1,N)<.5)*2-1).*log(rand(1,N)); % impulsive noise
A = randn(P); % weights
X = A*S; % mixed multivariate signals (P x N)

%% FastICA
[S_est, A_est, W_est] = fastica(X); % S_est is the estimated
independent components and A_est is the estimated mixing matrix
```

Please pay attention to the dimension of the input data matrix of `fastica`, which is different from that of PCA (`princomp`). But it does not mean that PCA and ICA operate on different types of data. The difference is only due to different styles and habits of programmers. Also, please use a tailored `fastica` function in the package of assignment 3 to complete your assignment.

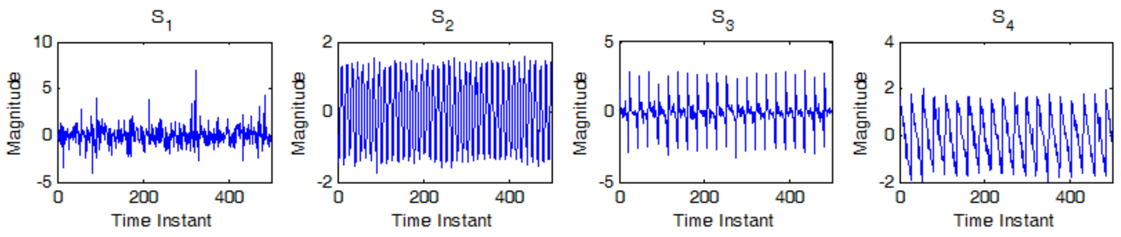
The original sources are shown below.



The mixed signals are shown below.



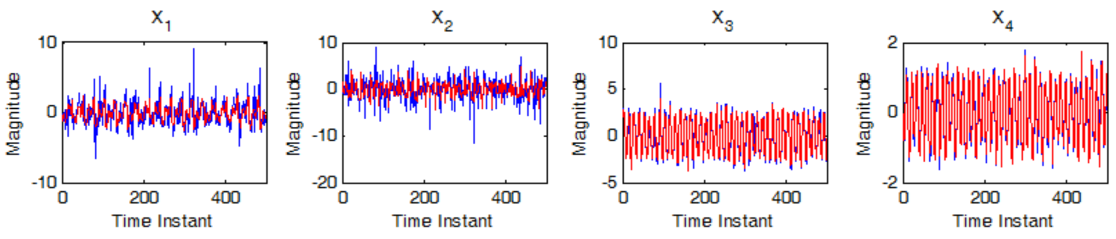
The sources estimated by FastICA are shown below.



We can see from the graph above that the 1st independent component is a noise component. So we can remove this component and then reconstruct the signal. Note that the sequence of the independent components may vary case by case (even in different implementations of the exactly same signal).

```
%% Reconstruction
X_rec = A_est(:,[2,3,4])*S_est([2,3,4],:);
```

The red curves in the graph below are the reconstructed signal (with the 1st independent component removed) and the blue curves are the original signal. We can see that the impulsive noise is satisfactorily removed (especially for the 1st and 2nd channels, which are seriously corrupted with the impulsive nosie).

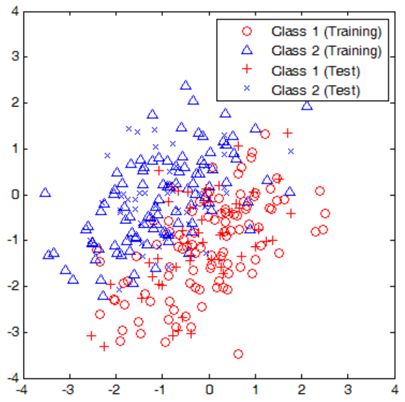


3. Pattern Recognition

The MATLAB Statistics Toolbox has a function `classify` to implement several types of classifiers. A simple example is given below.

Example: Two classes of samples are generated from two 2D Gaussian distributions.

```
% Set parameters
P = 2; % number of features
N_train = 100; % number of training samples
N_test = 50; % number of test samples

% Define the two (P) classes (two 2D normal distributions)
mu1 = [0; -1]; % mean of data1
sigma1 = [1, .5; .5, 1]; % covariance of data1
mu2 = [-1; 0]; % mean of data2
sigma2 = [1, .5; .5, 1]; % covariance of data2

% Generate training samples from the two distributions
train_data1 = mvnrnd(mu1,sigma1,N_train);
train_data2 = mvnrnd(mu2,sigma2,N_train);
% Generate test samples from the two distributions
test_data1 = mvnrnd(mu1,sigma1,N_test);
test_data2 = mvnrnd(mu2,sigma2,N_test);

% Plot the samples
figure
hold on; box on;
plot(train_data1(:,1), train_data1(:,2), 'ro');
plot(train_data2(:,1), train_data2(:,2), 'bo');
plot(test_data1(:,1), test_data1(:,2), 'r+');
plot(test_data2(:,1), test_data2(:,2), 'b+');
legend({'Class 1 (Training)','Class 2 (Training)', 'Class 1
(Test)','Class 2 (Test)'})
set(gca,'xlim',[-4 4],'ylim',[-4 4])
```



```
% Concatenate training/test data and specify the labels
train_samples = [train_data1;train_data2]; %  training samples
train_labels = [ones(N_train,1);zeros(N_train,1)]; % labels of
training samples: 1 for data1; 0 for data2
test_samples = [test_data1;test_data2]; % test samples
test_labels = [ones(N_test,1);zeros(N_test,1)]; % labels of test
samples: 1 for data1; 0 for data2
% LDA
classout =
classify(test_samples,train_samples,train_labels,'linear');
% Compute accuracy
acc = sum(classout==test_labels)/(N_test*2);
% Compute TP/TN/FP/FN and sensitivity/specificity
% label 0: positive; label 1: negative (case-dependent)
TP = sum((classout==test_labels)&(classout==0));
TN = sum((classout==test_labels)&(classout==1));
FP = sum((classout~=test_labels)&(classout==0));
FN = sum((classout~=test_labels)&(classout==1));
sensitivity = TP/(TP+FN);
specificity = TN/(TN+FP);
```

Next, an example of a 5-fold cross-validation (CV) on training samples only is provided. More precisely (as seen from the first two lines in the following code), available data for CV in this example are training data used in above examples, and these available data will be split into training samples and test samples in each round of CV.

```
%% Cross Validation
all_samples = train_samples; % all available data
all_labels = train_labels; % labels of all available data
K = 5; % K-fold CV
```

```
indices = crossvalind('Kfold',all_labels,K); % generate random
indices for CV
for k = 1:K % K iterations
    cv_test_idx = find(indices == k); % indices for test samples in
    one trial of validation
    cv_train_idx = find(indices ~= k); % indices for training samples
    in one trial of validation
    cv_classout =
    classify(all_samples(cv_test_idx,:),all_samples(cv_train_idx,:
    ),all_labels(cv_train_idx,:)); % classification
    cv_acc(k) =
    sum(cv_classout==all_labels(cv_test_idx,:))/(numel(cv_classout
    )); % compute classification accuracy
end
cv_acc_avg = mean(cv_acc); % averaged accuracy
```

Please refer to the MATLAB help for more details about the functions used in these examples of classification.