Structural Health Monitoring Tools (SHMTools)

Example Usages

LANL/UCSD Engineering Institute

LA-CC-14-046
LA-UR-14-21093

April 1, 2019

Example Usages

Contents

Data Set Descriptions

Experimental Procedure Description of the 3-Story Structure

Experimental Procedure Description of the Condition-Based Monitoring Example Data

Integrating Examples

Example using DAQ plus AR Parameters plus Mahalanobis Distance

Outlier Detection based on a Chi-square Distribution for the Undamaged Condition

Damage Location using AR Parameters from an Array of Sensors

Damage Location using ARX Parameters from an Array of Sensors

Appropriate Autoregressive Model Order

Modal Analysis

Optimal Sensor Placement Using Modal Analysis Based Approaches

Data Normalization for Outlier Detection using Modal Properties

Condition-Based Monitoring

Ball Bearing Fault Analysis

Gearbox Fault Analysis

Active Sensing

Sensor Diagnostics

National Instruments Ultrasonic Active Sensing DAQ

Active Sensing Feature Extraction Example

Outlier Detection

Assembling a Custom Detector

How to Use the Default Detectors

Parametric Detectors:

Outlier Detection based on the Nonlinear Principal Component Analysis

Outlier Detection based on the Factor Analysis Model

Outlier Detection based on Principal Component Analysis

Outlier Detection based on the Singular Value Decomposition

Outlier Detection based on the Mahalanobis Distance

Semi-Parametric Detectors:

Direct Use of Semi-Parametric Routines

Non-Parametric Detectors:

Direct Use of Non-Parametric Routines

Fast Metric Kernel Density Estimation

Published with MATLAB® R2013a

Base-Excited 3-Story Structure Data Sets

Contents

Structure Description

The three-story building structure shown with its basic dimensions in Figure 1 is used as a damage detection test bed structure.

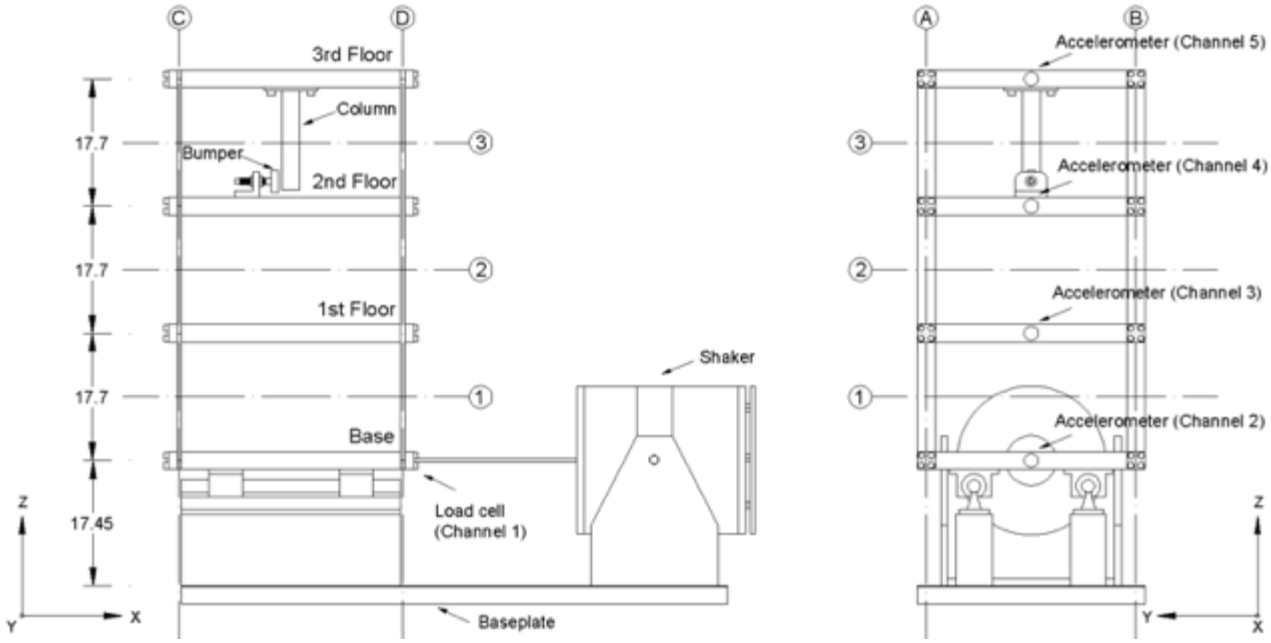


Figure 1

The structure consists of aluminum columns and plates assembled using bolted joints. The structure can slide on rails that allow movement in the x-direction only. At each floor, four aluminum columns (17.7x2.5x0.6 cm) are connected to the top and bottom aluminum plates (30.5x30.5x2.5 cm) forming a (essentially) four degree-of-freedom system. Additionally, a center column (15.0x2.5x2.5 cm) is suspended from the top floor. This column can be used to simulate damage by inducing nonlinear behavior when it contacts a bumper mounted on the next floor. The position of the bumper can be adjusted to vary the extent of impacting that occurs at a particular excitation level. In the context of SHM, this source of damage is intended to simulate fatigue cracks that can open and close or loose connections that can rattle under dynamic loading.

Data Acquisition System

An electrodynamic shaker provides a lateral excitation to the base floor along the centerline of the structure. The structure and shaker are mounted together on an aluminum baseplate (76.2x30.5x2.5 cm) and the entire system rests on rigid foam. The foam is intended to minimize extraneous sources of unmeasured excitation from being introduced through the base of the system. A load cell (Channel 1) with a nominal sensitivity of 2.2 mV/N was attached at the end of a stinger to measure the input force from the shaker to the structure. Four accelerometers (Channel 2-5) with nominal sensitivities of 1000 mV/g were attached at the centerline of each floor on the opposite side from the excitation source to measure the system response. Because the accelerometers are mounted at the centerline of each floor, they are insensitive to torsional modes of the structure. In addition, the shaker location and the linear bearings minimize the torsional excitation of the system.

A National Instruments PXI data acquisition system was used to collect and process the data. Analog output waveforms were generated using a PXI-4461 DAQ module and the response signals from the five sensors were acquired using a PXI-4472B DAQ module. ICP conditioning to the five sensor channels was provided by a PCB 482A16 signal conditioner. The analog output channel of the system, which provides the drive signal to the shaker, is input to a Techron 5530 Power Supply Amplifier that drives the shaker. The analog sensor signals were digitized at a rate of 2560 Hz and acquired in blocks of 65536 points. However, then the data were downsampled into 8192 data points at 3.125 ms intervals corresponding to a sampling frequency of 320 Hz. These sampling parameters yield time series 25.6 s in duration. A band-limited random excitation in the range of 20-150 Hz was used to excite the structure. This excitation signal was chosen in order to avoid the rigid body modes of the structure that are present below 20 Hz. The excitation level was set to 2.6 V RMS in the National Instruments system.

Data Sets Description

The structural state conditions can be categorized into four main groups, as shown in Table I. Note that for each state condition where performed 50 tests, yielding 50 time histories per channel for each state condition.

Table I

Label	State condition	Description
State#1	Undamaged	Baseline condition
State#2	Undamaged	Mass = 1.2 kg at the base
State#3	Undamaged	Mass = 1.2 kg on the 1st floor
State#4	Undamaged	87.5% stiffness reduction in column 1BD
State#5	Undamaged	87.5% stiffness reduction in column 1AD and 1BD
State#6	Undamaged	87.5% stiffness reduction in column 2BD
State#7	Undamaged	87.5% stiffness reduction in column 2AD and 2BD
State#8	Undamaged	87.5% stiffness reduction in column 3BD
State#9	Undamaged	87.5% stiffness reduction in column 3AD and 3BD
State#10	Damaged	Gap = 0.20 mm
State#11	Damaged	Gap = 0.15 mm
State#12	Damaged	Gap = 0.13 mm
State#13	Damaged	Gap = 0.10 mm
State#14	Damaged	Gap = 0.05 mm
State#15	Damaged	Gap = 0.20 mm and mass = 1.2 kg at the base
State#16	Damaged	Gap = 0.20 mm and mass = 1.2 kg on the 1st floor
State#17	Damaged	Gap = 0.10 mm and mass = 1.2 kg on the 1st floor

The first group is the baseline condition. The baseline condition is the reference structural state and is labeled State#1 in Table I. The bumper and the suspended column are included in the baseline condition, but the spacing between the bumper and the column was maintained in such a way that there were no impacts during the excitation. The second group includes the states with simulated operational and environmental variability. Such variability often manifests itself in changes in the stiffness or mass distribution of the structure, in order to simulate such operational and environmental condition changes, tests were performed with different mass-loading and stiffness conditions (State#2-9). The mass changes consisted of adding a 1.2 kg (approximately 19% of the total mass of each floor) to the first floor and to the base, as shown in Figure 2 when the mass is at the base. The stiffness changes were introduced by reducing the stiffness of one

or more of the columns by 87.5%. This process was executed by replacing the corresponded column with one that had half the cross sectional thickness in the direction of shaking.

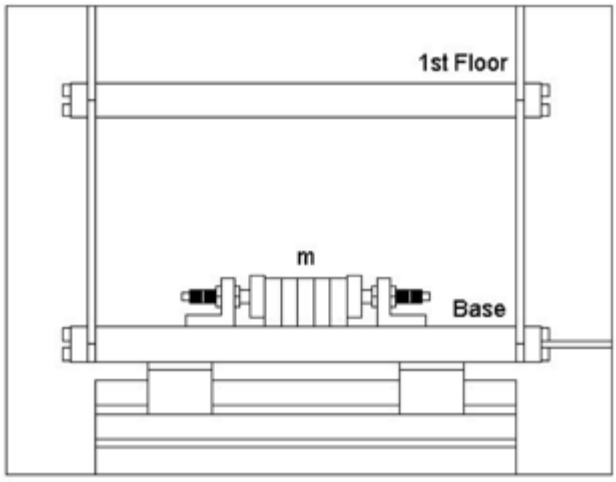


Figure 2

The third group includes the damaged state conditions; these were simulated by the introduction of nonlinearities into the structure using the bumper and the suspended column with different gaps between them, as shown in Figure 3. The gap between the bumper and the suspended column was varied (0.20, 0.15, 0.13, 0.10, and 0.05 mm) in order to introduce different levels of nonlinearity (State#10-14).

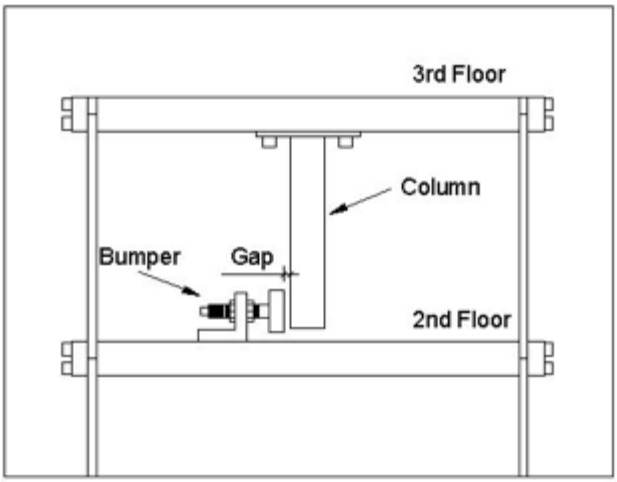


Figure 3

Finally, to create more realistic conditions, the fourth group includes state conditions with the simulated damage in addition to the mass and stiffness changes used to introduce simulated operational and environmental variation (State#15-17). The changes in mass and stiffness are such that the variations in the first natural frequencies are in the range of +/- 5 Hz for the state conditions with operational and environmental variations.

More details about the test structure as well as damaged scenarios can be found in Figueiredo et al. (2009).

File Descriptions

data3SS.mat - Each structural state condition is composed of 10 tests. The tests are stored in concatenated format. For instance, dataset(1:8192,1:5,1:10) corresponds to the data sets from Channel 1-5 of State #1 and dataset(:,1:5,91:100) corresponds to the data sets of State #10. Each time series has 8192 data points.

data3SS2009.mat - Each structural state condition is composed of 50 tests. Expands data3ss.mat. This dataset is available in the additional datasets download.

Information

References:

Figueiredo, E., Park, G., Figueiras, J., Farrar, C., & Worden, K. (2009). Structural Health Monitoring Algorithm Comparisons using Standard Data Sets. Los Alamos National Laboratory Report: LA-14393.

http://institute.lanl.gov/ei/software-and-data/

Author: Eloi Figueiredo

Date Created: September 01, 2009

Conditioned-Based Monitoring Example Data Set for Rotating Machinery

Contents

Structure Description

The example CBM data was collected via the SpectraQuest. Inc. Magnum Machinery Fault Simulator. The fault simulator can produce many faults but the damage states in the data below includes bearing and gearbox damage. The test bed consists of a main shaft driven by an electric motor which has a single rotation tach signal that reads the main shaft rotation in these experiments. The main shaft was set up to be supported by two ball bearings. The shaft is a 3/4 inch diameter steel shaft with a support length of 28.5 inches, center to center between bearings. Two aluminum masses were supported on the main shaft, one 8.5 inches to the right from the motor side bearing and the other 6.5 inches from the belt drive side bearing. A belt drive was used to drive a gearbox (description below) via belt drive. The belt drive consists of two double groove belt sheaves with a sheave ratio of ~1:3.71 where the smaller sheave was attached to the main shaft and the larger sheave to the gearbox shaft. The belt span is 13 inches and the tension was kept near the recommended value of 3.7 lbs. of force for a deflection to span ratio of 1:64. A magnetic break applied a torsional force to the pinion shaft of the gear box of approximately 1.9 lbs.-in.

Gearbox Data: Ratio: 1.5:1, Model: Hub City M2, Pitch Angle Gear: 56 degrees 19 minutes, Pitch Angle Pinion: 33 degrees 41 minutes, Pressure Angle for Gear and Pinion: 20 degrees, Material: Forged Steel, Backlash Tolerance: 0.001-0.005 inches, Pitch Diameter Pinion: 1.125 inches, Pitch Diameter Gear: 1.6875 inches. Number of teeth (Pinion): 18 Number of Teeth Gear: 27, Pinion Bearing: NSK 6202 (1 Bearing), Gear Bearing: (2 Bearings)

Main Shaft Ball Bearing Data: Manufacturer: MB Mfg., Model: ER-12K, Number of Rolling Elements: 8, Rolling Element Diameter: 1.318, Bearing Fault Frequencies are: Fundamental Train Frequency (Cage Speed): 3.048*Shaft Frequency Ballpass Frequency (Outer Race): 3.048*Shaft Frequency Ballpass Frequency (Inner Race: 4.95*Shaft Frequency Ball (Roller) Spin Frequency: 1.992*Shaft Frequency

Data Acquisition System

Channel 1: Tachyometer Channel 2: Accel. Mounted on Gearbox Channel 3: Accel. Mounted on Top of Bearing Housing Channel 4: Accel. Mounted on Side of Bearing Housing

Data Sets Description

Each instance contains 5 seconds of data sampled at 2048 Hz from 4 channels. Shaft speed is nominally constant at ~1000 rpm.

State 1: Baseline condition 1: Tachometer located on main shaft. Ball bearings supporting main shaft in healthy condition. Gearbox driven by belt drive. Gearbox in healthy condition.

State 2: Baseline condition 2: Tachometer located on main shaft. Ball bearings supporting main shaft in healthy condition. Gearbox driven by belt drive. Gearbox in healthy condition.

State 3: Main Shaft Ball (Roller) Spin Fault: Tachometer located on main shaft. Ball bearings supporting main shaft & have roller element fault. Gearbox in healthy condition.

State 4: Baseline condition 1: Tachometer located on main shaft. Fluid bearings supporting main shaft in healthy condition. Gearbox driven by belt drive. Gearbox in healthy condition.

State 5: Baseline condition 2: Tachometer located on main shaft. Fluid bearings supporting main shaft in healthy condition. Gearbox driven by belt drive. Gearbox in healthy condition.

State 6: Gear Box Worn tooth Fault: Tachometer located on main shaft. Fluid bearings supporting main shaft in healthy condition. Gearbox driven by belt drive. Gearbox suffers from worn tooth damage.

File Descriptions

data_CBM - Each structural state condition is composed of 64 tests. The tests are stored in concatenated format. For instance, dataset(1:10240,1,1:64) corresponds to the data sets from Channel 1 (tachyometer) of State #1.

Information

Author: Luke Robinson

Date Created: July 23, 2013

Published with MATLAB® R2013a

DAQ plus AR Parameters plus Mahalanobis Distance

Contents

Introduction

This example demonstrates how to sequence together a full structural health monitoring process. The Los Alamos National Laboratory Engineering Institute's 3-Story Test Structure is used as the test bed in conjunction with National Instruments data acquisition hardware.

The process is broken into two major stages: training and live testing. In the training, a set of data is acquired and processed in order to learn a model of the undamaged system. In live testing, one at a time, small blocks of data are acquired, processed, and classified as either "undamaged" or "damaged" based on the learned model and statistically derived thresholding.

The example starts by collecting a series of five second blocks of acceleration histories from the undamaged structure as it is being excited with band-limited white noise. It then fits a linear autoregressive model to each of the histories. The AR model coefficients will then serve as the feature vectors. The example then trains a Mahalanobis distance-based detector by learning the mean and covariance matrix of the feature vectors from the undamaged structure. It then calculates a threshold based on a 99% confidence.

The example then runs several test cases one at a time with the structure in different undamaged and damaged states. For each case, the example acquires a time series block, extracts the AR-based feature vector, calculates the Mahalanobis distance, and thresholds the Mahalanobis distance to determine if the structural state is either damaged or undamaged.

Requires data_simDAQ.mat dataset if in simulation mode.

References:

Figueiredo, E., Park, G., Figueiras, J., Farrar, C., & Worden, K. (2009). Structural Health Monitoring Algorithm Comparisons using Standard Data Sets. Los Alamos National Laboratory Report: LA-14393.

SHMTools functions called:

bandLimWhiteNoise_shm exciteAndAquire_shm arModelOrder_shm arModel_shm learnMahalanobis_shm scoreMahalanobis_shm plotDetectResults_shm

Author: Eric Flynn

Date Created: August 19, 2009

Flag if simulating DAQ with stored data

```
isSimulation=true;

if isSimulation
    load('data_simDAQ.mat','simTrainingData','simTestData');
end
```

Assign Data Acquisition Setting

Set up five channels on the National Instruments device, acquire at a sample rate of 320 Hz for a duration of 4096 samples.

```
adaptor='nidaq';
aiID='Dev5';
aiChans=0:4;
aoID='PXI1slot2';
sampleRate=320;
aiNumSamples=1600;
```

START OF TRAINING

Construct Excitation Waveforms

Generate the arrays of time series data for the excitation signal. The signals will be blocks of band-limited white noise from 20 to 150 Hz.

```
runCount=50;
arraySize=[aiNumSamples,runCount];
cutoffs=[20 150]/sampleRate*2;
rms=2.6;

exciteWaveform=bandLimWhiteNoise_shm(arraySize,cutoffs,rms);
```

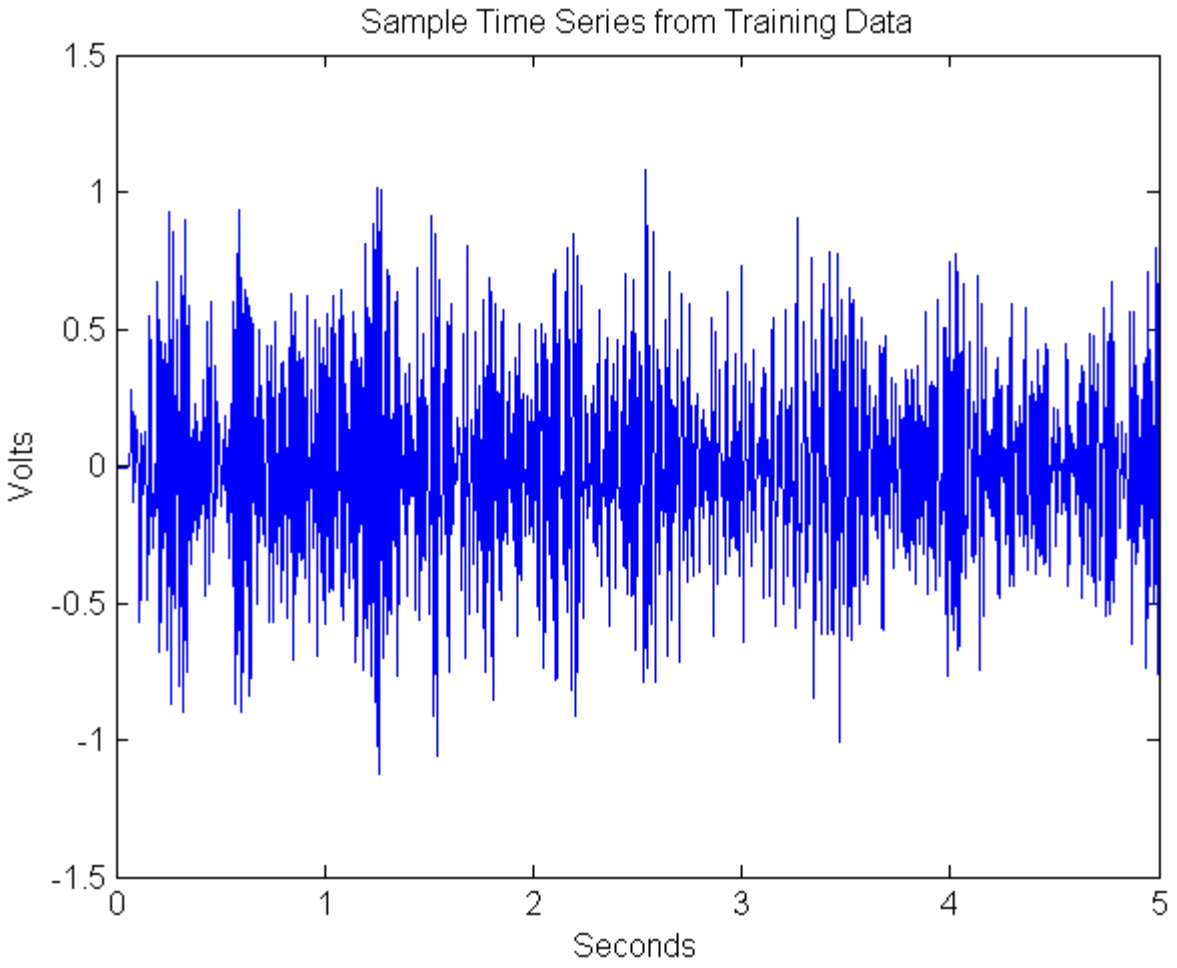
Acquire Time Series Data

Aquire the 50 sets of time series data to serve as the undamaged training sets.

```
if isSimulation
    trainingData=simTrainingData;
else
    trainingData=exciteAndAquire_shm(adaptor, aiID, aiChans, aoID, sampleRate, exciteWaveform,
aiNumSamples, runCount);
end
```

Plot some of the training time series data.

```
figure();
t=(0:aiNumSamples-1)/sampleRate;
plot(t,trainingData(:,5,1));
title('Sample Time Series from Training Data')
xlabel('Seconds'); ylabel('Volts');
```



Calculate Appropriate AR Model Order

Calculate an appropriate AR model order for reconstructing the time series using the Partial Autocorrelation Function Method.

```
data=trainingData(:,5,1);
method='PAF';

arOrder=arModelOrder_shm(data,method);
```

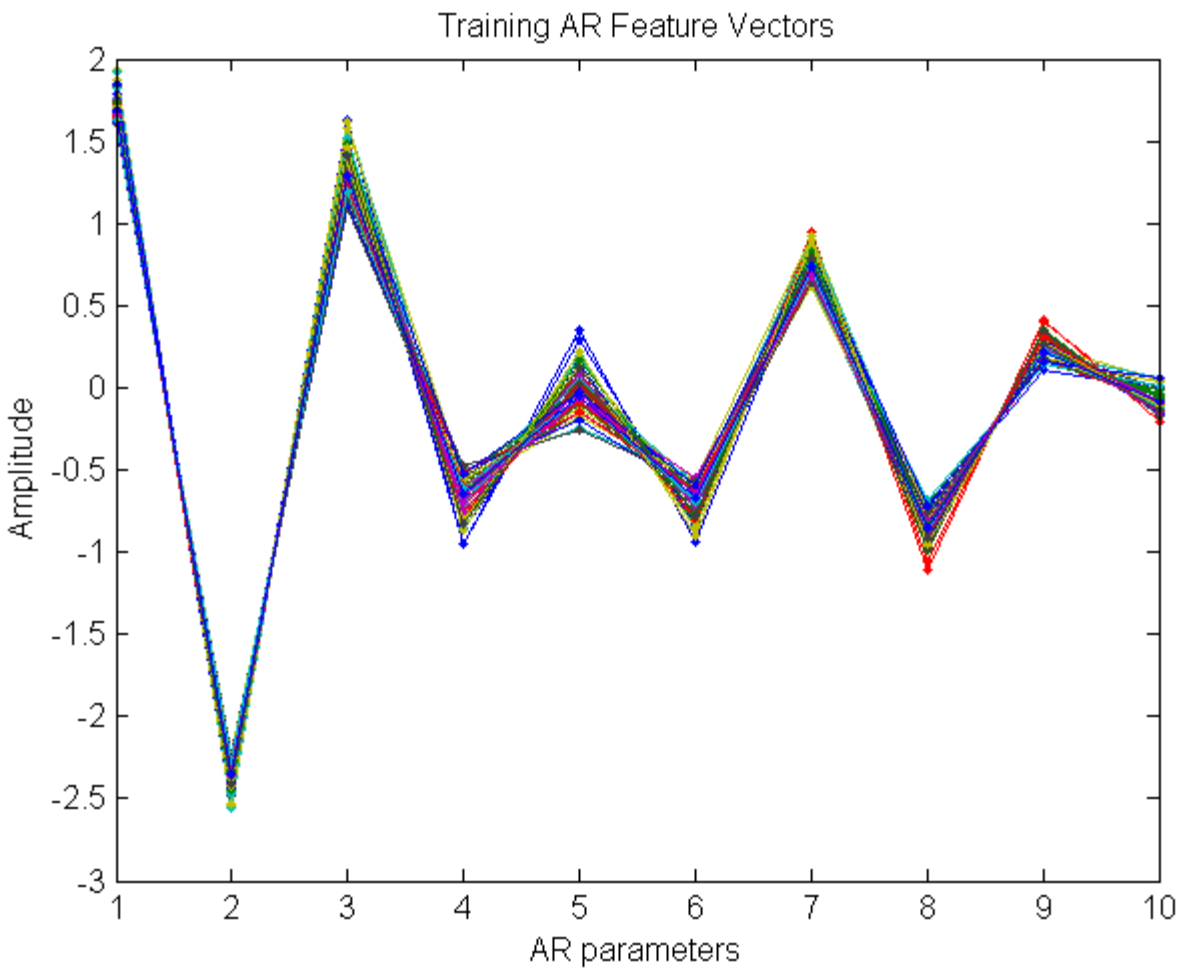
Extract AR Parameter Features

Extract the AR coefficients from channel five (top floor) of the acceleration time histories. These AR coefficients will serve as the feature vector.

```
[arParametersTrain]=arModel_shm(trainingData(:,5,:),arOrder);
```

Plot the AR feature vectors from the training sets.

```
figure();
plot(arParametersTrain','.-')
title('Training AR Feature Vectors')
xlabel('AR parameters'); ylabel('Amplitude')
set(gca,'XTick',1:arOrder,'Xlim',[1 arOrder])
```



Learn Mahalanobis Distance Model

Learn a model based on the Mahalanobis distance using the AR features from the training set.

```
[model]=learnMahalanobis_shm(arParametersTrain);
```

Determine Threshold

Determine Mahalanobis score threshold assuming AR feature vectors are Gaussian distributed. If the original feature vectors are multi-dimensional Gaussian, then the distribution of the square of the Mahalanobis distance will be chi-squared with degrees of freedom equal to the number of AR coefficients (model order).

```
dist='chi2';
dof=arOrder;
confidence=.99;
```

```
mThreshold = icdf(dist,confidence,dof);
```

END OF TRAINING

START OF LIVE TESTING

Set up Testing Process

Acquire and process test sets one at a time. The test will be run seven times with the structure in the following seven states: State 1,2,3: Undamaged, State 4: One loose joint, State 5: Two loose joints, State 6: Three loose joints, State 7: Four loose joints. This looping mimics the process an in-service SHM system might follow.

```
numTestCases=7;
runCount=1;
arraySize=[aiNumSamples,runCount];

testData=zeros(aiNumSamples,5,numTestCases);
arParametersTest=zeros(numTestCases,arOrder);
testScores=zeros(numTestCases,1);
damageState=zeros(numTestCases,1);

for i=1:numTestCases
```

```
    caseName=['Case ' num2str(i)];
```

Create excitation waveform

```
    exciteWaveform=bandLimWhiteNoise_shm(arraySize,cutoffs,rms);
```

Acquire Time Series Data

```
    if isSimulation
        testData(:,:,i)=simTestData(:,:,i);
    else
        pause(15)
        testData(:,:,i)=exciteAndAquire_shm(adaptor, aiID, aiChans, aoID, sampleRate,
exciteWaveform, aiNumSamples, runCount);
    end
```

Extract AR Parameter Features

```
    [arParametersTest(i,:)]=arModel_shm(testData(:,5,i),arOrder);
```

Apply Mahalanobis Scoring

The score is equal to the squared Mahalanobis distance from the training set

```
    testScores(i)=scoreMahalanobis_shm(arParametersTest(i,:),model);
```

Threshold the Score

```
    damageState(i)=-testScores(i)>mThreshold;
    if damageState(i), s='Damaged'; else s='Undamaged'; end
    fprintf(1,'Case %d: %s\n',i,s);
```

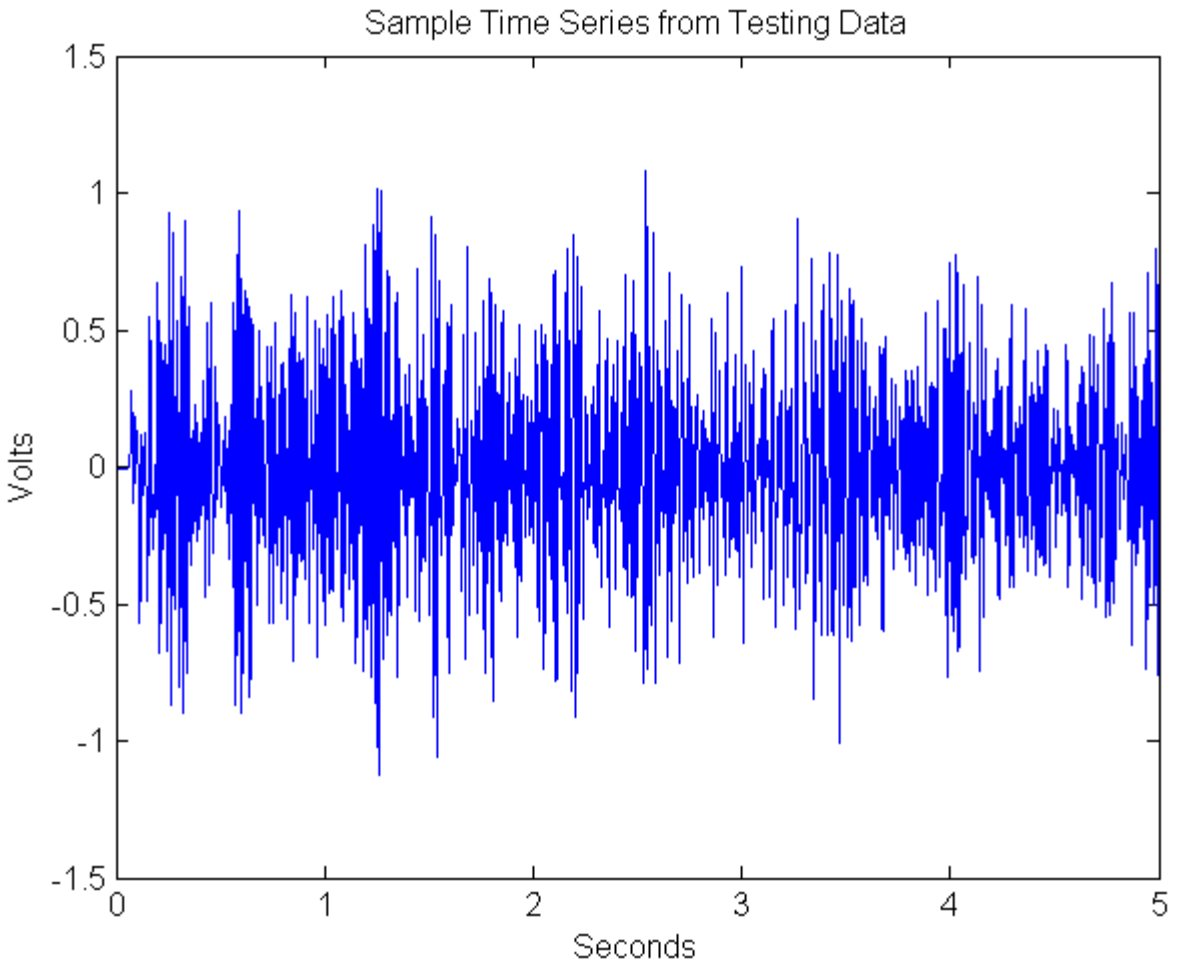
```
Case 1: Undamaged

Case 2: Undamaged

Case 3: Undamaged

Case 4: Damaged

Case 5: Damaged

Case 6: Damaged

Case 7: Damaged
```

```
end
```

END OF LIVE TESTING

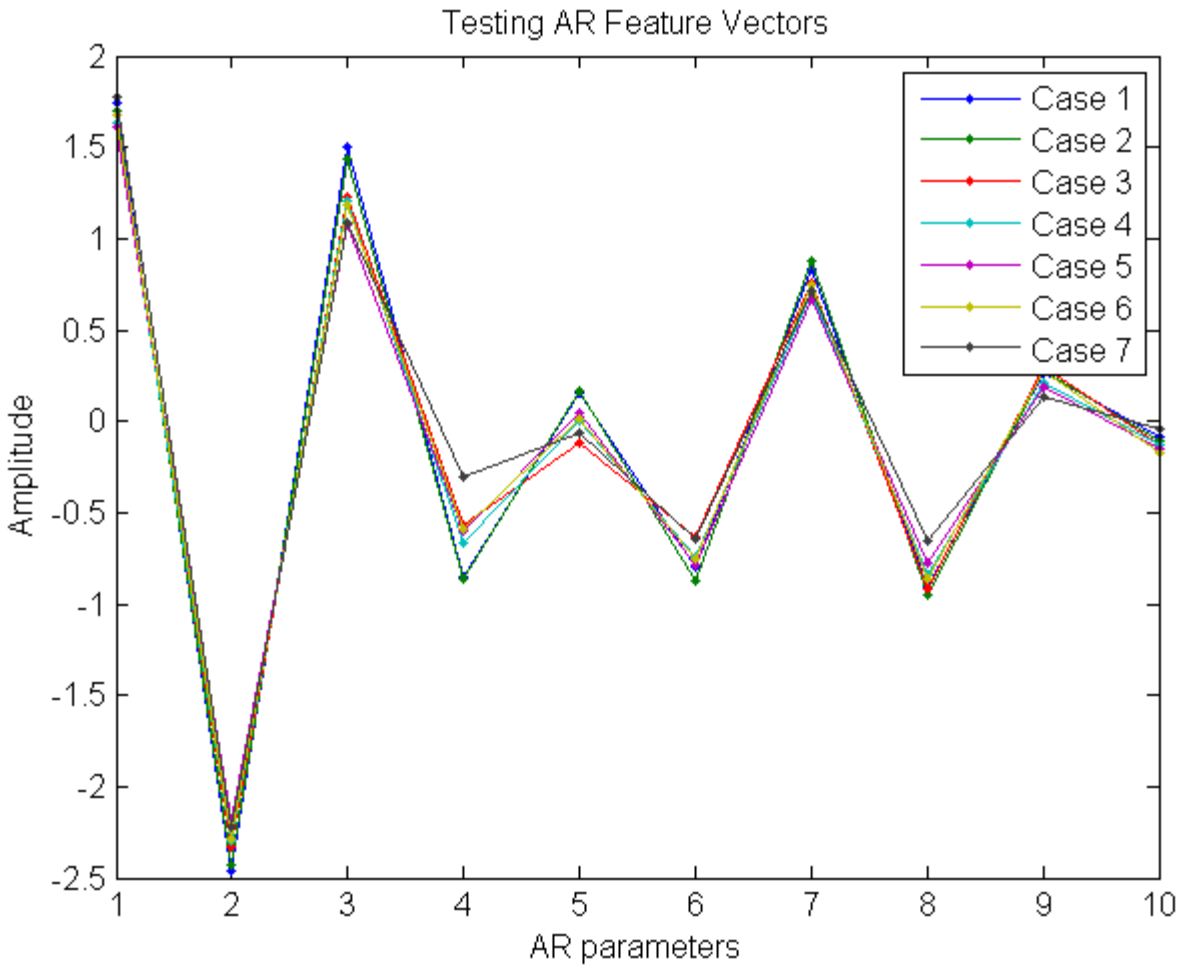
Plot some of the time series data from the test sets

```
figure();
t=(0:aiNumSamples-1)/sampleRate;
plot(t,trainingData(:,5,1));
title('Sample Time Series from Testing Data')
xlabel('Seconds'); ylabel('Volts');
```



Plot AR features from the test data. Note that by casual obeservation,
the feature vectors from the damaged cases don't seem much different
than those from the undamaged.

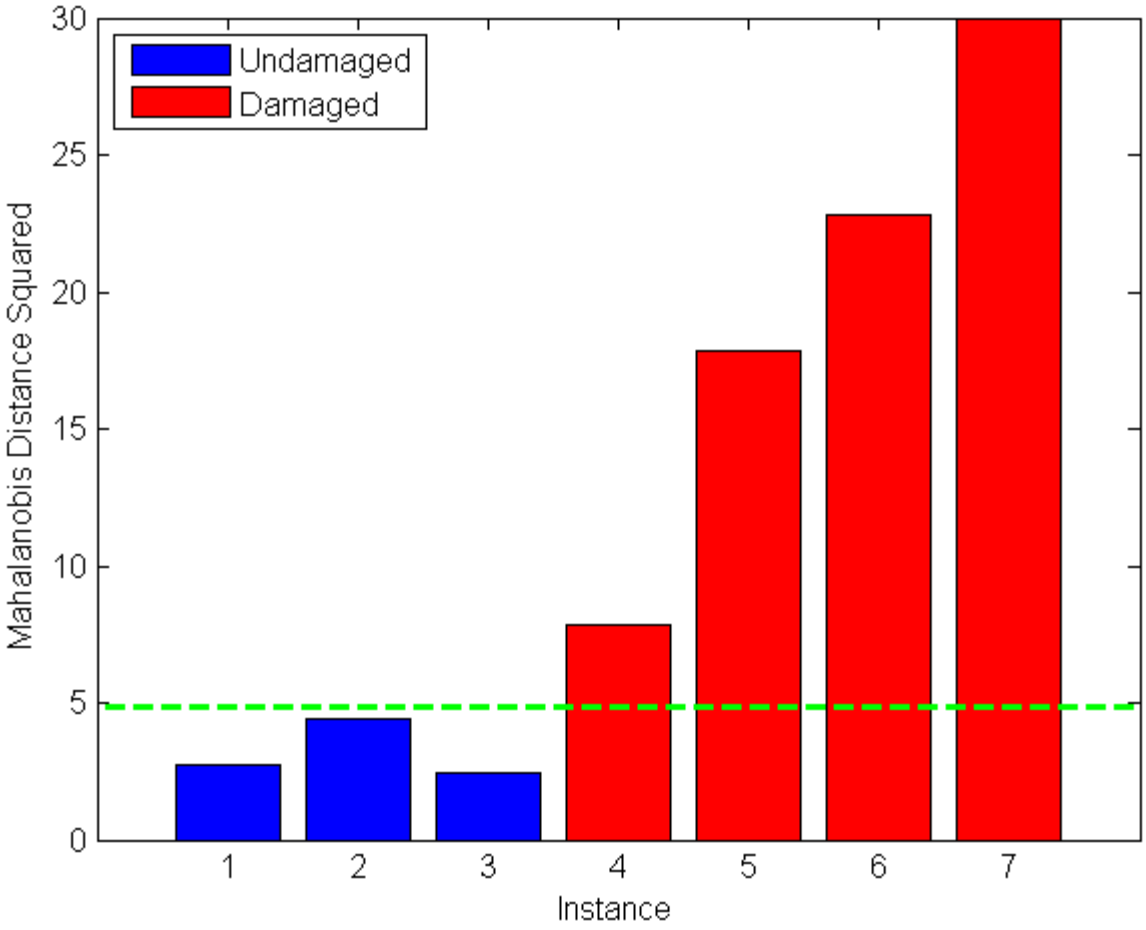
```
figure();
plot(arParametersTest','.-')
title('Testing AR Feature Vectors')
xlabel('AR parameters'); ylabel('Amplitude')
set(gca,'XTick',1:arOrder,'Xlim',[1 arOrder])
legend('Case 1','Case 2','Case 3','Case 4','Case 5','Case 6','Case 7');
```



Plot Detection Results

Plot the Mahalanobis distance for each case on a bar graph along with the threshold and the detected state (undamaged/damaged).

```
scores=sqrt(-testScores);
threshold=sqrt(mThreshold);
detectedStates=damageState;
stateNames={'Undamaged','Damaged'};
axesHandle=[];

plotScores_shm(scores, detectedStates, stateNames, threshold, false, false, axesHandle);

ylabel('Mahalanobis Distance Squared')
```



Published with MATLAB® R2013a

Outlier Detection based on Chi-Squared Distribution for Undamaged State

Contents

Introduction

The goal of this example is to discriminate acceleration time histories from undamaged and damaged condition based on a Chi-square distribution for the undamaged condition. Two different approaches are used for classification, namely, based on confidence intervals and hypothesis test.

The autoregressive (AR) parameters are used as damage-sensitive features and a machine learning algorithm based on the Mahalanobis distance is used to create damage indicators (DIs) invariant for feature vectors from the normal condition and that increase for feature vectors from damaged conditions.

Data sets from Channel 5 of the base-excited three story structure are used in this example usage. More details about the data sets can be found in 3-Story Data Sets.

Requires data3SS.mat dataset.

References:

SHMTools functions called:

```
arModel_shm
learnMahalanobis_shm
scoreMahalanobis_shm
```

Author: Elói Figueiredo

Date: September 01, 2009

Load Raw Data

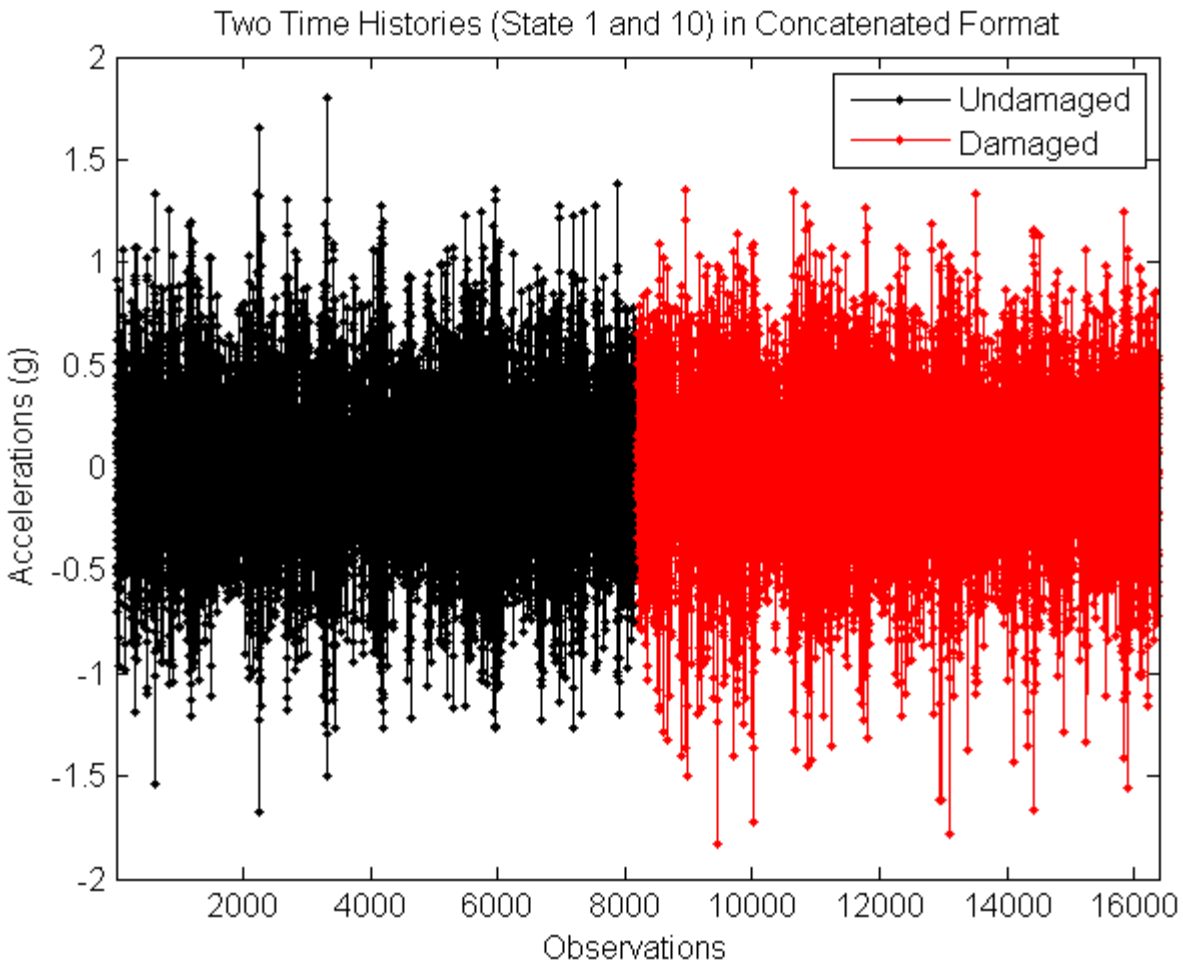
Load data set:

```
load('data3SS.mat');

data=dataset(:,5,:);
t=size(data,1);
```

Plot one time history from the baseline (State#1) and damaged (State#10) conditions:

```
figure
```

```
plot(1:t,data(:,1,1),'.-k')
hold on
plot(t+1:t*2,data(:,1,101),'.-r')
title('Two Time Histories (State 1 and 10) in Concatenated Format')
legend('Undamaged','Damaged')
xlabel('Observations')
ylabel('Accelerations (g)')
set(gca,'Xlim',[1 t*2])
```



Extraction of Damage-Sensitive Features

The AR(15) model parameters are extracted from the acceleration time histories.

AR model order:

```
arOrder=15;
```

Estimation of the AR parameters:

```
[arParameters]=arModel_shm(data,arOrder);
```

Feature vectors from all the undamaged cases:

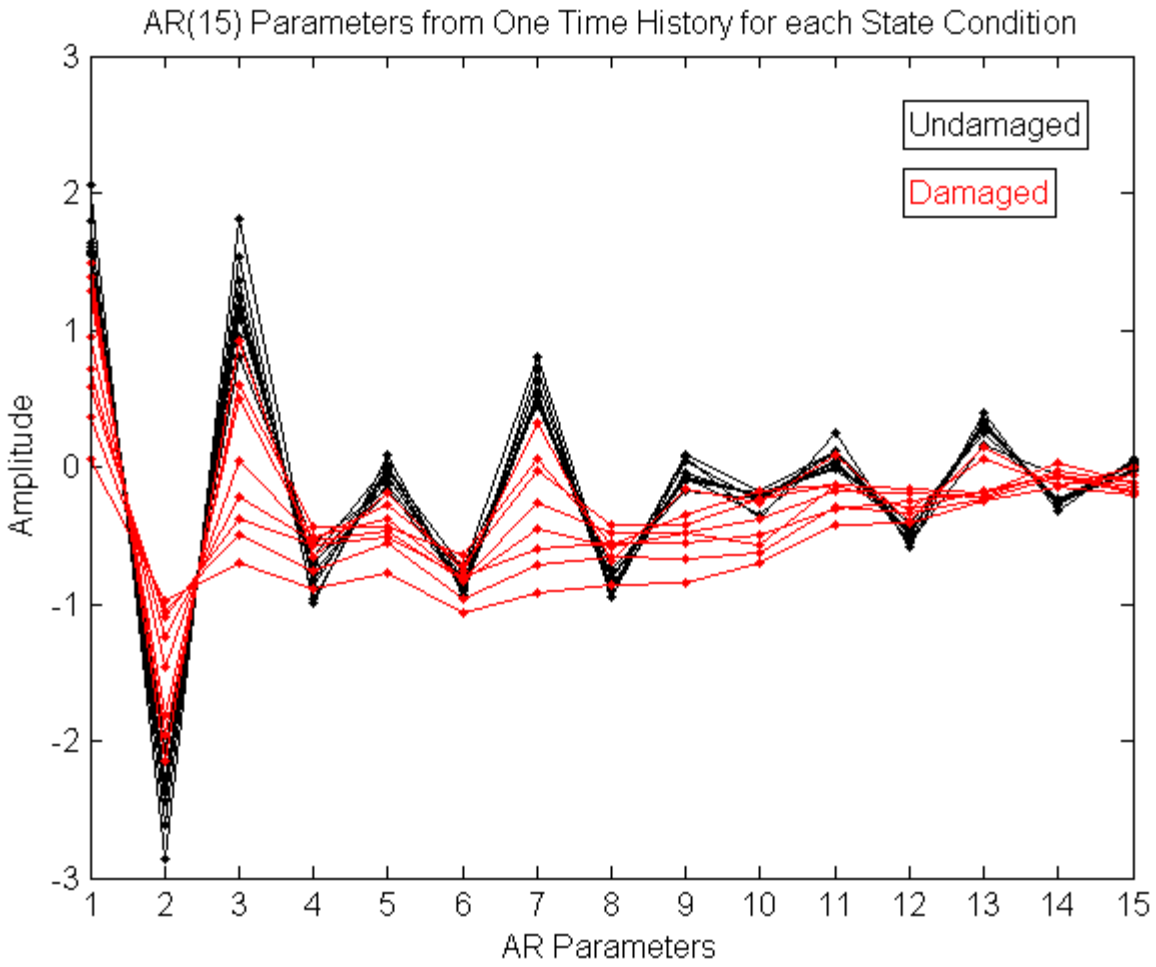
```
learnData = splitFeatures_shm(arParameters, states<10, [], []);
```

Feature vectors from all instances (undamaged and damaged):

```
scoreData=arParameters;
```

Plot test data:

```
figure
plot(1:arOrder,scoreData(10:10:90,:),'.-k')
hold on
plot(1:arOrder,scoreData(100:10:170,:),'.-r')
title(['AR(',num2str(arOrder),') Parameters from One Time History for each State Condition'])
xlabel('AR Parameters')
ylabel('Amplitude')
set(gca,'XTick',1:arOrder,'Xlim',[1 arOrder])
text(12,2.5,'Undamaged','Color','k','EdgeColor','k','BackgroundColor','w')
text(12,2.0,'Damaged','Color','r','EdgeColor','k','BackgroundColor','w')
```



Statistical Modeling For Feature Classification

First, each feature vector is reduced to one score (DI) by using the Mahalanonbis-based machine learning algorithm. Second, the Chi-square distribution is used to model the DIs from undamaged condition. (Note that the parametric distribution of the damaged condition is

not used because it lacks precision.)

Run the Mahalanobis-based Machine Learning Algorithm:

```
[model]=learnMahalanobis_shm(learnData);

[DI]=scoreMahalanobis_shm(scoreData,model); DI=-DI;
```

Flag and split all the instances into undamaged (0) and damaged (1):

```
stateFlag(1:90)=0; stateFlag(91:170)=1;
x=DI(1:90);
y=DI(91:170);
n=length(DI);
```

Define the Underlying Distribution of the Undamaged Condition

Histogram:

```
nbins=15;
h1=(max(x)-min(x))/nbins;
[n1,xout1]=hist(x,nbins);
```

Impose parameteric probability distribution:

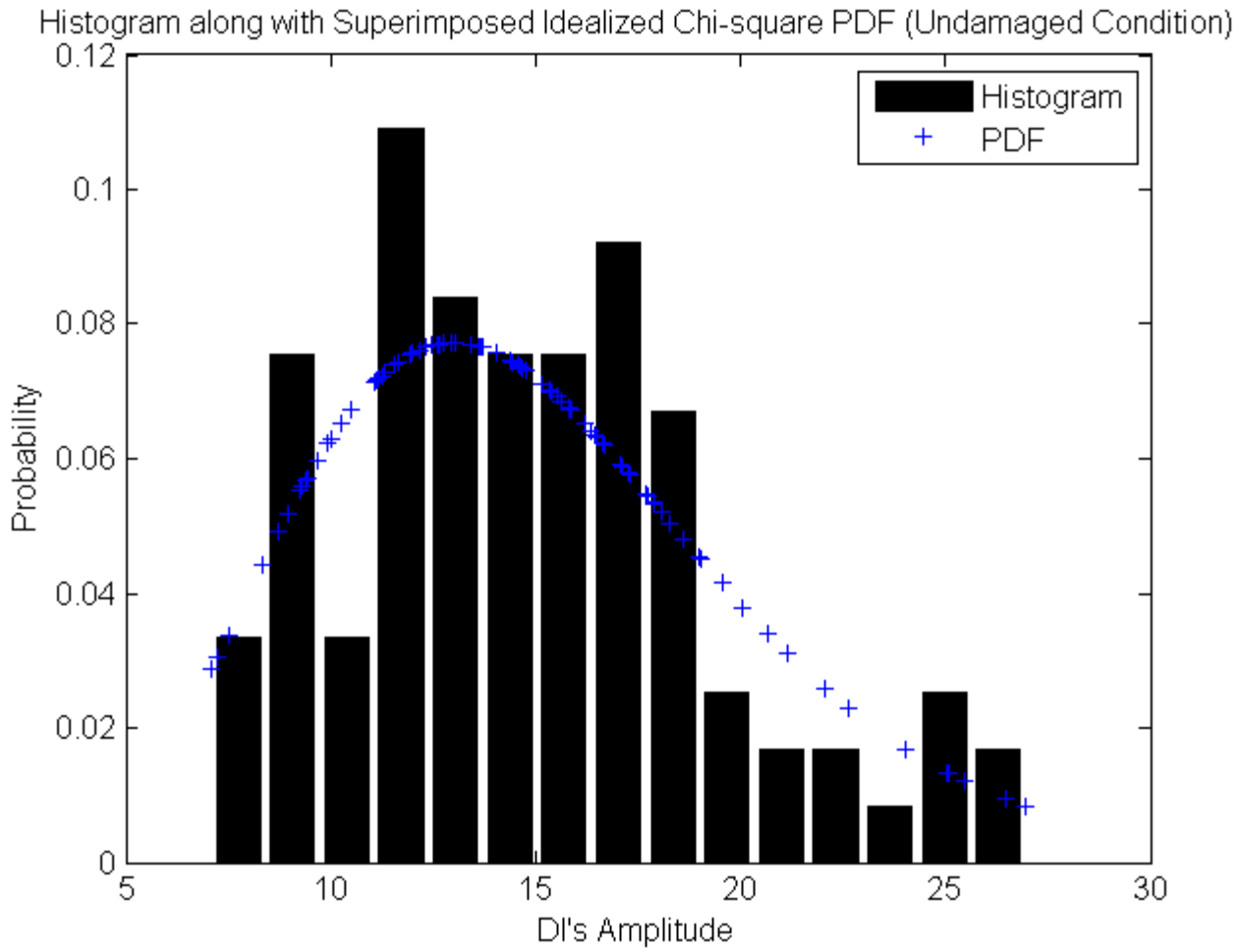
```
dist='chi2';
```

Estimate probability distribution function (PDF):

```
X = pdf(dist,x,arOrder);
```

Plot histogram along with superimposed idealized PDF:

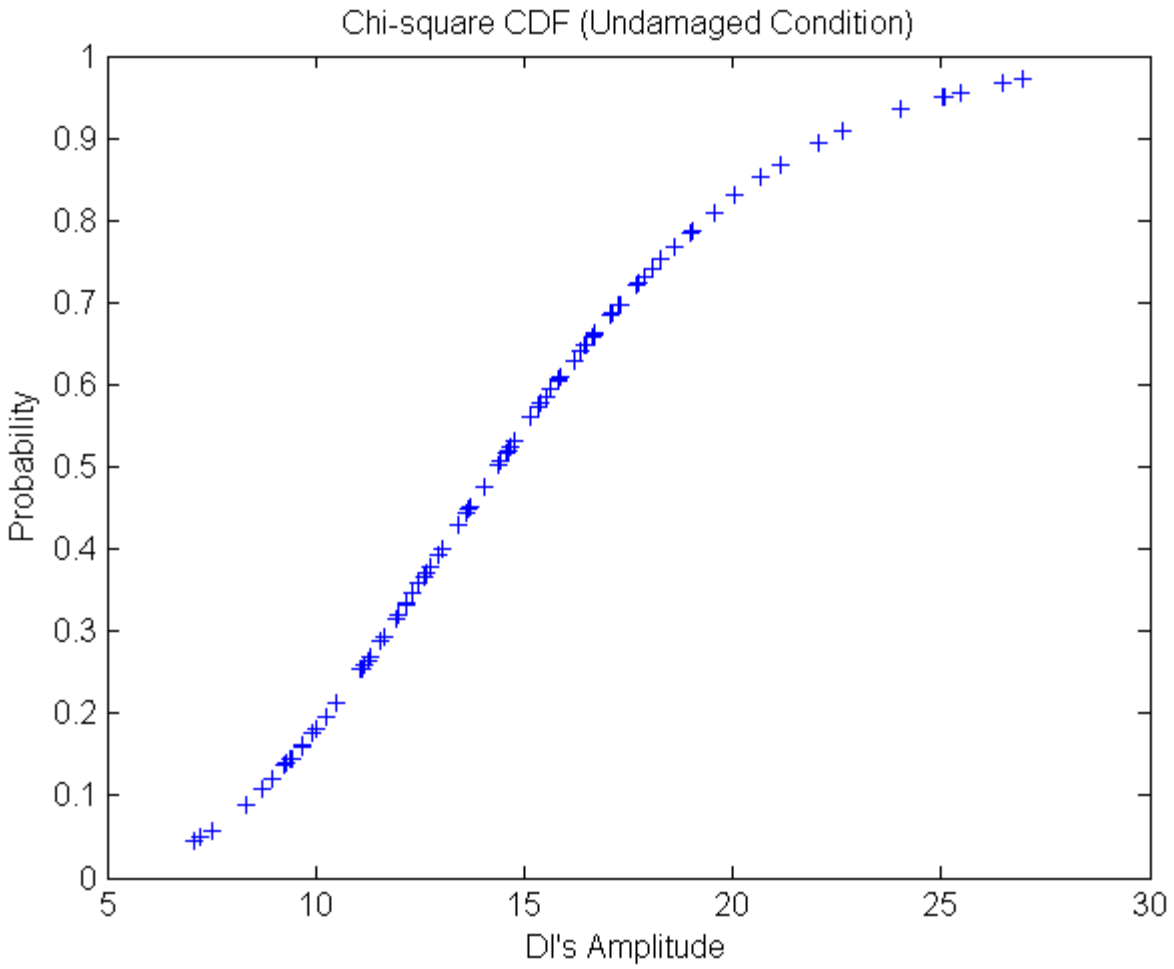
```
figure
bar(xout1,n1/(h1*length(x)),'k')
title('Histogram along with Superimposed Idealized Chi-square PDF (Undamaged Condition)')
hold on
plot(x,X,'+b')
xlabel('DI''s Amplitude')
ylabel('Probability')
legend('Histogram','PDF')
```



Note: Irregularities in the original distribution (histogram) most likely due to chance, are ignored by the smoothed distribution. Accordingly, any generalizations based on the smoothed distribution will tend to be more accurate than those based on the original distribution.

Estimate cumulative distribution function (CDF):

```
cdfx = cdf(dist,x,arOrder);

figure
plot(x,cdfx,'+b')
title('Chi-square CDF (Undamaged Condition)')
xlabel('DI''s Amplitude')
ylabel('Probability')
```



Confidence Interval

This section defines an upper threshold for feature classification based on information from the undamaged distribution. Note that feature classification can be done using either *hypothesis tests* or *confidence intervals*. Hypothesis tests only indicate whether or not an effect is present, whereas confidence intervals indicate the possible size of the effect.

Probability of false alarm or level of significance:

```
PFA=0.05;
```

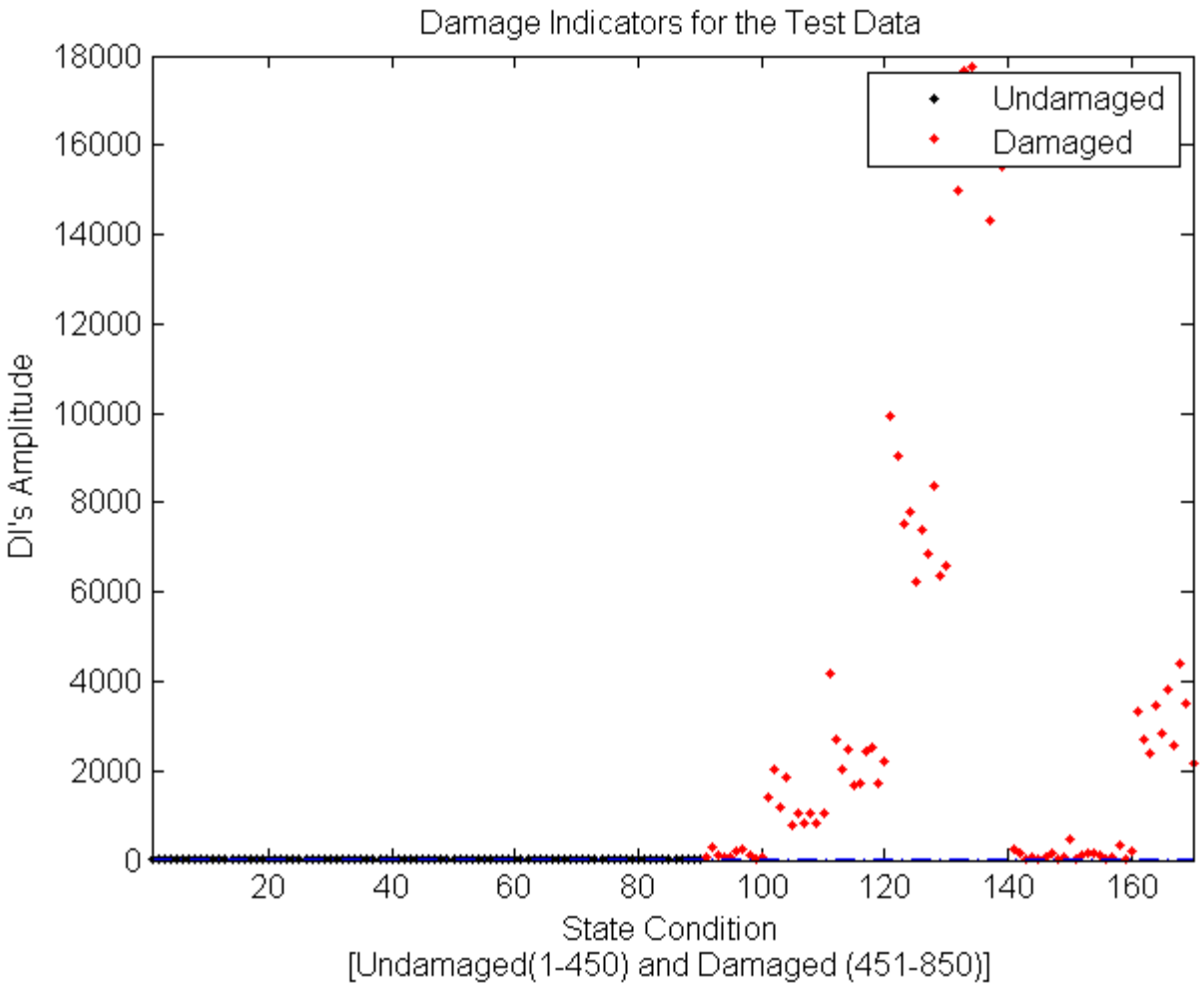
Threshold limit (or critical DI):

```
UCL = icdf(dist,(1-PFA),arOrder);
```

Plot DIs along with the threshold:

```
figure
plot(1:90,DI(1:90),'.k')
hold on
plot(91:170,DI(91:170),'.r')
title('Damage Indicators for the Test Data')
xlabel({'State Condition','[Undamaged(1-450) and Damaged (451-850)]'})
ylabel('DI''s Amplitude')
```

```
set(gca,'XLim',[1 length(DI)])
legend('Undamaged','Damaged')
line('XData',[0 length(DI)],'YData',[UCL UCL],'Color','b','LineWidth',1,'LineStyle','-.')
```



Number of Type I and Type II Error:

```
classState=zeros(1,length(DI));

for i=1:length(DI);

    if DI(i)>UCL; classState(i)=1; end

end

numErrorTypeI=length(find(classState==1 & stateFlag==0));
numErrorTypeII=length(find(classState==0 & stateFlag==1));

fprintf('Number of Type I Error is %g \n', numErrorTypeI)
fprintf('Number of Type II Error is %g \n', numErrorTypeII)
```

```
Number of Type I Error is 5
Number of Type II Error is 1
```

In order to define an effective threshold for outlier detection, it was adopted a 95% confidence interval of the upper-tail Chi-square distribution from the undamaged condition. The threshold was found from the false alarm constraint. Note that by changing the threshold, one can trade off probability of false alarm (PFA) and probability of detection (PD). The number of Type I and Type II errors gives information about 23 false positive indications of damage and 11 false negative indications of damage (roughly 4% of misclassifications).

Hypothesis Test

Statistical Hypothesis (*p-values*):

H_0 =Undamaged

H_1 =Damaged

Decision Rule:

The *p-values* for a test result represents the degree of rarity of that result given that the null hypothesis is true.

Decision:

Smaller *p-values* tend to discredit the null hypothesis H_0 and to support the alternative hipothesis H_1.

Pick a DI score up randomly:

```
aux=DI(80);

p_value = 1-cdf(dist,aux,arOrder)
```

```
p_value =

    0.3525
```

Hypothesis tests are useful to indicate whether or not an effect is present. Rather than associate the result to a predetermined level of significance, the *p-value* indicates the degree of rarity of a test result, i.e., for a given DI score, the *p-value* gives an indication of the probability of representing a time history from an undamaged condition. As a reference, for a level of confidence of 95%, the result supports the alternative hypotheses if *p-value* is lesser than 0.05.

Published with MATLAB® R2013a

Damage Location using AR Parameters from an Array of Sensors

Contents

Introduction

The goal of this example usage is to locate the source of damage in a structure based on the outlier/novelty detection. The autoregressive (AR) parameters are used as damage-sensitive features and a machine learning algorithm based on the Mahalanobis distance is used to create damage indicators (DIs) invariant for feature vectors from normal conditions and that increase when feature vectors are from damaged conditions.

Data sets of the base-excited three story structure are used in this example usage. More details about the data sets can be found in 3-Story Data Sets.

Requires data3SS.mat dataset.

References:

Figueiredo, E., Park, G., Figueiras, J., Farrar, C., & Worden, K. (2009). Structural Health Monitoring Algorithm Comparisons using Standard Data Sets. Los Alamos National Laboratory Report: LA-14393.

SHMTools functions called:

```
arModel_shm
learnMahalanobis_shm
scoreMahalanobis_shm
```


Author: Elói Figueiredo

Date Created: September 01, 2009

Load Raw Data

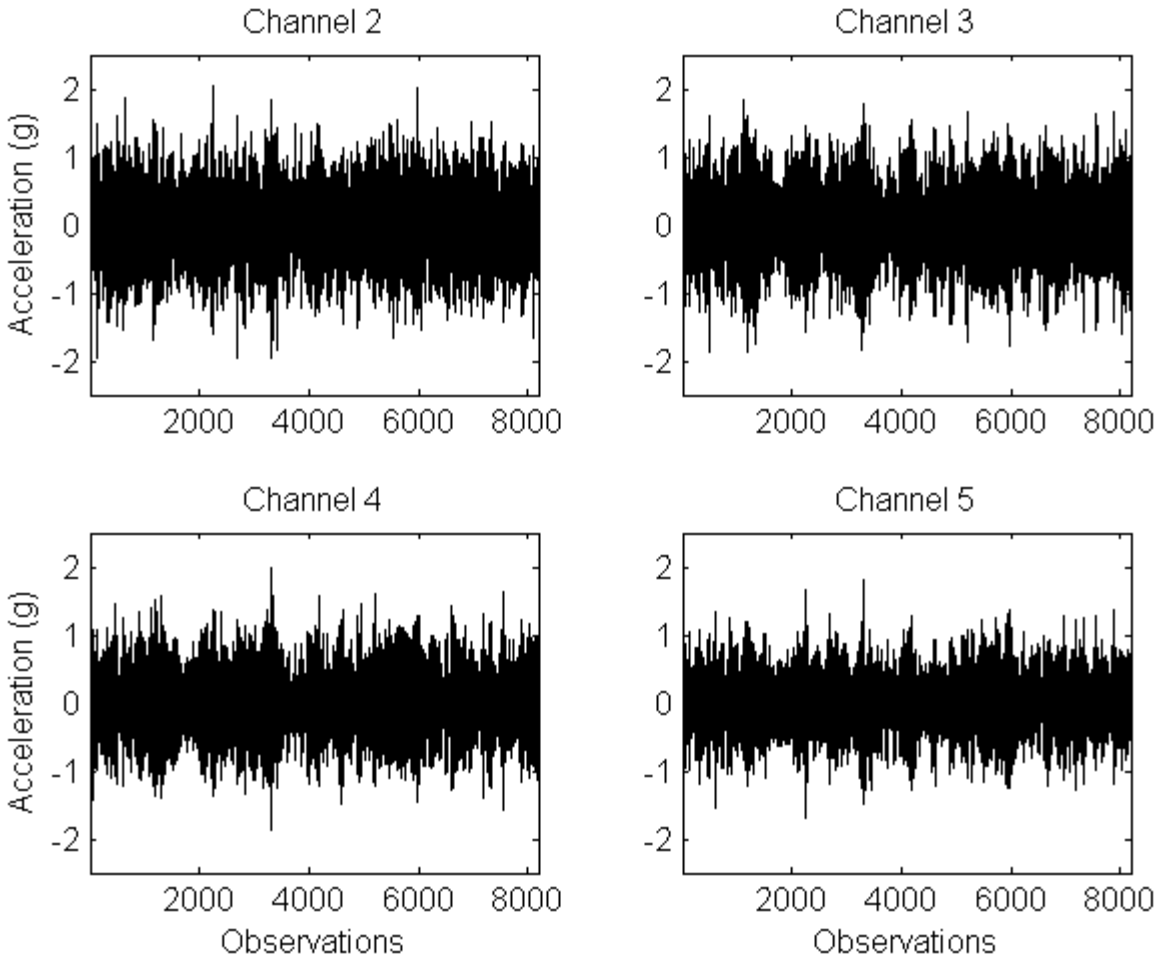
Load data set:

```
load('data3SS.mat');

data=dataset(:,2:5,:);
```

Plot time histories from the baseline condition (Channel 2-5):

```
figure

for i=1:4;
```

```
    subplot(2,2,i)
    plot(data(:,i,1),'k')
    title(['Channel ',num2str(i+1)])
    set(gca,'YTick',-2:2,'Xlim',[1 8192],'Ylim',[-2.5 2.5])

    if i==3 || i==4, xlabel('Observations'); end
    if i==1 || i==3, ylabel('Acceleration (g)'); end

end
```



Extraction of Damage-Sensitive Features

This section extracts the AR(15) model parameters from time histories of Channels 2-5 and plot the feature vectors for each instance.

AR model order:

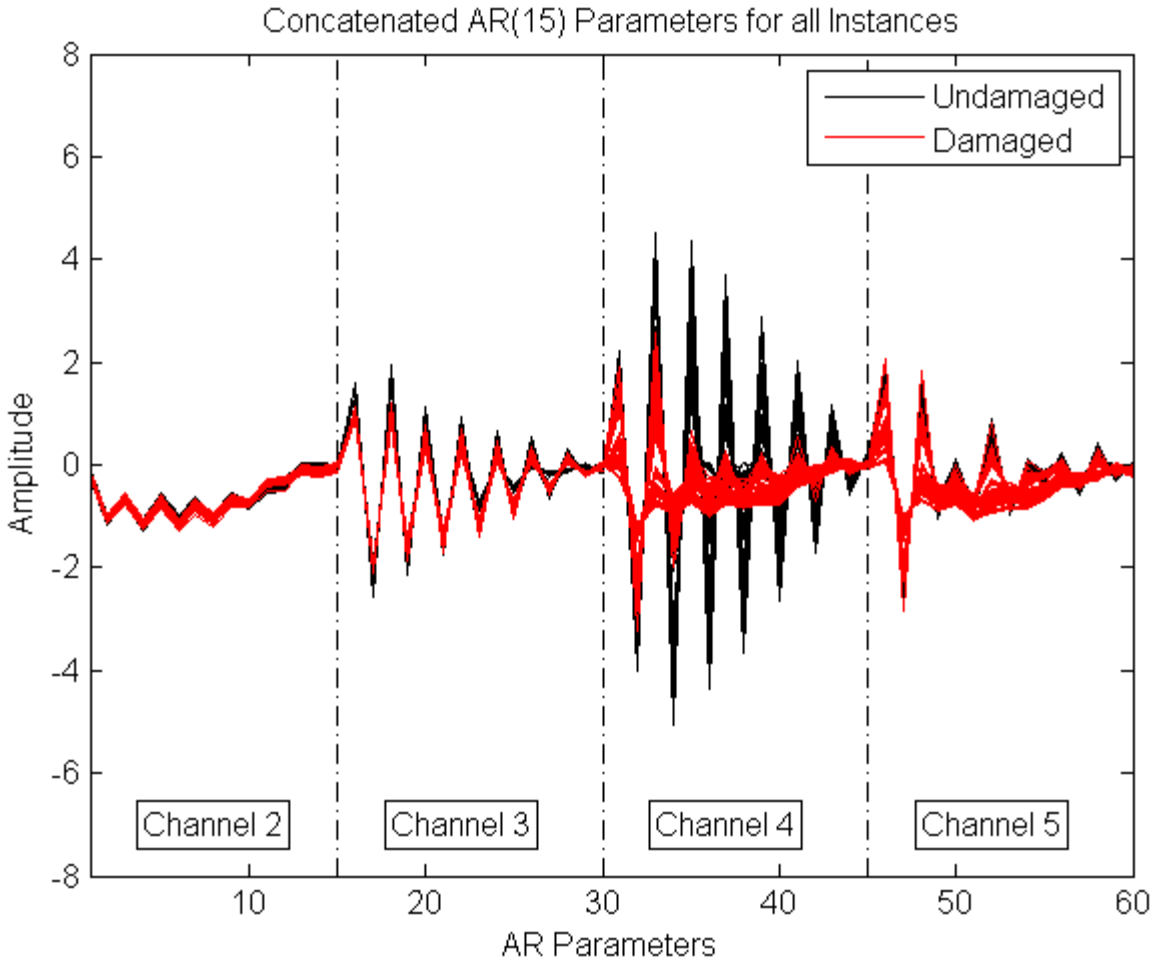
```
arOrder=15;
```

Estimation of the AR Parameters:

```
[arParameters]=arModel_shm(data,arOrder);

[n m]=size(arParameters);
```

Plot test data:

```
figure;
plot(1:m,arParameters(1:90,:)','k',1:m,arParameters(90:170,:)','r')
title(['Concatenated AR(',num2str(arOrder),') Parameters for all Instances'])
legend('Undamaged','Damaged')
xlabel('AR Parameters')
ylabel('Amplitude')
w=8;
set(gca,'Xlim',[1 m])
M(1,1:9)='Undamaged'; M(2,1:7)='Damaged';
legend([line('color','k');line('color','r')],M);
h(1)=line([m/4;m/4],[-w w],'color','k','lineStyle','-.');
h(2)=line([m/4*2;m/4*2],[-w w],'color','k','lineStyle','-.');
h(3)=line([m/4*3;m/4*3],[-w w],'color','k','lineStyle','-.');
text(4,-7,'Channel 2','Color','k','EdgeColor','k','BackgroundColor','w')
text(18,-7,'Channel 3','Color','k','EdgeColor','k','BackgroundColor','w')
text(33,-7,'Channel 4','Color','k','EdgeColor','k','BackgroundColor','w')
text(48,-7,'Channel 5','Color','k','EdgeColor','k','BackgroundColor','w')
```



Data Normalization for Novelty Detection

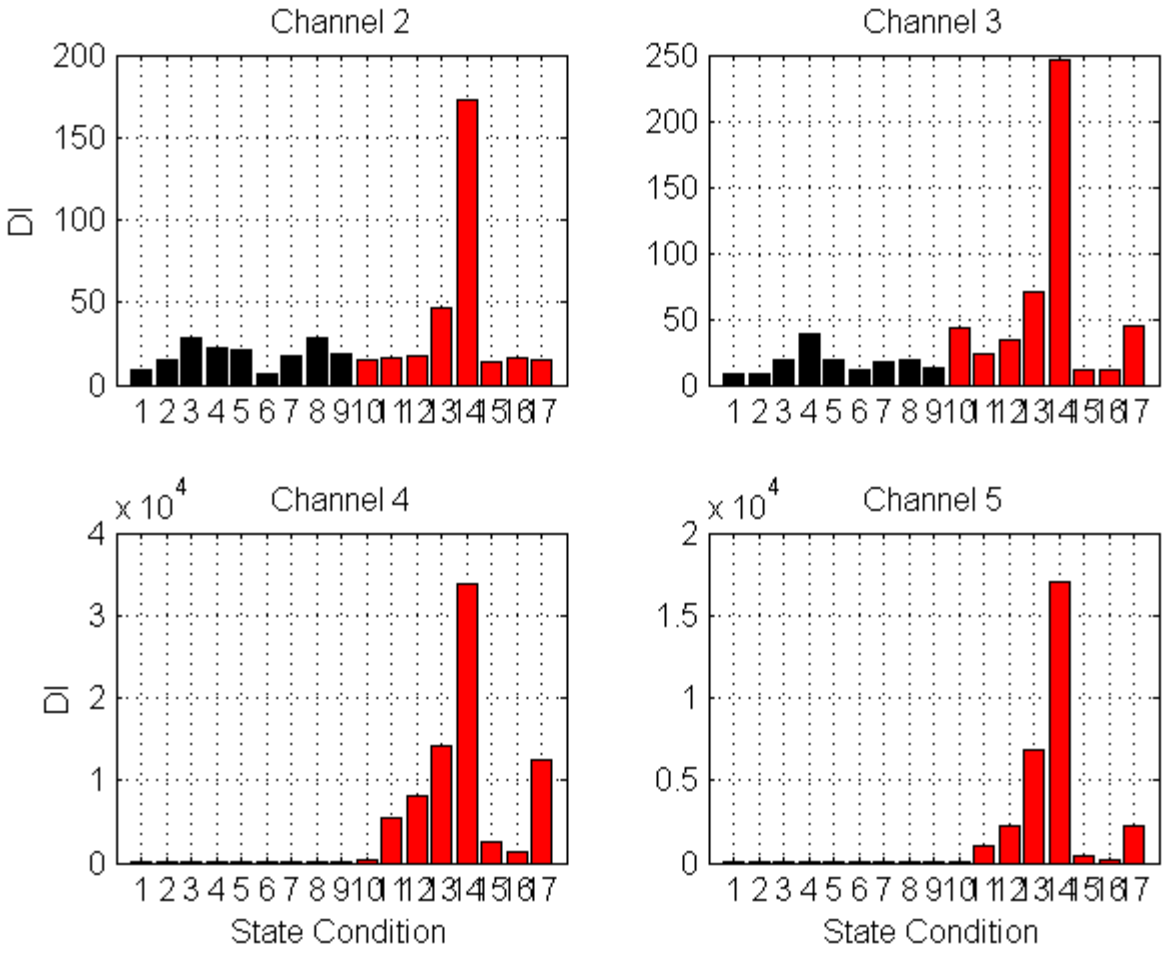
The Mahalanobis-based machine learning algorithm is used to normalize the features and reduce each feature vector to a score.

```
DI=zeros(17,4);
scoreData=zeros(17,arOrder);

cnt=1;

for i=1:4;

    for j=1:9;
        learnData(j*9-8:j*9,:)=arParameters(j*10-9:j*10-1,cnt:cnt+arOrder-1);
    end

    scoreData(1:17,:)=arParameters(10*(1:17),cnt:cnt+arOrder-1);

    [model]=learnMahalanobis_shm(learnData);

    DI(:,i)=scoreMahalanobis_shm(scoreData,model);

    cnt=cnt+arOrder;

end

DI=-DI;
```

Damage Location

Plot DIs from Channel 2-5:

```
figure

for i=1:4;

    subplot(2,2,i)
    bar(1:9,DI(1:9,i),'k'); hold on; bar(10:17,DI(10:17,i),'r')
    title(['Channel ',num2str(i+1)])
    set(gca,'Xlim',[0 length(DI)+1],'XTick',1:length(DI))
    grid on

    if i==3 || i==4, xlabel('State Condition'); end
    if i==1 || i==3, ylabel('DI'); end

end
```



The figure above shows that the extracted features from Channels 2-3 are lesser sensitive than from Channels 4 and 5 to discriminate the undamaged (1-9) and damaged (10-17) state conditions. This is an indication that the source of damage is located near to Channels 4 and 5.

See also:

Damage Location using ARX Parameters from an Array of Sensors

---

Published with MATLAB® R2013a

Damage Location using ARX Parameters from an Array of Sensors

Contents

Introduction

The goal of this example usage is to locate the source of damage in a structure based on the outlier/novelty detection. The autoregressive model with eXogenous (ARX) inputs parameters are used as damage-sensitive features and a machine learning algorithm based on the Mahalanobis distance technique is used to create a damage indicators (DIs) invariant for feature vectors from normal condition and that increase when feature vectors are from damaged conditions. This example usage intends to show the advantages of taking into account the input force to locate damaged in the structure. (See example to check the differences when using the AR parameters as damage-sensitive features.)

Data sets of the base-excited three story structure are used in this example usage. More details about the data sets can be found in 3-Story Data Sets.

Requires data3SS.mat dataset.

References:

SHMTools functions called:

```
arxModel_shm
learnMahalanobis_shm
scoreMahalanobis_shm
```

Author: Elói Figueiredo

Date Created: September 01, 2009

Load Raw Data

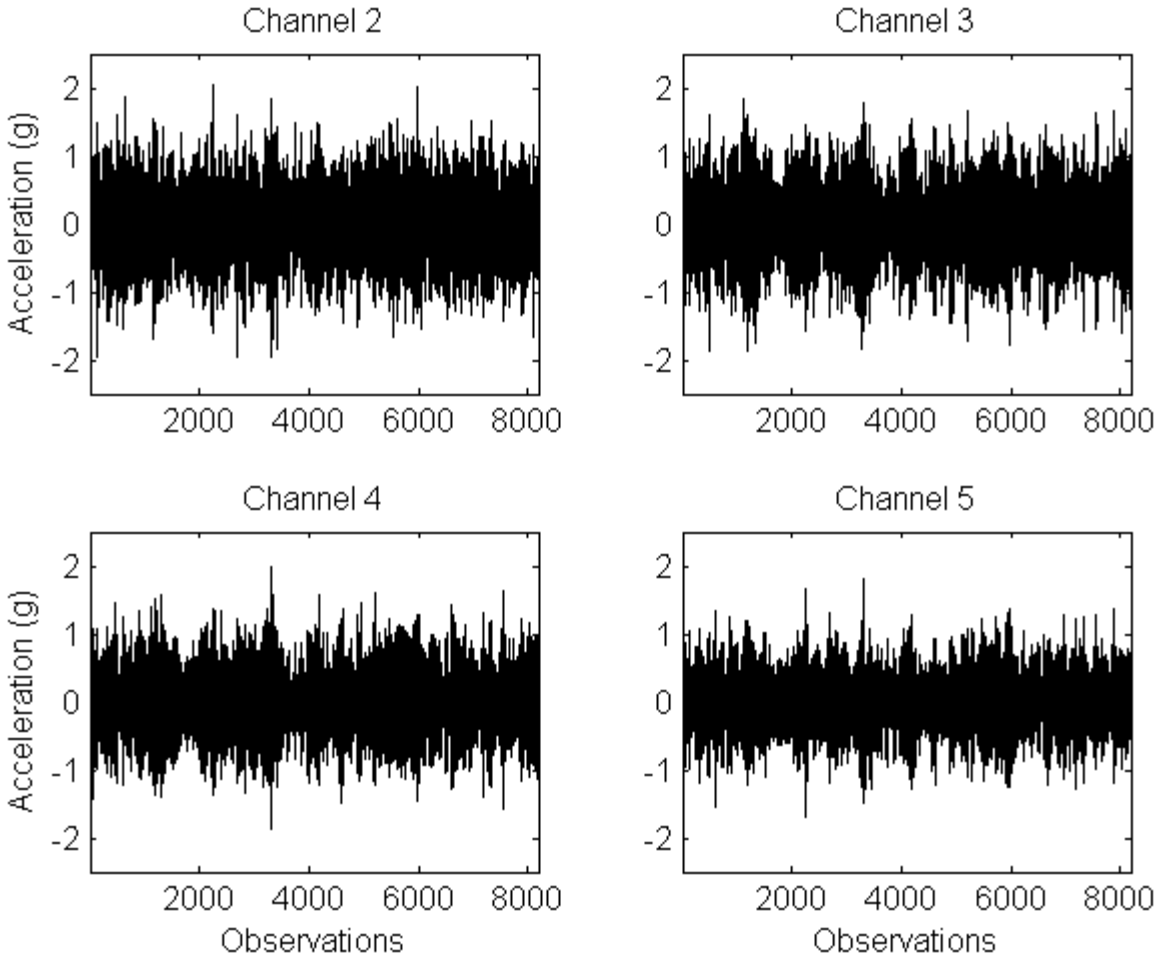
Load data set:

```
load('data3SS.mat');
```

Plot time histories from the baseline condition (Channel 2-5):

```
figure

for i=1:4;

    subplot(2,2,i)
    plot(dataset(:,i+1,1),'k')
```

```
    title(['Channel ',num2str(i+1)])
    set(gca,'YTick',-2:2,'Xlim',[1 8192],'Ylim',[-2.5 2.5])

    if i==3 || i==4, xlabel('Observations'); end
    if i==1 || i==3, ylabel('Acceleration (g)'); end

end
```



Extraction of Damage-Sensitive Features

Extraction of the ARX(10,5) model parameters from the time histories.

ARX model order:

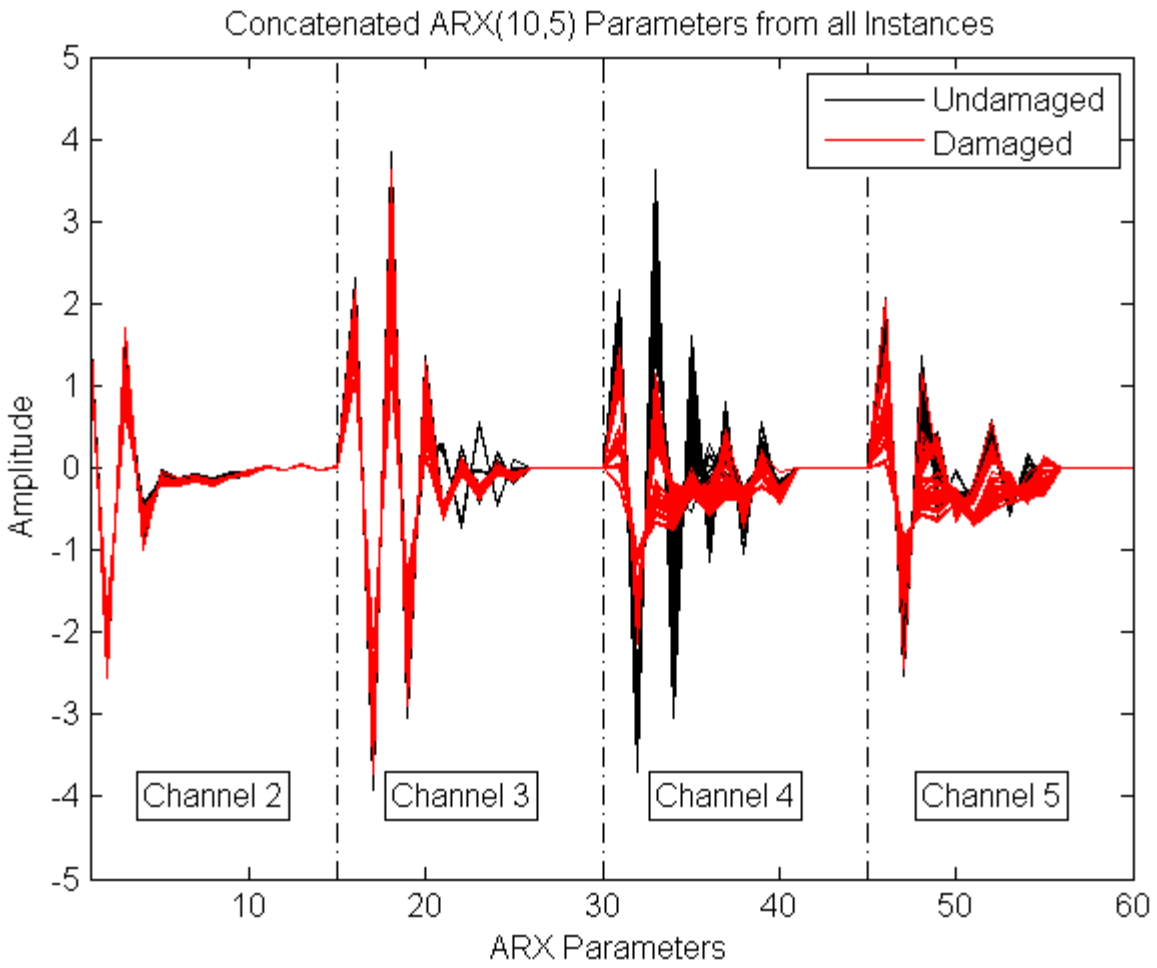
```
orders=[10 5 0];
arxOrder=15;
```

Estimation of the ARX Parameters:

```
[arxParameters]=arxModel_shm(dataset,orders);

[n m]=size(arxParameters);
```

Plot test data:

```
figure
plot(1:m,arxParameters(1:90,:)','k',1:m,arxParameters(90:170,:)','r')
title(['Concatenated ARX(',num2str(orders(1)),',',num2str(orders(2)),') Parameters from all
Instances'])
xlabel('ARX Parameters')
ylabel('Amplitude')
set(gca,'Xlim',[1 m])
M(1,1:9)='Undamaged'; M(2,1:7)='Damaged';
legend([line('color','k');line('color','r')],M);
w=5;
h(1)=line([m/4;m/4],[-w w],'color','k','lineStyle','-.');
h(2)=line([m/4*2;m/4*2],[-w w],'color','k','lineStyle','-.');
h(3)=line([m/4*3;m/4*3],[-w w],'color','k','lineStyle','-.');
text(4,-4,'Channel 2','Color','k','EdgeColor','k','BackgroundColor','w')
text(18,-4,'Channel 3','Color','k','EdgeColor','k','BackgroundColor','w')
text(33,-4,'Channel 4','Color','k','EdgeColor','k','BackgroundColor','w')
text(48,-4,'Channel 5','Color','k','EdgeColor','k','BackgroundColor','w')
```



Data Normalization for Novelty Detection

The Mahalanobis-based machine learning algorithm is used to normalize the features and to reduce each feature vector to a score.

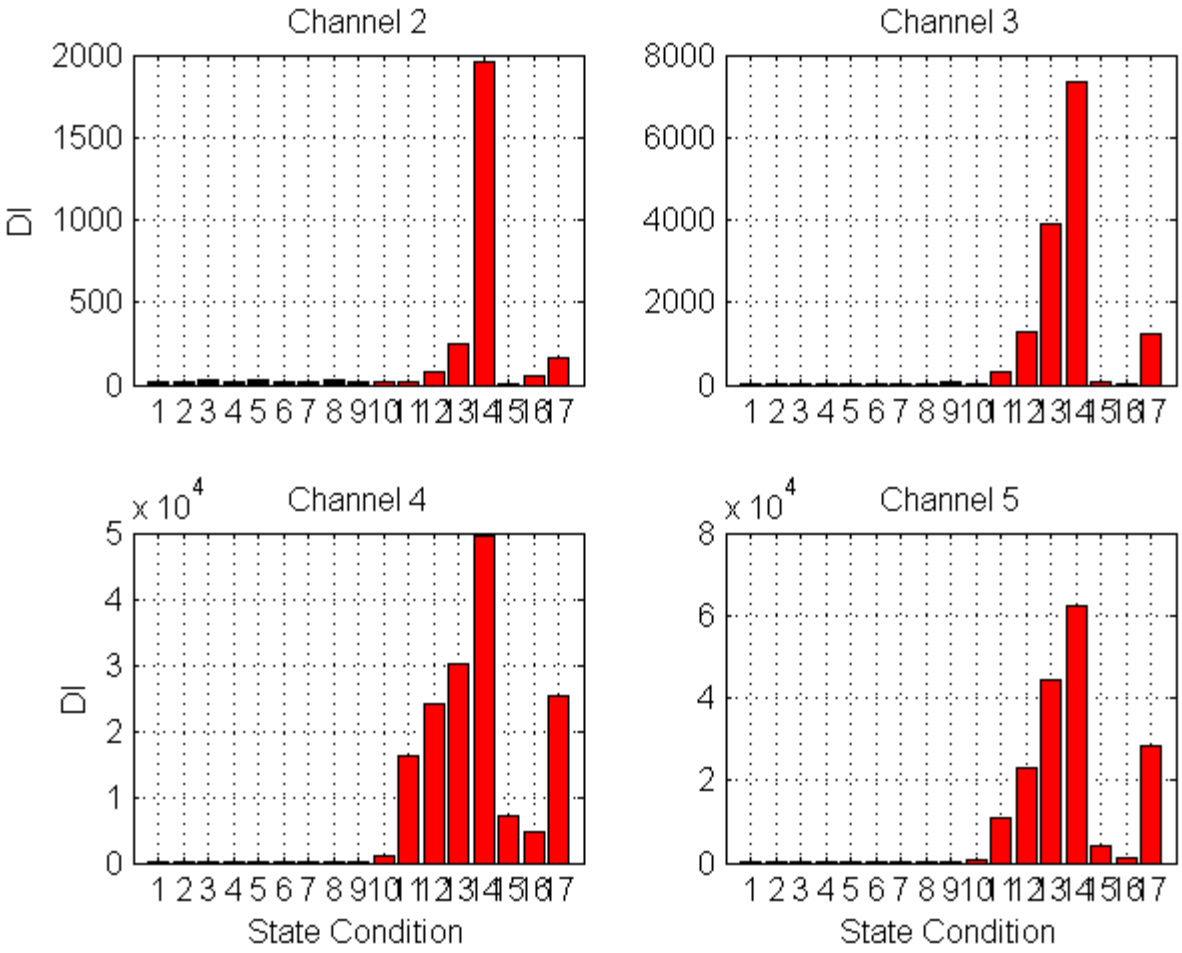
```
DI=zeros(17,5);
```

```
scoreData=zeros(17,arxOrder);

cnt=1;

for i=1:4;

    for j=1:9;
        learnData(j*9-8:j*9,:)=arxParameters(j*10-9:j*10-1,cnt:cnt+arxOrder-1);
    end

    scoreData(1:17,:)=arxParameters(10*(1:17),cnt:cnt+arxOrder-1);

    [model]=learnMahalanobis_shm(learnData);

    DI(:,i)=scoreMahalanobis_shm(scoreData,model);

    cnt=cnt+arxOrder;

end

DI=-DI;
```

Damage Location

Plot DIs from Channel 2-5:

```
figure

for i=1:4;

    subplot(2,2,i)
    bar(1:9,DI(1:9,i),'k'); hold on; bar(10:17,DI(10:17,i),'r')
    title(['Channel ',num2str(i+1)])
    set(gca,'Xlim',[0 length(DI)+1],'XTick',1:length(DI))
    grid on

    if i==3 || i==4, xlabel('State Condition'); end
    if i==1 || i==3, ylabel('DI'); end

end
```



The figure above shows that the ARX parameters perform better discrimination of the state conditions (in special at Channel 3) than the AR parameters. (See example.) Note that the location of damage can be done by comparing the number of outliers per channel.

See also:

Damage Location using AR Parameters from an Array of Sensors

Published with MATLAB® R2013a

Appropriate Autoregressive Model Order

Contents

Introduction

The goal of this example usage is to find out the appropriate autoregressive (AR) model order using an algorithm based on the partial autocorrelation function (PAF). One acceleration time history from the baseline condition is used to carry out the analysis.

Data sets from Channel 5 of the 3-story structure are used in this example usage. More details about the data sets can be found in 3-Story Data Sets.

Requires data3SS.mat dataset.

References:

Figueiredo, E., Park, G., Figueiras, J., Farrar, C., & Worden, K. (2009). Structural Health Monitoring Algorithm Comparisons using Standard Data Sets. Los Alamos National Laboratory Report: LA-14393.

SHMTools functions called:

`arModelOrder_shm`

Author: Elói Figueiredo

Date Created: September 01, 2009

Load Raw Data

Load data set:

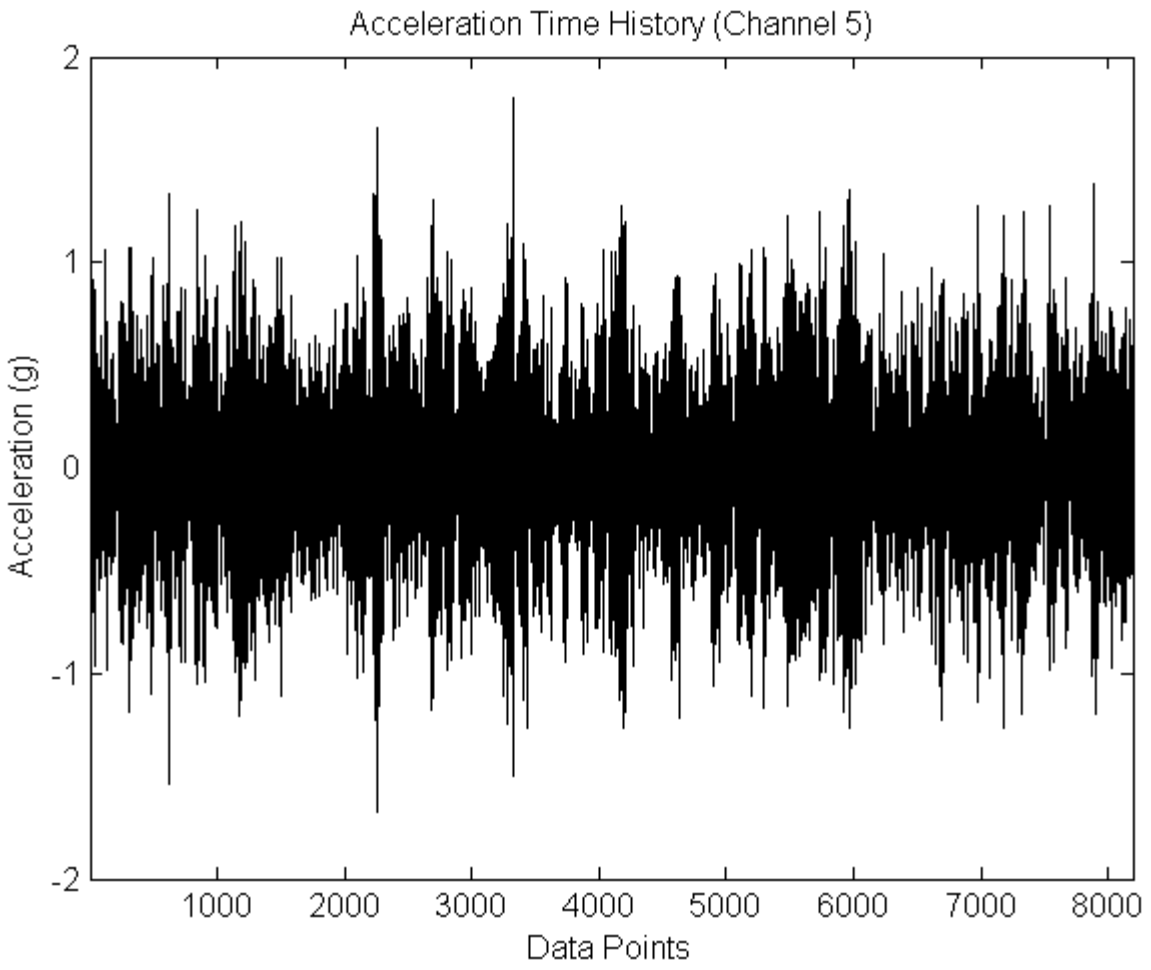
```
load('data3SS.mat');
```

Acceleration time history from the baseline condition (Channel 5):

```
data=dataset(:,5,1);
```

Plot time history:

```
figure;
plot(data,'k')
title('Acceleration Time History (Channel 5)')
xlabel('Data Points')
ylabel('Acceleration (g)')
set(gca,'YTick',-2:2,'Xlim',[1 8192],'Ylim',[-2 2])
```



Run Algorithm to find out the Appropriate AR Model Order

Using the PAF-based algorithm.

Set parameters:

```
method='PAF';
arOrderMax=30;
```

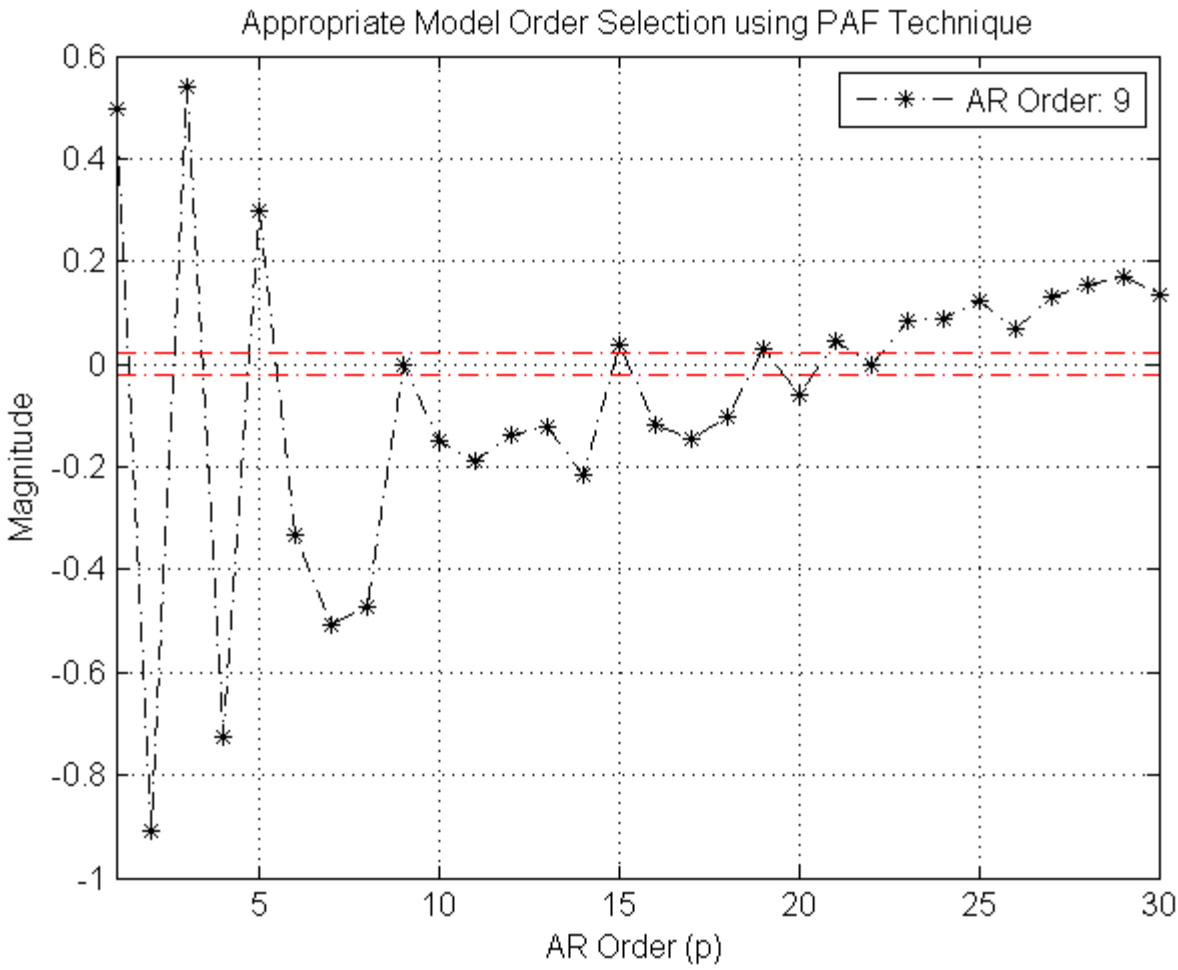
Run algorithm:

```
[meanARorder, arOrders, model]=arModelOrder_shm(data,method,arOrderMax,0.078);

outData=model.outData;
arOrderList=model.arOrderList;
CL=model.controlLimit;
```

Plot results along with the threshold:

```
figure
```

```
plot(arOrderList,outData,'-.*k')
title(['Appropriate Model Order Selection using ',method,' Technique'])
xlabel('AR Order (p)')
ylabel('Magnitude')
xlim([1 max(arOrderList)])
grid on
hold on

legend(['AR Order: ',num2str(meanARorder)])
line('XData',[1 max(arOrderList)],'YData',[CL(1) CL(1)],'Color','r','LineWidth',1,'LineStyle','-.')
if strcmp(method,'PAF'), line('XData',[1 max(arOrderList)],'YData',[CL(2)
CL(2)],'Color','r','LineWidth',1,'LineStyle','-.'); end
```



The PAF-based algorithm suggests an AR model of 9th order. This indication should be taken as a reference for a starting point. Other algorithms should be tried in order to find out a possible range of AR model orders. (Note that the arModelOrder_shm function contains other techniques, namely, the SVD, AIC, BIC, and RMS.)

Published with MATLAB® R2013a

Optimal Sensor Placement Using Modal Analysis Based Approaches

Contents

Introduction

Computes the 12-sensor optimal arrangments using the Fischer information method and the maximum norm method, plotting the resulting arrangements on the structure's geometry.

Requires data_OSPExampleModal.mat dataset.

References:

D. Kammer, “Sensor placement for on-orbit modal identification and correlation of large space structures,” Journal of Guidance, Control, and Dynamics, vol. 14, 1991, pp. 251-259.

M. Meo and G. Zumpano, “On the optimal sensor placement techniques for a bridge structure,” Engineering structures, vol. 27, 2005, pp. 1488–1497.

SHMTools functions called:

```
responseInterp_shm
addResp2Geom_shm
nodeElementPlot_shm
OSP_FisherInfoEIV_shm
getSensorLayout_shm
nodeElementPlot_shm
plotSensors_shm
OSP_MaxNorm_shm
```


Author: Eric Flynn

Date Created: June 26, 2009

Load Example Modal Data

Loads nodeLayout, elements, modeshapes, and respDOF

```
load ('data_OSPExampleModal.mat','nodeLayout','elements','modeShapes','respDOF')
```

Plot Mode Shapes

Convert the 3rd 1D mode vector to 2D response array using degree of freedom (DOF) definitions in respDOF

```
```

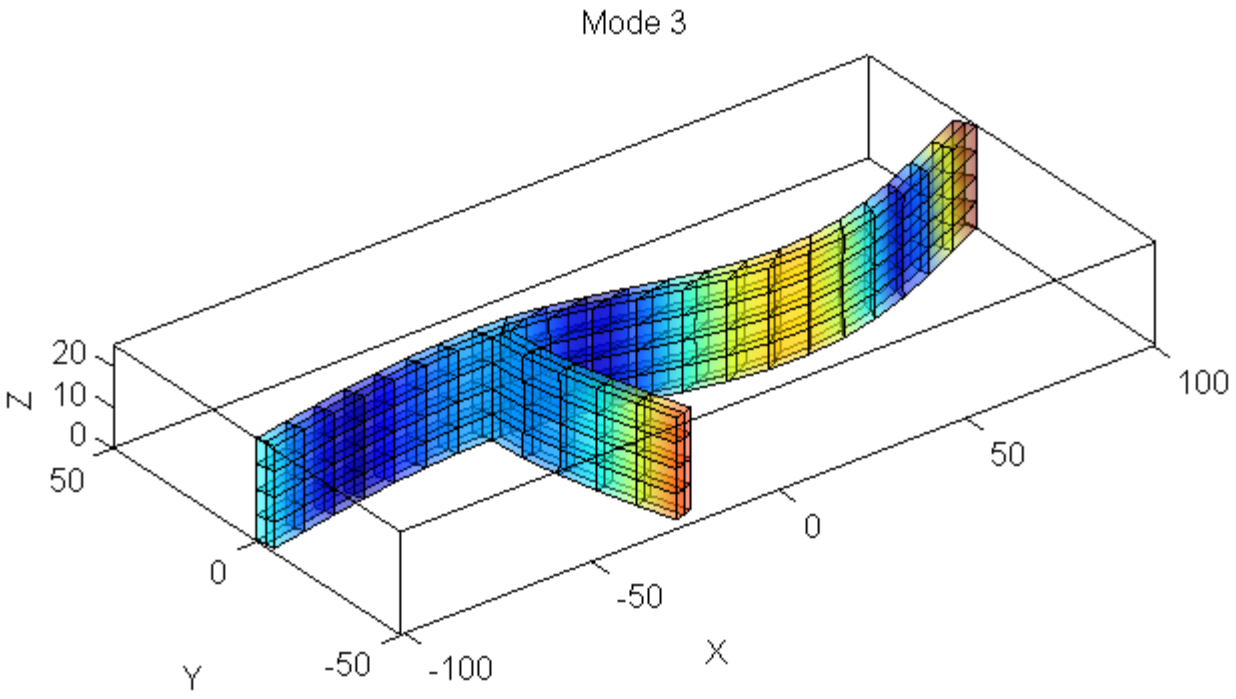
```
geomLayout=nodeLayout;
dispVec=modeShapes(:,3);
use3DInterp=false;

respXYZ=responseInterp_shm(geomLayout,dispVec,respDOF,use3DInterp);
```

Add the response vector to the node layout

```
[respLayout, respScale] = addResp2Geom_shm(geomLayout,respXYZ,[]);
```

Plot the response of the structure

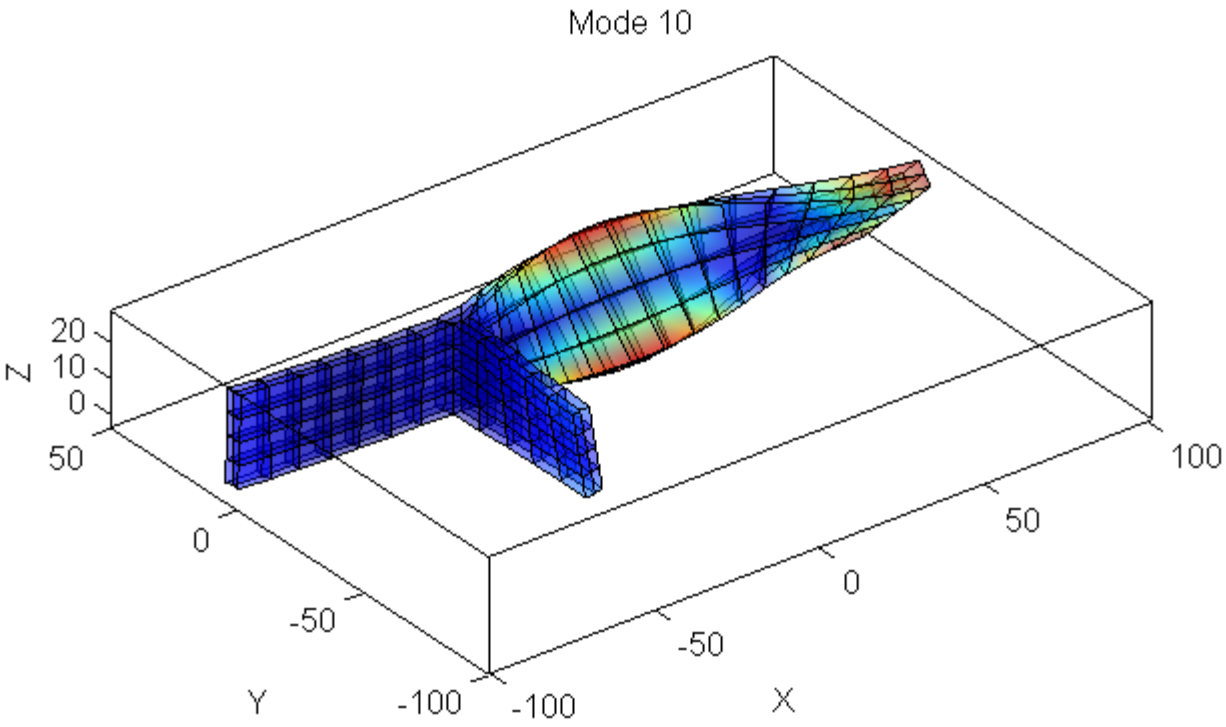
```
close all; aHandle=[];
nodeElementPlot_shm(respLayout,elements,respScale,aHandle);
xlabel('X');ylabel('Y');zlabel('Z'); title('Mode 3');
```



Do the same with the 10th mode vector

```
dispVec=modeShapes(:,10);
respXYZ=responseInterp_shm(geomLayout,dispVec,respDOF,use3DInterp);

[respLayout, respScale] = addResp2Geom_shm(geomLayout,respXYZ,[]);
```

```
nodeElementPlot_shm(respLayout,elements,respScale,aHandle);
xlabel('X');ylabel('Y');zlabel('Z'); title('Mode 10');
```



OSP Fisher Information Matrix, Effective Independence Method

Calculate the 12 optimal DOFs to place sensors by maximizing the determinant of the Fisher Information Matrix using the Equivalent Independence Method

```
numSensors=12;
covMatrix=[];

[opList,detQ] = OSP_FisherInfoEIV_shm(numSensors, modeShapes, covMatrix);
```

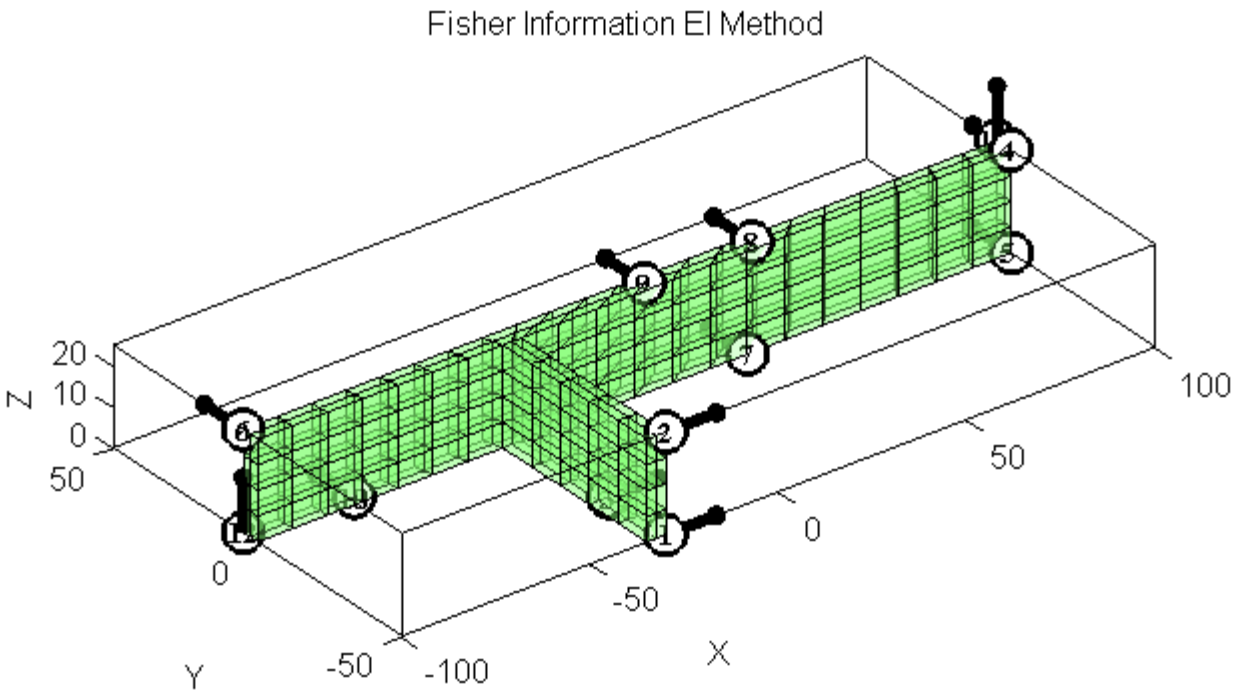
Convert 1D optimal DOF indices vector to 2D sensor layout array

```
sensorLayout=getSensorLayout_shm(opList,respDOF,nodeLayout);
```

Plot the sensors with the geometry

```
respScale=[];
aHandle=[];
```

```
aHandle=nodeElementPlot_shm(nodeLayout,elements,respScale,aHandle);
plotSensors_shm(sensorLayout,aHandle);
xlabel('X');ylabel('Y');zlabel('Z'); title('Fisher Information EI Method');
```



OSP Maximum Norm

Calculate the 12 optimal DOFs to place sensors by maximizing the norm of the response. The influence of the modes are weighted linearly. Sensors which are closer than a distance of 20 are "dueled" to maintain a minimum sepparation.

```
numSensors=12;
weights=13:-1:1;
dualingDistance=20;
geomLayout=nodeLayout;

[opList] =OSP_MaxNorm_shm(numSensors, modeShapes, weights, dualingDistance, respDOF, geomLayout);
```

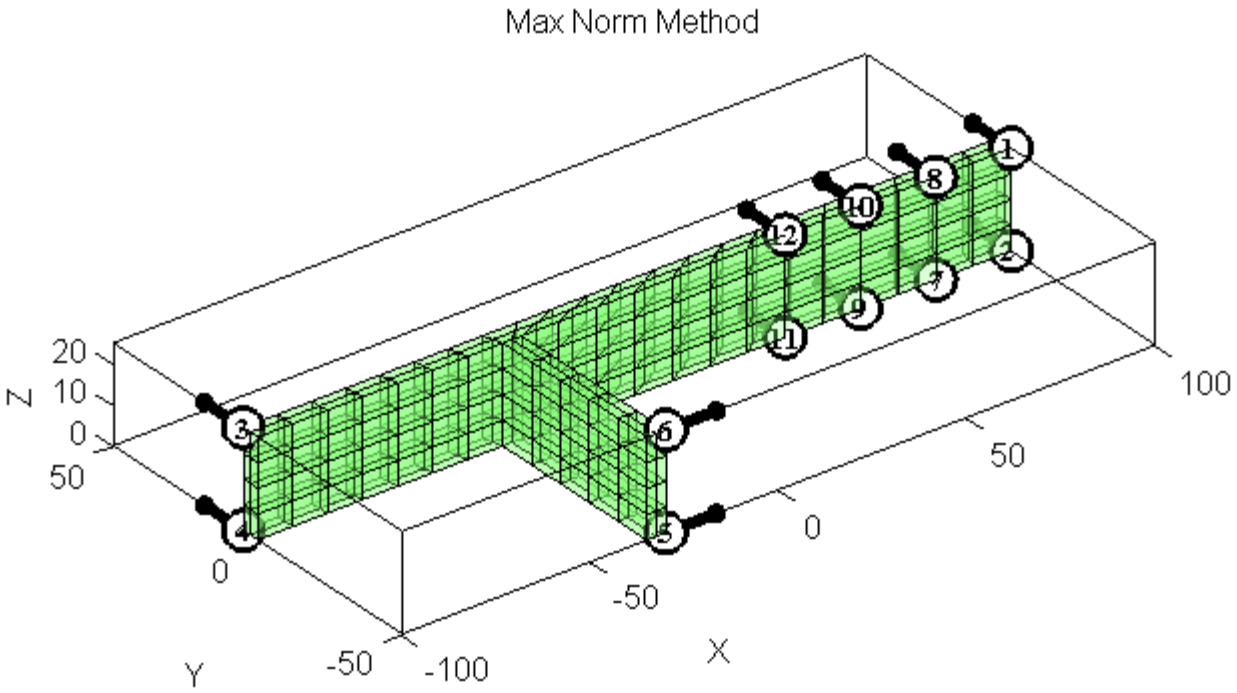
Convert 1D optimal DOF indices vector to 2D sensor layout array

```
sensorLayout=getSensorLayout_shm(opList,respDOF,nodeLayout);
```

Plot the sensors with the geometry

```
```

```
respScale=[];
aHandle=[];

aHandle=nodeElementPlot_shm(nodeLayout,elements,respScale,aHandle);
plotSensors_shm(sensorLayout,aHandle);
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Max Norm Method');
```



Published with MATLAB® R2013a

Data Normalization for Outlier Detection using Modal Properties

Contents

Introduction

The goal of this example usage is to discriminate time histories from undamaged and damaged conditions based on the outlier/novelty detection using the transfer function between Channel 1 (input force) and Channel 5 (output acceleration). The natural frequencies are used as damage-sensitive features and a machine learning algorithm based on nonlinear principal component analysis (NLPCA) is used to create damage indicators (DIs) invariant for feature vectors from normal condition and that increase when feature vectors are from damaged condition. Additionally, the receiver operating characteristic (ROC) curve is applied to evaluate the performance of this classifier.

Requires Neural Networking Toolbox and data3ss.mat dataset.

References:

Figueiredo, E., Park, G., Figueiras, J., Farrar, C., & Worden, K. (2009). Structural Health Monitoring Algorithm Comparisons using Standard Data Sets. Los Alamos National Laboratory Report: LA-14393.

Sohn, H., Worden, K., & Farrar, C. R. (2002). Statistical Damage Classification under Changing Environmental and Operational Conditions. Journal of Intelligent Material Systems and Structures , 13 (9), 561-574.

SHMTools functions called:

```
frf_shm
rpfit_shm
learnNLPCA_shm
scoreNLPCA_shm
ROC_shm
```


Author: Elói Figueiredo

Date Created: September 01, 2009

Load Raw Data

Load data set:

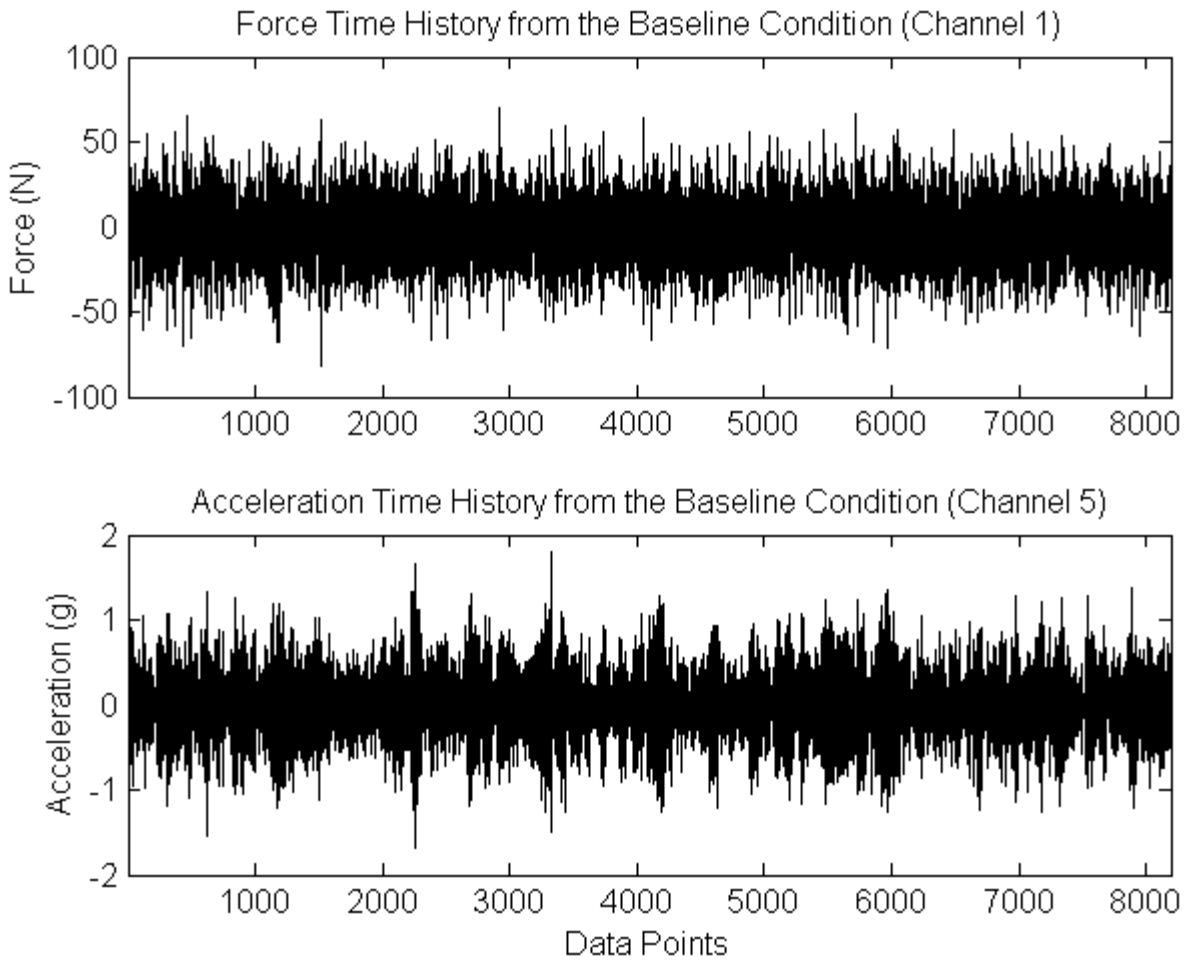
```
load('data3SS.mat');
```

Split into input force (Channel 1) and output acceleration time histories (Channel 5):

```
inputForce(:,:)=dataset(:,1,:);
outputAcc(:,:)=dataset(:,5,:);
```

Plot one force and one acceleration time history from the baseline condition:

```
figure

subplot(2,1,1)
plot(inputForce(:,1),'k')
title('Force Time History from the Baseline Condition (Channel 1)')
ylabel('Force (N)')
axis([1 8192 -100 100])

subplot(2,1,2)
plot(outputAcc(:,1),'k')
title('Acceleration Time History from the Baseline Condition (Channel 5)')
xlabel('Data Points')
ylabel('Acceleration (g)')
axis([1 8192 -2 2])
```



Extraction of Damage-Sensitive Features

Set parameters:

Matrix size:

```
```

```
[n m]=size(outputAcc);
```

Sampling frequency:

```
sf=320;
```

Window size to compute the FFT:

```
blockSize=n/4;
```

Frequency resolution:

```
df=sf/blockSize;
```

Parameter initialization:

```
G=zeros(blockSize,m);
```

Compute frequency response functions (FRFs):

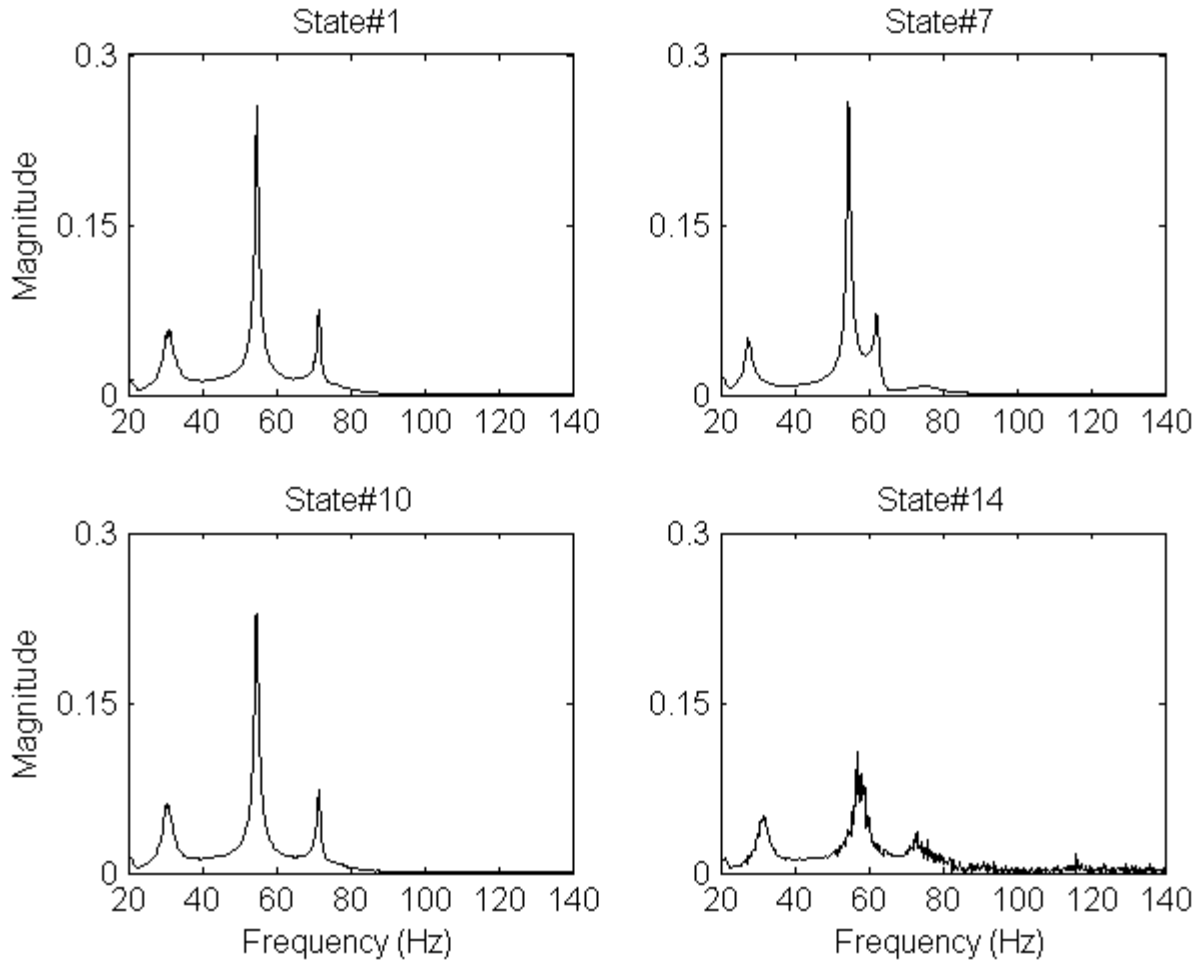
```
G=frf_shm(dataset(:,[1,5],:),blockSize,0.5,@hann, true);
```

Generate frequency vector

```
freqvec = linspace(0,160,blockSize/2)';
```

Plot four FRFs:

```
state=[1 7 10 14];

figure

for i=1:4;

    subplot(2,2,i)
    plot(freqvec,abs(G(:,state(i)*10)),'k')
    title(['State#',num2str(state(i))])
    set(gca,'Xlim',[20 140],'Ylim',[0 0.3],'Ytick',[0 0.15 0.3])

    if i==3 || i==4, xlabel('Frequency (Hz)'); end
    if i==1 || i==3, ylabel('Magnitude'); end

end
```



Frequency range to fit the FRF at each identified natural frequency:

```
frq=[26 36
    50 60
    65 75];
```

Number of modes to fit in each range:

```
nmodes=[1
    1
    1];
```

Number of extra terms to include:

```
exterms=4;
```

Run rpfit function to extract the natural frequencies as damage-sensitive features:

```
[res, freq] = rpfit_shm (G, df, frq, nmodes, exterms);
```

Training data:

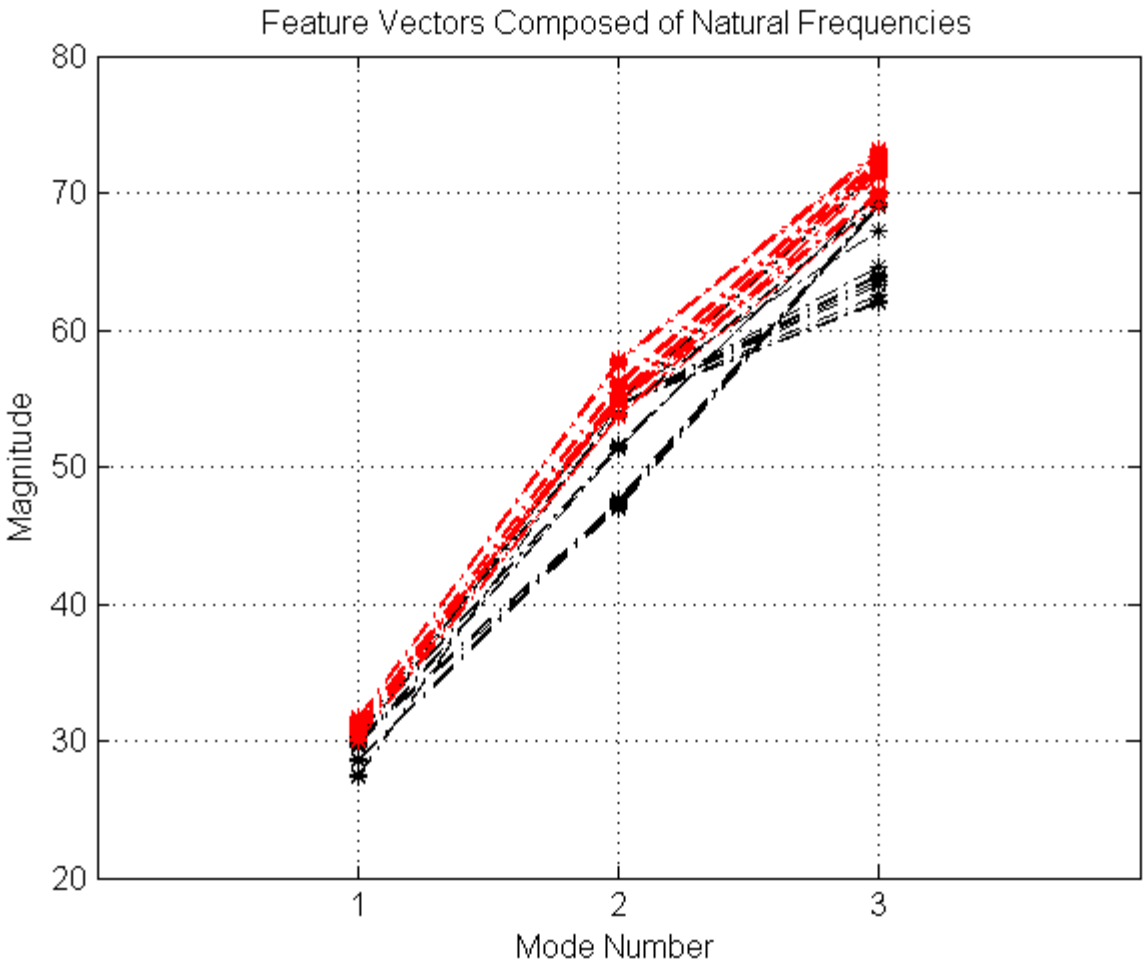
```
learnData=freq(:, 1:90)';
```

Test data:

```
scoreData=freq';
```

Plot damage-sensitive features:

```
figure
plot(1:3,scoreData(1:90,1:3),'-.*k',1:3,scoreData(91:170,1:3),'-.*r')
set(gca,'XLim',[0 4],'XTick',0:4,'XTickLabel',[' ';'1';'2';'3';' '],'YLim',[20 80])
title('Feature Vectors Composed of Natural Frequencies')
xlabel('Mode Number')
ylabel('Magnitude')
grid on
```



Statistical Modeling for Feature Classifiction

The NLPCA-based machine learning algorithm assumes two nodes in the bottleneck layer in order to represent the changes due to operational and environmental variations.

Run machine learning algorithm:

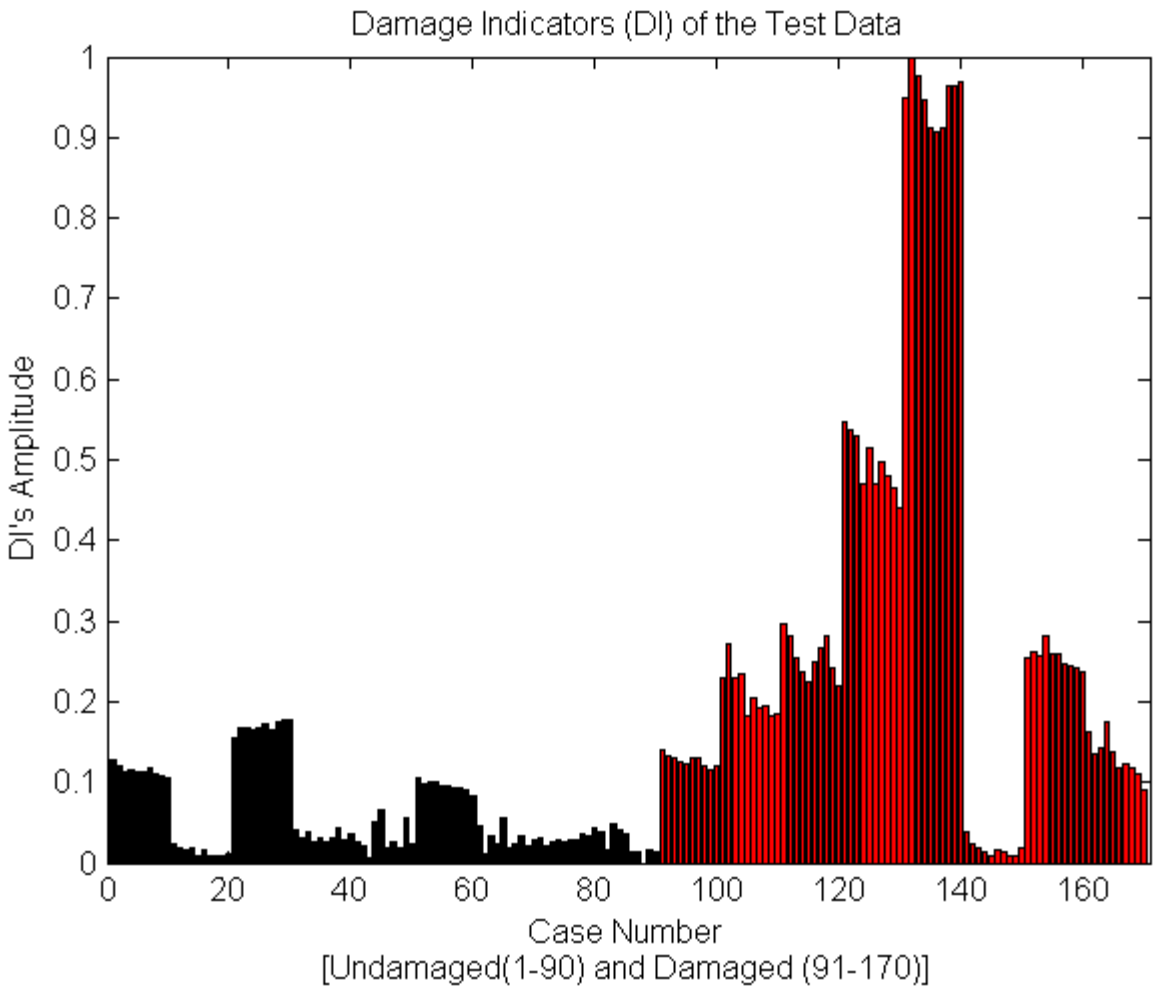
```
[model]=learnNLPCA_shm(learnData,2,5,5);

[DI]=scoreNLPCA_shm(scoreData,model);
```

Normalization procedure:

```
DIn=scaleMinMax_shm(-DI, 1, [0,1]);
```

Plot DIs:

```
figure;
bar(1:90,DIn(1:90),'k'); hold on; bar(91:170,DIn(91:170),'r')
title('Damage Indicators (DI) of the Test Data')
xlabel({'Case Number','[Undamaged(1-90) and Damaged (91-170)]'})
ylabel('DI''s Amplitude')
xlim([0 171])
```



ROC Curve

Flag all instances:

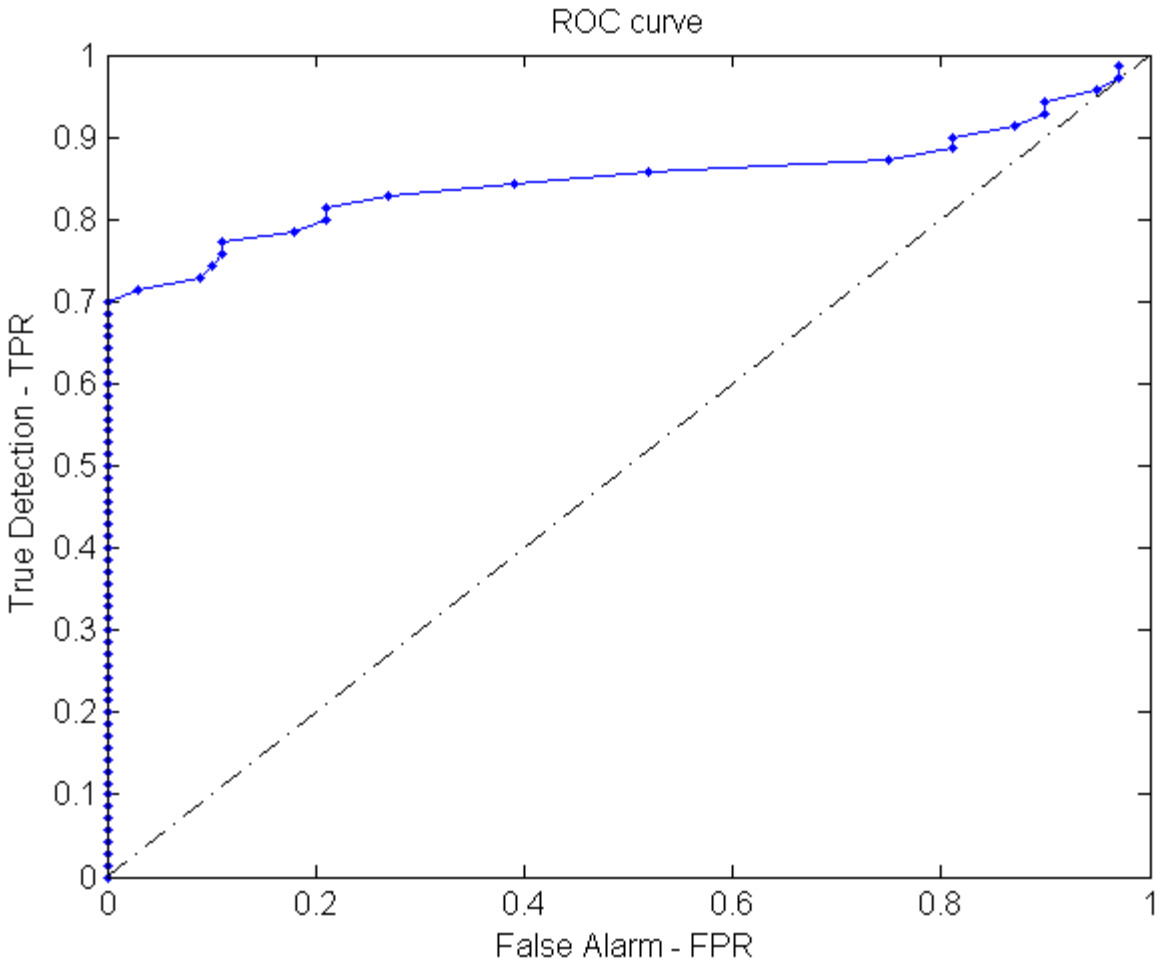
```
flag(1:100,1)=0; flag(101:170,1)=1;
```

Run ROC curve algorithm:

```
[TPR,FPR]=ROC_shm(DI,flag);
```

Plot ROC curve:

```
figure
plot(FPR,TPR,'.-b');
title('ROC curve')
xlabel('False Alarm - FPR')
ylabel('True Detection - TPR')
hold on
line('XData',[0 1],'YData',[0 1],'Color','k','LineWidth',1,'LineStyle','-.')
```



The ROC curve suggests that the approach (damage-sensitive features along with machine learning algorithm) is not appropriate to discriminate undamaged and damaged state conditions. (More details about ROC curves see example.)

Published with MATLAB® R2013a

Condition Based Monitoring Ball Bearing Fault Analysis

Contents

Introduction:

This example goes through a recommended semi-automated procedure outlined in Vibration-based Condition Monitoring by R. Randall. The data was collected from a machinery fault simulator with roller bearings supporting a shaft with a gear box attached to it and being driven by a belt drive. The script proceeds to remove periodic frequency components generated by the gearbox which tends to drown out the bearing fault vibrations. To remove the periodic components the signals first needed to be resampled to the gear shaft angular domain. The random components associated with bearing damage are separated and the spectral kurtosis for a number of healthy and damaged signals are compared to determine if filtering and demodulation are appropriate for the signal to maximize the detection of the ball bearing roller fault. In this example it turns out that the random component signal is optimal and no demodulation was needed. Then some damage features are computed for a number of instances containing modified vibration signals from a healthy bearing and a bearing with a roller element fault to compare the detection capabilities of each damage feature using receiver operating characteristic curse.

Requires data_CBM.mat dataset.

References:

[1] Randall, Robert., Vibration-based Condition Monitoring, Wiley and Sons, 2011.

SHMTools functions called:

```
arsAccel_shm
arsTach_shm
crestFactor_shm
demean_shm
discRandSeparation_shm
envelope_shm
fastKurtogram_shm
import_CBMData_shm
plotROC_shm
plotKurtogram_shm
ROC_shm
statMoments_shm
```

Author: Luke Robinson

Date Created: July 25, 2013

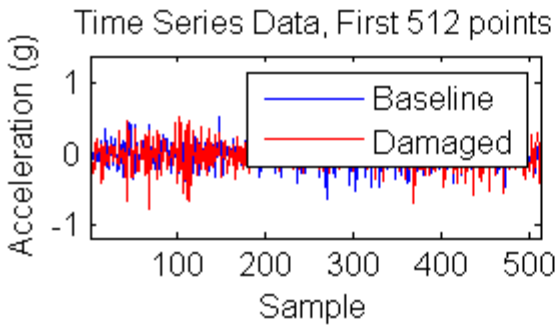
Begin Bearing Damage Analysis Script

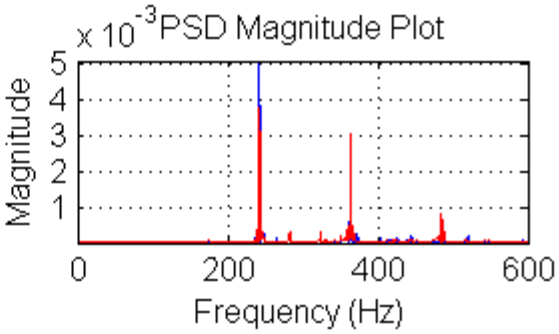
```
clear; clc

% Load Desired Data States and Channels for Outter Race Bearing Damage w/
% Channel 3: Accel Mounted on Top of Bearing Hounsing
[dataset, damageStates, stateList, Fs] = import_CBMData_shm ();

states = (stateList == 5) | (stateList == 6);
channels = [1,3]; % tachyometer and accel

X = dataset(:,channels,states);
damageStates = damageStates(states);
stateList = stateList(states);

iBaseline = find(stateList == 5);
iDamage = find(stateList == 6);
[X] = demean_shm(X);
```

1) Look at an Example Time and Frequency Series

```
%Plot Instance Number ##
instance = 16;

figure;
subplot(3,2,1) %Time Series Comparison
plot(X(:,2,iBaseline(instance)),'b')
hold on;
plot(X(:,2,iDamage(instance)),'r')
axis tight;
xlim([1 512]);
title('Time Series Data, First 512 points');
xlabel('Sample');
ylabel('Acceleration (g)');
legend('Baseline','Damaged')

subplot(3,2,2) %Frequency Domain Comparison
[psdMatrix, f, is1sided] = psdWelch_shm (X(:,2,[iBaseline(instance),...
    iDamage(instance)]), [], [], [], Fs, []);
plot(f,psdMatrix(:,1,1),'b');
hold on;
plot(f,psdMatrix(:,1,2),'r');
axis tight;
xlim([0 600]);
grid on;
title('PSD Magnitude Plot');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

% From the raw time signal and FFT you cannot gain much information at the
% raw stage for bearing damage. From a priori knowledge of the shaft speed
% and gear box data. It is known that the fundamental gear mesh frequency
```

```
% is approximately 120Hz and its harmonics are integer multiples of the
% fundamental mesh frequency i.e. (120Hz 240Hz 360Hz...) which can be
% clearly seen to be the dominate frequency components even though the
% data in question was retrieved from a sensor relatively distant from the
% gear box. These periodic components need to be removed from the vibration
% signal.
```





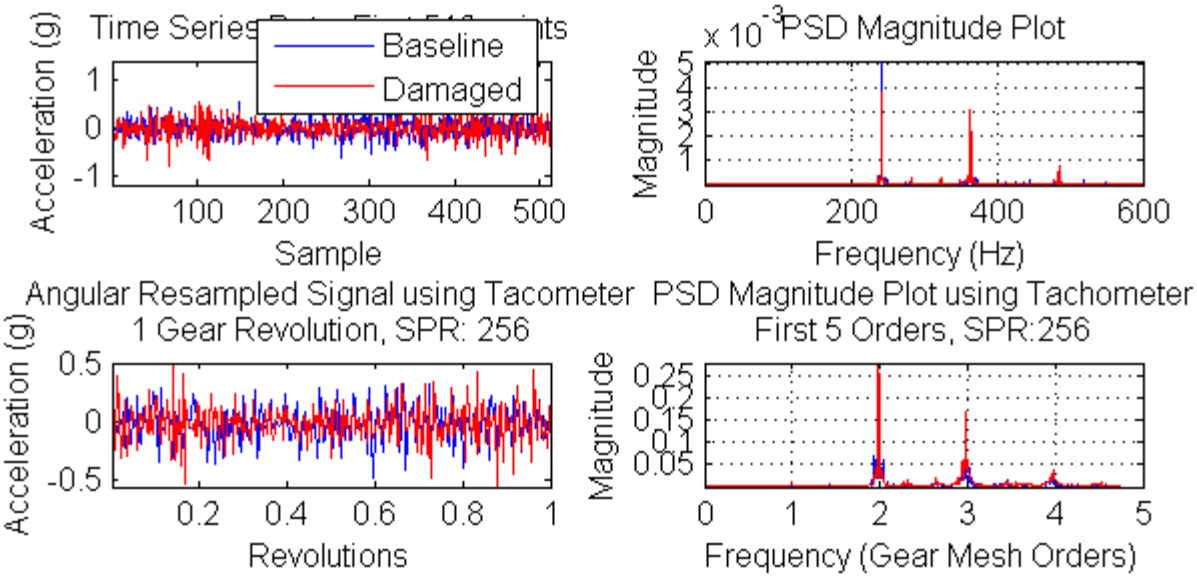
2A) Order Track using ARSTach.

```
% The data in this example was retrieved from a system that had minor speed
% fluctuations in the main shaft speed. The shaft speed variation was on
% the order or +/- 3RPM. As such it may be possible to simply use the
% signalARSAccel algorithm to order track the signal but for the sake of
% discussion both are used. signalARSTach uses a single pulse per rotation
% signal to resample a time domain signal that may have large speed
% fluctuations into a vibration signal tracked to orders of the shaft
% rotation in an equally space angular domain. This improves periodic
% frequency components that would have smeared from shaft speed
% fluctuations. The tachometer is located on the main shaft but the gear
% box is separated by a belt drive with a gear ratio equal to 1:3.71 and
% must be accounted for to resample to the gearbox shaft. nFilter is used
% for various anti-aliasing functionality in signalARSTach. signalARSTach
% by default uses a Kaiser windowed fir filter with a beta shape function
% set to 4 as its filter type.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%signalARSTach Input:
```

```
nFilter = 255;        %Anti-Alias Filter Length
samplesPerRev = 256; %Desired Samples per Rev
gearRatio = 1/3.71;   %Main Shaft:Gear Saft Ratio
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[xARSMatrixT, samplesPerRev] = arsTach_shm (X, nFilter, samplesPerRev, gearRatio);
```

Compare Resampled Angualar Series and Frequency Domain Content

```
subplot(3,2,3) %Angular Series Comparison
plot((1:samplesPerRev)/samplesPerRev,xARSMatrixT(1:samplesPerRev,1,iBaseline(instance)),'b')
hold on; plot((1:samplesPerRev)/samplesPerRev,xARSMatrixT(1:samplesPerRev,1,iDamage(instance)),'r')
axis tight;
title({'Angular Resampled Signal using Tacometer';...
    ['1 Gear Revolution, SPR: ' num2str(samplesPerRev)]});
xlabel('Revolutions');
ylabel('Acceleration (g)');

subplot(3,2,4) %Frequency Domain Comparison
[psdMatrix, f, is1sided] = psdWelch_shm (xARSMatrixT(:,1,[iBaseline(instance),...
    iDamage(instance)]), [], [], [], samplesPerRev/27, []);
plot(f,psdMatrix(:,1,1),'b');
hold on;
plot(f,psdMatrix(:,1,2),'r');
grid on;
axis tight;
xlim([0 5]);
title({'PSD Magnitude Plot using Tachometer';...
    ['First 5 Orders, SPR:' num2str(samplesPerRev)]});
xlabel('Frequency (Gear Mesh Orders)');
ylabel('Magnitude');
```



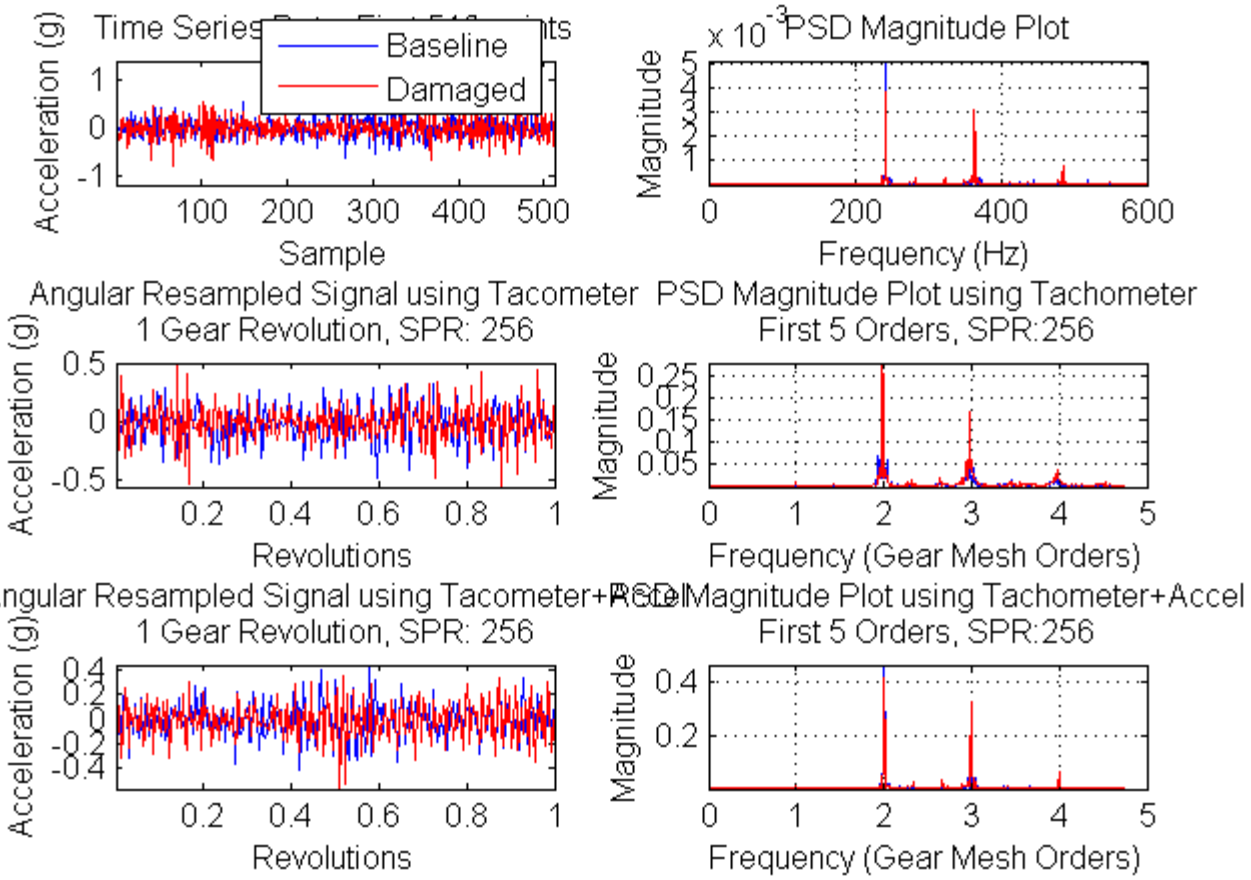
2B) Use signalARSAccel to further refine the angular resampling.

```
% signalARSAccel can be performed on vibration signals that have low
% varying speed or have been previously order tracked using a tachometer
% signal. Here we can gain some improvement over the previous method since
% the tachometer signal was not directly correlated to the gear shaft but
% have to be estimated. Due to round off error of the gear ratio as well as
% possible belt slipping. Some reduction in periodic smearing can be gained
% by angular resampling again using the phase information extracted from a
% periodic component of a vibration signal.

% With the knowledge that the gear consists of 27 teeth, and the sampling
% frequency resampled to a specified samplesPerRev. It can be determined
% by looking at the frequency domain that the 2nd harmonic is a good
% candidate to resample the signal too.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%signalARSAccel Input:
Fs = samplesPerRev;  %Sampling Frequency in FFT to Get Fc and Fsb
nFilter = 129;       %Anti-Aliasing/BandPass Filter Length
%samplesPerRev       %Keep as 1024
Fc = 53.65;          %Cycles/Rev (Chosen Harmonic Central Frequency)
Fsb = 0.45;          %Cycles/Rev (First Order Side Band Distance)
meshOrder = 2;       %2nd Order Harmonic
nGearTeeth = 27;     %27 Max Gear Teeth on Gear
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
[xARSMatrixA,samplesPerRev] = arsAccel_shm
(xARSMatrixT,Fs,nFilter,samplesPerRev,Fc,Fsb,meshOrder,nGearTeeth);
```

Compare Resampled Angualar Series and Frequency Domain Content

```
subplot(3,2,5) %Angular Series Comparison
plot((1:samplesPerRev)/samplesPerRev,xARSMatrixA(1:samplesPerRev,1,iBaseline(instance)),'b')
hold on; plot((1:samplesPerRev)/samplesPerRev,xARSMatrixA(1:samplesPerRev,1,iDamage(instance)),'r')
axis tight;
title({'Angular Resampled Signal using Tacometer+Accel';...
    ['1 Gear Revolution, SPR: ' num2str(samplesPerRev)]});
xlabel('Revolutions');
ylabel('Acceleration (g)');

subplot(3,2,6) %Frequency Domain Comparison
[psdMatrix, f, is1sided] = psdWelch_shm (xARSMatrixA(:,1,[iBaseline(instance),...
    iDamage(instance)]), [], [], [], samplesPerRev/27, []);
plot(f,psdMatrix(:,1,1),'b');
hold on;
plot(f,psdMatrix(:,1,2),'r');
axis tight;
grid on;
xlim([0 5]);
title({'PSD Magnitude Plot using Tachometer+Accel';...
    ['First 5 Orders, SPR:' num2str(samplesPerRev)]});
xlabel('Frequency (Gear Mesh Orders)');
ylabel('Magnitude');
```



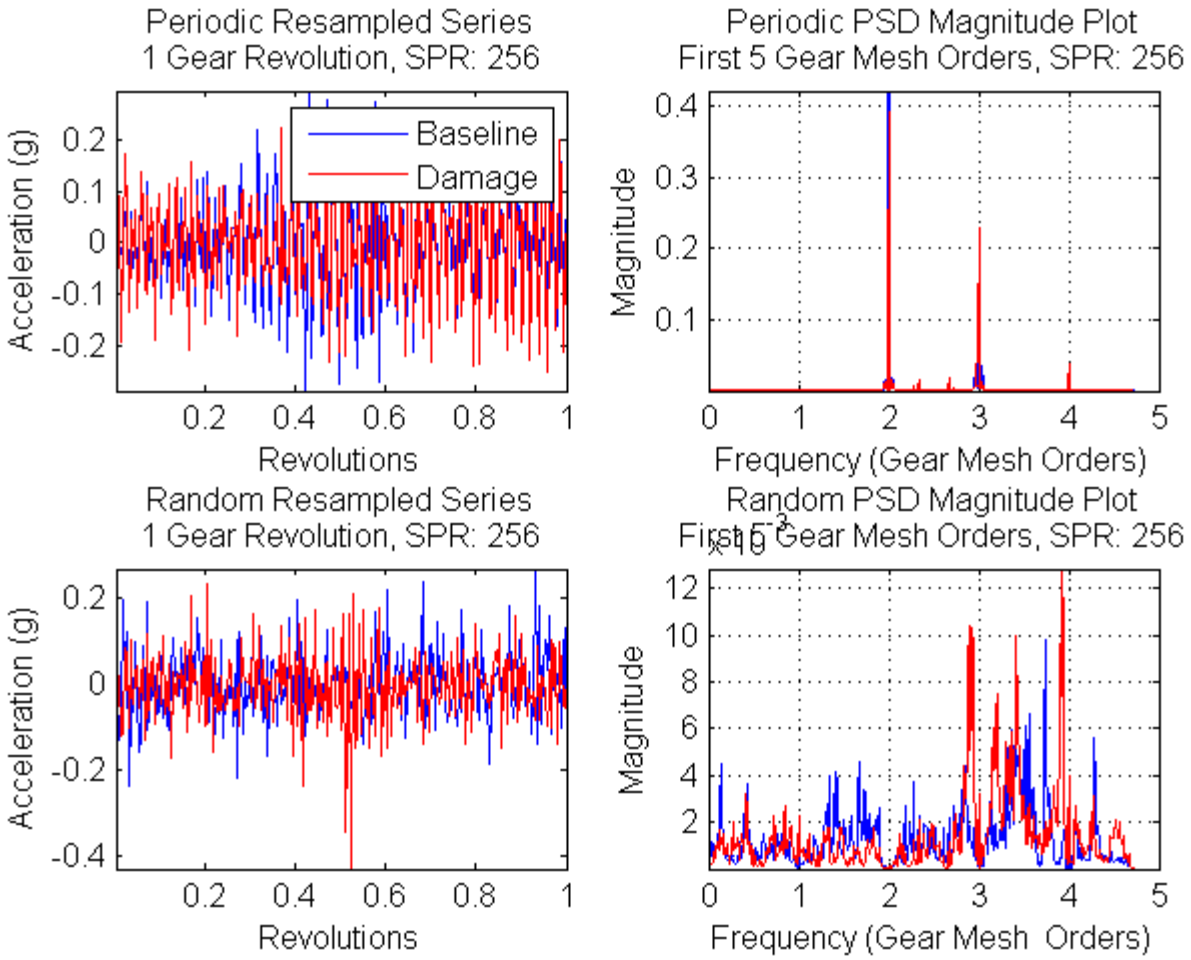
3) Remove Order Tracked Frequency using Discrete Random Separation

```
% Discrete/Random Separation uses two windows of the same signal with a
% specified delay or sample separation that are used to estimate an optimal
% filter to filter out the periodic or discrete frequency components from a
% signal. The filter is non-causal so filter delay and window separation
% have to be accounted for. Recommended values for nWin would be equal to
% the samples per revolution, nFFT = 2*nWin, and nWinDelay set to an
% integer multiple of nWin. Once the periodic signal if filtered out it is
% removed from the original signal using subtraction and the random signal
% is the signal minus the gear mesh harmonics.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%signalDRS Input:
nWin = samplesPerRev;       %Window Size
nOvlap = 0;         %Window Overlap
nFFT = 2*nWin;        %Number of FFT Bins (Filter Size)
nWinDelay = nWin;  %Window Separation Delay
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[xPMatrix, xRMatrix] = discRandSeparation_shm (xARSMatrixA, nWin, nOvlap, nFFT ,nWinDelay);
```

Compare Resampled Angualar Series and Frequency Domain Content

```
figure;
```

```
subplot(2,2,1) %Discrete Series Comparison
plot((1:samplesPerRev)/samplesPerRev,xPMatrix(1:samplesPerRev,1,iBaseline(instance)),'b')
hold on; plot((1:samplesPerRev)/samplesPerRev,xPMatrix(1:samplesPerRev,1,iDamage(instance)),'r')
axis tight;
title({'Periodic Resampled Series';...
       ['1 Gear Revolution, SPR: ' num2str(samplesPerRev)]});
xlabel('Revolutions');
ylabel('Acceleration (g)');
legend('Baseline','Damage')

subplot(2,2,2) %Discrete Frequency Domain Comparison
[psdMatrix, f, is1sided] = psdWelch_shm (xPMatrix(:,1,[iBaseline(instance),...
    iDamage(instance)]), [], [], [], samplesPerRev/27, []);
plot(f,psdMatrix(:,1,1),'b');
hold on;
plot(f,psdMatrix(:,1,2),'r');
axis tight;
grid on;
xlim([0 5]);
title({'Periodic PSD Magnitude Plot';...
      ['First 5 Gear Mesh Orders, SPR: ' num2str(samplesPerRev)]});
xlabel('Frequency (Gear Mesh Orders)');ylabel('Magnitude');

subplot(2,2,3) %Random Series Comparison
plot((1:samplesPerRev)/samplesPerRev,xRMatrix(1:samplesPerRev,1,iBaseline(instance)),'b')
hold on; plot((1:samplesPerRev)/samplesPerRev,xRMatrix(1:samplesPerRev,1,iDamage(instance)),'r')
axis tight;
title({'Random Resampled Series';...
       ['1 Gear Revolution, SPR: ' num2str(samplesPerRev)]});
xlabel('Revolutions');
ylabel('Acceleration (g)');

subplot(2,2,4) %Random Frequency Domain Comparison
[psdMatrix, f, is1sided] = psdWelch_shm (xRMatrix(:,1,[iBaseline(instance),...
    iDamage(instance)]), [], [], [], samplesPerRev/27, []);
plot(f,psdMatrix(:,1,1),'b');
hold on;
plot(f,psdMatrix(:,1,2),'r');
axis tight;
grid on;
xlim([0 5]);
title({'Random PSD Magnitude Plot';...
       ['First 5 Gear Mesh Orders, SPR: ' num2str(samplesPerRev)]});
xlabel('Frequency (Gear Mesh  Orders)');
ylabel('Magnitude');
```



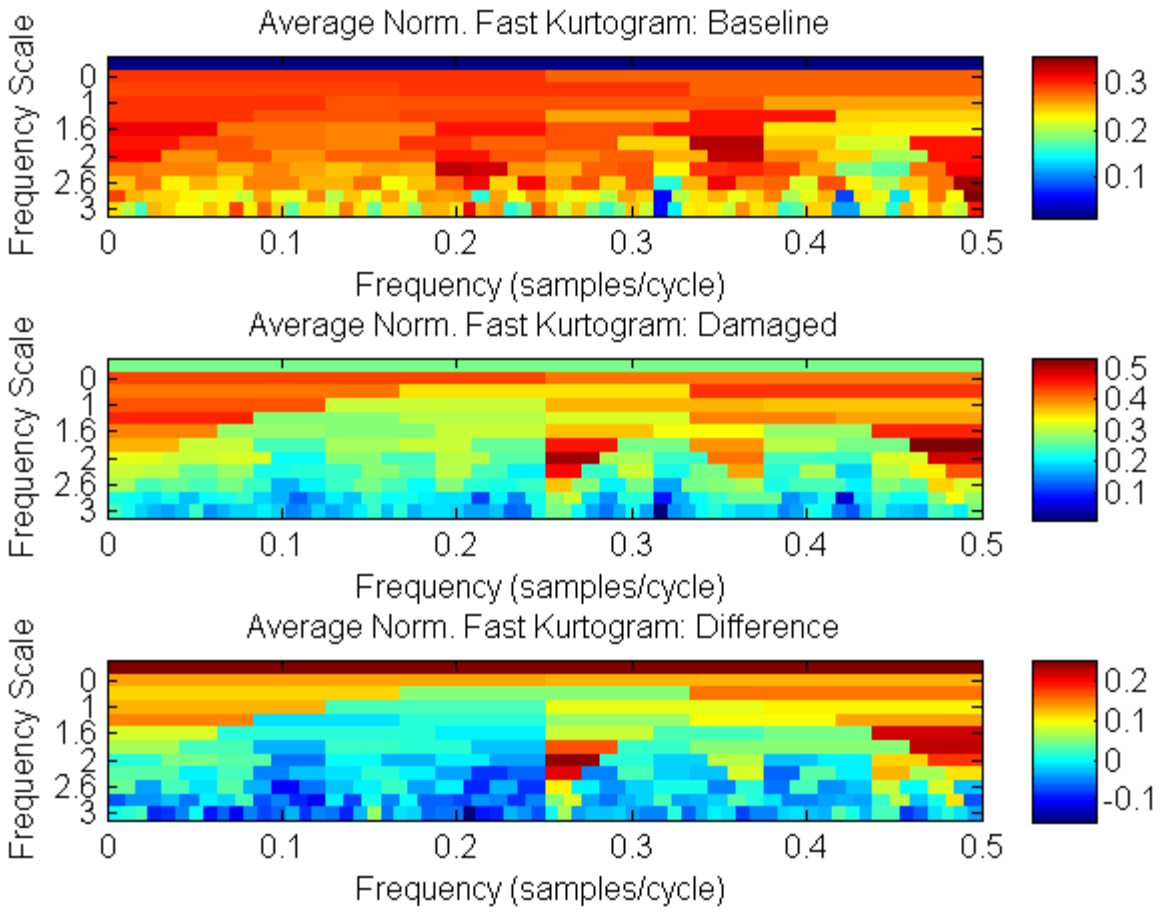
4) Look at Normalized Average Fast Spectral Kurtosis Image (Random Component)

```
% Looking at the spectral kurtosis after the gear mesh frequencies have
% been removed from the signal shows which frequency bands have the largest
% spectral kurtosis and which may be optimal frequency bands for
% demodulation to improve damage detection features. The fast spectral
% kurtosis images were normalized to their maximum values then averaged to
% get a more general idea of the differences not just in magnitude but
% frequency band. Some trials may experience random transients which have a
% very large effect on the spectral kurtosis but are not necessarily
% damage.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%fastKurtogram Input:
Fs = 1;            %Normalized Frequency
nKScale = 6;       %Number of Frequency Band Scales
flagPave3 = true;  %Include 1/3 Frequency Bands
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[fastKurtMatrix,levels,f] = fastKurtogram_shm (xRMatrix, Fs, nKScale, flagPave3);

%Normalize FSK Images to max values
for i = 1:64
fastKurtMatrix(:,:,1,iBaseline(i)) = fastKurtMatrix(:,:,1,iBaseline(i))./...
    max(max(fastKurtMatrix(:,:,1,iBaseline(i))));
fastKurtMatrix(:,:,1,iDamage(i)) = fastKurtMatrix(:,:,1,iDamage(i))./...
    max(max(fastKurtMatrix(:,:,1,iDamage(i))));
```

```
end

%Determine Average FSK for Baseline and Damaged Conditions
fSKMeanBaseline = sum(fastKurtMatrix(:,:,1,1:iBaseline(end)),4)./...
    length(iBaseline);
fSKMeanDamage = sum(fastKurtMatrix(:,:,1,iDamage(1):iDamage(end)),4)./...
    length(iDamage);
```

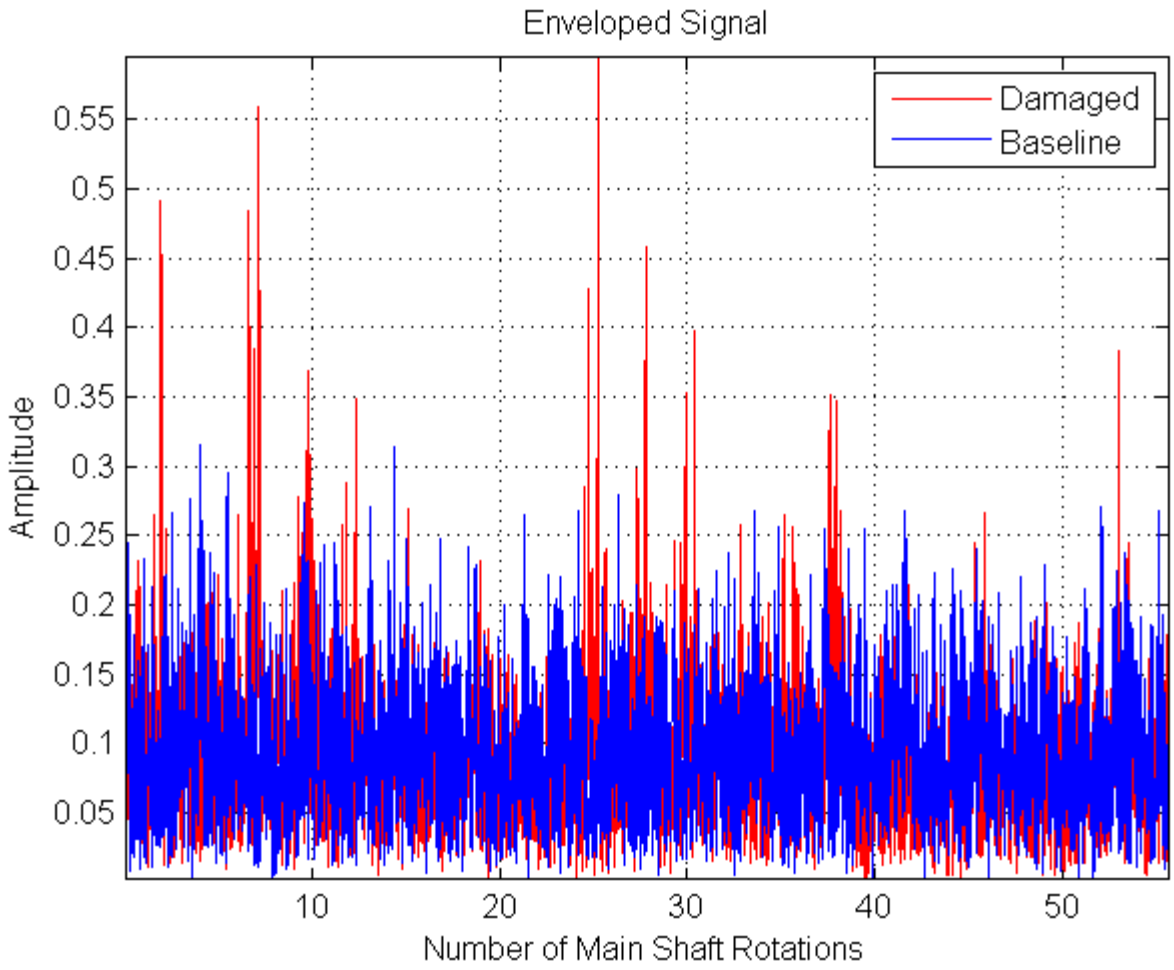
Plot FSK Images for Baseline Damage and Difference.

```
figure;
ax = subplot(3,1,1);
plotKurtogram_shm (fSKMeanBaseline, [],[], f, levels, ax);
title({'Average Norm. Fast Kurtogram: Baseline'}); ylabel('Frequency Scale');
xlabel('Frequency (samples/cycle)');

ax = subplot(3,1,2);
plotKurtogram_shm (fSKMeanDamage, [],[], f, levels, ax);
title({'Average Norm. Fast Kurtogram: Damaged'}); ylabel('Frequency Scale');
xlabel('Frequency (samples/cycle)')

ax = subplot(3,1,3);
plotKurtogram_shm ((fSKMeanDamage-fSKMeanBaseline), [],[], f, levels, ax);
title({'Average Norm. Fast Kurtogram: Difference'})
ylabel('Frequency Scale'); xlabel('Frequency (samples/cycle)');

% From looking at the Average Spectral Kurtosis of the Baseline Condition
% and the Damage Condition it appears that the frequency band with the most
% kurtosis is the entire range from 0 to the Nyquist frequency. So no
% demodulation is done in this example.
```



5) Compute the Demodulated Enveloped Signal of the Random Component

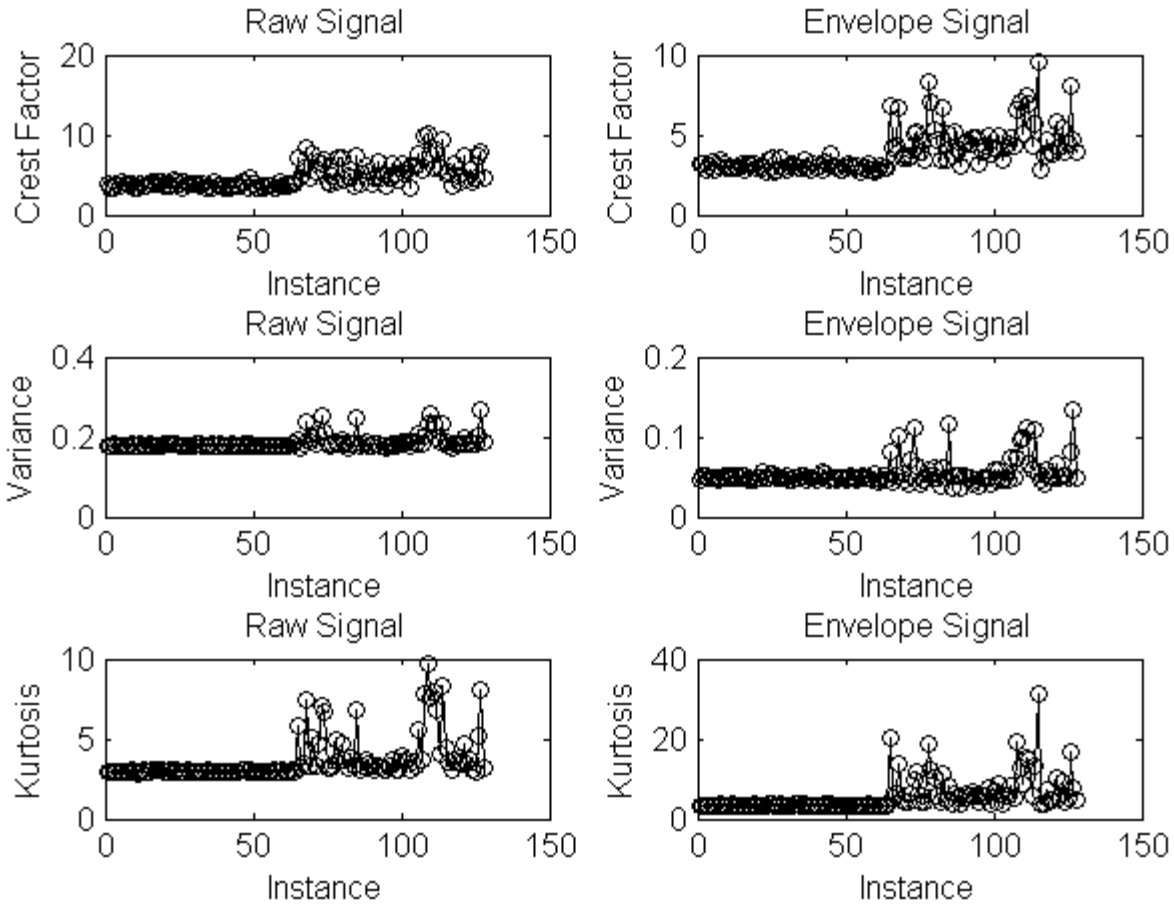
```
[envelopeMatrix] = envelope_shm (xRMatrix);

figure;
rotation = (1:size(envelopeMatrix,1))/gearRatio/samplesPerRev;
plot(rotation,envelopeMatrix(:,1,iDamage(instance)),'r');
hold on;plot(rotation,envelopeMatrix(:,1,iBaseline(instance)));
title('Enveloped Signal');xlabel('Number of Main Shaft Rotations');
ylabel('Amplitude');axis tight;grid on;legend('Damaged','Baseline')

% The plot provided shows an example of when impulses from the ball
% bearing elements are actively hitting a fault.
```



6) Look at Some Feature Types

```
% To be compared are 6 damage types. The following damage types are crest
% factor, kurtosis, and variance which are calculated using the enveloped
% signal of the random component after gear mesh frequencies have been
% removed. To see if any improvement in detectability has been achieved a
% comparison is made against the same damage features of the raw signal
% with no processing at all which has been used in some studies of ball
% bearing damage.

%Compute Matrix Damage Features

%Raw Signal Damage Features
[cfRaw] = crestFactor_shm(X(:,2,:));
[statisticsFV] = statMoments_shm (X(:,2,:));
[varianceRaw] = statisticsFV(:,2);
[kurtRaw] = statisticsFV(:,4);

%Envelope Signal Damage Features
[cf] = squeeze(crestFactor_shm(envelopeMatrix));
[statisticsFV] = statMoments_shm (envelopeMatrix);
[variance] = statisticsFV(:,2);
[kurt] = statisticsFV(:,4);

%Plot Damage Features
features = [cfRaw,cf,varianceRaw,variance,kurtRaw,kurt];
featNames = {'Raw Crest Factor','Envelope Crest Factor','Raw Variance',...
```

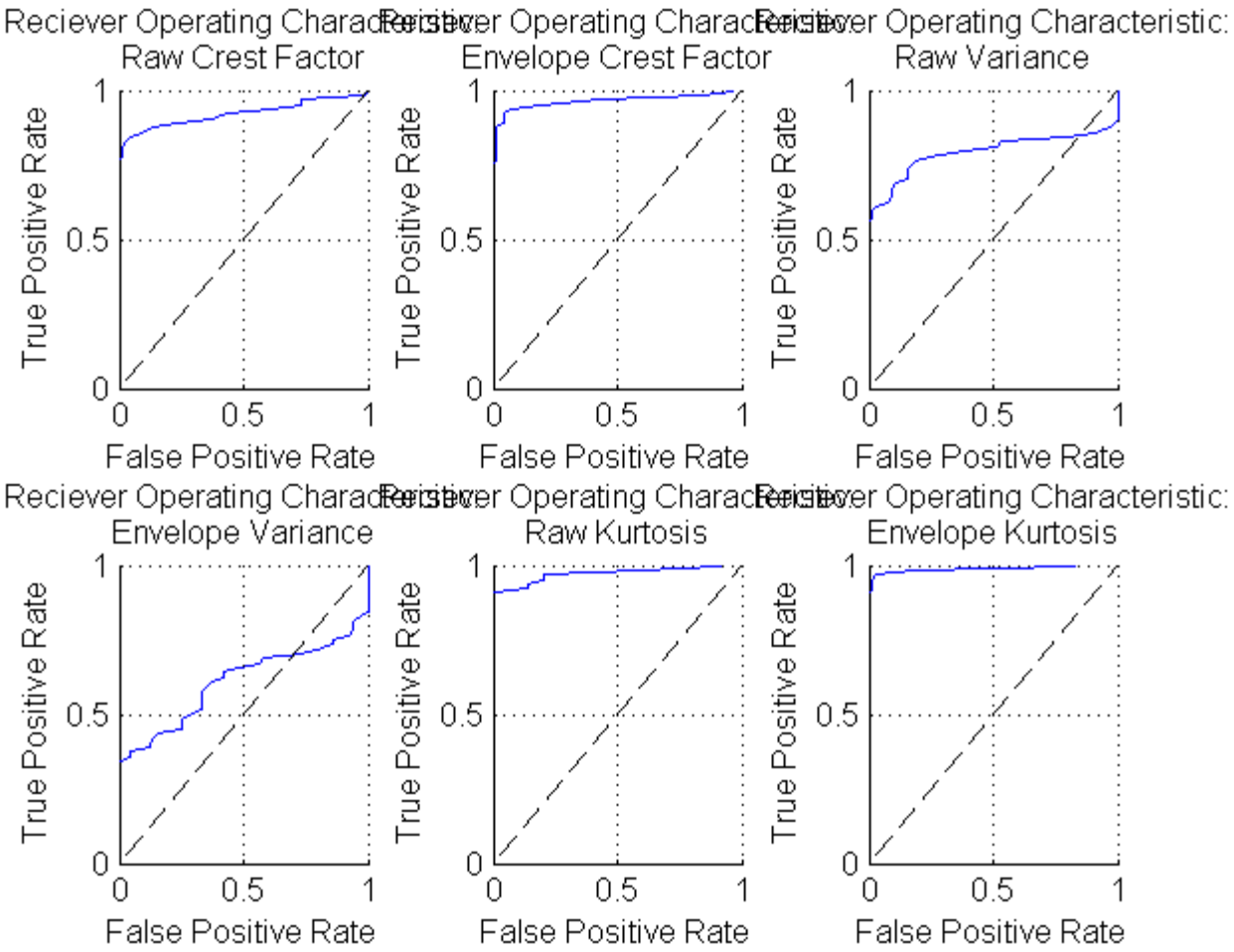
```
    'Envelope Variance','Raw Kurtosis','Envelope Kurtosis'};
plotFeatures_shm(features,[],[],{'Raw Signal','Envelope Signal',...
    'Raw Signal','Envelope Signal','Raw Signal','Envelope Signal'},...
    {'Crest Factor','Crest Factor','Variance','Variance','Kurtosis',...
    'Kurtosis'},[]);
```



7) Plot ROC Curves

```
% To compare the damage features detectability statistically, receiver
% operating characteristic curves can be used to show the probability of a
% detection vs. the probability of false alarm. Damage features with a high
% probability of detection to false alarm rate are optimal detectors. From
% the ROC curves the kurtosis and creast factor of the enveloped signal
% have slightly better performance than usign the raw signal.

numPts = [];
thresholdType = 'above';

figure;

for i=1:length(featNames)
    ax = subplot(2,3,i);
    [TPR, FPR] = ROC_shm (features(:,i), damageStates, [], thresholdType);
    plotROC_shm (TPR, FPR, 'linear', ax);
    hold on;
    plot([0 1],[0 1],'--k');
    grid on;
```

```
    axis([0 1 0 1]);
    title({'Reciever Operating Characteristic:',featNames{i}});
end
```



Published with MATLAB® R2013a

Condition Based Monitoring Gearbox Fault Analysis

Contents

Introduction:

This usage script focuses on extracting a number of damage features for gearbox diagnostics and compares them statistically to determine which would be a good candidate for detecting worn tooth damage of a gearbox vibrations signal. Vibration signals were collected of over a number of instances for a baseline healthy state as well as worn tooth damage state. The bearings on the main shaft used were fluid film bearings which supported the main shaft that drove the gear box. The usage script begins by loading the vibration signals and angular resampling the vibration signal to a specified samples per revolution of the gear shaft. The resampled signal is compared to the raw time signal to demonstrate the improvement of the gear mesh components. The power spectral densities are looked at to see if any visible damage has occurred between the damage state and the baseline state. The residual difference and band pass mesh signals are filtered for the angular resampled signal using fir filtering methods which are used by various gearbox damage features. Then the script plots some time frequency domain figures of merit to see if any useful information can be extracted. Of the four time frequency domains presented the continuous wavelet scalogram is chosen for further processing. The Hoelder exponent is computed from the continuous wavelet scalogram for damage detection in a later damage feature extraction method. Ten damage features are computed from the signals processed earlier on and compared using receiver operating characteristic curves to see which have better performance for the data set.

Requires data_CBM.mat dataset.

References:

[1] Randall, Robert., Vibration-based Condition Monitoring, Wiley and Sons, 2011.

[2] Lebold, M.; McClintic, K.; Campbell, R.; Byington, C.; Maynard, K., Review of Vibration Analysis Methods for Gearbox Diagnostics and Prognostics, Proceedings of the 54th Meeting of the Society for Machinery Failure Prevention Technology, Virginia Beach, VA, May 1-4, 2000, p. 623-634.

SHMTools functions called:

arsTach_shm crestFactor_shm cwtScalogram_shm demean_shm dwvd_shm filter_shm fir1_shm fm0_shm fm4_shm hoelderExp_shm import_CBMData_shm lpcSpectrogram_shm m6a_shm m8a_shm na4m_shm nb4m_shm rms_shm plotPSD_shm plotROC_shm plotTimeFreq_shm plotScalogram_shm psdWelch_shm ROC_shm statMoments_shm stft_shm window_shm

Author: Luke Robinson

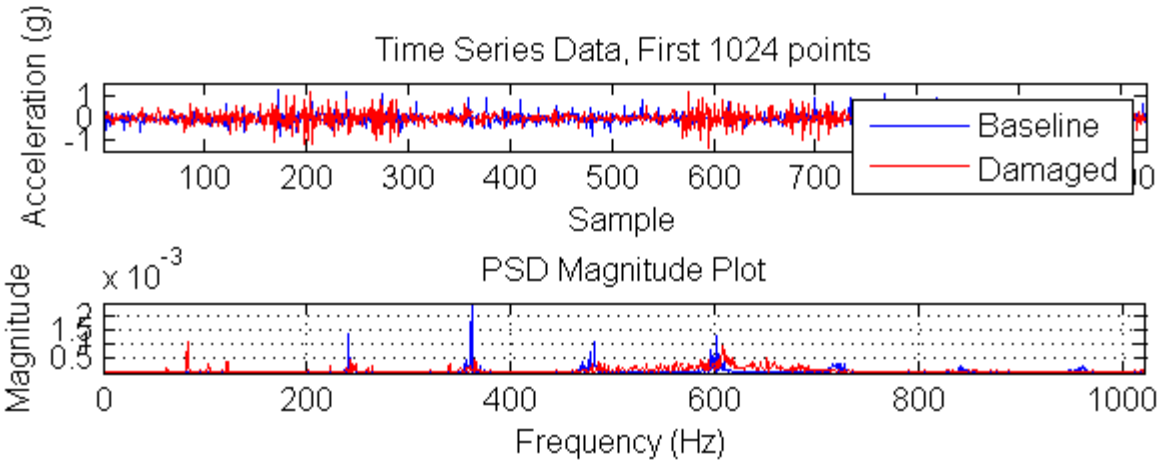
Date Created: July 25, 2013

Begin Gear Box Damage Analysis Script

```
clear; clc

% Load Desired Data States and Channels for Outter Race Bearing Damage w/
% Channel 2: Accel Mounted on Gearbox
[dataset, damageStates, stateList, Fs] = import_CBMData_shm ();

states = (stateList == 1) | (stateList == 3);
channels = [1,2]; % tachyometer and accel

X = dataset(:,channels,states);
damageStates = damageStates(states);
stateList = stateList(states);

iBaseline = find(stateList == 1);
iDamage = find(stateList == 3);
[X] = demean_shm(X);
```

1) Look at an Example Time and Frequency Series

```
%Plot instance Number ##
instance = 3;

figure;
subplot(4,1,1) %Time Series Comparison
plot(X(:,2,iBaseline(instance)),'b')
hold on;
plot(X(:,2,iDamage(instance)),'r')
axis tight;
xlim([1 1024]);
title('Time Series Data, First 1024 points');
xlabel('Sample');
ylabel('Acceleration (g)');
legend('Baseline','Damaged')

subplot(4,1,2) %Frequency Domain Comparison
[psdMatrix, f, is1sided] = psdWelch_shm (X(:,2,[iBaseline(instance),...
    iDamage(instance)]), [], [], [], Fs, []);
plot(f,psdMatrix(:,1,1),'b');
hold on;
plot(f,psdMatrix(:,1,2),'r');
axis tight;
grid on;
title('PSD Magnitude Plot');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
```

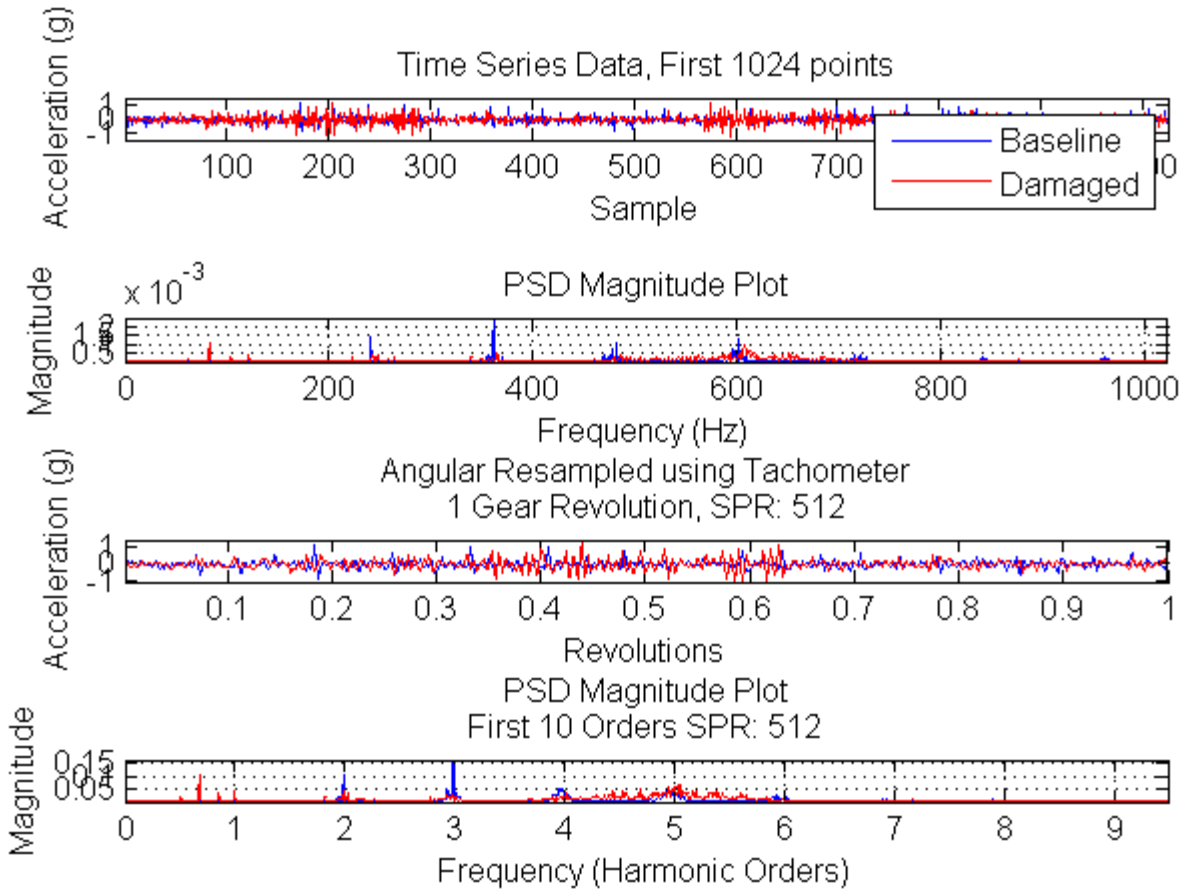


2) Order Track using ARSTach and ARSAccel for further refinement.

```
% The data in this example was retrieved from a system that had minor speed
% fluctuations in the main shaft speed. The shaft speed variation was on
% the order or +/- 3RPM. signalARSTach uses a single pulse per rotation
% signal to resample a time domain signal that may have large speed
% fluctuations into a vibration signal tracked to orders of the shaft
% rotation in an equally space angular domain. This improves periodic
% frequency components that would have smeared from shaft speed
% fluctuations. The tachometer is located on the main shaft but the gear
% box is separated by a belt drive with a gear ratio equal to 1:3.71 and
% must be accounted for to resample to the gearbox shaft. nFilter is used
% for various anti-aliasing functionality in signalARSTach. signalARSTach
% by default uses a Kaiser windowed fir filter with a beta shape function
% set to 4 as its filter type.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%arsTach_shm Input:
nFilter = 255;        %Anti-Alias Filter Length
samplesPerRev = 512;  %Desired Samples per Rev
gearRatio = 1/3.71;   %Main Shaft:Gear Saft Ratio
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[xARSMatrixT, samplesPerRev] = arsTach_shm(X, nFilter, samplesPerRev, gearRatio);
```

Compare Resampled Angualar Series and Frequency Domain Content

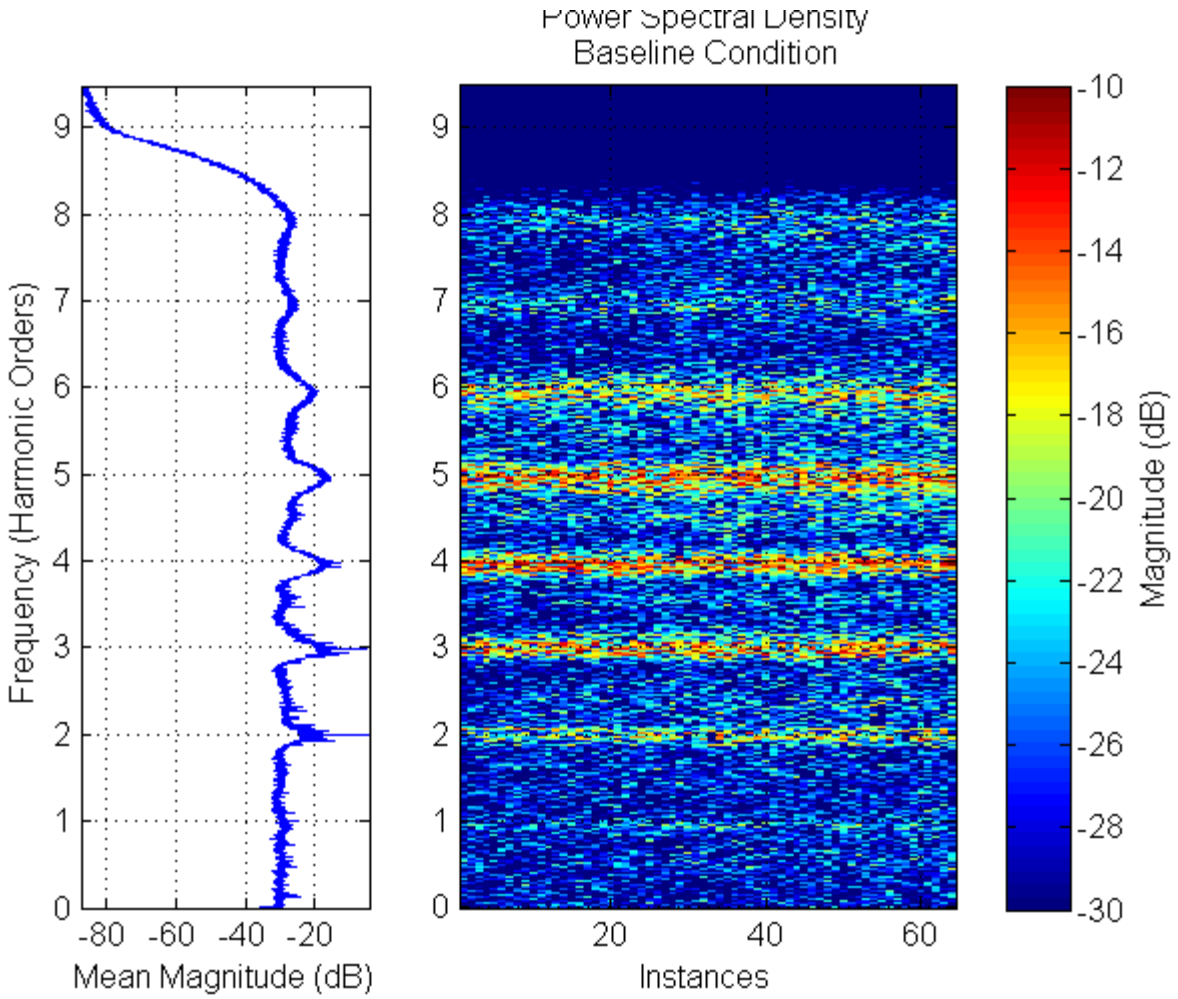
```
subplot(4,1,3) %Angular Series Comparison
plot((1:samplesPerRev)/samplesPerRev,xARSMatrixT(1:samplesPerRev,1,iBaseline(instance)),'b')
hold on; plot((1:samplesPerRev)/samplesPerRev,xARSMatrixT(1:samplesPerRev,1,iDamage(instance)),'r')
axis tight;
title({'Angular Resampled using Tachometer';...
    ['1 Gear Revolution, SPR: ' num2str(samplesPerRev)]});
xlabel('Revolutions');
ylabel('Acceleration (g)');

subplot(4,1,4) %Frequency Domain Comparison
[psdMatrix, f, is1sided] = psdWelch_shm (xARSMatrixT(:,1,[iBaseline(instance),...
    iDamage(instance)]), [], [], [], samplesPerRev/27, []);
plot(f,psdMatrix(:,1,1),'b');
hold on;
plot(f,psdMatrix(:,1,2),'r');
axis tight;
grid on;
title({'PSD Magnitude Plot';...
    ['First 10 Orders SPR: ' num2str(samplesPerRev)]});
xlabel('Frequency (Harmonic Orders)')
;ylabel('Magnitude');
```

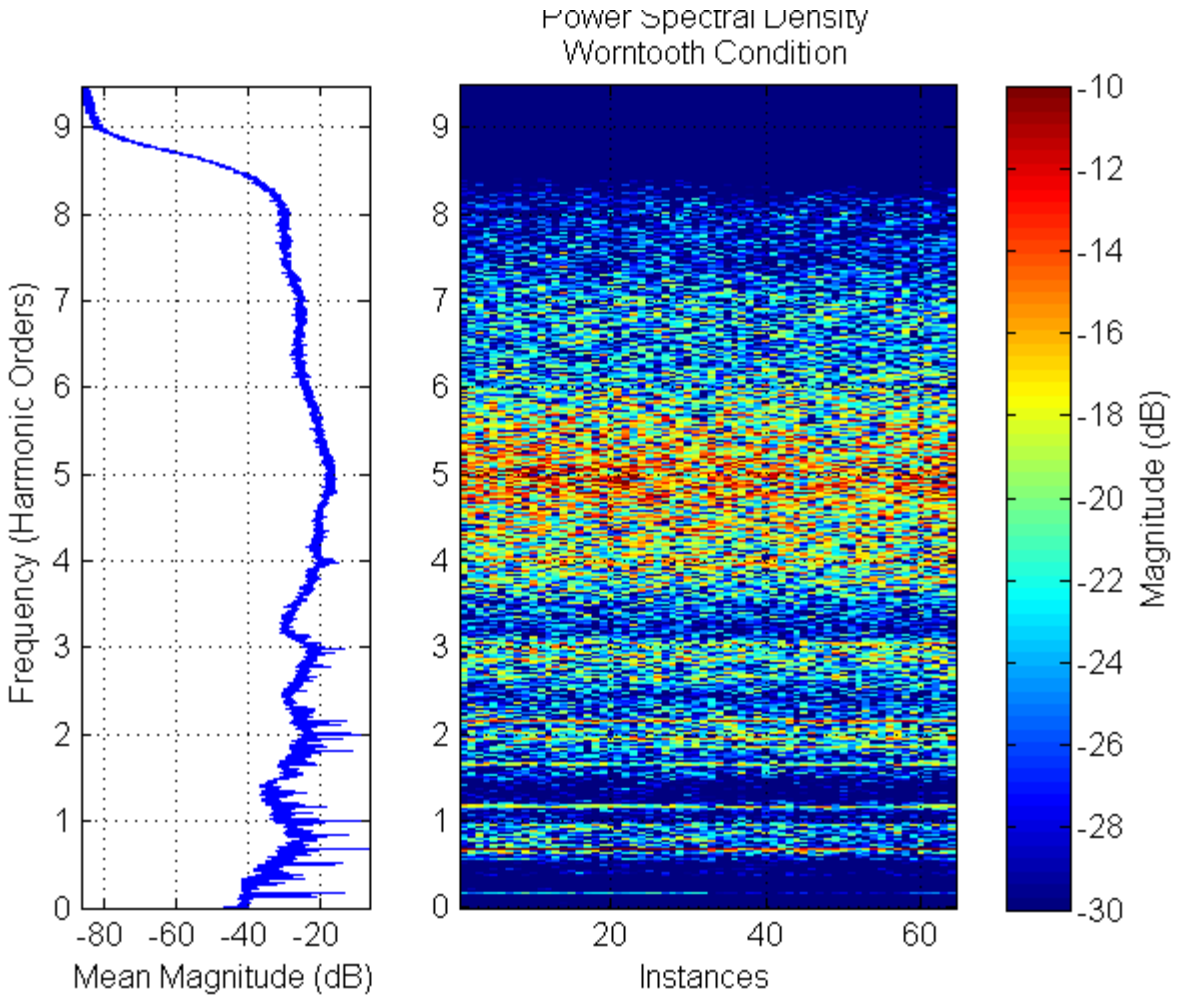


3) Look at Average Power Spectral Density for the Baseline Case

```
% Comparing the power spectral densities it can be seen that the
% periodic gear mesh harmonics begin to smear and frequency energy not
```

```
% associated with the gear meshing increases as the gear teeth begin to
% wear.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% psdWelch_shm Input:
nWin = 2^(nextpow2(size(xARSMatrixT,1))-1);
nOvlap = nWin*.75;
nFFT = nWin*2;
Fs = samplesPerRev/27;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[psdMatrix, f,is1sided] = psdWelch_shm (xARSMatrixT, nWin, nOvlap, nFFT, Fs, []);

[ax] = plotPSD_shm(psdMatrix(:,:,iBaseline), 1, is1sided, f, true, true, []);
axes(ax(1));
caxis([-30, -10]);
title({'Power Spectral Density';'Baseline Condition'});
axes(ax(2));
ylabel('Frequency (Harmonic Orders)');

[ax] = plotPSD_shm(psdMatrix(:,:,iDamage), 1, is1sided, f, true, true, []);
axes(ax(1));
caxis([-30, -10]);
title({'Power Spectral Density';'Worntooth Condition'});
axes(ax(2));
ylabel('Frequency (Harmonic Orders)');
```





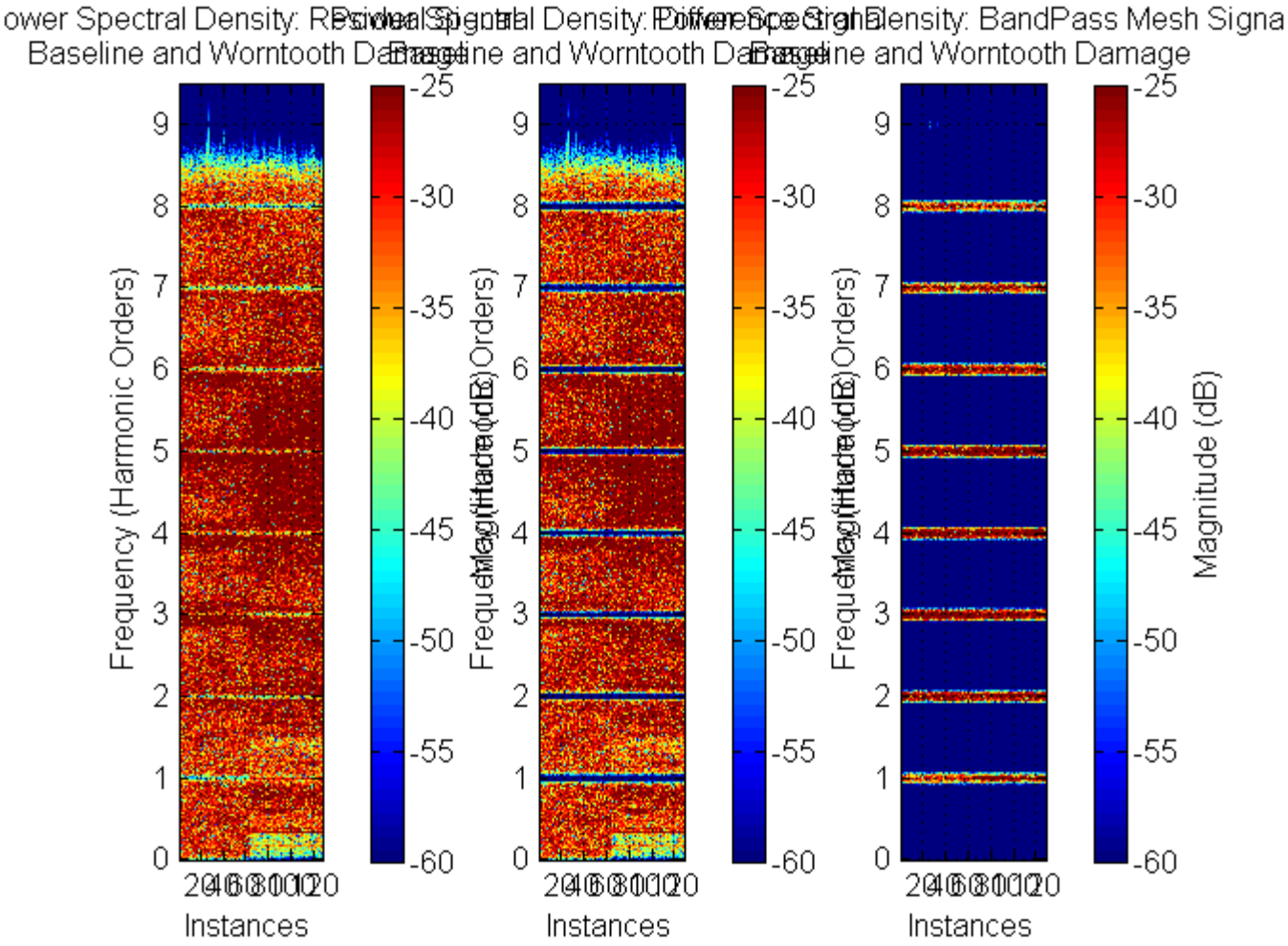
4) Filter xARS to get Residual, Difference and Band Pass Signal.

```
% Filtering of the angular resampled signal is used to determine three
% traditional processed signals used in gearbox damage detection. Here the
% signals are filtered and plotted for comparison. The residual signal
% consists of the angular resampled signal with the shaft and gear mesh
% frequencies filtered out. To do this a narrow band fir filter is used and
% set to filter ate gear mesh orders with a width of an estimate of the
% first order sideband. The difference signal is similar to the residual
% signal but its band width is set to be a little larger than that of the
% residual signal filter to also filter out first order sidebands. The band
% pass gear mesh signal is the angular resampled signal with all frequency
% components filtered out except gear mesh harmonics and the first order
% sidebands.


%Filter Out Drive Shaft
nGearTeeth = 27;
Fs = samplesPerRev;        %Cycles/Rev
fDrive = 1;                %Cycles/Rev
fHarmonic = nGearTeeth;    %Cycles/Rev
fSideBand = 1;             %Cycles/Rev

%Constant Filtering Parameters
nFilter = 511;
[win] = window_shm ('kaiser', [nFilter,4], []);
```

```
nDelay = ceil((nFilter-1)/2);

%Filter Out Drive Shaft Frequency
Wn = fDrive./Fs;
filterType = 'high';

[filterCoef] = fir1_shm (nFilter, Wn, filterType, win);
[y] = filter_shm (xARSMatrixT, filterCoef);

%Residual Signal Filtering out Gear Mesh(Initialize Filter)
index = 1; Fc = index*fHarmonic; xResidual = y; filterType = 'bandstop';
filterContinue = true;
while filterContinue == true
    Wn = [Fc-fSideBand,Fc+fSideBand]./Fs;
    [filterCoef] = fir1_shm (nFilter, Wn, filterType, win);
    [xResidual] = filter_shm (xResidual, filterCoef);
    index = index+1; Fc = fHarmonic*index;
    if (Fc+fSideBand)>= Fs/2; filterContinue = false; end
end
%Remove Filter Delay
xResidual = xResidual(1+nDelay:end,:,:);

%Difference Signal Filtering out Gear Mesh (Initialize Filter)
index = 1; Fc = index*fHarmonic; xDifference = y; filterType = 'bandstop';
filterContinue = true;
while filterContinue == true
    Wn = [Fc-2*fSideBand,Fc+2*fSideBand]./Fs;
    [filterCoef] = fir1_shm (nFilter, Wn, filterType, win);
    [xDifference] = filter_shm (xDifference, filterCoef);
    index = index+1; Fc = fHarmonic*index;
    if (Fc+2*fSideBand)>= Fs/2; filterContinue = false; end
end
%Remove Filter Delay
xDifference = xDifference(1+nDelay:end,:,:);

%Band Pass Mesh Signal Filtering out All but Gear Mesh(Initialize Filter)
index = 0;Fc = fHarmonic/2; xBandPassMesh = y; filterType = 'bandstop';
filterContinue = true;
while filterContinue == true
    if index == 0
        Wn = [.00001, Fc+(fHarmonic/2-fSideBand)]./Fs;
    else
        Wn = [Fc-(fHarmonic/2-fSideBand),Fc+(fHarmonic/2-fSideBand)]./Fs;
    end
    [filterCoef] = fir1_shm (nFilter, Wn, filterType, win);
    [xBandPassMesh] = filter_shm (xBandPassMesh, filterCoef);
    index = index+1; Fc = fHarmonic*(index+0.5);
    if (((Fc+(fHarmonic/2-fSideBand))/Fs)>= 0.5);
        Wn = [Fc-(fHarmonic/2-fSideBand),Fs/2-.0001]./Fs;
        [filterCoef] = fir1_shm (nFilter, Wn, filterType, win);
        [xBandPassMesh] = filter_shm (xBandPassMesh, filterCoef);
        filterContinue = false;
    end
end
%Remove Filter Delay
xBandPassMesh = xBandPassMesh(1+nDelay:end,:,:);
```

Look at Average Power Spectral Density for New Signals

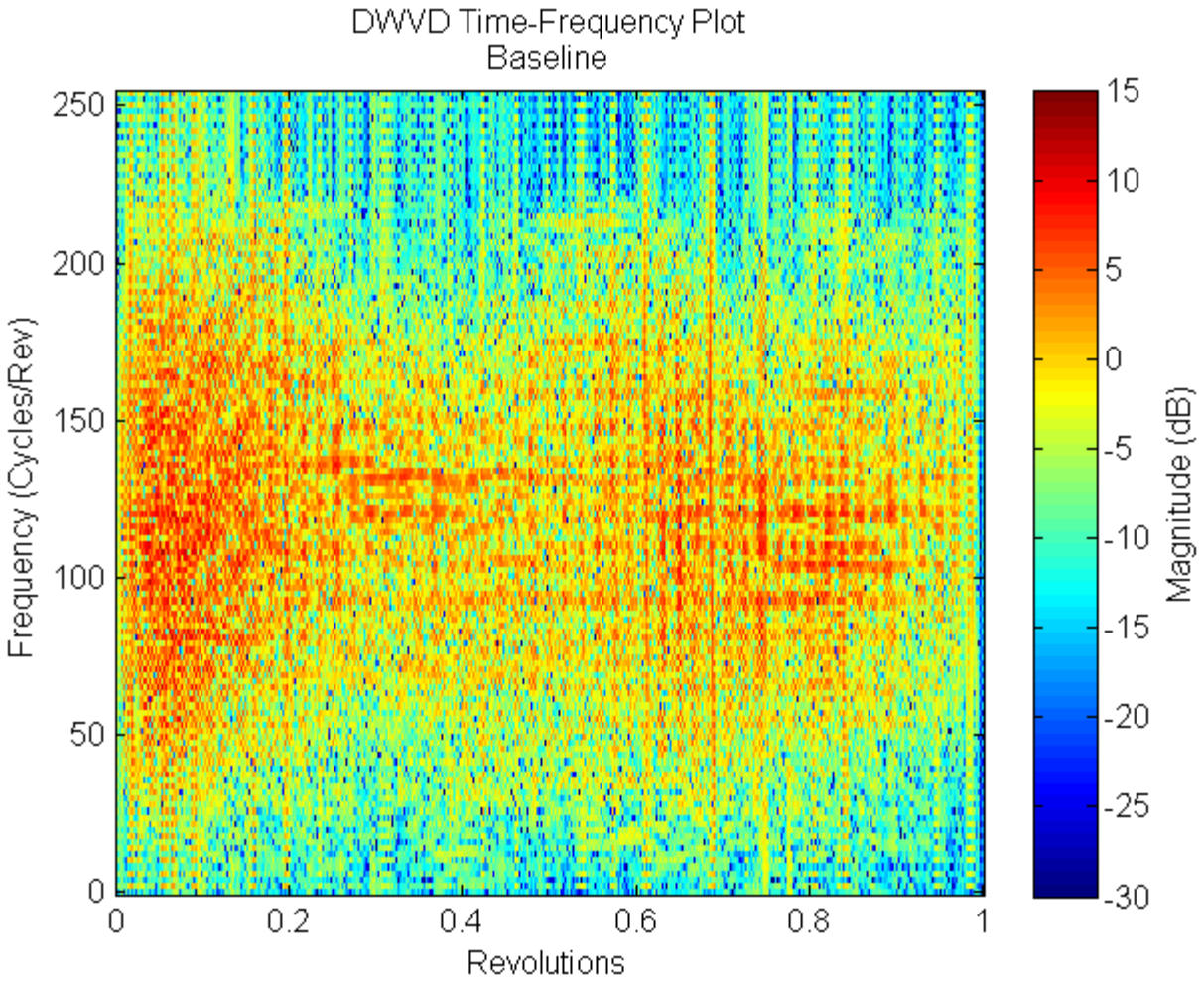
```
% Of the four time frequency plots shown, the continuous wavelet scalogram
% presents the best option for detection nonlinearities in a signal. The
% gear mesh impulses can be detected in the high frequency bin. By design
% continuous wavelet scalograms have good frequency resolution and poor
% time resolution for low frequencies and good time resolution but poor
% frequency resolution at high frequencies. Impulses create broad band
% noise and this can be seen in the high frequency range of the scalogram
% which directly corresponds to gear teeth impacts that are extracted using
% the Hoelder exponent. The other time frequency plots, in order to get
% good time resolution the must use small window sizes which cause a lot of
% smearing in the frequency content.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% psdWelch_shm Input:
nWin = 2^(nextpow2(size(xResidual,1))-1);
nOvlap = nWin*.75;
nFFT = nWin*2;
Fs = samplesPerRev/nGearTeeth;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure;

%Plot PSD of all instances of the residual signal
[psdMatrix, f, is1sided] = psdWelch_shm (xResidual, nWin, nOvlap, nFFT, Fs, []);

ax = subplot(1,3,1);
plotPSD_shm(psdMatrix, 1, is1sided, f, true, false, ax);
caxis([-60, -25]);
title({'Power Spectral Density: Residual Signal',...
    'Baseline and Worntooth Damage'})
ylabel('Frequency (Harmonic Orders)');

%Plot PSD of all instances of the difference signal
[psdMatrix, f, is1sided] = psdWelch_shm (xDifference, nWin, nOvlap, nFFT, Fs, []);

ax = subplot(1,3,2);
[ax] = plotPSD_shm(psdMatrix, 1, is1sided, f, true, false, ax);
caxis([-60, -25]);
title({'Power Spectral Density: Difference Signal',...
    'Baseline and Worntooth Damage'})
ylabel('Frequency (Harmonic Orders)');

%Plot PSD of all instances of the band pass mesh signal
[psdMatrix, f, is1sided] = psdWelch_shm (xBandPassMesh, nWin, nOvlap, nFFT, Fs, []);

ax = subplot(1,3,3);
[ax] = plotPSD_shm(psdMatrix, 1, is1sided, f, true, false, ax);
caxis([-60, -25]);
title({'Power Spectral Density: BandPass Mesh Signal',...
    'Baseline and Worntooth Damage'})
ylabel('Frequency (Harmonic Orders)');
```

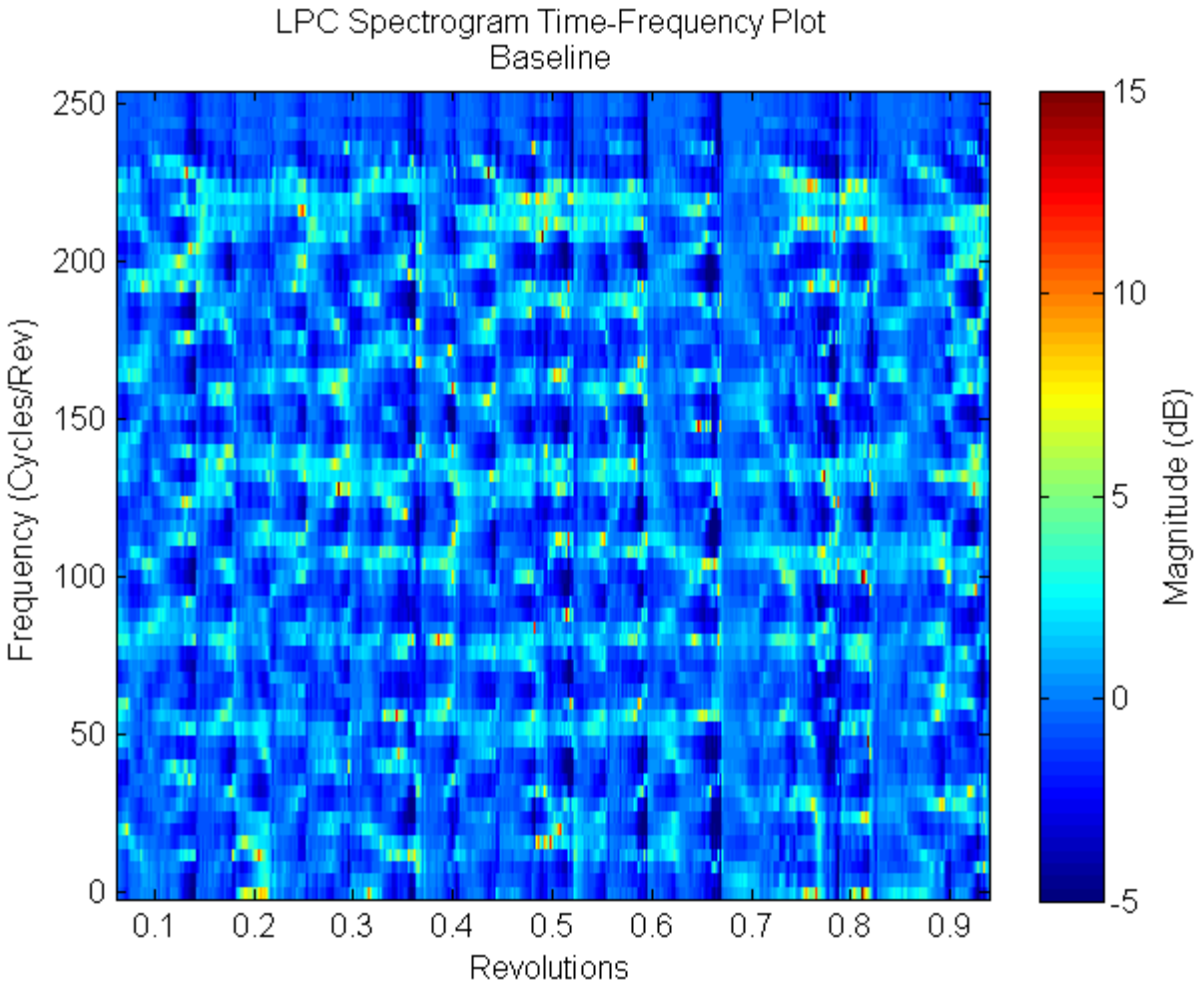


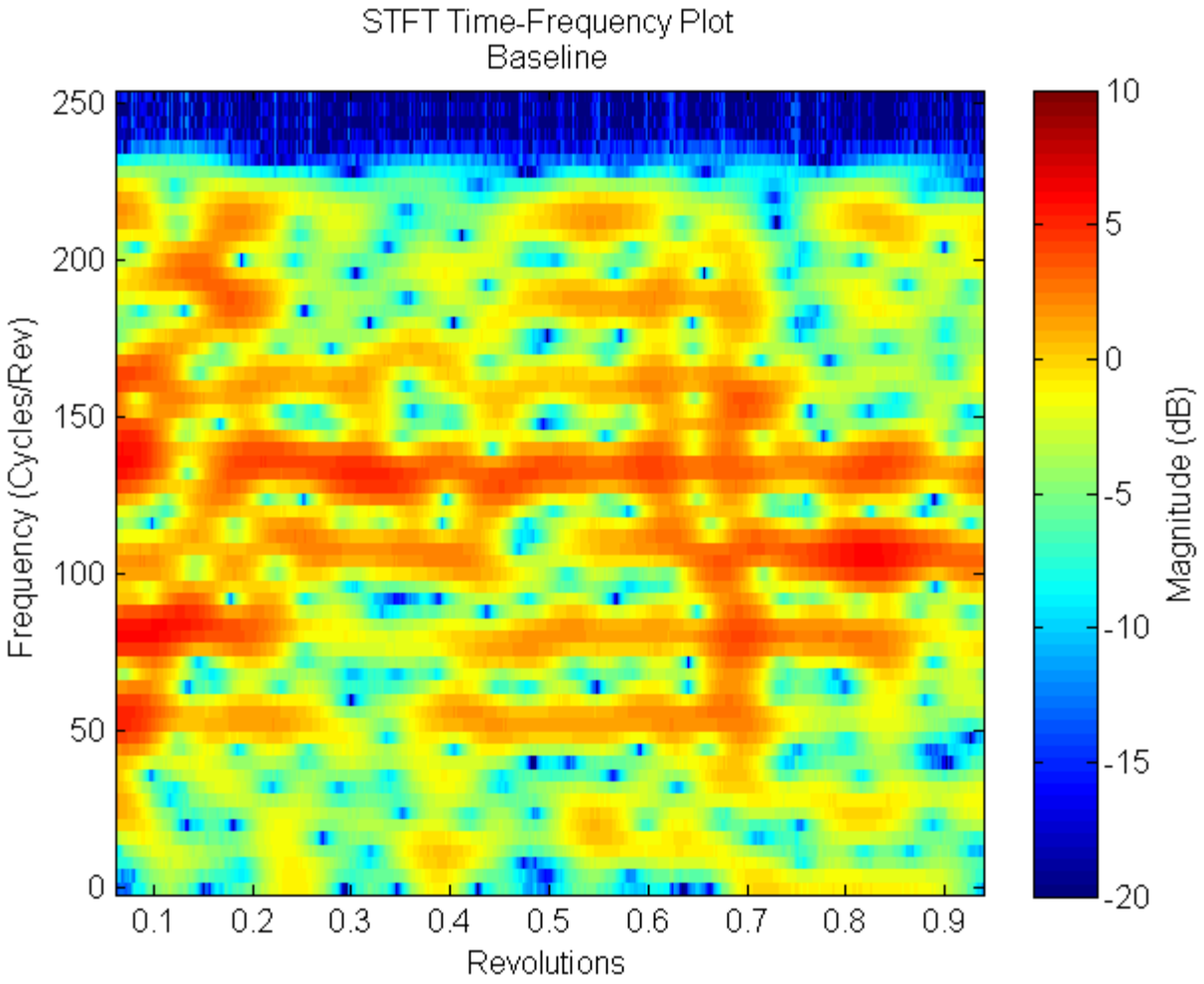
5) Look at Time Frequency Domain

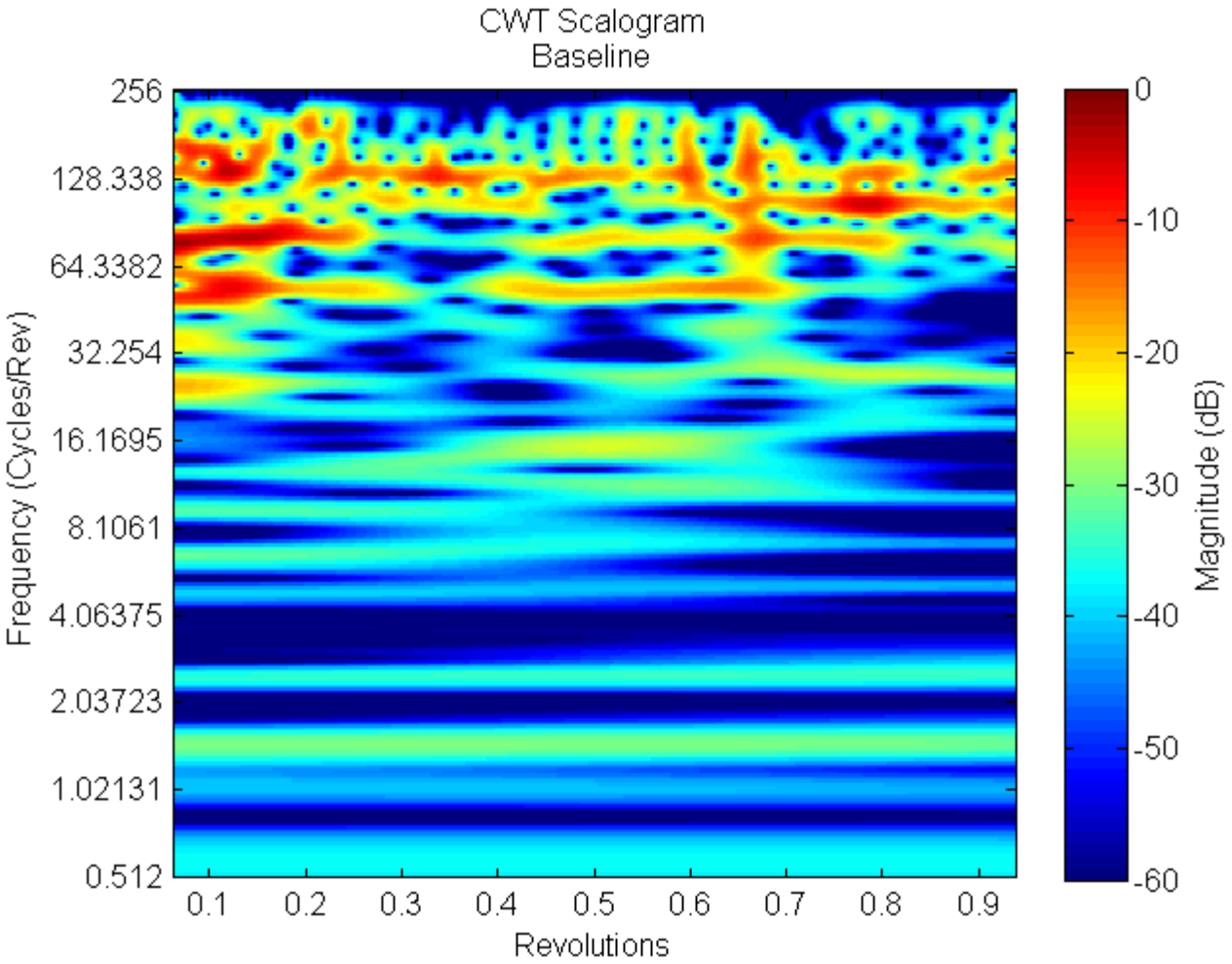
```
for i = 1:2
    index = 4;
    instance = [iBaseline(index) iDamage(index)];
    stateTitle = {'Baseline' 'Damaged'};

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % dwvd_shm Input:
    nWin = 65;
    nOvlap = nWin-1;
    nFFT = nWin*2;
    Fs = samplesPerRev;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    [dwvdMatrix, f, t] = dwvd_shm (xARSMatrixT(1:samplesPerRev,:,instance(i)), nWin, nOvlap, nFFT,
Fs);

    %Plot Discrete Wignere-Ville
    plotTimeFreq_shm (dwvdMatrix, [], [], t, f, [], []);
    caxis([-30, 15]);
    title({'DWVD Time-Frequency Plot';char(stateTitle(i))});
    ylabel('Frequency (Cycles/Rev)');
    xlabel('Revolutions');

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % lpcSpectrogram_shm Input:
```

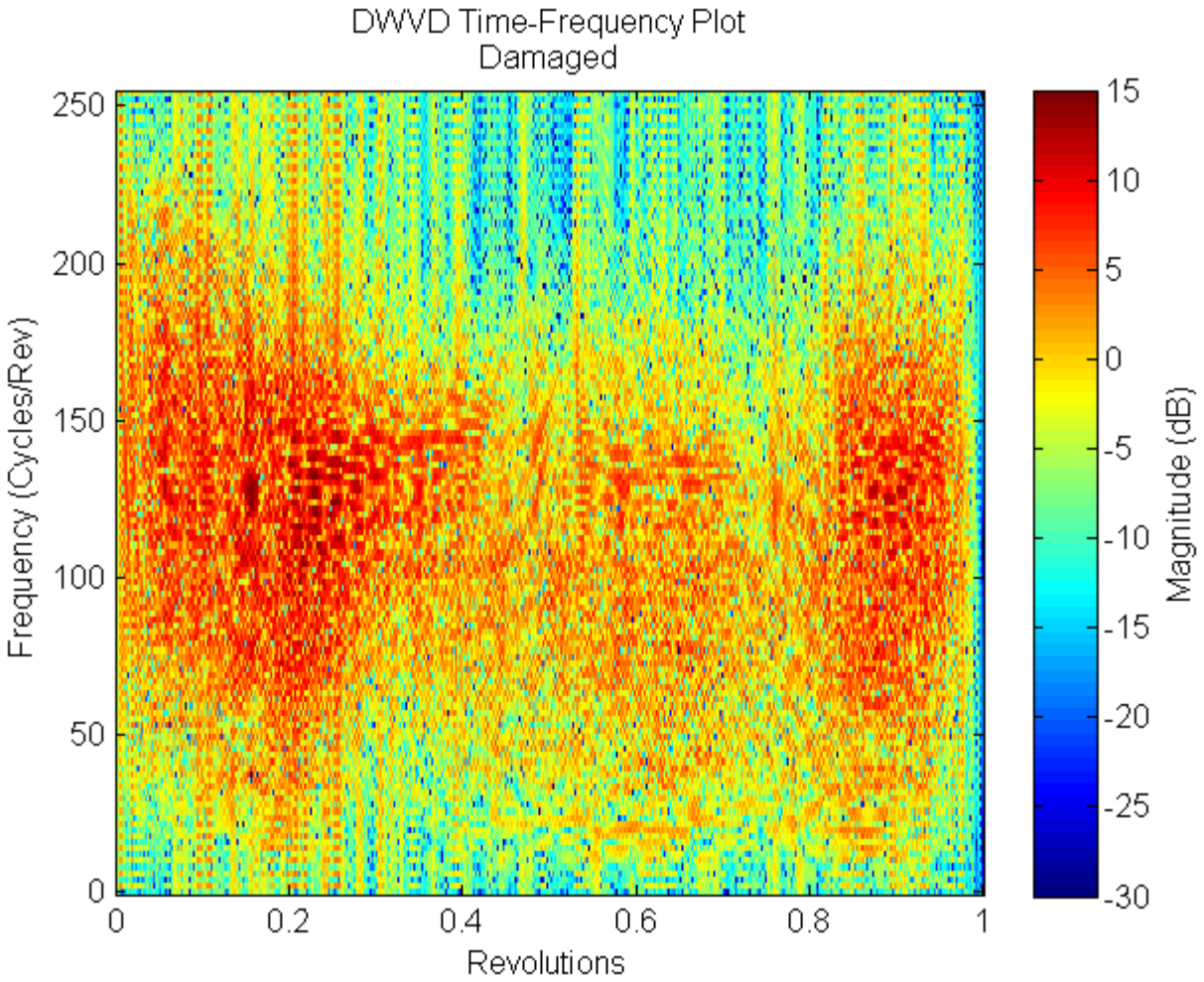
```
    modelOrder = 42;
    nWin = 64;
    nOvlap = nWin-1;
    nFFT = nWin*2;
    Fs = samplesPerRev;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    [lpcSpecMatrix, f, t] = lpcSpectrogram_shm (xARSMatrixT(1:samplesPerRev,:,instance(i)),
modelOrder, nWin,...
        nOvlap, nFFT, Fs);

    %Plot LPC Spectrogram
    plotTimeFreq_shm (lpcSpecMatrix, [], [], t, f, [], []);
    caxis([-5, 15]);
    title({'LPC Spectrogram Time-Frequency Plot';char(stateTitle(i))});
    ylabel('Frequency (Cycles/Rev)');
    xlabel('Revolutions');

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % stft_shm Input:
    nWin = 64;
    nOvlap = nWin-1;
    nFFT = nWin*2;
    Fs = samplesPerRev;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    [stftMatrix, f, t] = stft_shm (xARSMatrixT(1:samplesPerRev,:,instance(i)), nWin, nOvlap, nFFT,
Fs);

    % Plot STFT
    plotTimeFreq_shm (stftMatrix, [], [], t, f, [], []);
    caxis([-20, 10]);
    title({'STFT Time-Frequency Plot';char(stateTitle(i))});
    ylabel('Frequency (Cycles/Rev)');
    xlabel('Revolutions');


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % cwtScalogram_shm Input:
    Fs = samplesPerRev;
    fMin = [];
    fMax = [];
    nScale = 256;
    waveOrder = [];
    waveType = [];
    useAnalytic = true;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    [scaloMatrix,f,timeVector] = cwtScalogram_shm (xARSMatrixT(1:samplesPerRev,:,instance(i)),
Fs,...
        fMin, fMax, nScale, waveOrder, waveType, useAnalytic);

    % Plot CWT
    plotScalogram_shm (scaloMatrix, [], [], t, f, []);
    caxis([-60, 0]);
    title({'CWT Scalogram';char(stateTitle(i))});
    ylabel('Frequency (Cycles/Rev)');
    xlabel('Revolutions');
end
```

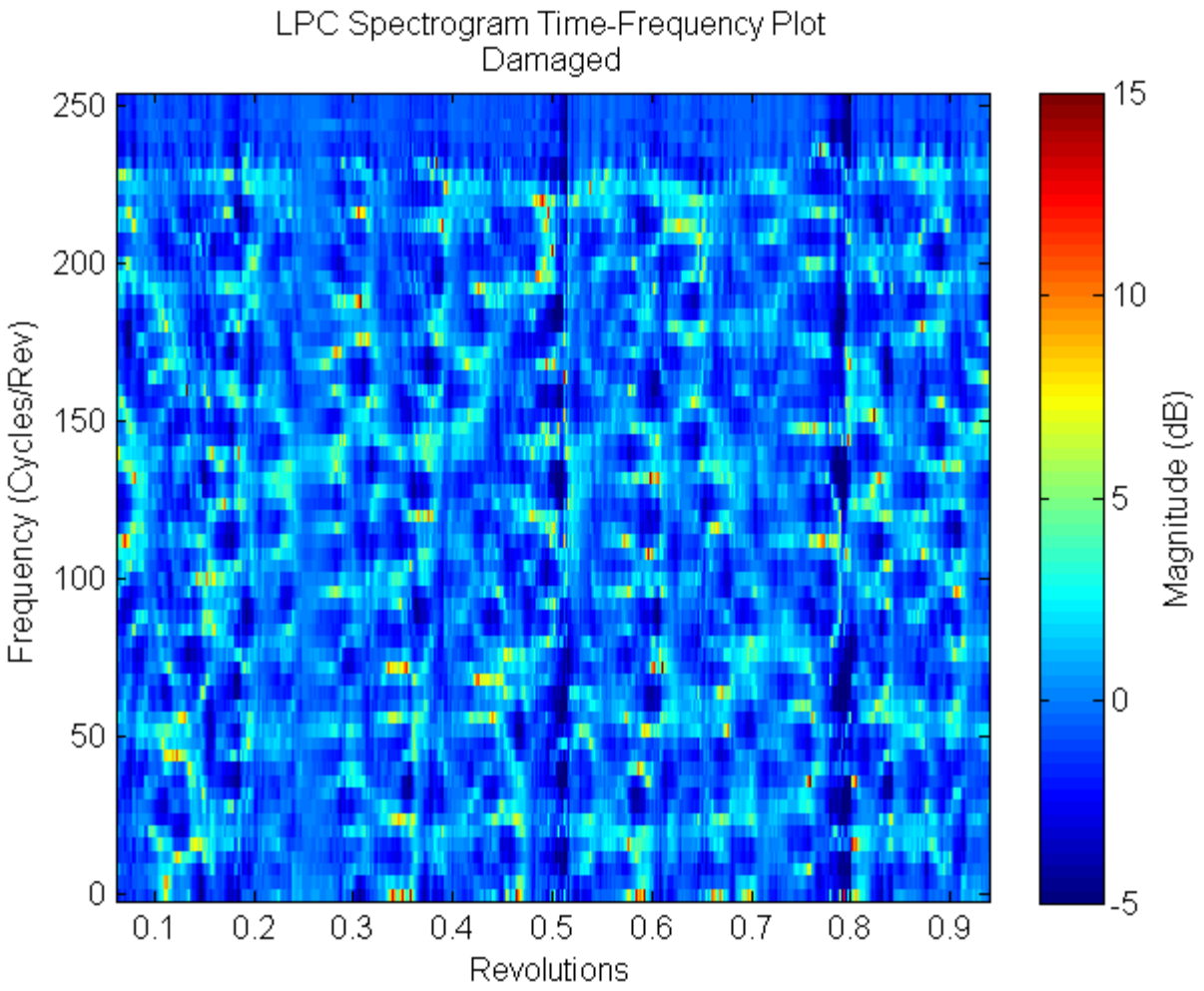

DWVD Time-Frequency Plot
Baseline
Frequency (Cycles/Rev)
250
200
150
100
50
0
0
0.2
0.4
0.6
0.8
1
Revolutions
15
10
5
0
-5
-10
-15
-20
-25
-30
Magnitude (dB)

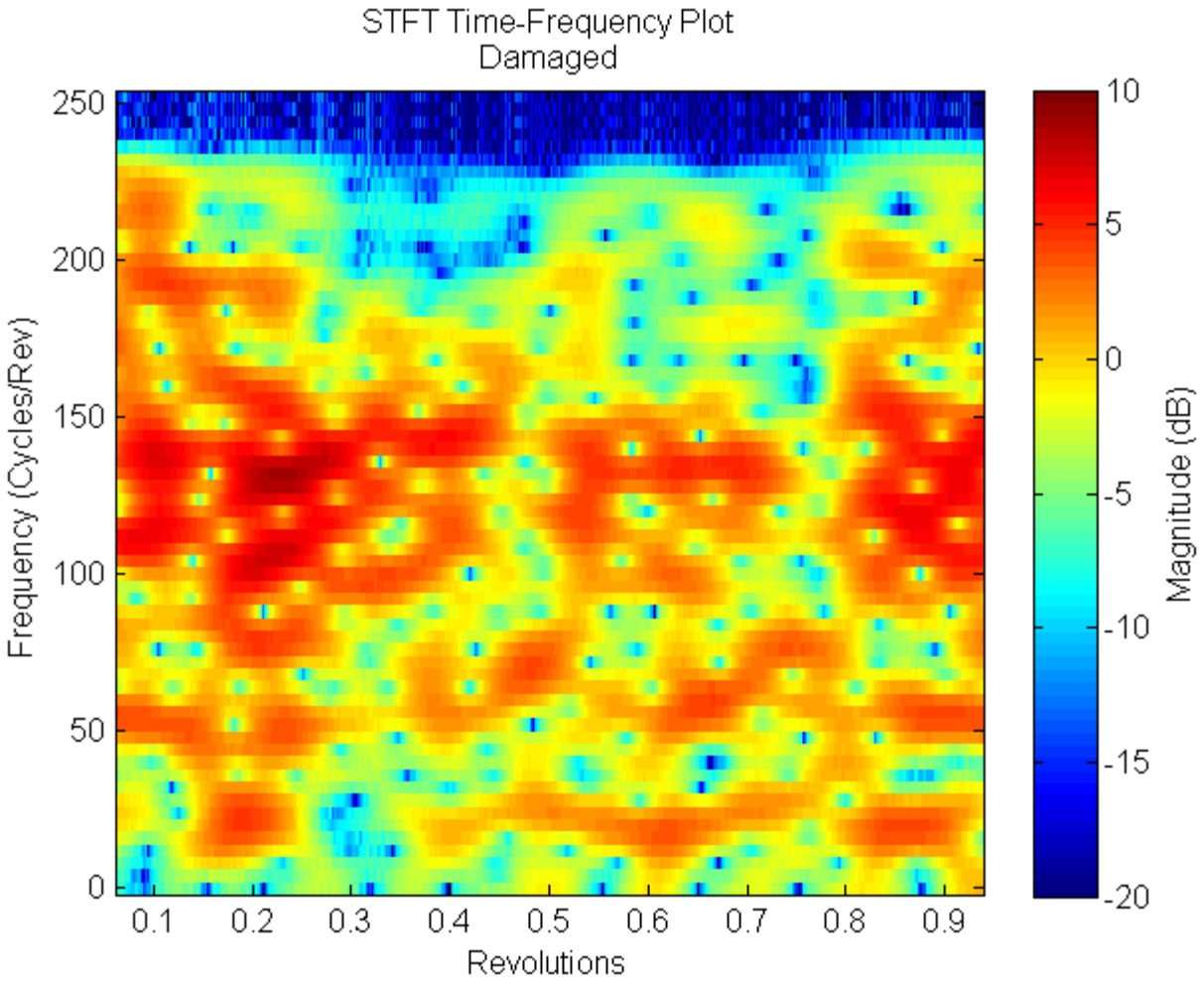

LPC Spectrogram Time-Frequency Plot
Baseline
Frequency (Cycles/Rev)
250
200
150
100
50
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Revolutions
15
10
5
0
-5
Magnitude (dB)

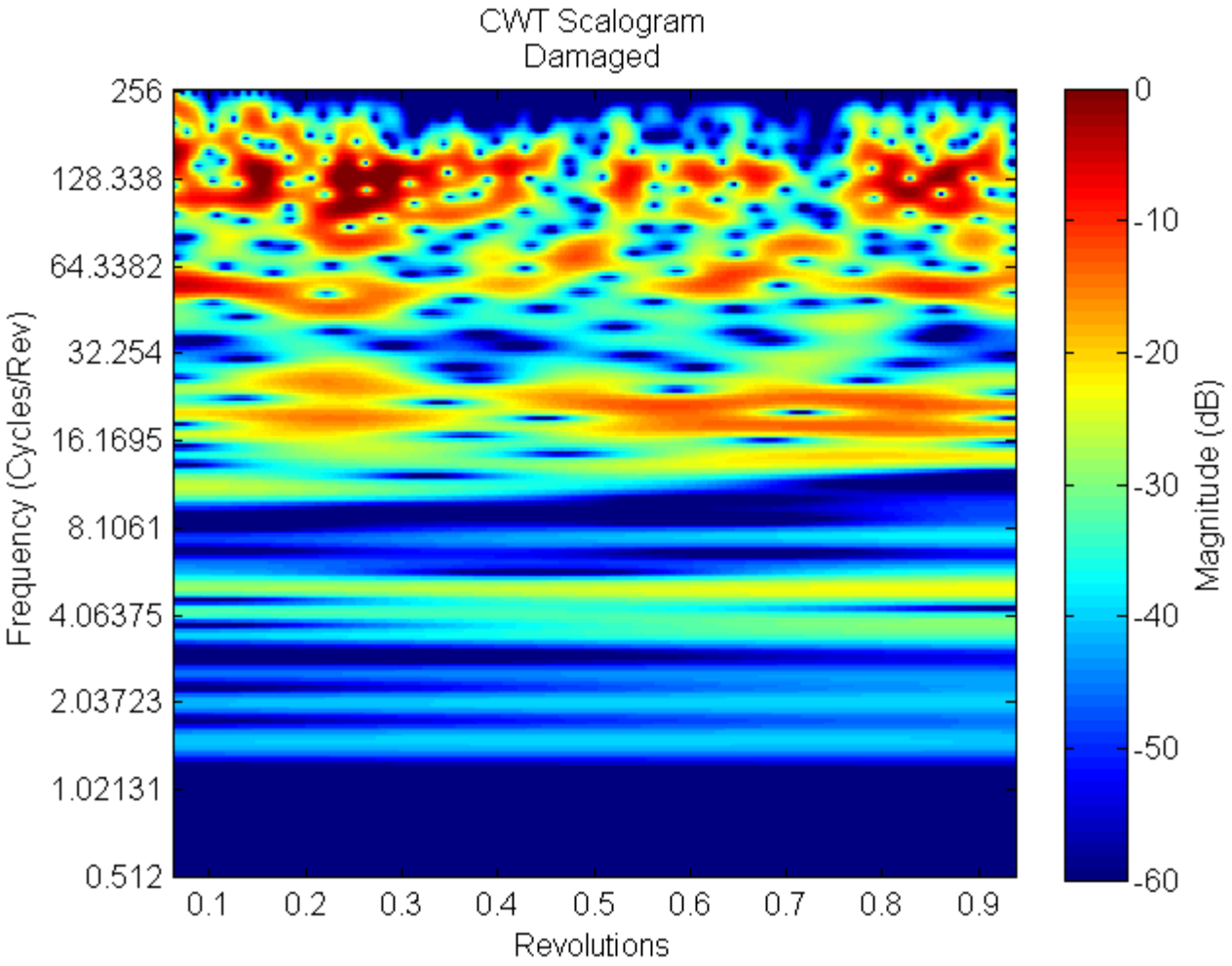

STFT Time-Frequency Plot
Baseline
Frequency (Cycles/Rev)
250
200
150
100
50
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Revolutions
10
5
0
-5
-10
-15
-20
Magnitude (dB)


CWT Scalogram
Baseline
256
128.338
64.3382
32.254
16.1695
8.1061
4.06375
2.03723
1.02131
0.512
Frequency (Cycles/Rev)
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Revolutions
0
-10
-20
-30
-40
-50
-60
Magnitude (dB)


DWVD Time-Frequency Plot
Damaged
Frequency (Cycles/Rev)
Revolutions
Magnitude (dB)
250
200
150
100
50
0
0
0.2
0.4
0.6
0.8
1
15
10
5
0
-5
-10
-15
-20
-25
-30


LPC Spectrogram Time-Frequency Plot
Damaged
Frequency (Cycles/Rev)
250
200
150
100
50
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Revolutions
15
10
5
0
-5
Magnitude (dB)


STFT Time-Frequency Plot
Damaged
Frequency (Cycles/Rev)
250
200
150
100
50
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Revolutions
10
5
0
-5
-10
-15
-20
Magnitude (dB)

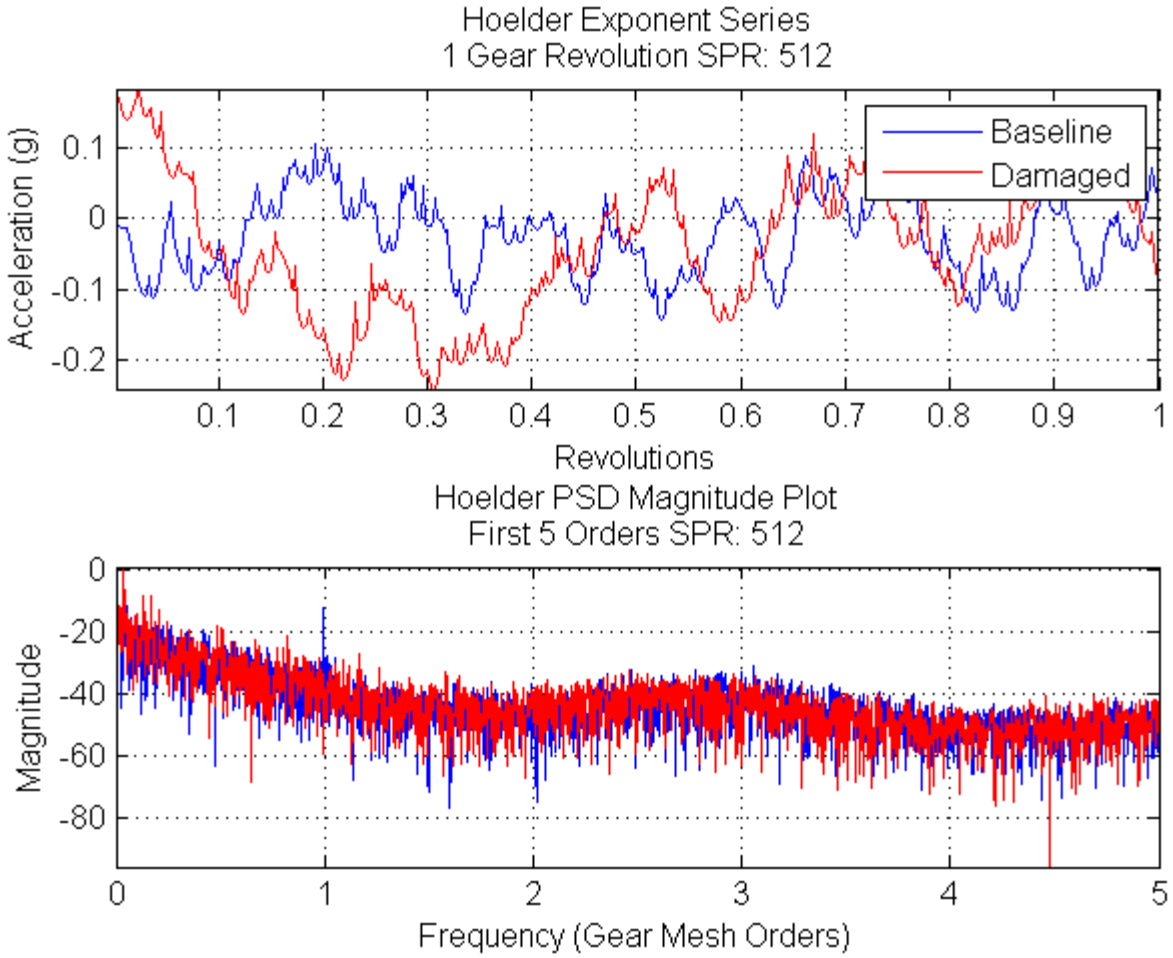


6) Get Hoelder Series from CWTScalo

```
% The Hoelder exponent is a measure of the slope of the energy content in a
% time frequency domain. As high frequency energy rises and falls so too
% does the value of the Hoelder exponent. As previously stated this picks
% up on impacts or nonlinearities in a signal that cause increases in high
% energy content. By looking at the frequency domain of the Hoelder
% exponent for a gear it is possible to track the impulses of gear teeth
% and as the wear the magnitude of these impulses in the Hoelder domain
% will begin to decay making it useful in monitoring gear teeth wear.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% cwtScalogram_shm Input:
Fs = samplesPerRev;
fMin = [];
fMax = [];
nScale = 64;
waveOrder = [];
waveType = [];
useAnalytic = true;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[scaloMatrix,f,t] = cwtScalogram_shm (xARSMatrixT, Fs,...
    fMin, fMax, nScale, waveOrder, waveType, useAnalytic);
hoelderMatrix = hoelderExp_shm (scaloMatrix, f);
hoelderMatrix = demean_shm(hoelderMatrix);
```

Plot Hoelder Series in Time and Frequency Domain

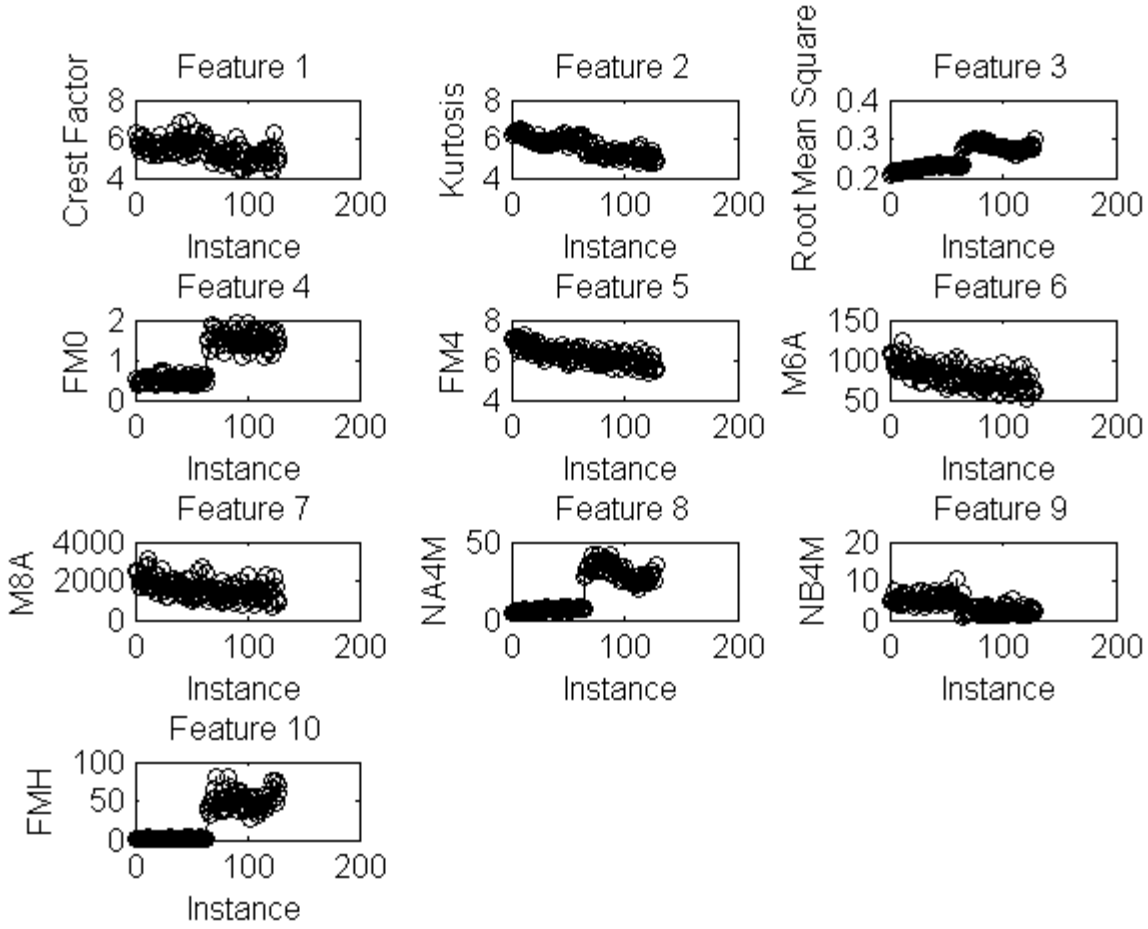
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% psdWelch Input:
nWin = 2^(nextpow2(size(xResidual,1))-1);
nOvlap = nWin*.75;
nFFT = nWin*2;
Fs = samplesPerRev/nGearTeeth;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[psdMatrix, f] = psdWelch_shm (hoelderMatrix, nWin, nOvlap, nFFT, Fs, []);
psdMatrix = 10.*log10(psdMatrix);

figure; instance = 6;
subplot(2,1,1) %Angular Series Comparison
plot((1:samplesPerRev)/samplesPerRev,hoelderMatrix(1:samplesPerRev,1,iBaseline(instance)),'b')
hold on;
plot((1:samplesPerRev)/samplesPerRev,hoelderMatrix(1:samplesPerRev,1,iDamage(instance)),'r')
axis tight;
grid on;
title({'Hoelder Exponent Series'...
    ['1 Gear Revolution SPR: ' num2str(samplesPerRev)]});
xlabel('Revolutions');
ylabel('Acceleration (g)');
legend('Baseline','Damaged')

subplot(2,1,2) %Frequency Domain Comparison
plot(f,psdMatrix(:,1,iBaseline(instance)),'b');
hold on;
plot(f,psdMatrix(:,1,iDamage(instance)),'r');
axis tight;
grid on;
xlim([0 5]);
title({'Hoelder PSD Magnitude Plot';...
    ['First 5 Orders SPR: ' num2str(samplesPerRev)]});
xlabel('Frequency (Gear Mesh Orders)');
ylabel('Magnitude');
```



7) Look at Some Common Feature Types

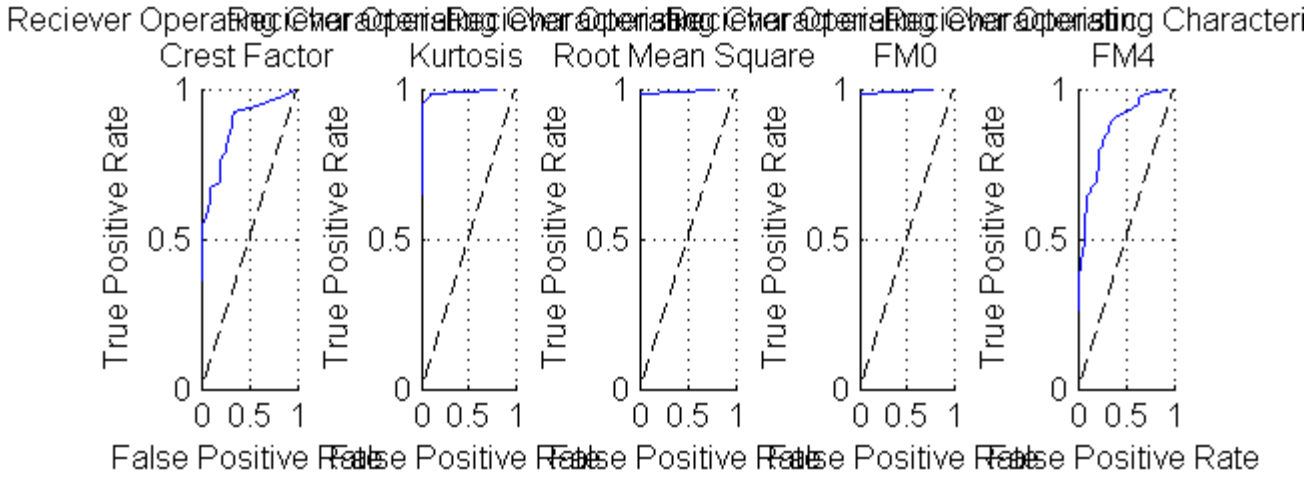
```
% Several features are compared here. Raw signal features like crest factor
% kurtosis and root mean square are good at detection damage but the damage
% features suffer from a lack of localization in determining the cause of
% change. Increases in load and speed can cause a large change in the value
% of these features. FM0 uses the resampled signal and track the gear mesh
% orders which makes it tuned to the gear in question. FMH is similar to
% FM0 but uses the Hoelder series instead giving it improved separation in
% the damage features between baseline and worn tooth states. The residual
% and difference signals monitor changes in the frequency content other
% than that of gear mesh harmonics and sidebands and the band pass mesh
% signal features monitor the kurtosis of the enveloped signal of
% primarily gear mesh harmonics.

%Raw Signal Damage Damage Features
[cf] = crestFactor_shm(X(:,2,:));
[statisticsFV] = statMoments_shm (X(:,2,:));
[kurt] = statisticsFV(:,4);
[rms] = rms_shm(X(:,2,:));

%Resampled Signal Damage Features
fundMeshFreq = fHarmonic/samplesPerRev;
trackOrders = [1 2 3];
nFFT = [];
nBinSearch = 3;
[fm0] = fm0_shm (xARSMatrixT,fundMeshFreq,trackOrders,nFFT,nBinSearch);
```

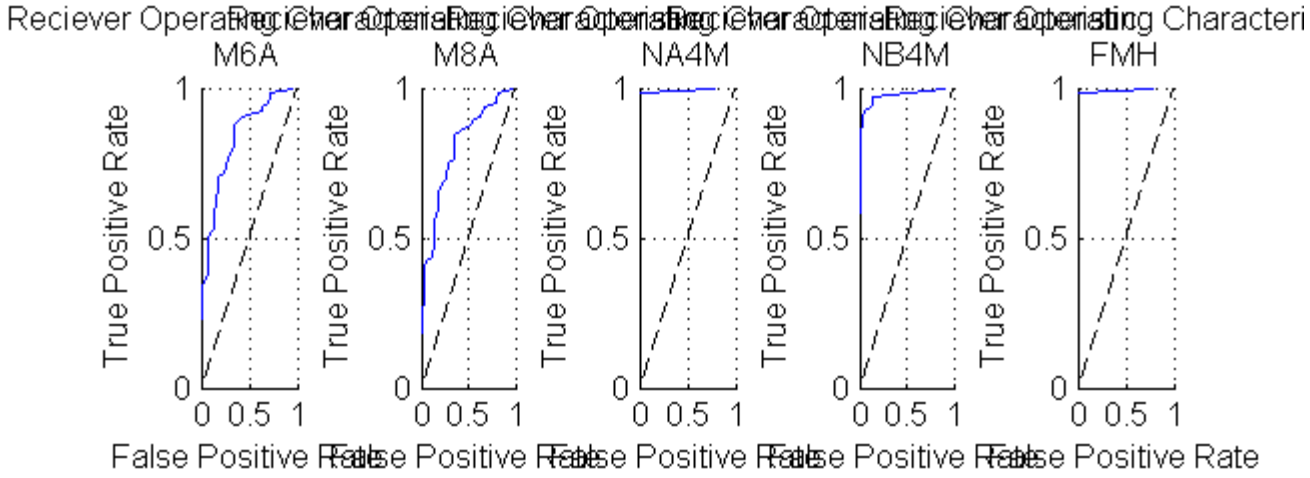
```
%Residual Signal Damage Features
[fm4] = fm4_shm (xResidual);
[m6a] = m6a_shm (xResidual);
[m8a] = m8a_shm (xResidual);

%Difference Signal Damage Features
[na4mBase, m2] = na4m_shm (xDifference(:,:,iBaseline), []);
[na4mDamage, temp] = na4m_shm (xDifference(:,:,iDamage), m2);
na4m = [na4mBase;na4mDamage];

%Bandpass Mesh Signal
[nb4mBase, m2] = nb4m_shm (xBandPassMesh(:,:,iBaseline), []);
[nb4mDamage, temp] = nb4m_shm (xBandPassMesh(:,:,iDamage), m2);
nb4m = [nb4mBase;nb4mDamage];

%Hoelder Signal Damage Features
fundMeshFreq = fHarmonic/samplesPerRev;
trackOrders = [1 2 3];
nFFT = [];
nBinSearch = 3;
[fmH] = fm0_shm (hoelderMatrix,fundMeshFreq,trackOrders,nFFT,nBinSearch);

%Plot Damage Features
features = [cf,kurt,rms,fm0,fm4,m6a,m8a,na4m,nb4m,fmH];
featNames = {'Crest Factor','Kurtosis','Root Mean Square','FM0','FM4',...
    'M6A','M8A','NA4M','NB4M','FMH'};
plotFeatures_shm(features,[],[],[],featNames,[]);
```



8) Compare Damage Features Statistically - Plot ROC Curves

```
%To compare the damage features detectability statistically, receiver
%operating characteristic curves can be used to show the probability of
%detection vs. the probability of false alarm. Damage features with a high
%probability of detection to false alarm rate are optimal detectors. From
%the ROC curves the four best performing damage features for the data
%provided were NA4M, root mean square, FM0 and FMH which all had perfect
%detection in this data set.

figure;
thresholdTypes = {'below','below','above','above','below','below',...
    'below','above','below','above'};

for i=1:length(featNames)
    ax = subplot(2,5,i);
    [TPR, FPR] = ROC_shm (features(:,i), damageStates, [], thresholdTypes{i});
    plotROC_shm (TPR, FPR, 'linear', ax);
    hold on;
    plot([0 1],[0 1],'--k');
    grid on;
    axis([0 1 0 1]);
    title({'Reciever Operating Characteristic:',featNames{i}});
end
```





Published with MATLAB® R2013a

Example Usage: Sensor Diagnostics

Contents

Introduction

The goal of this example is to perform the piezoelectric sensor diagnostic process to check the operational status of piezoelectric sensors in SHM applications. Both sensor fractures and debonding between the sensor and a host structure can be automatically identified. The basic principle of this technique is to track the chagnes in capacitance value of piezoelectric materials for sensor diagnostics. Because the capacitance is temperature senstive, this algorithm uses an array of sensors to instaneously establishs a baseline, which can be robust against temperature variations.

The data sets are measured from twelve piezoelectric patches (1/2 inch diameter) installed on a thin (1/8th thickness) aluminum plates. Three of the senosrs were improperly installed, one with 80% debonding, two with sensor fracture. The following process shows the identification of these faulty sensors

Requires dataSensorDiagnostic.mat dataset.

References:

Overly, T.G., Park, G., Farinholt, K.M., Farrar, C.R. "Piezoelectric Active-Sensor Diagnostics and Validation Using Instantaneous Baseline Data," IEEE Sensors Journal, in press.

Park, G., Farrar, C.R., Rutherford, C.A., Robertson, A.N., 2006, "Piezoelectric Active Sensor Self-diagnostics using Electrical Admittance Measurements," ASME Journal of Vibration and Acoustics, 128(4), 469-476.

Park, G., Farrar, C.R., Lanza di Scalea, F., Coccia, S., 2006, "Performance Assessment and Validation of Piezoelectric Active Sensors in Structural Health Monitoring," Smart Materials and Structures, 15(6), 1673-1683.

Park, S., Park, G., Yun, C.B., Farrar, C.R., 2009, "Sensor Self-Diagnosis Using a Modified Impedance Model for Active-Sensing Structural Health Monitoring," International Journal of Structural Health Monitoring, 8(1),71-82.

SHMTools functions called:

```
sdFeature_shm
sdAutoclassify_shm
sdPlot_shm
```


Author: Tim Overly, Gyuhae Park

Date: August 20th, 2009

Load Raw Data

Additionally you can load sd_ex.mat file, which are composed of all healthy sensors.

```
load('dataSensorDiagnostic.mat');
```

Feature Extraction

```
slope=sdFeature_shm(sd_ex_broken);
```

Sensor Status Classification

```
[Sensor_status, data_for_plotting]=sdAutoclassify_shm(slope, 0.02)
```

```
Sensor_status =

    1.0000         0    9.1757
    2.0000         0    9.4774
    3.0000    2.0000    7.2611
    4.0000         0    9.2205
    5.0000         0    9.2718
    6.0000    1.0000   10.5018
    7.0000         0    9.3875
    8.0000         0    9.3981
    9.0000         0    9.2593
   10.0000    2.0000    8.6806
   11.0000         0    9.0976
   12.0000         0    9.5094


data_for_plotting =

       A: [12x5 double]
     pos: 9
     ave: 9.3108e-09
    list: {[2x1 double]  [6]}
```

The first column of Sensor_status is the sensor id number.

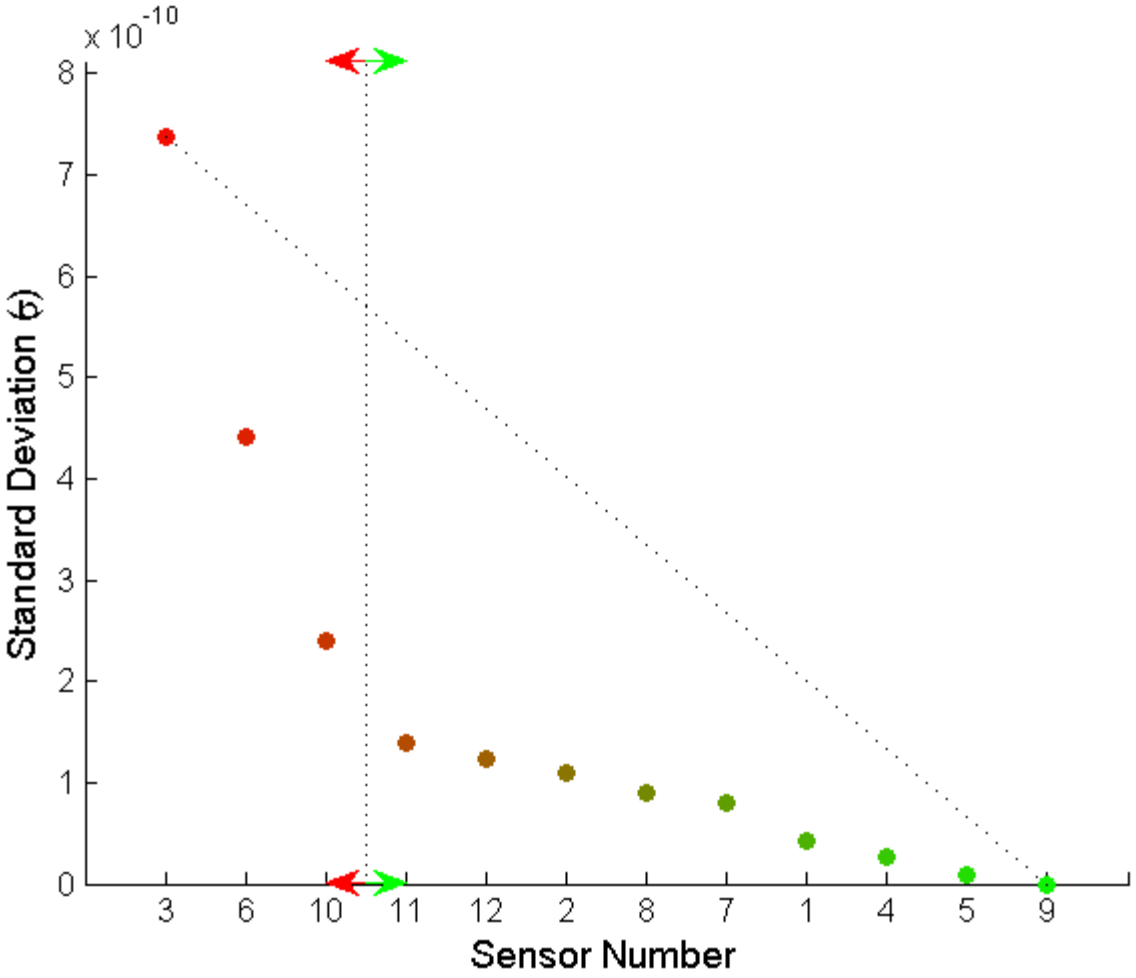
In the second column, 0: health sensors, 1: De-bonded sensors, 2: broken sensors.

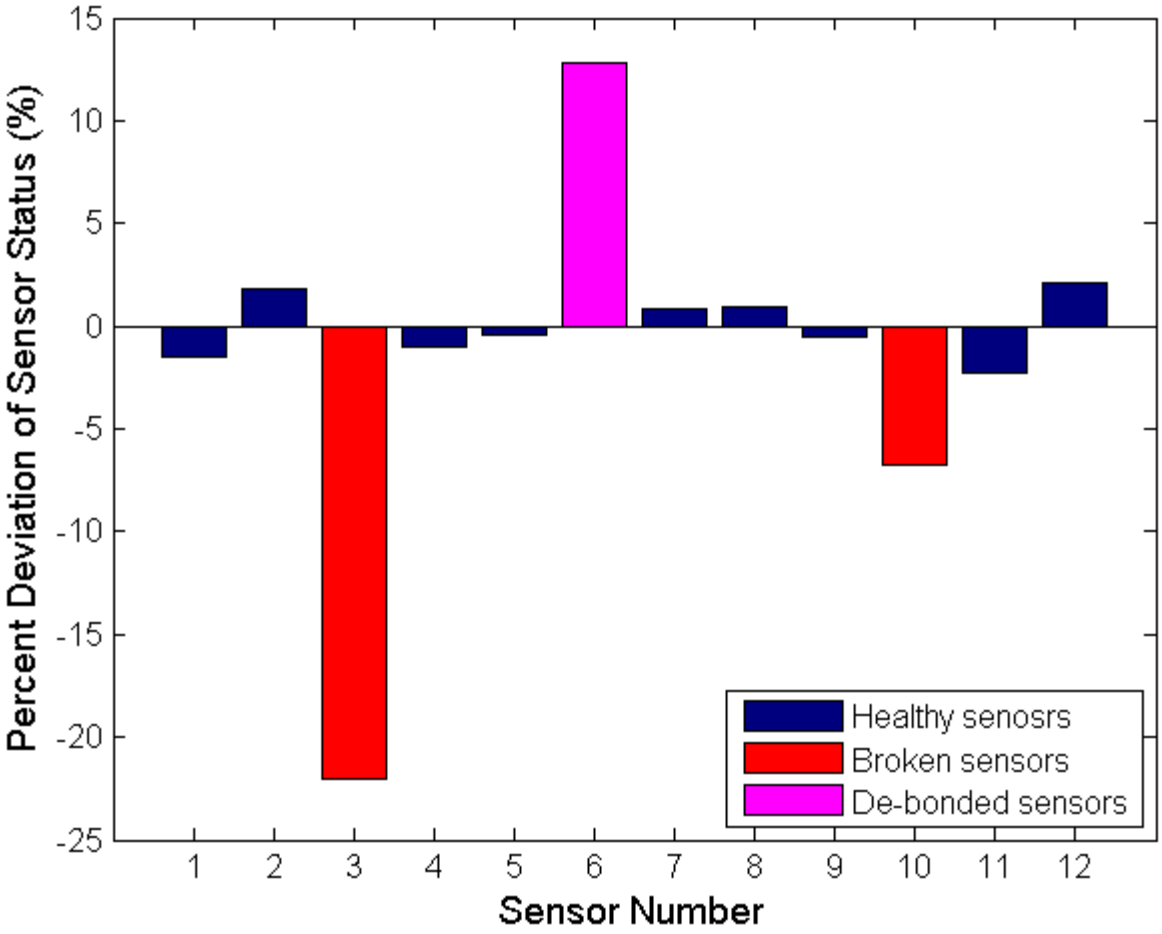
The third column shows the capactive value of each sensor in nF.

The second output (data_for_plotting) is used for graphically illustrating the results.

Plotting Results

```
sdPlot_shm(data_for_plotting)
```


x 10^-10
8
7
6
5
4
3
2
1
0
Standard Deviation (σ)
3
6
10
11
12
2
8
7
1
4
5
9
Sensor Number



The first figure shows the outcome of the automated identification process based on Overly et al. Sensors 3,6,10 are identified faulty and the remaining sensors are healthy.

The second figures show the quantitative results. Blue bars are healthy sensors, red bars are broken sensors, and magenta bars are debonded sensors. This figure shows the percent deviations from the mean values of the healthy sensors.

Published with MATLAB® R2013a

National Instruments DAQ for Ultrasonic Active Sensing

Contents

Introduction

In this example we acquire a set of ultrasonic wave propagation data using National Instruments hardware. In particular, we interface with an NI-FGEN device for high speed waveform actuation, an NI-SCOPE device for high speed digitizing, and an NI-SWITCH device for multiplexing among a set of actuating and sensing transducers. NI-TCLK is invoked to syncronize the highspeed actuation and sensing.

Here, we implement an array of five piezo-electric discs which are each capable of both actuation and sensing. In in each run, the switch connects one transducer to the NI-FGEN waveform generator, and a second transducer to an NI-SCOPE digitizer. We then generate an 80 kHz gaussian modulated sine wave at the actuation transducer, and acquire the incident response wave at the sensing transducer after it has propogated through the structure. The switch then connects the next actuator-sensor pair in lineup, and the process repeats. A wideband amplifier is connected in line with the waveform generator in order to provide sufficient power to the transducers.

The idea behind this sensing scheme is that given some baseline response when the structure is in a known "healthy" condition, the introduction of damage (cracking, corosion, etc) will lead to changes in the response due to mechanisms such as scattering and/or damping.

For structural health monitoring, the next step is to carry out a feature extraction procedure. The example example_activeSensingFeature demonstrates an appropriate feature extraction procedure for the type of active sensing data acquired here.

Requires Instrument Control Toolbox

Data sets can be downloaded from:

http://institute.lanl.gov/ei

References:

www.ni.com

R. Andrew Swartz, Eric Flynn, Daniel Backman, R. Jason Hundhausen, and Gyuhae Park. Active piezoelectric sensing for damage identification in honeycomb aluminum panels. In Proceedings of the IMAC-XXIV. SEM, 2006.

SHMTools functions called:

```
NI_SWITCH_Init_shm
NI_SCOPE_SetOptions_shm
NI_SCOPE_InitConfig_shm
NI_FGEN_SetOptions_shm
NI_FGEN_InitConfig_shm
getGausModSin_shm
NI_FGEN_PrepWave_shm
NI_TCLK_SyncPrep_shm
buildPairList_shm
NI_multiplexSession_shm
```

Author: Eric Flynn

Date Created: June 22, 2009

Clear Instrumentation

```
instrreset();
```

Initialize NI-SWITCH

Initialize NI-SWITCH switch device. "deviceID" must correspond to the ID assigned to the switch device in NI-MAX.

```
deviceID='Switch';

niSwitchHandle=NI_SWITCH_Init_shm(deviceID);
```

Set NI-SCOPE Options

Assign analog input options. "deviceID" must correspond to the ID assigned to the digitizer device in NI-MAX.

```
deviceID='Digitizer';
channelList=0;
minSampleRate=2e7;
minNumPts=100000;

niScopeOptions=NI_SCOPE_SetOptions_shm(deviceID,channelList,minSampleRate,minNumPts);
```

Initialize and Configure NI-SCOPE

Initialize analog input and apply settings

```
[niScopeHandle, niScopeOptions]=NI_SCOPE_InitConfig_shm(niScopeOptions);
```

Set NI-FGEN Options

Assign analog output options. "deviceID" must correspond to the ID assigned to

the function generator device in NI-MAX.

```
deviceID='FGen';
channelNum=0;
sampleRate=niScopeOptions.actualSampleRate;
gain=6;

niFgenOptions=NI_FGEN_SetOptions_shm(deviceID,channelNum,sampleRate,gain);
```

Initialize and Configure NI-FGEN

Initialize analog output and apply settings

```
niFgenHandle=NI_FGEN_InitConfig_shm(niFgenOptions);
```

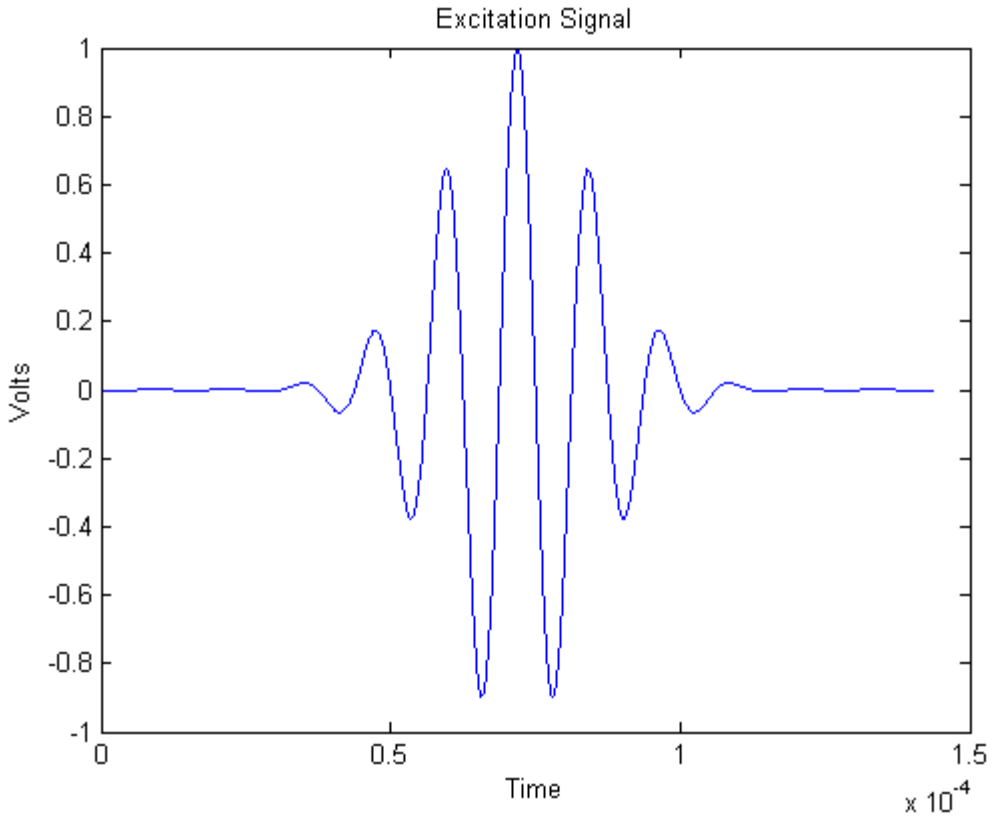
Construct Excitation Signal

Construct an 80kHz Gaussian-modulated sinusoid for excitation

```
centerFreq=80000;
sampleRate=minSampleRate;
normBandwidth=.25;
numPoints=[];

[gausModSin, numPoints]=getGausModSin_shm(centerFreq,sampleRate,normBandwidth,numPoints);
```

Plot Excitation Signal

```
figure
t=(0:numPoints-1)/sampleRate;
plot(t,gausModSin);
title('Excitation Signal');
xlabel('Time');
ylabel('Volts')
```



Prepare Excitation Waveform

Send the excitation waveform to the analog output hardware

```
waveform=gausModSin;

NI_FGEN_PrepWave_shm(niFgenHandle,waveform,niFgenOptions);
```

Prepare AO/AI Syncronization

Use NI-TCLK to syncronize the output and input

```
session1=niFgenHandle;
session2=niScopeHandle;

niTclkSyncHandle=NI_TCLK_SyncPrep_shm(session1,session2);
```

Build pair list

Construct a list of transducer pairs for pitch-catch based actuating and sensing

```
channels=[0 1 2 3];
isBiDirectional=false;
isPitchCatch=true;
isPulseEcho=false;
pairList=buildPairList_shm(channels,isBiDirectional,isPitchCatch,isPulseEcho)
```

```
pairList =

     0     0     0     1     1     2
     1     2     3     2     3     3
```

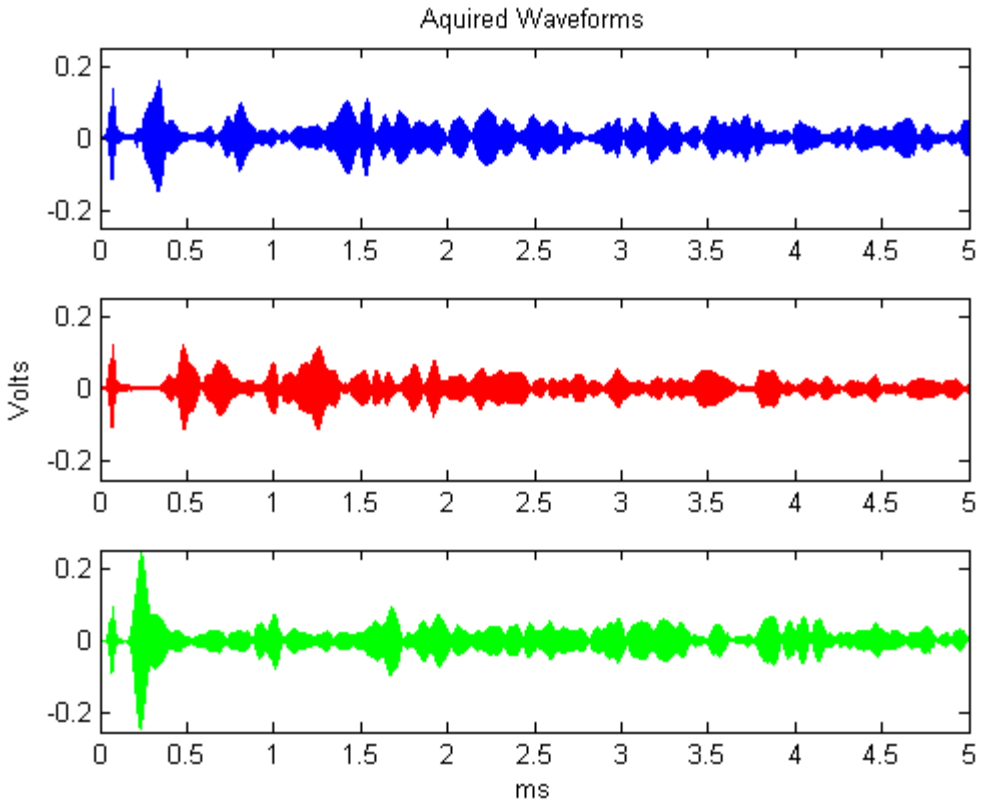
Run Multiplex Session

Cycle through all one-directional actuator/sensor pair combinations, actuating and sensing for each. Average five waveforms for each, and pause 50 ms between measurements. The output channel sepparation corresponds to an NI PXI-2527 switch in dual 16x1 2-wire mode.

```
numberOfAverages=5;
delay=50;
outputChSep=16;

[waveforms]=NI_multiplexSession_shm(niTclkSyncHandle, niSwitchHandle, niScopeHandle,
niScopeOptions, pairList, numberOfAverages, delay, outputChSep);
```

Plot waveforms acquired from pairs 1, 5 and 8

```
figure
t=(0:length(waveforms)-1)/sampleRate*1000;
subplot(3,1,1);
plot(t,waveforms(:,1),'b');
title('Aquired Waveforms');
ylim([-.25 .25]);

subplot(3,1,2);
plot(t,waveforms(:,4),'r');
ylabel('Volts');
ylim([-.25 .25]);

subplot(3,1,3);
plot(t,waveforms(:,6),'g');
xlabel('ms');
ylim([-.25 .25]);
```



Reset instrumentation

```
instrreset;
```

Published with MATLAB® 7.5

Ultrasonic Active Sensing Feature Extraction

Contents

Introduction

In this example we load in a raw set of active sensing data, process it according to the geometry of the plate structure, and extract features relative to the presence of damage.

The test structure was a 0.01 inch concave-shaped plate approximately 48 inches on one side. The plate was instrumented with 32 piezoeletric tranducers which served as both actuators and sensors to form 492 actuator-sensor pairs. Damage was simulated using a two inch neodymium magnet.

The data acquisition system cycled through the actuator-sensor pairs, one at a time, inducing a gaussian windowed sinusoid at the actuator and sensing the propogated wave at the sensor. This was done once before damage was applied and then again after damage. It is assumed that the damage modifies the received waveform through scattering.

This example builds an array of points on the structure for detecting damage at. Individual features are then extracted from the measured waveforms by estimating the wave group velocity and establishing line-of-sight contraints. The final result is a map of the sums of the feature vectors at each point on the structure.

For proper structural health monitoring, the features produced in this example need to be used to build and test against a statistical model in order to decide the damage state of the structure.

Requires data_example_ActiveSense.mat dataset.

SHMTools functions called:

`plotBorder_shm`

```
plotSensors_shm
buildContainedGrid_shm
propagationDist2Points_shm
getPropDist2Boundary_shm
sensorPairLineOfSight_shm
incoherentMatchedFilter_shm
estimateGroupVelocity_shm
distance2Index_shm
extractSubsets_shm
flexLogicFilter_shm
sumMultDims_shm
fill2DMap_shm
plot2DMap_shm
```

Author: Eric Flynn

Date Created: June 22, 2009

Configuration Parameters

Select subset (or all) of sensors to process (0-31)

```
sensorSubset=[0 2 5 7 11 12 15 17 19 21 24 25 27 28 30];
```

Resolution, in inches, of imaging on plate

```
POISpacing=.5;
```

Sample actuator-sensor pair index for plotting

```
samplePairI=4;
```

Make the figure backgrounds white for publishing

```
set(0,'DefaultFigureColor',[1 1 1]);
```

Load Data and DAQ Parameters

Load the data

```
load('data_example_ActiveSense.mat','waveformBase','waveformTest','sensorLayout','pairList',...
    'borderStruct','sampleRate','actuationWaveform','damageLocation');
```

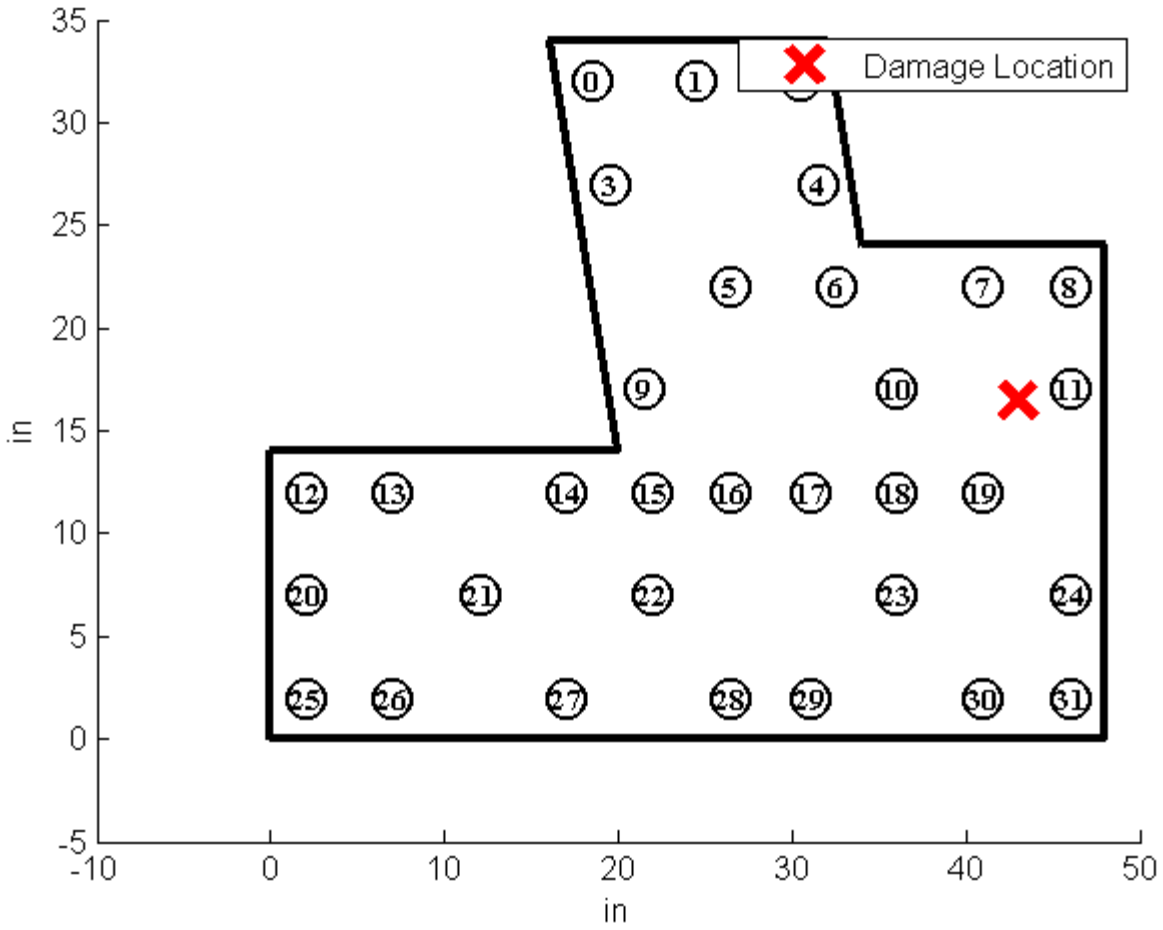
Collect Border Line Segments into One Array

The line segments defining the border are stored as a structure of arrays. Combine them into a single array by concatinating them.

```
borderComb=structCell2Mat_shm(borderStruct);
```

Plot the boundaries, sensors, and damage location

```
figure
axisHandle=axes;
axisHandle=plotBorder_shm(borderComb,axisHandle);
plotSensors_shm(sensorLayout,axisHandle);
hold on
dp=plot(damageLocation(1), damageLocation(2), 'xr', 'MarkerSize', 15, 'LineWidth', 4);
legend(dp,'Damage Location');
xlabel('in');
ylabel('in');
hold off
```



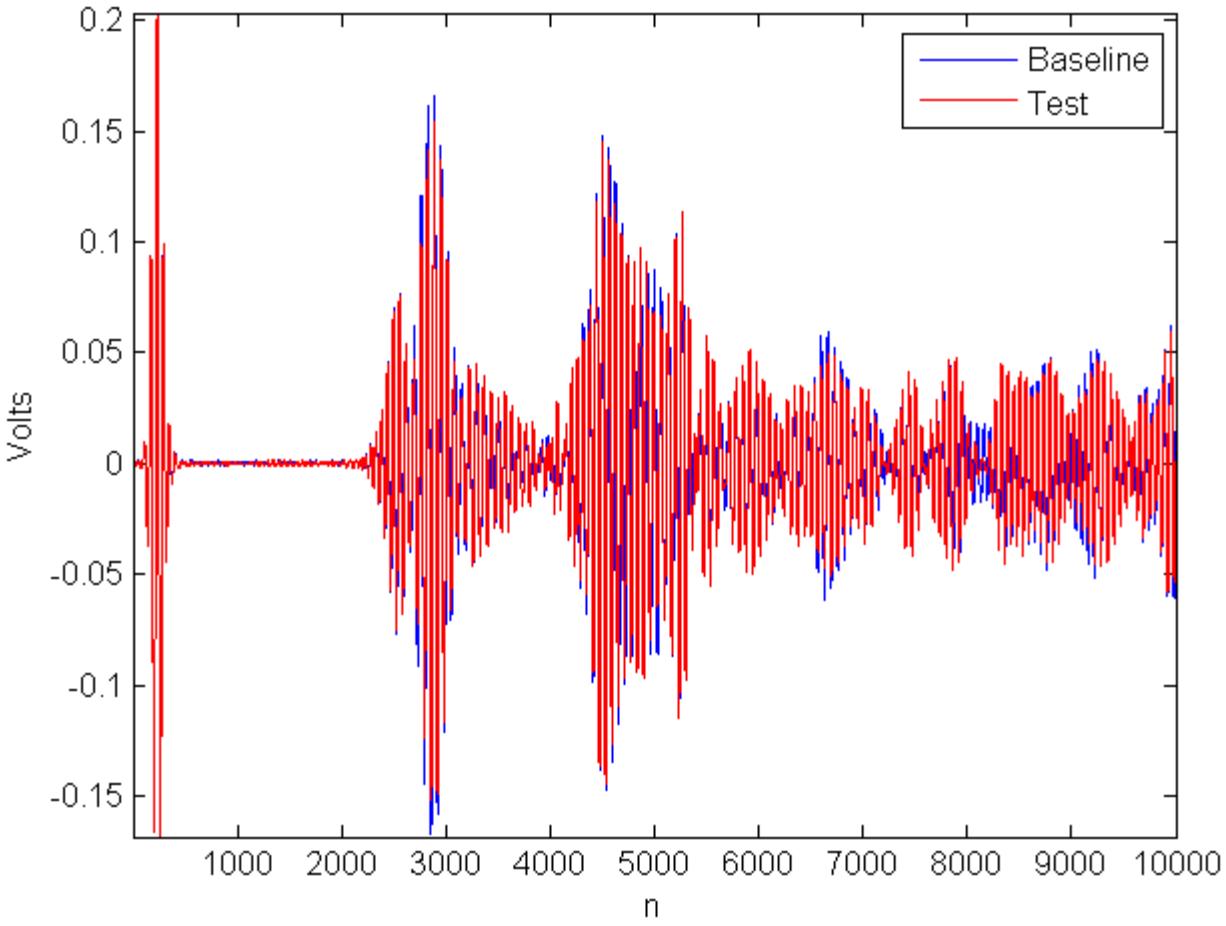
Extract Data for Sensor Subset

Extract the data relevent to the chosen subset of sensors

```
[pairListSub, sensorLayoutSub, waveformBaseSub, waveformTestSub] = reduce2PairSubset_shm...
    (sensorSubset, sensorLayout, pairList,waveformBase, waveformTest);
```

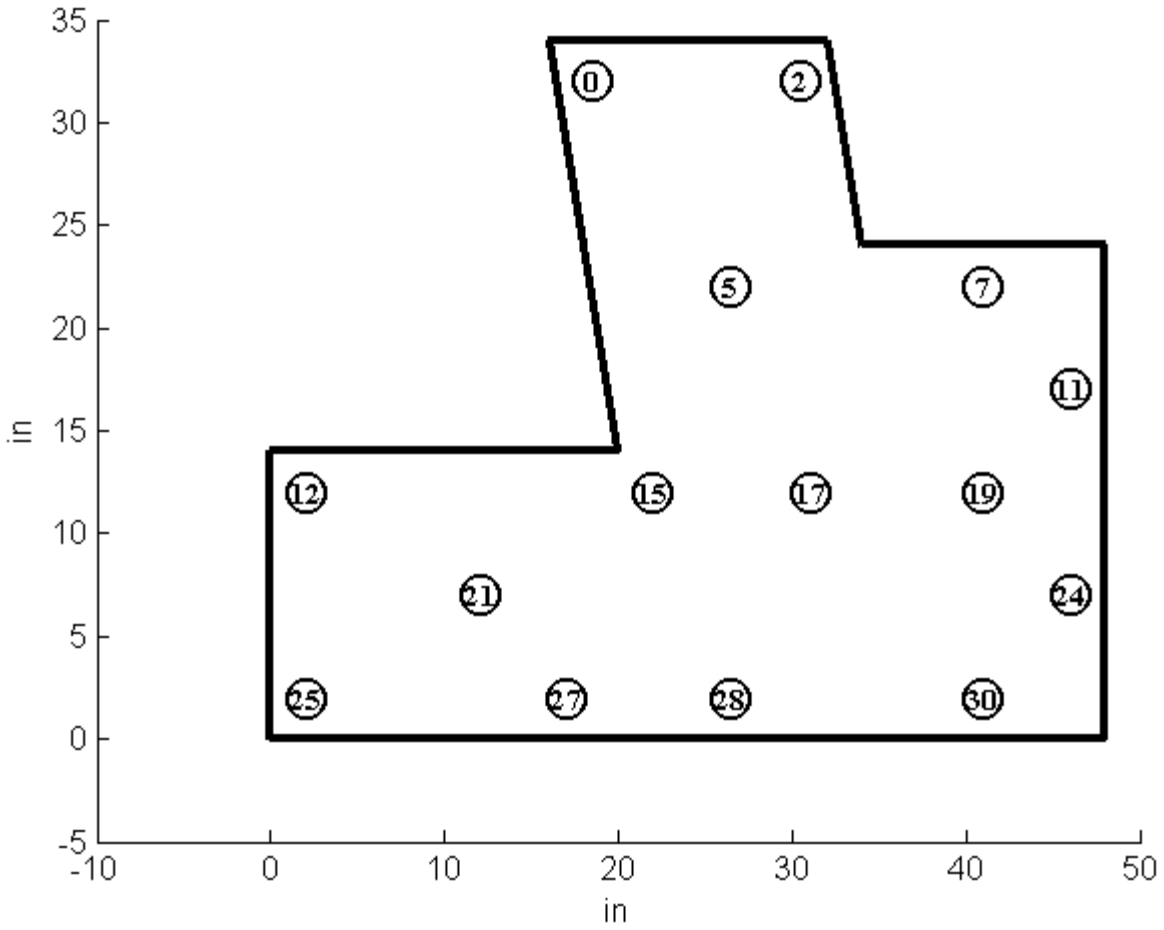
Plot an example waveform

```
figure
plot(waveformBaseSub(:,samplePairI,1)); hold on
plot(waveformTestSub(:,samplePairI,1),'r')
legend('Baseline','Test');
xlabel('n');
ylabel('Volts');
axis tight;
hold off
```



Plot the chosen subset of sensors

```
figure
axisHandle=plotBorder_shm(borderComb,gca);
plotSensors_shm(sensorLayoutSub,axisHandle);
xlabel('in');
ylabel('in');
hold off
```



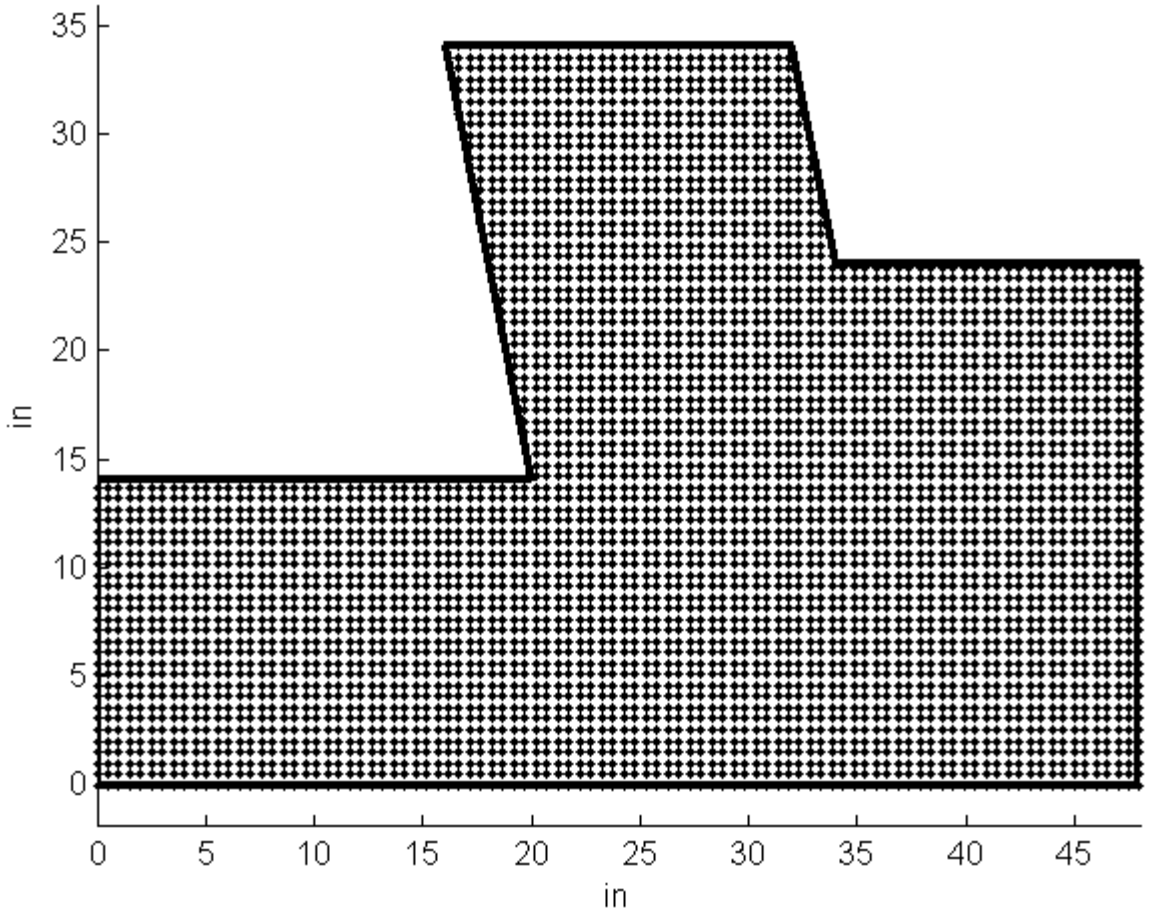
Build Contained Grid of Points

Construct uniform list of points of interest (POIs)

```
xSpacing=POISpacing;
ySpacing=POISpacing;

[pointList, pointMask, xMatrix, yMatrix] = buildContainedGrid_shm(borderStruct, xSpacing, ySpacing);
```

Plot grid of points

```
figure
plotBorder_shm(borderComb,gca); hold on
plot(pointList(1,:),pointList(2,:),'k.');
axis equal;
xlabel('in');
ylabel('in');
hold off;
```



Propogation Distance to Points

Calculate propogation distance from transducer pairs to POIs

```
propDistance=propagationDist2Points_shm(pairListSub,sensorLayoutSub,pointList);
```

Propogation Distance to Boundary

Calculate the propogation distance from transducer pairs to boundaries

```
[propDist,minPropDist] = getPropDist2Boundary_shm(pairListSub, sensorLayoutSub, borderComb);
```

Line of Sight

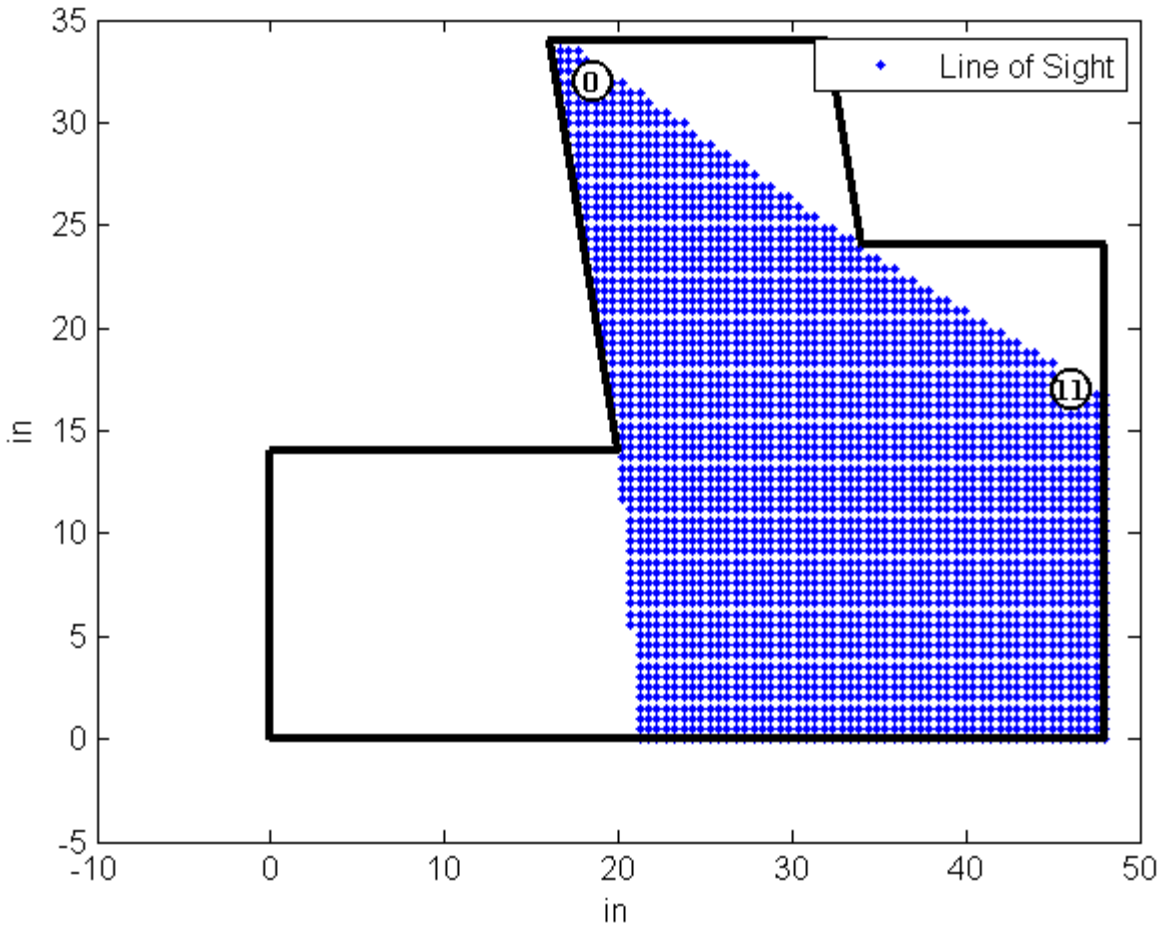
Determine line of sight from transducer pairs to POIs

```
[pairLineOfSight sensorLineOfSight] = sensorPairLineOfSight_shm (pairListSub, sensorLayoutSub,
pointList, borderComb);
```

Plot the line of sight points for the sample pair

```
[singlePair, pairLayout] = reduce2PairSubset_shm(pairListSub(:, samplePairI), sensorLayoutSub,
```

```
pairListSub,[],[]);

figure
plot(pointList(1,pairLineOfSight(samplePairI,:)), pointList(2,pairLineOfSight(samplePairI,:)),'b.')
legend('Line of Sight');
xlabel('in');
ylabel('in');
hold on
plotBorder_shm(borderComb,gca);
plotSensors_shm(pairLayout,gca);
```



Distance Compare

Compare the distance to the POIs to the distance to the nearest boundary

```
distance=propDistance;
maxDistance=minPropDist;
distanceAllowance=0;

[belowMaxDistance]=bsxfun(@lt,distance-distanceAllowance,maxDistance);
```

Estimate Group Velocity

Estimate the Group Velocity of the Wave

Filter the waveforms

```
waveform=waveformBaseSub;
matchedWaveform=actuationWaveform;
filteredWaveform2=incoherentMatchedFilter_shm(waveform,matchedWaveform);
```

Calculate the group velocity

```
waveform=filteredWaveform2;
actuationWidth=length(actuationWaveform);
lineOfSight=[];

[estSpeed, speedList]=estimateGroupVelocity_shm(waveform, pairListSub, sensorLayoutSub, sampleRate,
actuationWidth, lineOfSight);
```

Distance 2 Index

Translate propogation distances to waveform indices using group velocity

```
wavespeed=estSpeed;
offset=actuationWaveform;

indices=distance2Index_shm(propDistance,sampleRate,wavespeed,offset);
```

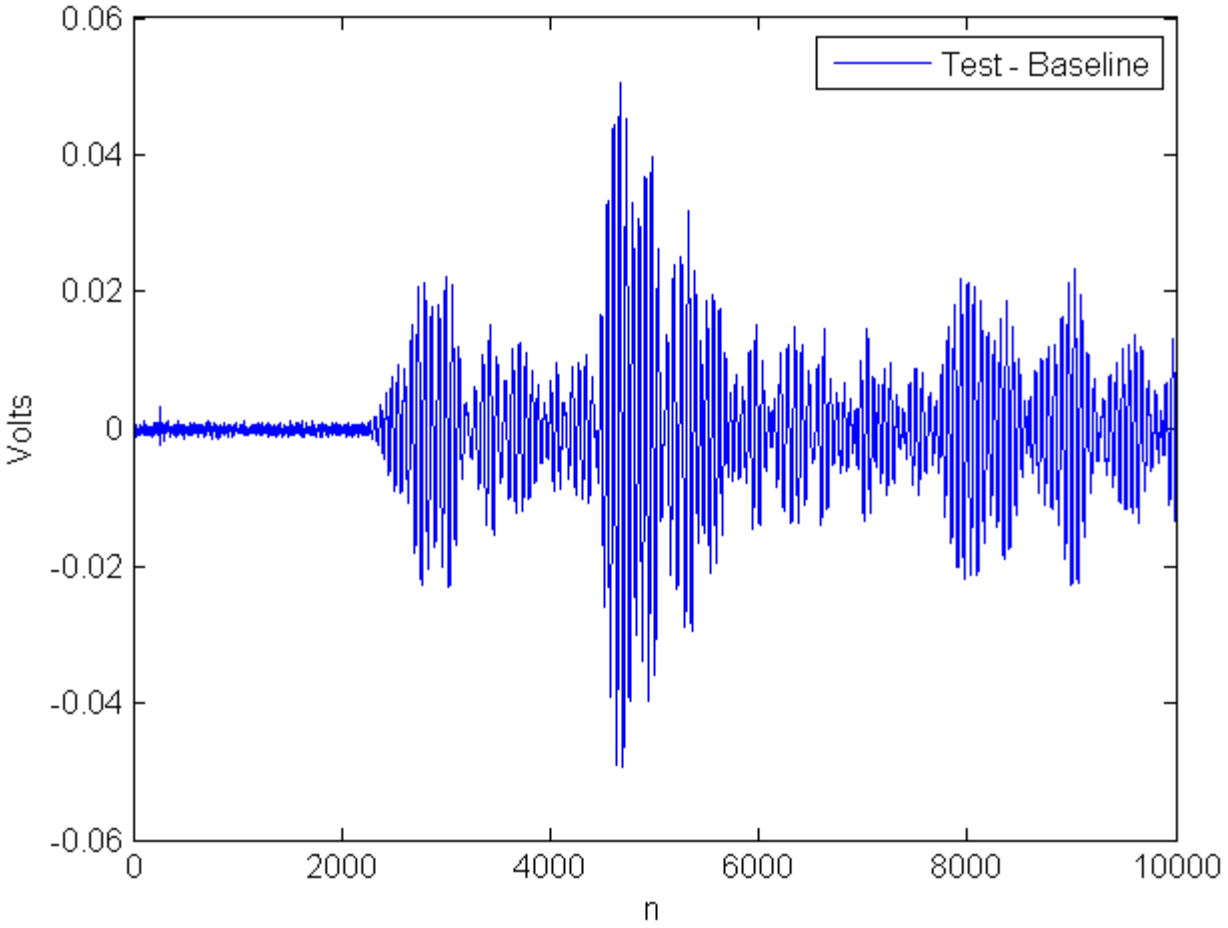
Difference

Subtract the baseline waveforms from the test waveforms

```
dataDifference=waveformTestSub-waveformBaseSub;
```

Plot sample waveform difference

```
figure
plot(dataDifference(:,samplePairI,1))
legend('Test - Baseline');
xlabel('n');
ylabel('Volts');
```



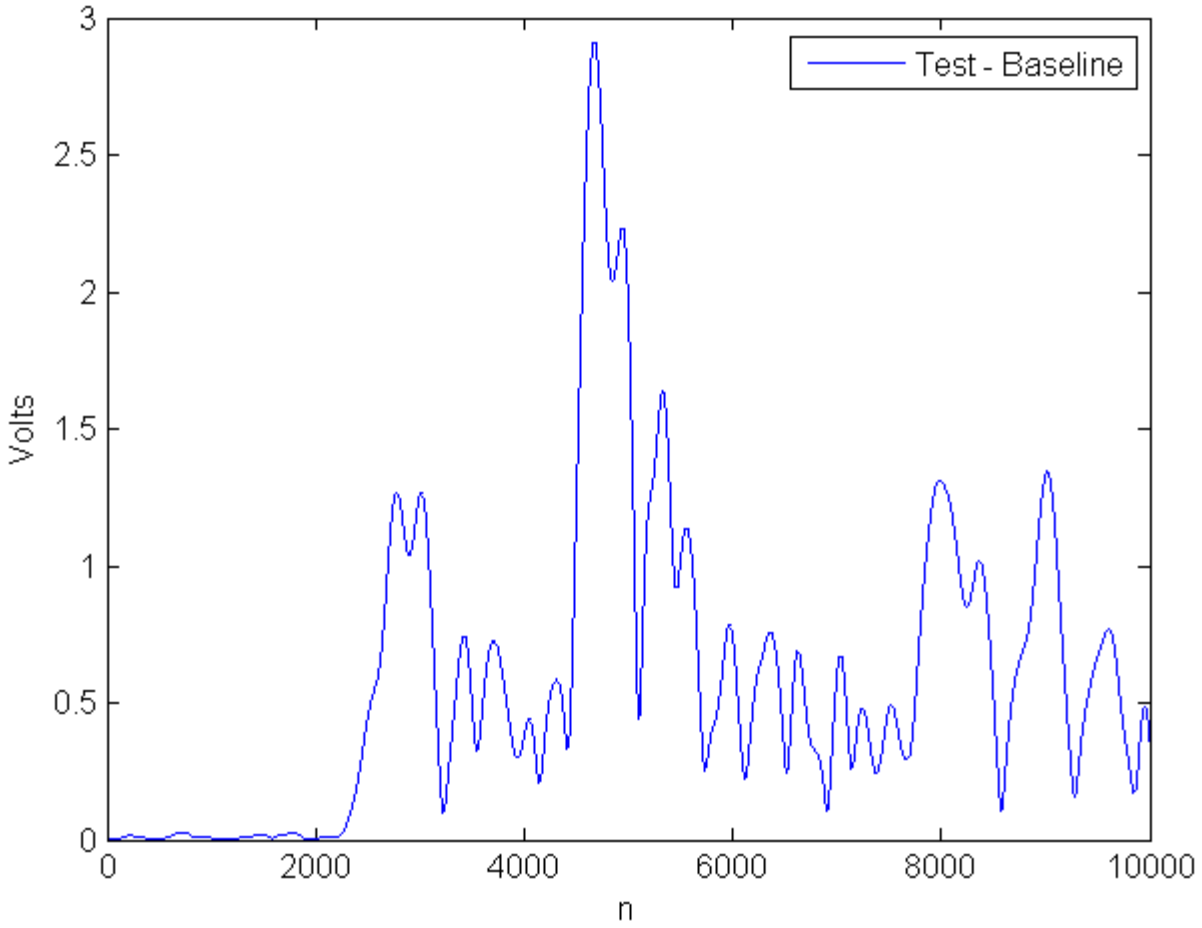
Incoherent Matched Filter

Apply incoherent matched filter to waveform difference

```
waveform=dataDifference;
matchedWaveform=actuationWaveform;

filteredWaveform=incoherentMatchedFilter_shm(waveform,matchedWaveform);
```

Plot sample filtered waveform difference

```
figure
plot(filteredWaveform(:,samplePairI,1))
legend('Test - Baseline');
xlabel('n');
ylabel('Volts');
```



Extract Subset

Extract the matched filter value for each POI using time of flight indices

```
data=filteredWaveform;
startIndices=indices;
subsetLength=1;

dataSubset=extractSubsets_shm(data,startIndices,subsetLength);
```

Apply Logic Filters

Zero-out contributions from transducer pairs without line of sight

```
data=dataSubset;
logicFilter=pairLineOfSight;

[filteredData]=flexLogicFilter_shm(data,logicFilter);
```

Zero-out contributions from transduer pairs that are closer to a boundary than the POIs

```
data=filteredData;
logicFilter=belowMaxDistance;
```

```
[filteredData]=flexLogicFilter_shm(data,logicFilter);
```

Sum Dimensions

Sum across transducer pairs for each POI

```
data=filteredData;
dimensions=[1 2];

dataSum=sumMultDims_shm(data,dimensions);
```

Fill 2D Map

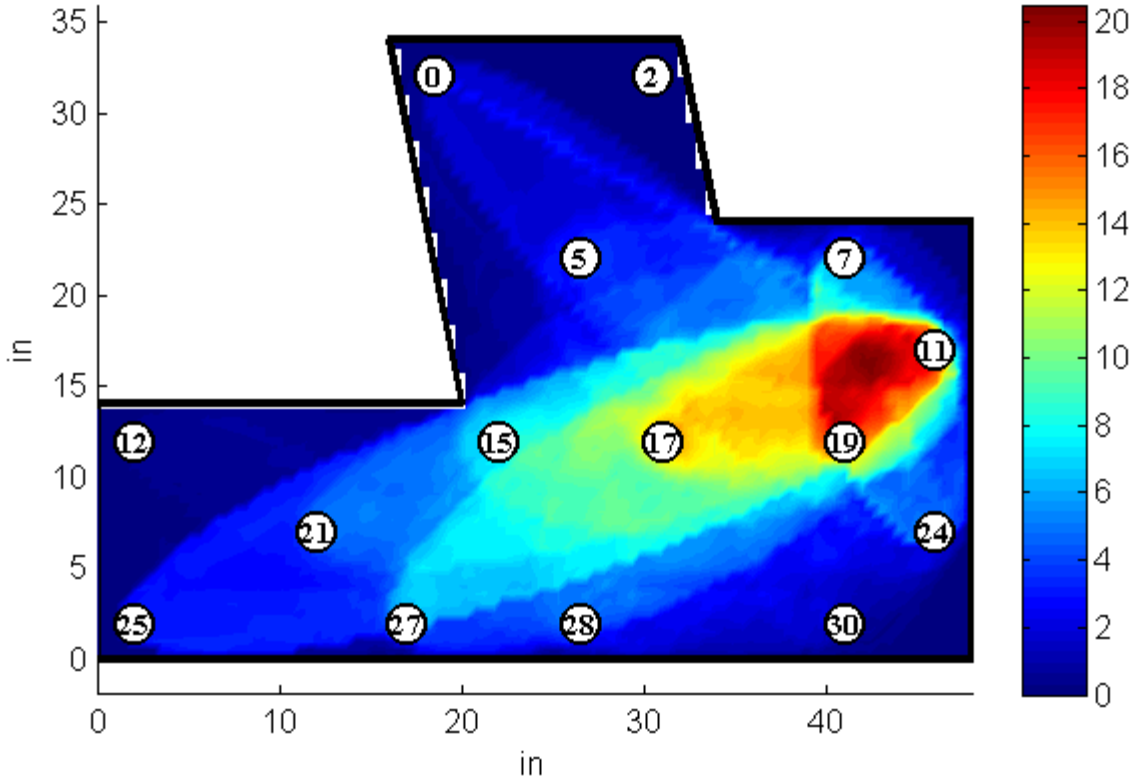
Translate POI list into 2D Map using mask

```
data1D=dataSum;
mask=pointMask;

dataMap2D=fill2DMap_shm(data1D,mask);
```

Plot 2D Map

Plot 2DmMap of POI levels

```
sensorLayout=sensorLayoutSub;

figure
axisHandle=plot2DMap_shm(xMatrix,yMatrix,dataMap2D,gca);

axisHandle=plotBorder_shm(borderComb,axisHandle);
axisHandle=plotSensors_shm(sensorLayout,axisHandle);
xlabel('in');
ylabel('in');
```



Published with MATLAB® R2013a

Example Usage: Assemble a Custom Detector

Contents

Introduction

In this example we show how to assemble a custom detector from the various functions available in the library. This is simple: one just needs to call the "assembleOutlierDetector" which then walks the user through the available routines by category. The three main categories are: Nonparametric detectors, Semiparametric detectors, and Parametric detectors. So let's try this.

Author: Samory Kpotufe

Date Created: August 21, 2009

Calling the assembly routine.

This can be done from any directory. It takes a default parameter "dirBase" which should be a directory containing subdirectories corresponding to the main categories of detectors described above. If this parameter is not provided then, the directory containing the file "assembleOutlierDetector.m" is used.

Assembling a Nonparametric routine.

These are based on kernel density estimation, and we'd be asked to choose a kernel method. The available kernel methods are in "NonParametricDetectors/Kernels". We present an excerpt of the call sequence below.

```
assembleOutlierDetector_shm('NonParam')

DirBase was not provided, we'll be using the following base
directory:
 /home/samory/Research/LANL/SoftwarePack/

Types of detectors are:
 [1] Non parametric detectors
 [2] Semi parametric detectors
 [3] Parametric detectors

Enter a type (number): 1
You chose to assemble a non parametric detector.

Options for <learning methods,scoring methods> pairs are:
 [1] <learnKernelDensity_shm.m, scoreKernelDensity_shm.m>

Choose one (number): 1

learnKernelDensity_shm arguments are:
```

```
 [1] Xtrain
 [2] H
 [3] kernelFun
 [4] bs_method

Select an argument (number) to set, or write 0 to continue: 3
 Enter a value for argument "kernelFun": @epanechnikovKernel_shm

Select an argument (number) to set, or write 0 to continue: 4
 Enter a value for argument "bs_method": 2

Select an argument (number) to set, or write 0 to continue: 0
 Done, assembled training routine:
 "/home/samory/Research/LANL/SoftwarePack//AssembledDetectors/
 trainDetector_NonParam.m",
 to be used with "detectOutlier" detection routine.
```

Assembling a Semiparametric routine.

These routines work by first partitioning the data, then learning a parametric model on each partition (the available parametric model now is a multi-dimensional Gaussian). The user will be given a list a partitioning method. These can be found in "SemiParametricDetectors/PartitioningAlgorithms".

Assembling a Parametric routine

This is the simplest type to assemble. The user is just given a list of the available paremetric routines and given the option to set their parameters.

Using the assembled detectors.

The just assembled detector would be save in the directory "AssembledDetectors" which is at the same level as the file "assembleOutlierDetector.m".

The assembled detectors are training routines to be used in conjunction with the detection routine 'detectOutlier_shm'. Each of the assembled routines have the same function signatures and can be used the same way as the default 'trainOutlierDetector' routine. In the two examples default usage and default thresholding, the calls to 'trainOutlierDetector' can be replaced by calls to any of the custom routines we just assembled.

Published with MATLAB® R2013a

Example Usage: How to Use the Default Detectors

Contents

Introduction

Here we show how to get started with using the default mechanism provided to the user. There are two important functions: trainOutlierDetector_shm, and detectOutlier_shm. The first is used to learn a model (mixture of Gaussians) over the training data, this model is saved in the working directory and used by subsequent calls to detectOutlier. Also a threshold is passed to the detection routine. In this example we show a different way to select the threshold besides the default way shown in the default usage example.

The data used in this example is from the 3-story structure. More details about the data sets can be found in 3-Story Data Sets.

Requires data3SS4.mat dataset.

SHMTools functions called:

```
arModel_shm
trainOutlierDetector_shm
detectOutlier_shm
```

Author: Samory Kpotufe

Date Created: August 19, 2009

Load data

The data here is in the form of time series in a 3 dimensional matrix (time, sensors, instances) and also a state vector representing the various environmental conditions under which the data is collected.

```
load('data3SS.mat');
```

For this example, we will break each 8192 point time series into 4, 2048 point time series.

```
timeData = zeros(2048,5,680);
timeDataStates = zeros(1,680);
for i=1:4
    timeData(:,:,i:4:680)=dataset((1+2048*(i-1)):(2048*i),:,:);
    timeDataStates(:,i:4:680)=states(:,:);
end
```

Extract some features using your favorite function, but first pick N of the instances (each time series reading over all sensors). Each instance is then transformed into a feature vector: the returned matrix has the form (instances, features).

```
N = 680;
Idx = randperm(size(timeData, 3));
Idx = Idx(1:N);
Xdata = arModel_shm(timeData(:, :, Idx));
Xstates = timeDataStates(Idx);
```

Now set 80% of states 1:9 aside as the training data, these states correspond to undamaged readings. We'll then test on the remaining 20% of 1:9 and on the "damaged" states 10:17.

```
Idx = logical(ismember(Xstates, [1:9]));
Xundamaged = Xdata(Idx, :);
nUndamaged = size(Xundamaged, 1);
nTrain = round(0.8*nUndamaged);
Xtrain = Xundamaged(1:nTrain, :);
Xtest = [Xundamaged(nTrain+1:nUndamaged, :); Xdata(~Idx, :)];
nTest = size(Xtest, 1);
```

Now set labels for the test data, 0 corresponds to undamaged, and 1 to damaged.

number of undamaged in test.

```
nTest_0 = nUndamaged -nTrain;
```

test labels

```
testLabels = [zeros(nTest_0, 1); ones(nTest -nTest_0, 1)];
```

Train a model over the undamaged data

The call below will learn a mixture of k gaussians, and select a threshold over the likelihoods values of the feature vectors under the learned distribution. The threshold is selected as follows: first a 'normal' distribution is learned over the likelihood values of the feature vectors; then the threshold is picked so that 0.9 (confidence) mass of the learned distribution is above the threshold. Any distribution supported by the 'mle' function can be used.

```
k = 5;
confidence = 0.9;
distForScores = 'normal';
trainOutlierDetector_shm(Xtrain, k, confidence, [], distForScores);
```

```
****************** TRAIN DAMAGE_DETECTOR ***************************
Start learning model of undamaged conditions ----
Learning treshold at the 90.00 percent cutoff  ----
The threshold picked is 78.33
Learning a confidence model
Saving the models into model file UndamagedModel.mat
```

Test on the saved model and threshold.

Note that we can also specify our threshold and many other parameters, see the help on detectOutlier.

```
[results, confidences, scores] = detectOutlier_shm(Xtest);
```

```
****************** DETECT OUTLIER ***************************
Loading models in model file UndamagedModel.mat
Start test on n= 392 data points of dimension D = 25
Threshold used: 78.3327 ... Anything over : 0.90 likelihood of damage will be flagged as "1 =
Damage"
```

Report the detector's performance.

Various error rates

```
TotalErr = sum(results ~= testLabels)/nTest;
falsePositiveErr = sum(results(1:nTest_0) ~= 0)/nTest_0;
falseNegativeErr = sum(results(nTest_0+1:end)~=1)/(nTest - nTest_0);
fprintf('\n Total error: %2.2f\n False Positive rate: %2.2f\n False Negative rate: %2.2f\n',
TotalErr, falsePositiveErr, falseNegativeErr);
```

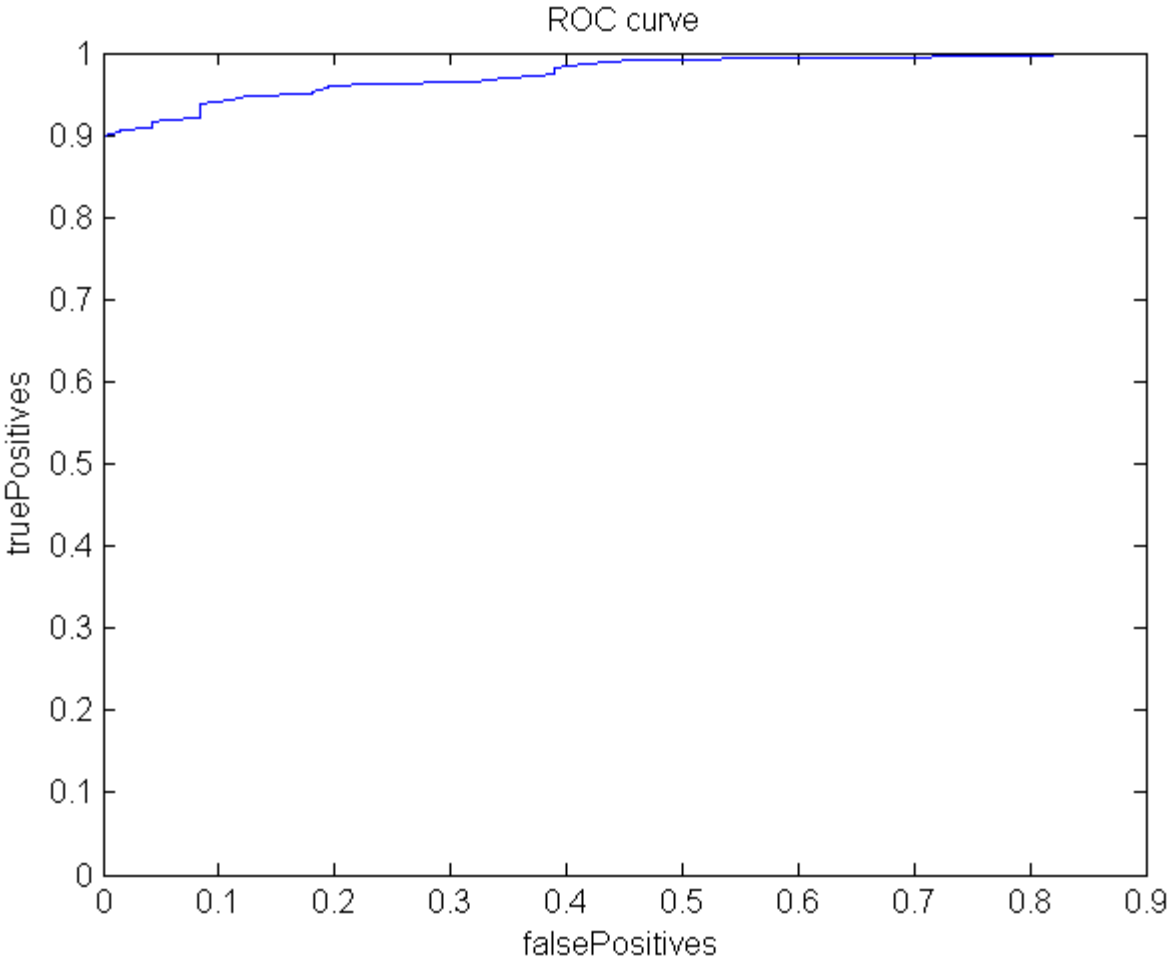
```
 Total error: 0.09
 False Positive rate: 0.38
 False Negative rate: 0.03
```

Compute ROC curve

```
[truePositives,falsePositives] = ROC_shm(scores,testLabels);
```

Now plot the curve

```
figure;
plot(falsePositives, truePositives);
xlabel('falsePositives');
ylabel('truePositives');
title('ROC curve');
```



Published with MATLAB® R2013a

Outlier Detection based on Nonlinear Principal Component Analysis

Contents

Introduction

The goal of this example usage is to discriminate time histories from undamaged and damaged condition based on outlier detection. The first four statistical moments are used as damage-sensitive features and a machine learning algorithm based on nonlinear principal component analysis (NLPCA) is used to create damage indicators (DIs) invariant for feature vectors from normal structural condition and that increase when feature vectors are from damaged structural conditions.

Data sets from Channel 5 of the 3-story structure are used in this example usage. More details about the data sets can be found in 3-Story Data Sets.

Requires Neural Networking Toolbox and data3ss.mat dataset.

References:

Figueiredo, E., Park, G., Figueiras, J., Farrar, C., & Worden, K. (2009). Structural Health Monitoring Algorithm Comparisons using Standard Data Sets. Los Alamos National Laboratory Report: LA-14393.

Sohn, H., Worden, K., & Farrar, C. R. (2002). Statistical Damage Classification under Changing Environmental and Operational Conditions. Journal of Intelligent Material Systems and Structures , 13 (9), 561-574.

Kramer, M. A. (1991). Nonlinear Principal Component Analysis using Autoassociative Neural Networks. AIChE Journal , 37 (2), 233-243.

SHMTools functions called:

```
arModel_shm
learnNLPCA_shm
scoreNLPCA_shm
```


Author: Elói Figueiredo

Date Created: September 01, 2009

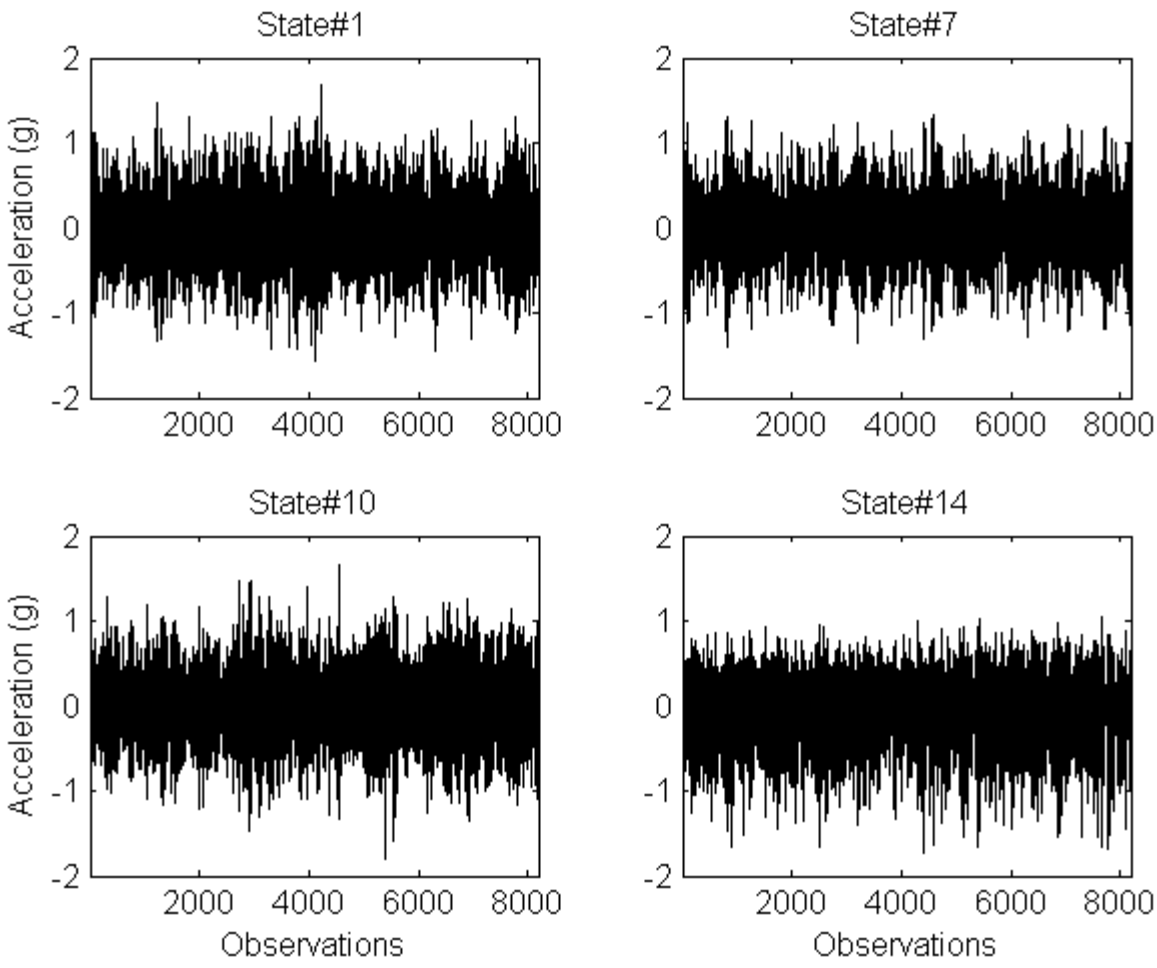
Load Raw Data

Load data set composed of acceleration time histories:

```
load('data3SS.mat');
```

Plot one acceleration time history (Channel 5) from four state conditions:

```
states=[1 7 10 14];

figure

for i=1:4

    subplot(2,2,i)
    plot(dataset(:,5,states(i)*10),'k')
    title(['State#',num2str(states(i))])
    if i==3 || i==4,xlabel('Observations'); end
    if i==1 || i==3, ylabel('Acceleration (g)'); end
    set(gca,'YTick',-2:2,'Xlim',[1 8192],'Ylim',[-2 2])

end
```



Extraction of Damage-Sensitive Features

The first four statistical moments (mean, standard deviation, skewness and kurtosis) are extracted from each time history and stored into a feature vector. Note that the data for the training process is not used later in the test process.

Estimation of the statistical moments:

```
[statMoments]=statMoments_shm(dataset(:,5,:));
```

Training data (undamaged feature vectors):

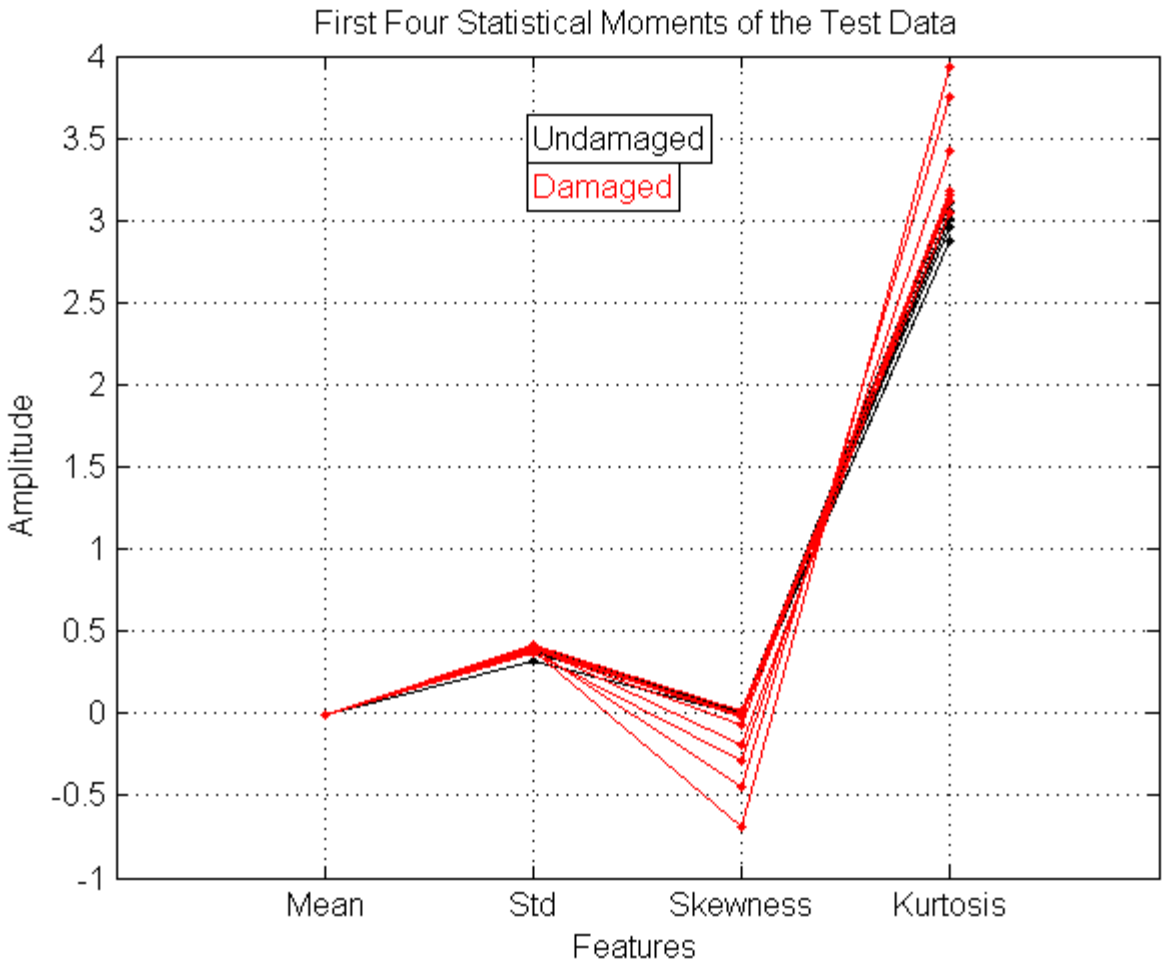
```
clear learnData
for i=1:9;
    learnData(i*9-8:i*9,:)=statMoments(i*10-9:i*10-1,:);
end
```

Test data (9 undamaged and 8 damaged feature vectors):

```
scoreData=statMoments(10:10:170,:);

[n m]=size(scoreData);
```

Plot test data:

```
labels{1}='Mean'; labels{2}='Std'; labels{3}='Skewness'; labels{4}='Kurtosis';

figure
plot(1:m,scoreData(1:9,:),'.-k',1:m,scoreData(10:17,:),'.-r')
title('First Four Statistical Moments of the Test Data')
xlabel('Features')
ylabel('Amplitude')
set(gca,'XTick',1:4,'XTickLabel',labels,'Xlim',[0 5])
text(2,3.5,'Undamaged','Color','k','EdgeColor','k','BackgroundColor','w')
text(2,3.2,'Damaged','Color','r','EdgeColor','k','BackgroundColor','w')
grid on
```



Statistical Modeling for Feature Classifiction

The NLPCA-based machine learning algorithm is used to create DIs invariant under feature vectors from the undamaged structural condition. The two nodes at the bottleneck layer represent the changes in mass and stiffness. Four nodes are assumed in both mapping layers.

Training:

```
[model]=learnNLPCA_shm(learnData,2,4);
```

Scoring:

```
[DI]=scoreNLPCA_shm(scoreData,model);
```

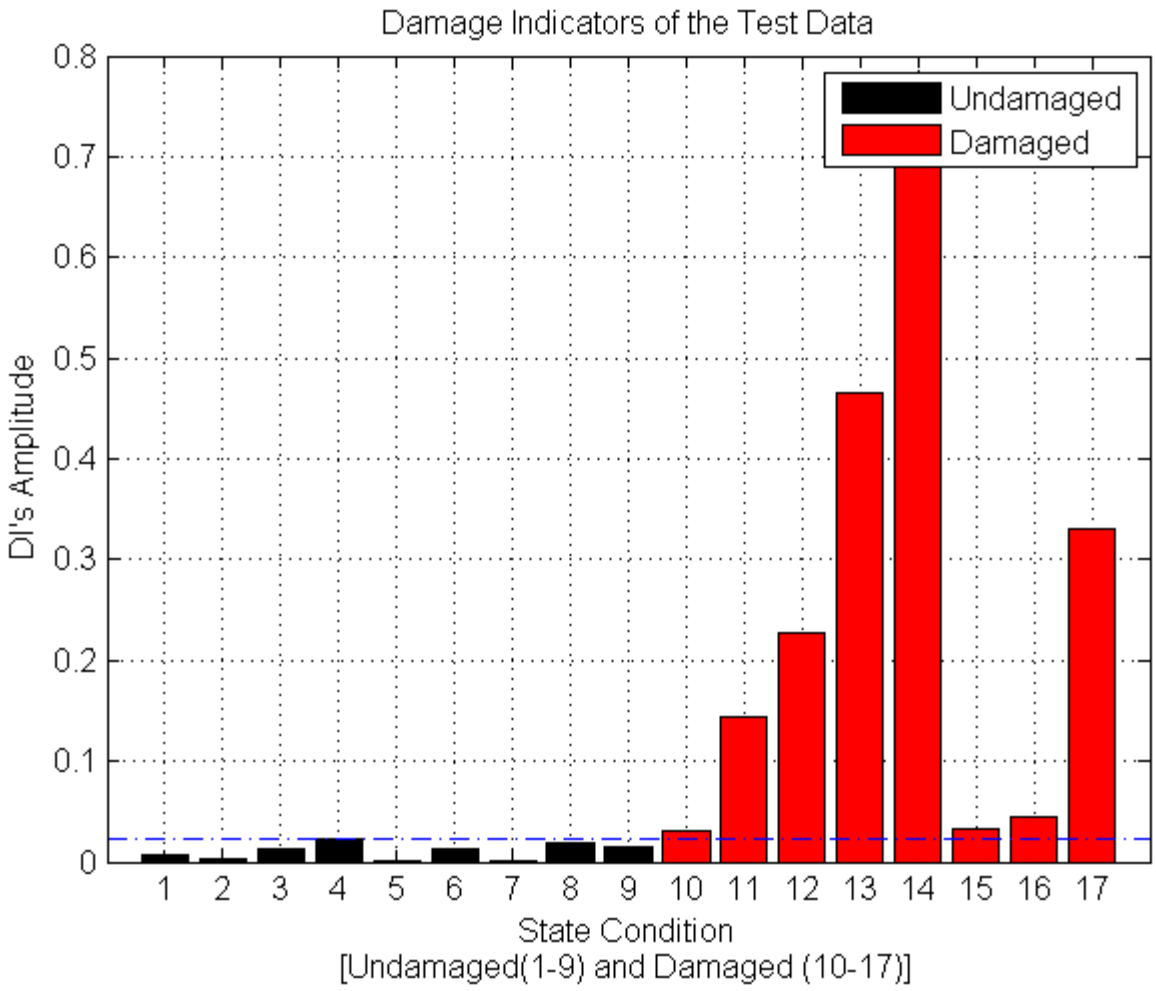
Plot Damage Indicators

Threshold based on the 95% cut-off over the training data:

```
[threshold]=scoreNLPCA_shm(learnData,model);
threshold=sort(-threshold);
UCL=threshold(round(length(threshold)*0.95));
```

Plot DIs:

```
figure;
bar(1:9,-DI(1:9),'k'); hold on; bar(10:17,-DI(10:17),'r')
title('Damage Indicators of the Test Data')
xlabel({'State Condition','[Undamaged(1-9) and Damaged (10-17)]'})
ylabel('DI''s Amplitude')
set(gca,'Xlim',[0 n+1],'XTick',1:n,'XTickLabel',1:n)
line('XData',[0 n+1],'YData',[UCL UCL],'Color','b','LineWidth',1,'LineStyle','-.')
legend('Undamaged','Damaged')
grid on
```



Note that the performance of this algorithm can be improved by changing the architecture of the network, such as by increasing the number of nodes in the mapping layers.

See also:

Outlier Detection Based on the Factor Analysis Model

Outlier Detection Based on Principal Component Analysis

Outlier Detection Based on the Singular Value Decomposition

Outlier Detection Based on the Mahalanobis Distance

Published with MATLAB® R2013a

Outlier Detection based on Factor Analysis

Contents

Introduction

The goal of this example usage is to discriminate time histories from undamaged and damaged condition based on outlier detection. The parameters from an autoregressive (AR) model are used as damage-sensitive features and a machine learning algorithm based on the factor analysis (FA) model is used to create damage indicators (DIs) invariant for feature vectors from normal structural condition and that increase when feature vectors are from damaged structural condition.

Data sets from Channel 5 of the 3-story structure are used in this example usage. More details about the data sets can be found in 3-Story Data Sets.

Requires data3SS.mat dataset.

References:

Kullaa, J. (2003). Is Temperature Measurement Essential in Structural Health Monitoring? Proceedings of the 4th International Workshop on Structural Health Monitoring 2003: From Diagnostic & Prognostics to Structural Health Monitoring (pp. 717-724). DEStech Publications, Inc.

SHMTools functions called:

```
arModel_shm
learnFactorAnalysis_shm
scoreFactorAnalysis_shm
```


Author: Elói Figueiredo

Date Created: September 01, 2009

Load Raw Data

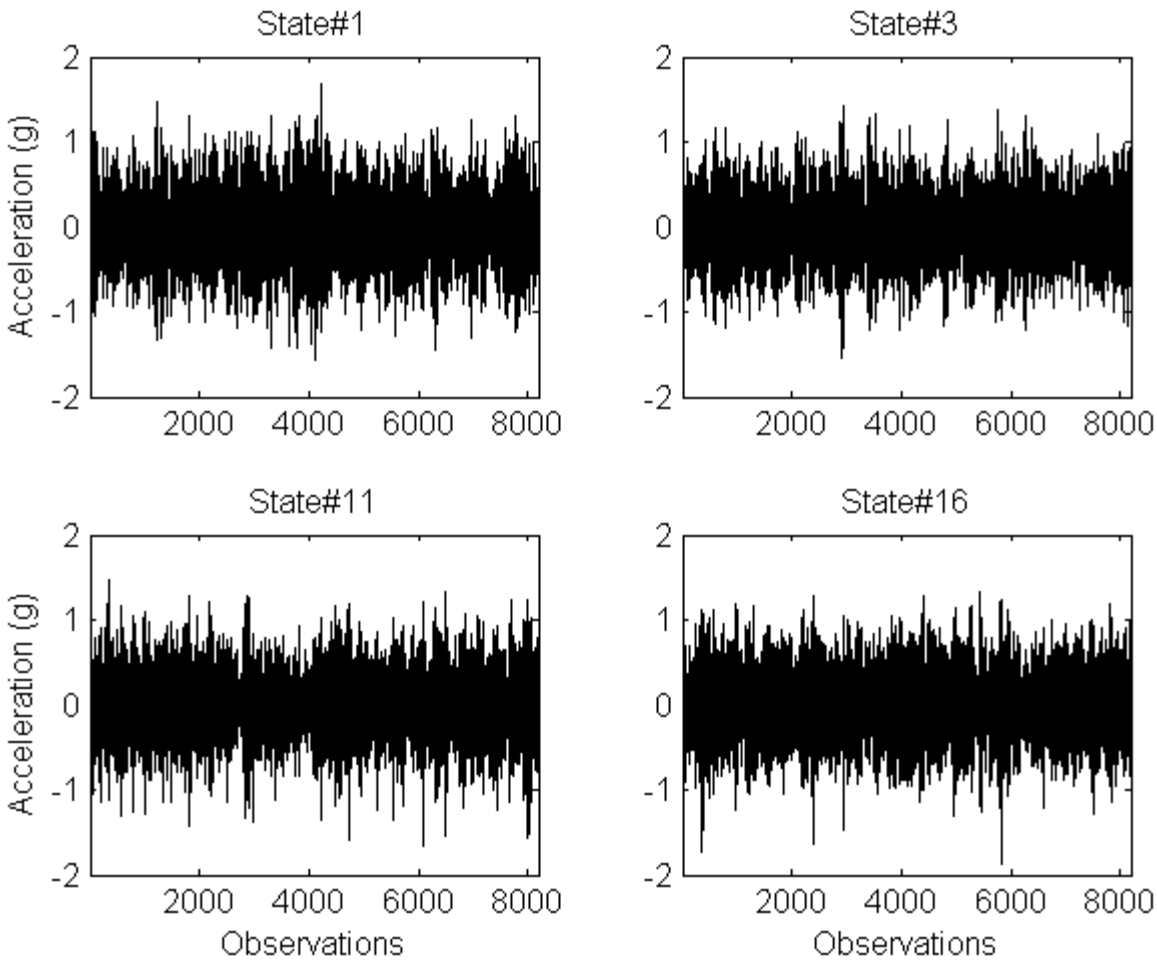
Load data set composed of acceleration time histories:

```
load('data3SS.mat');
```

Plot one acceleration time history from four state conditions:

```
states=[1 3 11 16];

figure

for i=1:4
```

```
    subplot(2,2,i)
    plot(dataset(:,5,states(i)*10),'k')
    title(['State#',num2str(states(i))])
    if i==3 || i==4,xlabel('Observations'); end
    if i==1 || i==3, ylabel('Acceleration (g)'); end
    set(gca,'YTick',-2:2,'Xlim',[1 8192],'Ylim',[-2 2])

end
```



Extraction of Damage-Sensitive Features

AR parameters are extracted from acceleration time histories. It is assumed an AR(15) model. The order of the model was picked from the lower-bound of the range given by the optimization methods available in this package. (For more details see example usage)

AR model order:

```
arOrder=15;
```

Estimation of the AR parameters:

```
[arParameters]=arModel_shm(dataset(:,5,:),arOrder);
```

Training data (undamaged feature vectors):

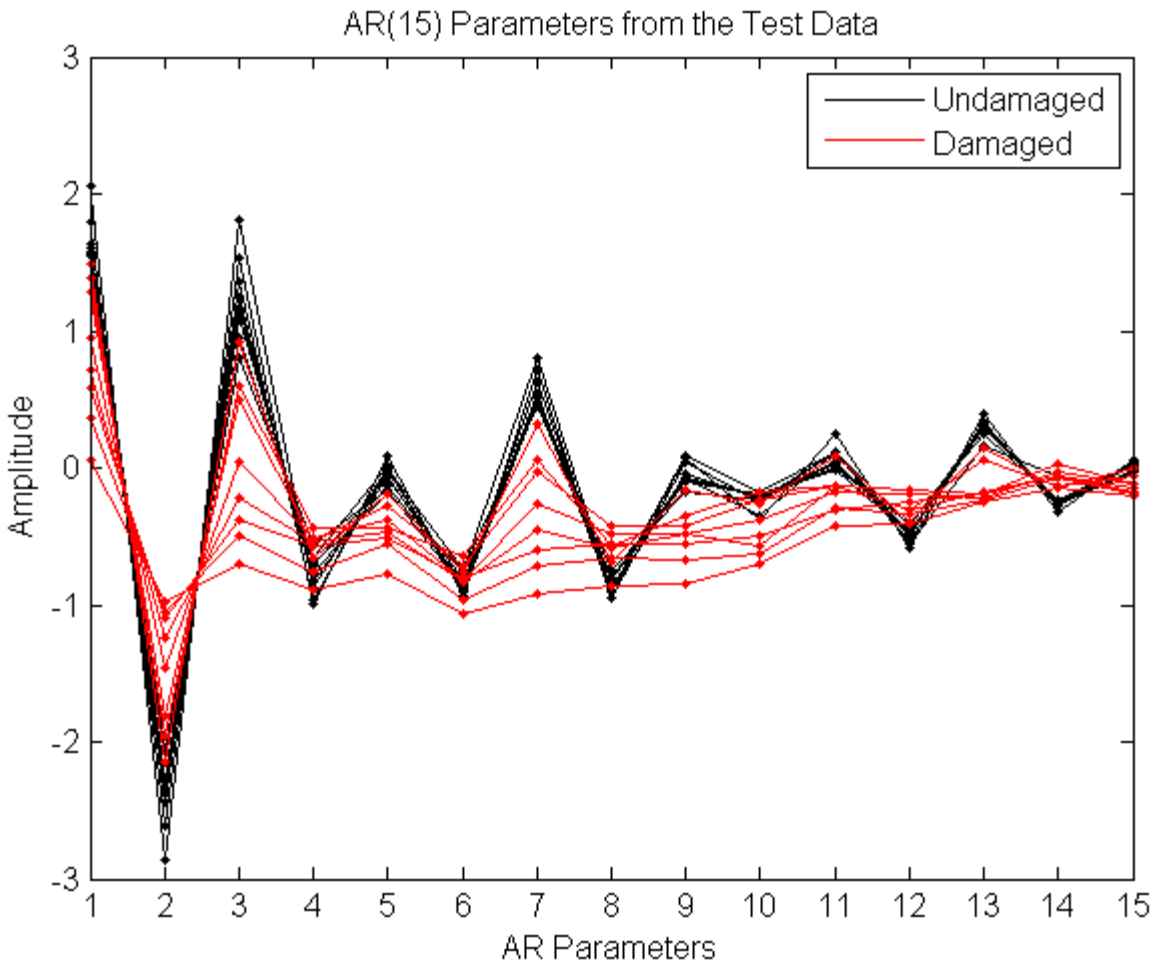
```
clear learnData
for i=1:9;
    learnData(i*9-8:i*9,:)=arParameters(i*10-9:i*10-1,:);
end
```

Test data (9 undamaged and 8 damaged feature vectors):

```
scoreData=arParameters(10:10:170,:);
```

Plot test data:

```
figure
plot(1:arOrder,scoreData(1:9,:),'.-k',1:arOrder,scoreData(10:17,:),'.-r')
title(['AR(',num2str(arOrder),') Parameters from the Test Data'])
xlabel('AR Parameters')
ylabel('Amplitude')
set(gca,'XTick',1:arOrder,'Xlim',[1 arOrder])
M(1,1:9)='Undamaged'; M(2,1:7)='Damaged';
legend([line('color','k');line('color','r')],M);
```



Note: The curves in the figure above correspond to 17 feature vectors from the undamaged (State#1-9) and damaged conditions (State#10-17).

Statistical Modeling for Feature Classifiction

The FA-based machine learning algorithm is used to create DIs invariant under feature vectors from the undamaged condition. In this case, two unobserved variables are assumed to quantify the influence of the operational and environmental variations (the changes in mass and stiffness).

Training:

```
[model]=learnFactorAnalysis_shm(learnData,2,'thomson');
```

```
Warning: Some unique variances are zero.
```

Scoring:

```
[DI]=scoreFactorAnalysis_shm(scoreData,model);
```

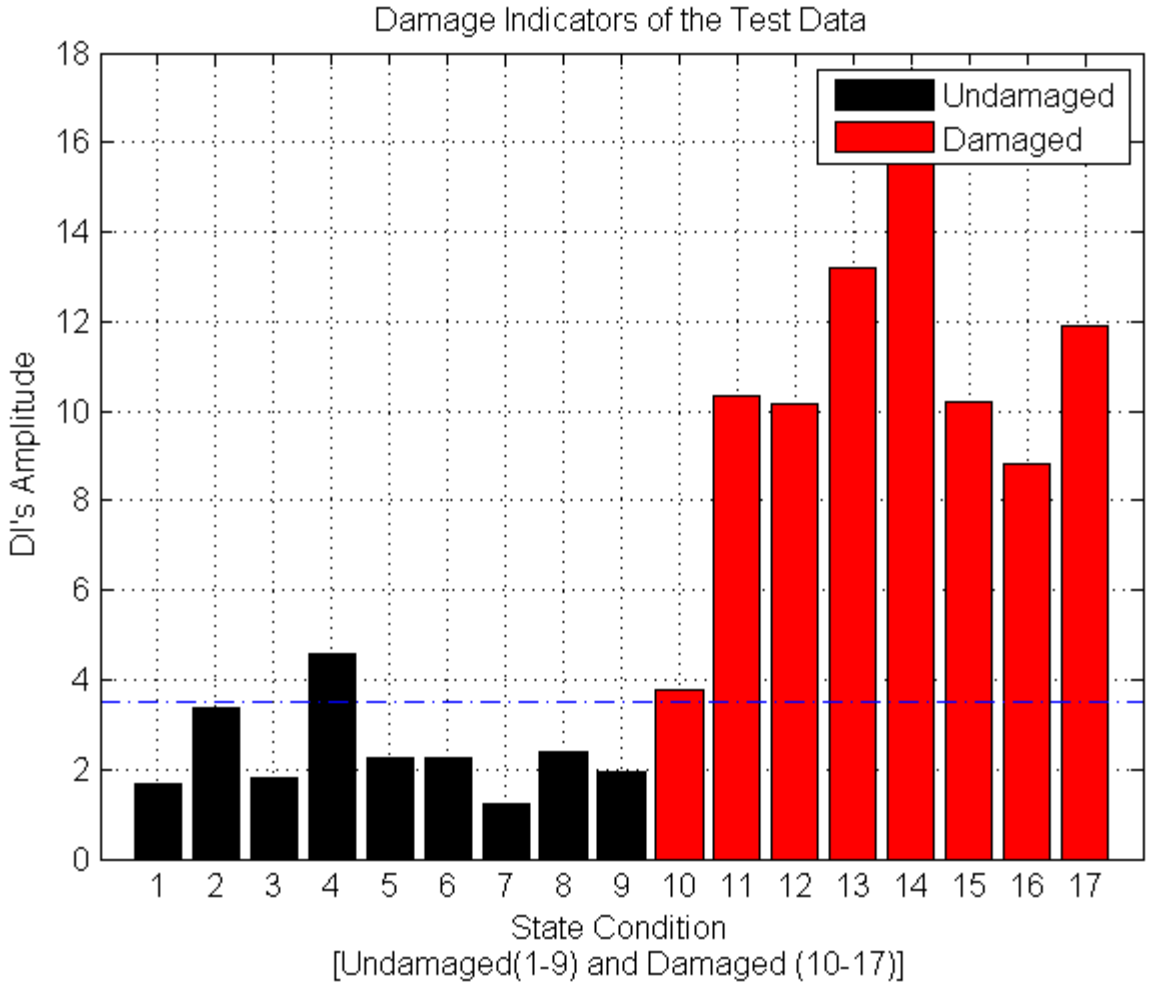
Plot Damage Indicators

Threshold based on the 95% cut-off over the training data:

```
[threshold]=scoreFactorAnalysis_shm(learnData,model);
threshold=sort(-threshold);
UCL=threshold(round(length(threshold)*0.95));
```

Plot DIs:

```
figure;
bar(1:9,-DI(1:9),'k'); hold on; bar(10:17,-DI(10:17),'r')
title('Damage Indicators of the Test Data')
xlabel({'State Condition','[Undamaged(1-9) and Damaged (10-17)]'})
ylabel('DI''s Amplitude')
line('XData',[0 17+1],'YData',[UCL UCL],'Color','b','LineWidth',1,'LineStyle','-.')
set(gca,'XTick',1:length(DI))
legend('Undamaged','Damaged')
grid on
```



The figure shows that the FA-based machine learning algorithm is able to discriminate all the damaged state conditions. However, it cannot avoid one false-positive indication of damaged (State#4). Note however that this result can be improved by increasing either the number of unobserved variables or the number of feature vectors in the training data.

See also:

Outlier Detection Based on Nonlinear Principal Component Analysis

Outlier Detection Based on Principal Component Analysis

Outlier Detection Based on the Singular Value Decomposition

Outlier Detection Based on the Mahalanobis Distance

Published with MATLAB® R2013a

Outlier Detection based on Principal Component Analysis

Contents

Introduction

The goal of this example usage is to discriminate time histories from undamaged and damaged condition based on outlier detection. The root mean square (RMS) errors of an autoregressive (AR) model are used as damage-sensitive features and a machine learning algorithm based on the principal component analysis (PCA) is used to create damage indicators (DIs) invariant for feature vectors from normal structural condition and that increase when feature vectors are from damaged structural condition.

Data sets of an array of sensors (Channel 2-5) of the base-excited three story structure are used in this example usage. More details about the data sets can be found in 3-Story Data Sets.

Requires data3SS.mat dataset.

References:

Figueiredo, E., Park, G., Figueiras, J., Farrar, C., & Worden, K. (2009). Structural Health Monitoring Algorithm Comparisons using Standard Data Sets. Los Alamos National Laboratory Report: LA-14393.

SHMTools functions called:

```
arModel_shm
learnPCA_shm
scorePCA_shm
```


Author: Elói Figueiredo

Date: September 01, 2009

Load Raw Data

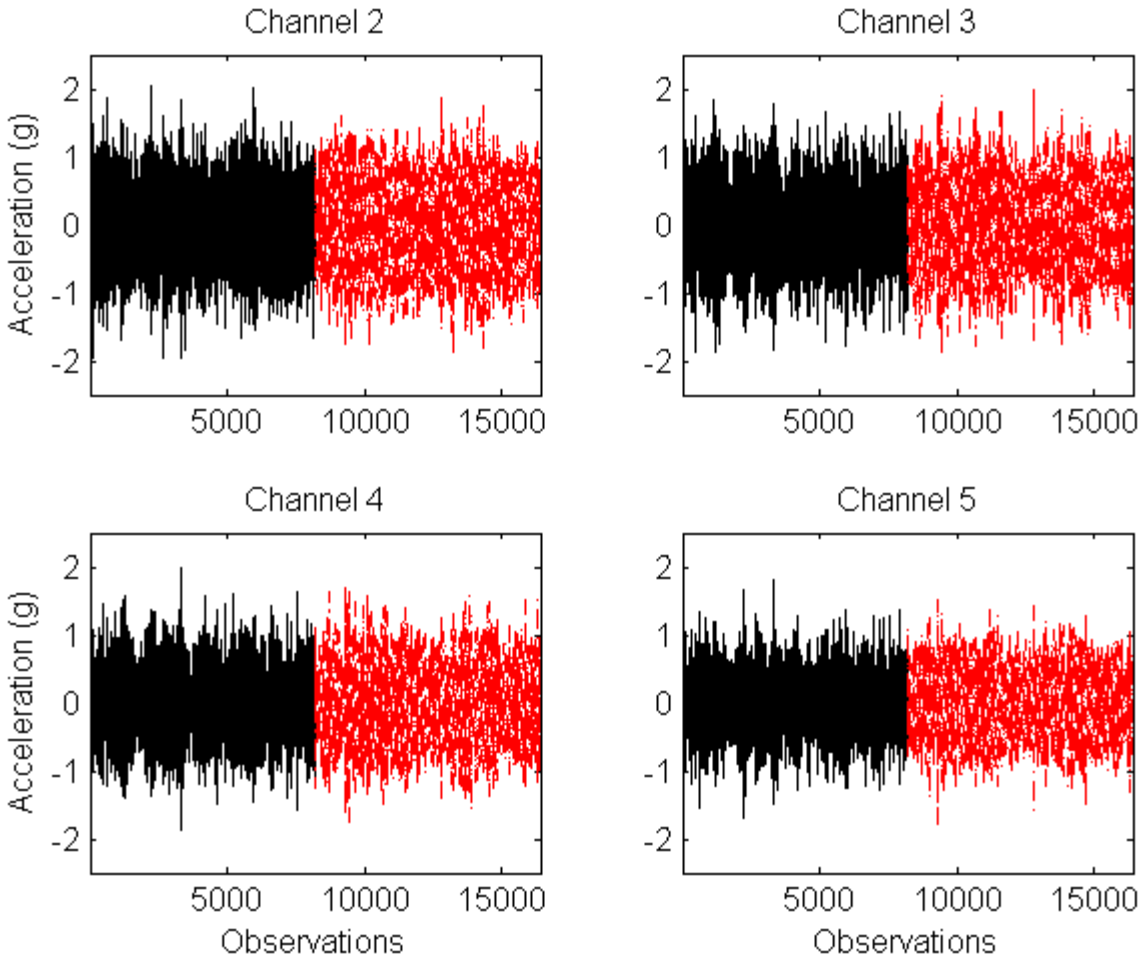
Note that the data sets are composed of acceleration time histories from Channel 2-5.

Load data set:

```
load('data3SS.mat');

data=dataset(:,2:5,:);

[t m n]=size(data);
```

Plot time history from the baseline condition (Channel 2-5):

```
label{1}='Channel 2'; label{2}='Channel 3'; label{3}='Channel 4';
label{4}='Channel 5';

figure

for i=1:m;

    subplot(2,2,i)
    plot(1:t,data(:,i,1),'k',t+1:t*2,data(:,i,91),'-.r')
    title([label{i}])
    set(gca,'YTick',-2:2,'Xlim',[1 t*2],'Ylim',[-2.5 2.5])

    if i==3 || i==4, xlabel('Observations'); end
    if i==1 || i==3, ylabel('Acceleration (g)'); end

end
```



The figure above plots time histories from State#1 (baseline condition, black) and State#10 (lowest level of damage, red) in concatenated format.

Extraction of Damage-Sensitive Features

This section returns the RMS errors of an AR(15) model of Channels 2-5 in concatenated format. This way, any condition is classified based on a feature vector composed of features from all sensors.

AR model order:

```
arOrder=15;
```

Estimation of AR Parameters:

```
[arParameters RMSE]=arModel_shm(data,arOrder);
```

Training Data:

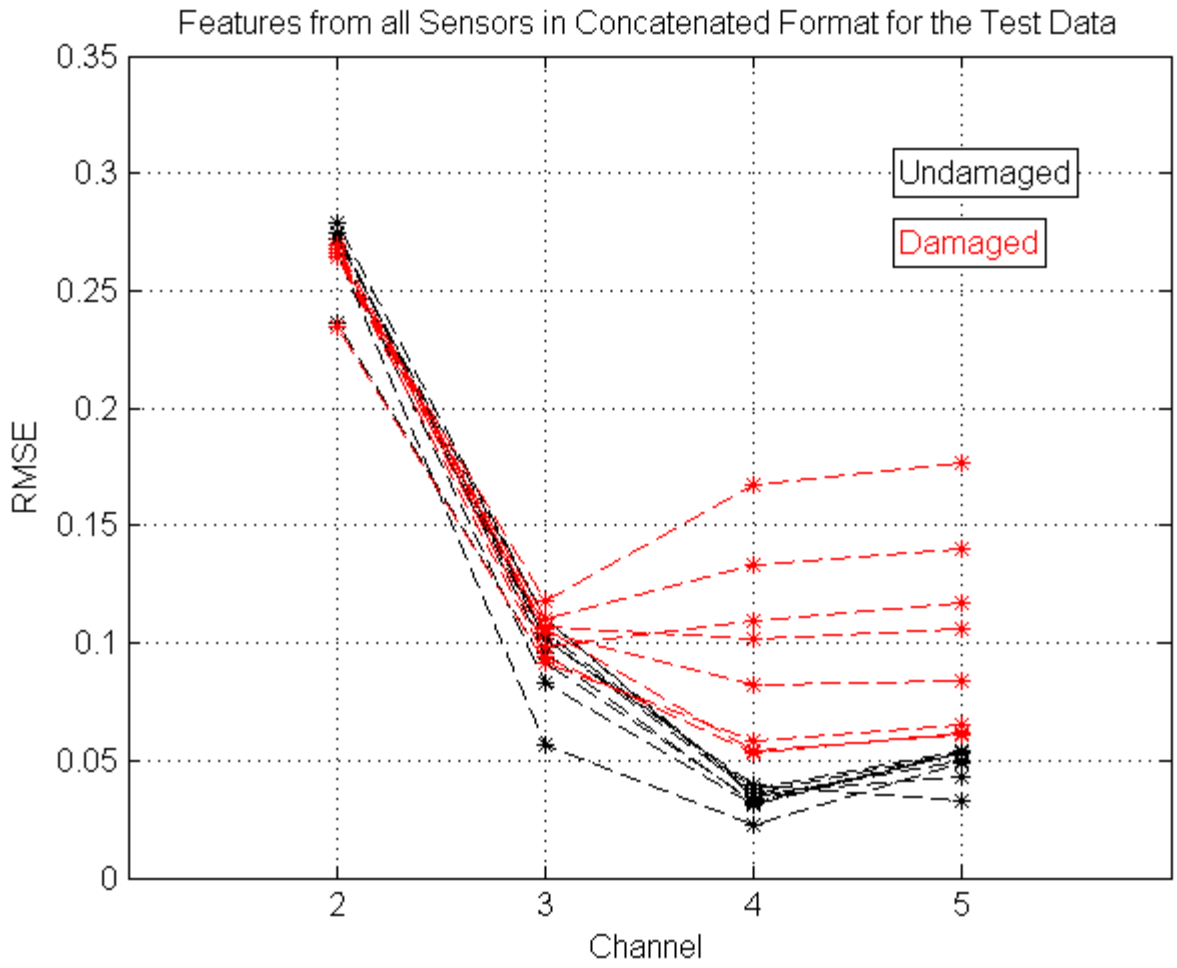
```
clear learnData
for i=1:9;
    learnData(i*9-8:i*9,:)=RMSE(i*10-9:i*10-1,:);
end
```

Test Data:

```
scoreData=RMSE(10:10:170,:);

[n m]=size(scoreData);
```

Plot test data:

```
figure
plot(1:m,scoreData(1:9,:)','*--k',1:m,scoreData(10:17,:)','*--r')
title('Features from all Sensors in Concatenated Format for the Test Data')
xlabel('Channel')
ylabel('RMSE')
set(gca,'Xlim',[0 m+1],'XTick',1:m,'XTickLabel',2:5)
text(3.7,0.30,'Undamaged','Color','k','EdgeColor','k','BackgroundColor','w')
text(3.7,0.27,'Damaged','Color','r','EdgeColor','k','BackgroundColor','w')
grid on
```



Statistical Modeling for Feature Classifiction

The PCA-based machine learning algorithm is used to normalize the features and to reduce each feature vector to a score (also called DI).

```
[model]=learnPCA_shm(learnData);

[DI]=scorePCA_shm(scoreData,model);
```

Outlier Detection

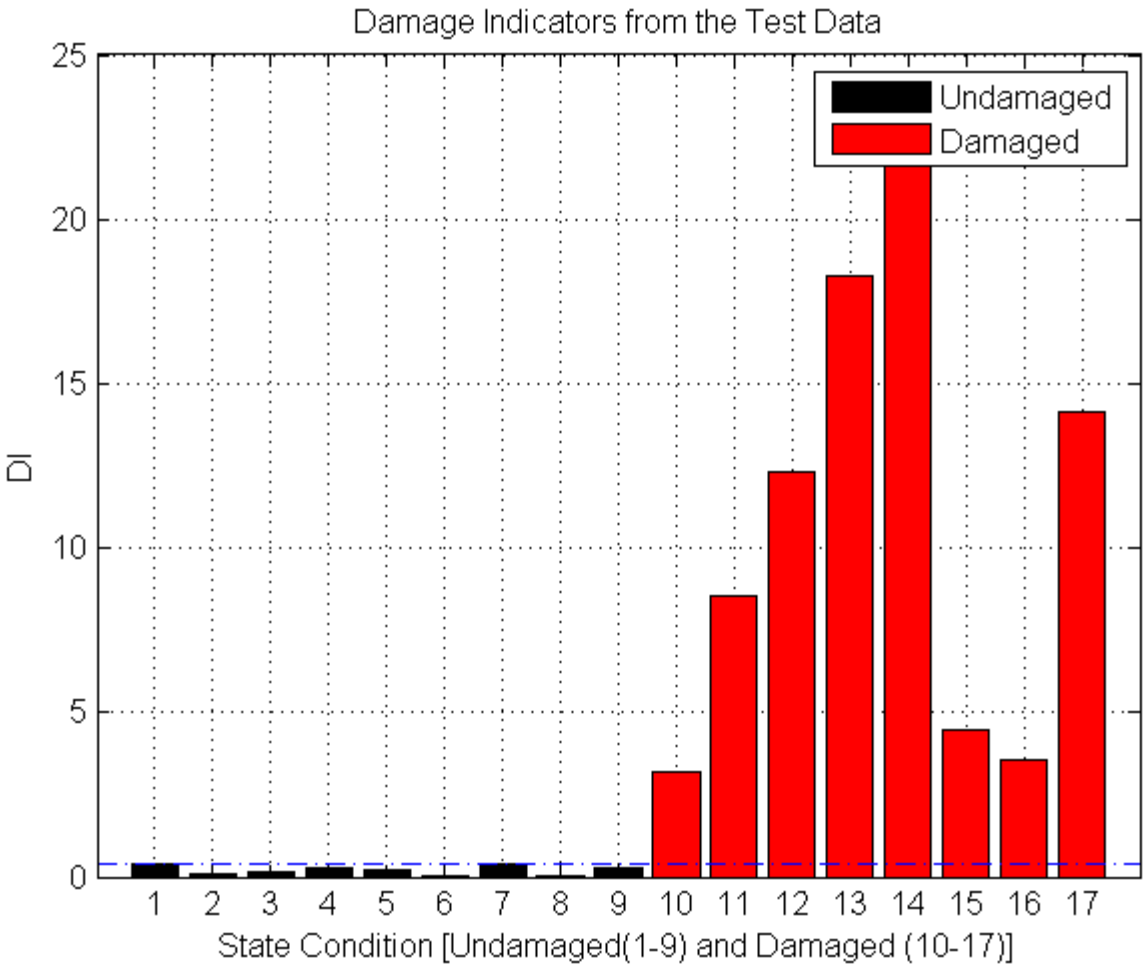
Threshold based on the 95% cut-off over the training data:

```
[threshold]=scorePCA_shm(learnData,model);
threshold=sort(-threshold);
UCL=threshold(round(length(threshold)*0.95));
```

Plot DIs:

```
figure
bar(1:9,-DI(1:9),'k'); hold on; bar(10:17,-DI(10:17),'r')
title('Damage Indicators from the Test Data')
set(gca,'Xlim',[0 n+1],'XTick',1:n)
line('XData',[0 n+1],'YData',[UCL UCL],'Color','b','LineWidth',1,'LineStyle','-.')
xlabel('State Condition [Undamaged(1-9) and Damaged (10-17)]')
```

```
ylabel('DI');
legend('Undamaged','Damaged')
grid on
```



The figure above shows that the approach for damage detection, based on PCA-based machine learning algorithm along with the RMS errors of an AR(15) model from Channel 2-5, is able to discriminate the undamaged (1-9) and damaged (10-17) state conditions without any false-negative and false-positive indications of damage.

See also:

Outlier Detection based on Nonlinear Principal Component Analysis

Outlier Detection based on the Factor Analysis Model

Outlier Detection based on the Singular Value Decomposition

Outlier Detection based on the Mahalanobis Distance

Published with MATLAB® R2013a

Outlier Detection based on Singular Value Decomposition

Contents

Introduction

The goal of this example usage is to discriminate time histories from undamaged and damaged condition based on outlier detection. The parameters from an autoregressive (AR) model are used as damage-sensitive features and a machine learning algorithm based on the singular value decomposition (SVD) technique is used to create damage indicators (DIs) invariant for feature vectors from normal structural condition and that increase when feature vectors are from damaged structural condition. Additionally, the receiver operating characteristic (ROC) curve is applied to evaluate the performance of the classification algorithm. In this example usage each time history of the data sets is split into four segments in order to increase the number of instances available.

Data sets from Channel 5 of the base-excited three story structure are used in this example usage. More details about the data sets can be found in 3-Story Data Sets.

Requires data_3ss4.mat dataset.

SHMTools functions called:

```
arModel_shm
learnSVD_shm
scoreSVD_shm
ROC_shm
```

Author: Elói Figueiredo

Date: September 01, 2009

Load Raw Data

In this case each time history of the original data (Channel 5) is split into four segments.

Load acceleration time histories from Channel 5:

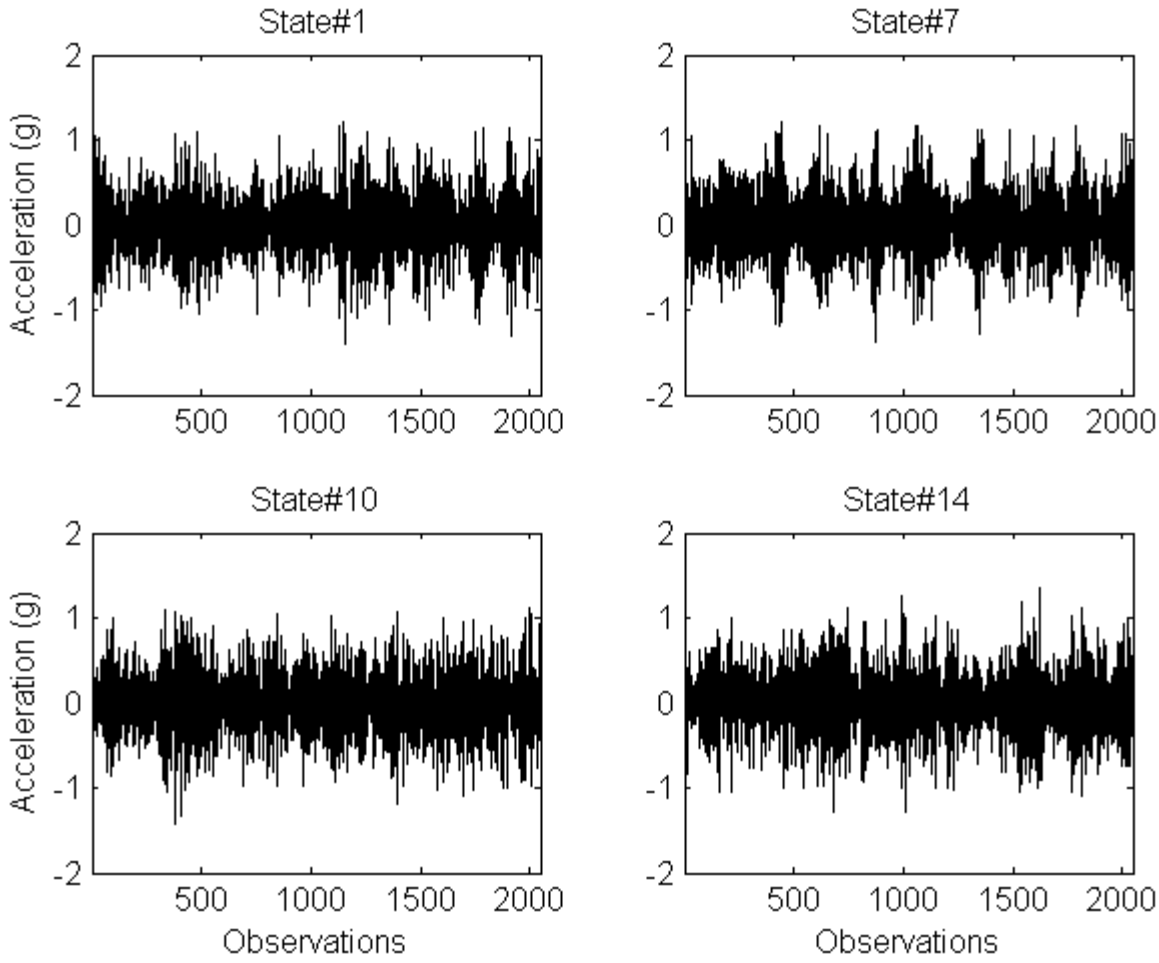
```
load('data3SS.mat');
breakPoint=400;
```

For this example, we will break each 8192 point time series into 4, 2048 point time series.

```
timeData = zeros(2048,1,680);
for i=1:4
    timeData(:,:,i:4:680)=dataset((1+2048*(i-1)):(2048*i),5,:);
end
```

Plot one segment of one acceleration time history from four state conditions:

```
states=[1 7 10 14];

figure

for i=1:4

    subplot(2,2,i)
    plot(timeData(:,1,states(i)*10),'k')
    title(['State#',num2str(states(i))])
    if i==3 || i==4,xlabel('Observations'); end
    if i==1 || i==3, ylabel('Acceleration (g)'); end
    set(gca,'YTick',-2:2,'Xlim',[1 length(timeData)],'Ylim',[-2 2])

end
```



Extraction of Damage-Sensitive Features

Extraction of the AR(15) model parameters from the segments of acceleration time histories. The order of the model was picked from the lower-bound of the range given by the optimization methods available in this package. (For more details see example)

Set AR model order:

```
arOrder=15;
```

Estimation of AR parameters:

```
[arParameters]=arModel_shm(timeData,arOrder);
```

Training feature vectors:

```
learnData=arParameters(1:breakPoint,:);
```

Test feature vectors:

```
scoreData=arParameters;

[n m]=size(arParameters);
```

Statistical Modeling for Feature Classifiction

In the context of data normalization process, the SVD-based machine learning algorithm is used to create DIs invariant under feature vectors from undamaged structural conditions.

Training:

```
[model]=learnSVD_shm(learnData,0);
```

Scoring:

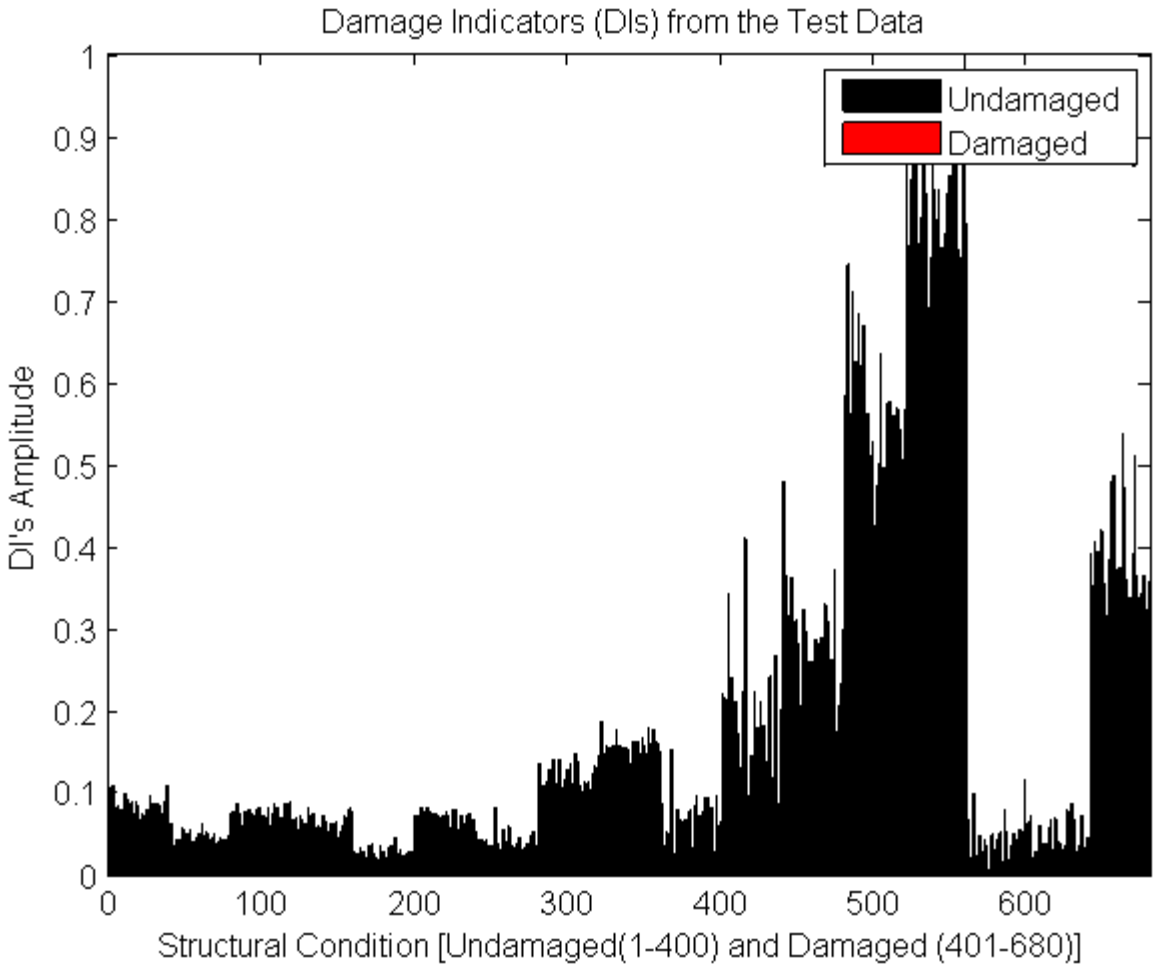
```
[DI]=scoreSVD_shm(scoreData,model);
```

Normalization procedure:

```
DIn=scaleMinMax_shm(-DI, 1, [0,1]);
```

Plot DIs:

```
figure
bar(1:breakPoint,DIn(1:breakPoint),'k');
hold on;
bar(breakPoint+1:n,DIn(breakPoint+1:end),'r')
title('Damage Indicators (DIs) from the Test Data')
xlabel(['Structural Condition [Undamaged(1-',num2str(breakPoint),') and Damaged
(',num2str(breakPoint+1),'-',num2str(n),')]'])
ylabel('DI''s Amplitude')
set(gca,'XLim',[0 n+1])
legend('Undamaged','Damaged')
```



Receiver Operating Characteristic Curve

Flag all the instances:

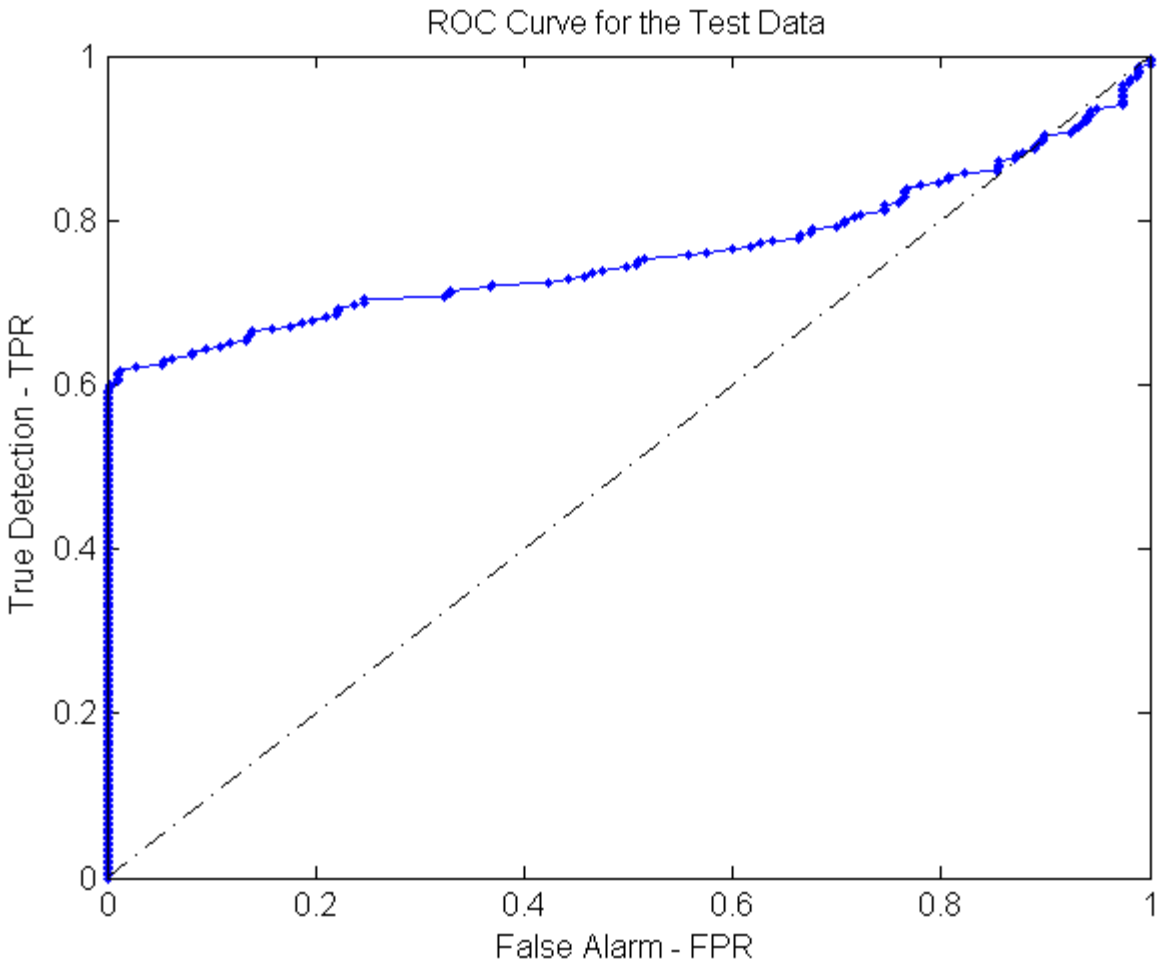
```
flag(1:breakPoint,1)=0; flag(breakPoint+1:n,1)=1;
```

Run ROC curve algorithm:

```
[TPR,FPR]=ROC_shm(DI,flag);
```

Plot ROC curve:

```
figure
plot(FPR,TPR,'.-b')
title('ROC Curve for the Test Data')
xlabel('False Alarm - FPR')
ylabel('True Detection - TPR')
hold on
line('XData',[0 1],'YData',[0 1],'Color','k','LineWidth',1,'LineStyle','-.')
set(gca,'XTick',0:0.2:1,'YTick',0:0.2:1)
```



The ROC curve in the figure above shows that there is no linear threshold able to discriminate all the undamaged and damaged instances, when using the AR(15) parameters as damage-sensitive features and the SVD-based machine learning algorithm. Note that the diagonal line divides the ROC space in areas of good (left) or bad (right) classification. Note that the optimal point (no false negatives/positives) is in the upper-left corner of the plot.

See also:

Outlier Detection based on Nonlinear Principal Component Analysis

Outlier Detection Based on the Factor Analysis Model

Outlier Detection Based on Principal Component Analysis

Outlier Detection Based on the Mahalanobis Distance

Published with MATLAB® R2013a

Outlier Detection based on Mahalanobis Distance

Contents

Introduction

The goal of this example usage is to discriminate undamaged and damaged structural state conditions based on outlier detection. The parameters from an autoregressive (AR) model are used as damage-sensitive features and a machine learning algorithm based on the Mahalanobis distance is used to create damage indicators (DIs) invariant for feature vectors from normal structural condition and that increase when feature vectors are from damaged structural conditions.

Data sets of an array of sensors from Channel 2-5 of the base-excited three story structure are used in this example usage. More details about the data sets can be found in 3-Story Data Sets.

Requires data3SS.mat dataset.

References:

Worden, K., & Manson, G. (2000). Damage Detection using Outlier Analysis. Journal of Sound and Vibration , 229 (3), 647-667.

SHMTools functions called:

```
arModel_shm
learnMahalanobis_shm
scoreMahalanobis_shm
```


Author: Elói Figueiredo

Date Created: September 01, 2009

Load Raw Data

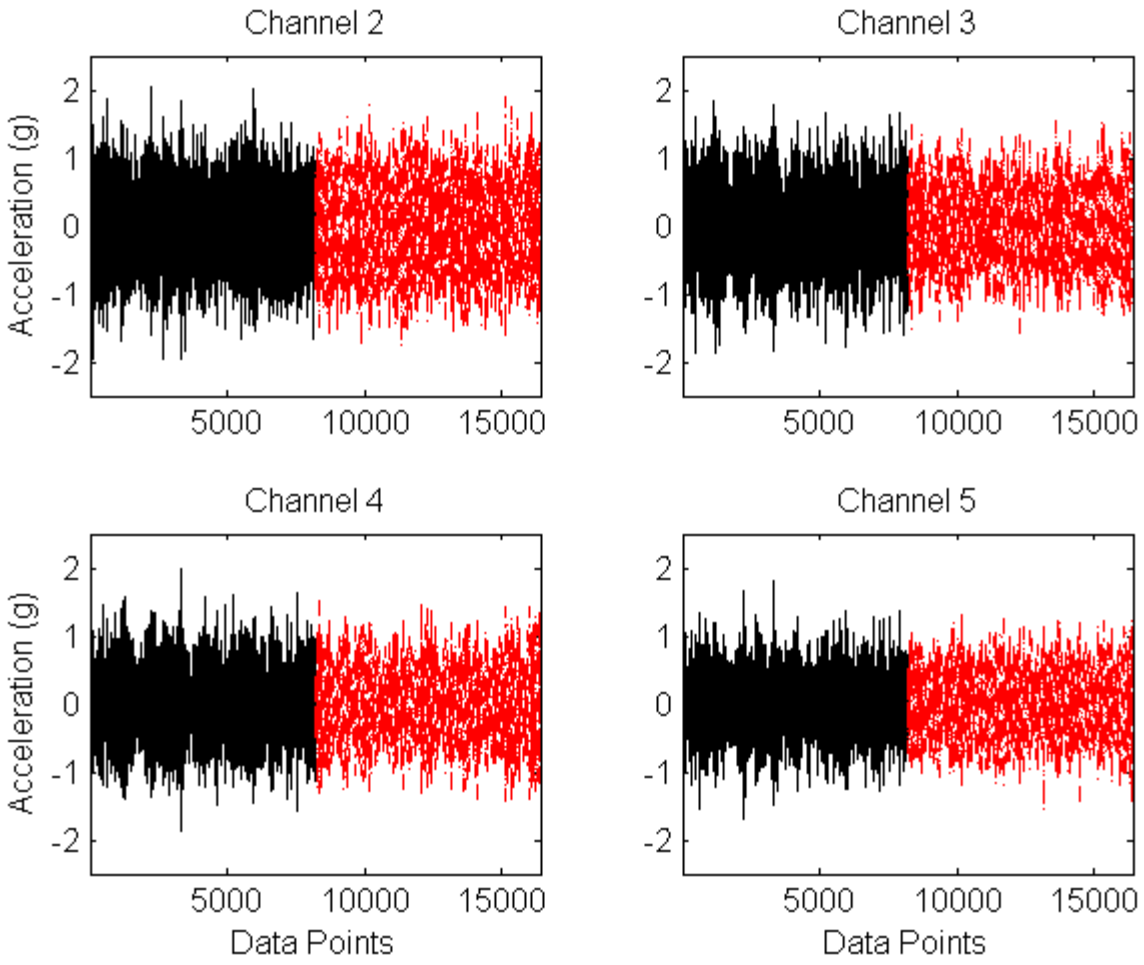
Load data set:

```
load('data3SS.mat');

data=dataset(:,2:5,:);

[t]=size(data,1);
```

Plot time histories (Channel 2-5) from State#1 (baseline condition) and State#16:

```
label{1}='Channel 2'; label{2}='Channel 3'; label{3}='Channel 4'; label{4}='Channel 5';
```

```
figure

for i=1:4;

    subplot(2,2,i)
    plot(1:t,data(:,i,1),'k',t+1:t*2,data(:,i,151),'-.r')
    title([label{i}])
    set(gca,'YTick',-2:2,'Xlim',[1 t*2],'Ylim',[-2.5 2.5])

    if i==3 || i==4, xlabel('Data Points'); end
    if i==1 || i==3, ylabel('Acceleration (g)'); end

end
```



The figure above plots time histories from State#1 (baseline condition, black) and State#16 (damaged with simulated operational changes, red) in concatenated format.

Extraction of Damage-Sensitive Features

This section estimates the AR(15) model parameters from the time histories of Channels 2-5 and plot the feature vectors for each instance (or condition).

AR model order:

```
arOrder=15;
```

Estimation of AR Parameters:

```
[arParameters]=arModel_shm(data,arOrder);
```

Training Data:

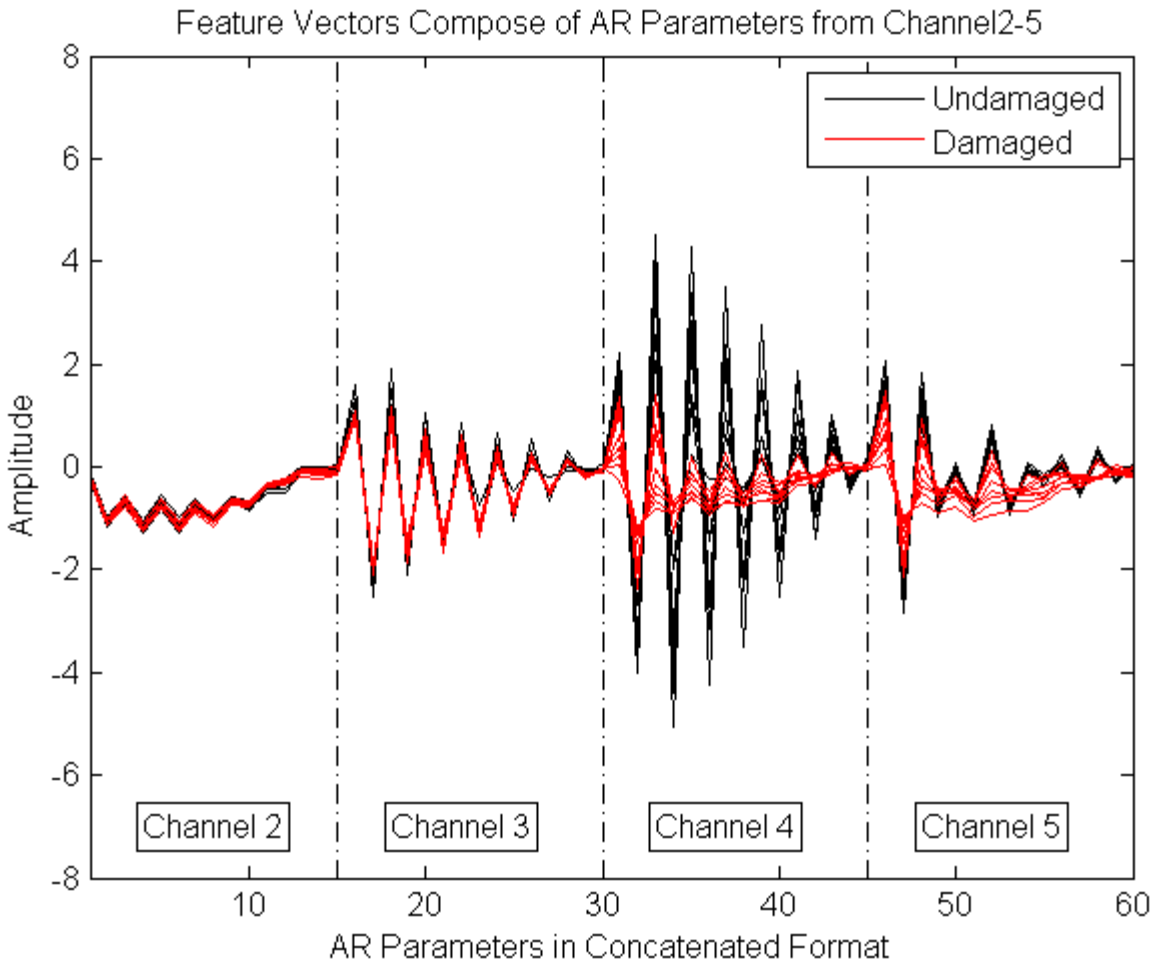
```
clear learnData
for i=1:9;
    learnData(i*9-8:i*9,:)=arParameters(i*10-9:i*10-1,:);
end
```

Test Data:

```
scoreData=arParameters(10:10:170,:);

[n m]=size(scoreData);
```

Plot test data:

```
figure
plot(1:m,scoreData(1:9,:)','k',1:m,scoreData(10:17,:)','r')
title('Feature Vectors Compose of AR Parameters from Channel2-5')
legend('Undamaged','Damaged')
xlabel('AR Parameters in Concatenated Format')
ylabel('Amplitude')
w=8;
set(gca,'Xlim',[1 m],'YLim',[-w w])
M(1,1:9)='Undamaged'; M(2,1:7)='Damaged';
legend([line('color','k');line('color','r')],M);
h(1)=line([m/4;m/4],[-w w],'color','k','lineStyle','-.');
h(2)=line([m/4*2;m/4*2],[-w w],'color','k','lineStyle','-.');
h(3)=line([m/4*3;m/4*3],[-w w],'color','k','lineStyle','-.');
text(4,-7,'Channel 2','Color','k','EdgeColor','k','BackgroundColor','w')
text(18,-7,'Channel 3','Color','k','EdgeColor','k','BackgroundColor','w')
text(33,-7,'Channel 4','Color','k','EdgeColor','k','BackgroundColor','w')
text(48,-7,'Channel 5','Color','k','EdgeColor','k','BackgroundColor','w')
```



Statistical Modeling for Feature Classifiction

The Mahalanobis-based machine learning algorithm is used to normalize the features and reduce each feature vector into a score.

```
[model]=learnMahalanobis_shm(learnData);

[DI]=scoreMahalanobis_shm(scoreData,model);
```

Outlier Detection

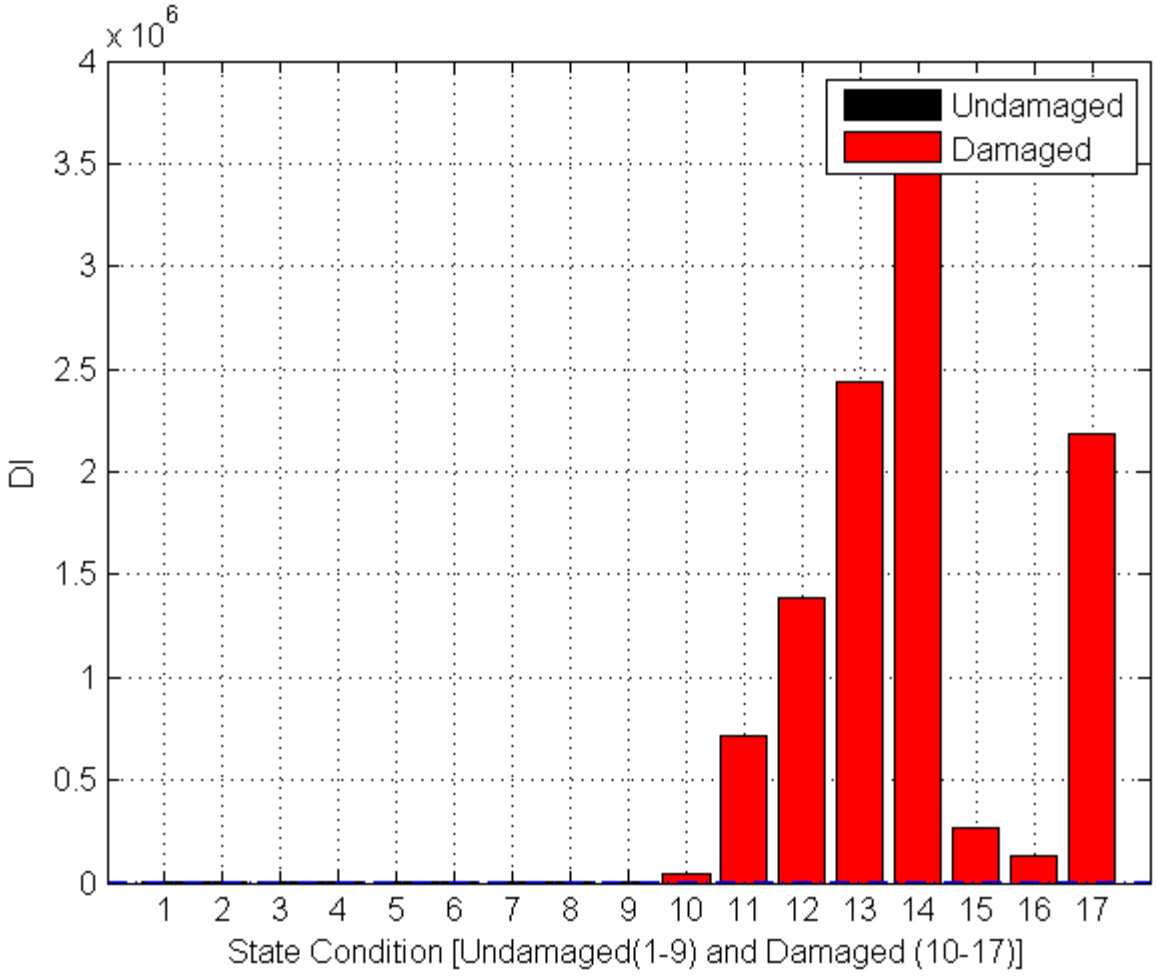
Threshold based on the 95% cut-off over the training data:

```
[threshold]=scoreMahalanobis_shm(learnData,model);
threshold=sort(-threshold);
UCL=threshold(round(length(threshold)*0.95));
```

Plot DIs:

```
figure
bar(1:9,-DI(1:9),'k'); hold on; bar(10:17,-DI(10:17),'r')
set(gca,'Xlim',[0 n+1],'XTick',1:n)
line('XData',[0 n+1],'YData',[UCL UCL],'Color','b','LineWidth',1,'LineStyle','-.')
xlabel('State Condition [Undamaged(1-9) and Damaged (10-17)]')
ylabel('DI');
```

```
legend('Undamaged','Damaged')
grid on
```



The figure above shows that the approach for damage detection, based on Mahalanobis distance along with the AR(15) parameters from Channel 2-5, is able to discriminate all the undamaged (1-9) and damaged (10-17) state conditions.

See also:

Outlier Detection based on Nonlinear Principal Component Analysis

Outlier Detection based on the Factor Analysis Model

Outlier Detection based on Principal Component Analysis

Outlier Detection based on the Singular Value Decomposition

Published with MATLAB® R2013a

Example Usage: Direct Use of Semi-Parametric Routines

Contents

Introduction

Here we show how to directly use the Semiparametric routines while bypassing the "trainOutlierDetector" routine.

The data used in this example is from the 3-story structure. More details about the data sets can be found in 3-Story Data Sets.

Requires data3SS4.mat dataset.

SHMTools functions called:

```
arModel_shm
learnSemiParametricModel_shm
scoreGMM_shm
```

Author: Samory Kpotufe

Date Created: August 19, 2009

Load data

The data here is in the form of time series in a 3 dimensional matrix (time, sensors, instances) and also a state vector representing the various environmental conditions under which the data is collected.

```
load('data3SS.mat');
```

For this example, we will break each 8192 point time series into 4, 2048 point time series.

```
timeData = zeros(2048,5,680);
timeDataStates = zeros(1,680);
for i=1:4
    timeData(:,:,i:4:680)=dataset((1+2048*(i-1)):(2048*i),:,:);
    timeDataStates(:,i:4:680)=states(:,:);
end
```

Extract some features using your favorite function, but first pick N of the instances (each time series reading over all sensors). Each instance is then transformed into a feature vector: the returned matrix has the form (instances, features).

```
```

```
N = 400;
Idx = randperm(size(timeData, 3));
Idx = Idx(1:N);
Xdata = arModel_shm(timeData(:, :, Idx));
Xstates = timeDataStates(Idx);
```

Now set 80% of states 1:9 aside as the training data, these states correspond to undamaged readings. We'll then test on the remaining 20% of 1:9 and on the "damaged" states 10:17.

```
Idx = logical(ismember(Xstates, [1:9]));
Xundamaged = Xdata(Idx, :);
nUndamaged = size(Xundamaged, 1);
nTrain = round(0.8*nUndamaged);
Xtrain = Xundamaged(1:nTrain, :);
Xtest = [Xundamaged(nTrain+1:nUndamaged, :); Xdata(~Idx, :)];
nTest = size(Xtest, 1);
```

Now set labels for the test data, 0 corresponds to undamaged, and 1 to damaged.

number of undamaged in test.

```
nTest_0 = nUndamaged -nTrain;
```

test labels

```
testLabels = [zeros(nTest_0, 1); ones(nTest -nTest_0, 1)];
```

Train a model over the undamaged data

The next call learns a mixture of k gaussians over the undamaged data and returns the parameters of this model in dModel. The partition function is one of those in "SemiParametricDetectors/PartitioningAlgorithms/" or should have the same behavior as one of those functions (including signature). The "MMFun" is a Mixture Model function from "SemiParametricDetectors/ParametricMixtures" or should have the same behavior.

```
partitionFun = @kMedians_shm;
MMFun = @learnGMM_shm;
k = 5;
dModel = learnGMMSemiParametricModel_shm(Xtrain, partitionFun,k);
```

Pick a threshold from the training data.

We will first obtain the "scores" over the training data, that is the log-likelihoods that are given by the learned distribution. Then we learn a distribution of these scores, and pick a threshold so that 90% of the training data (undamaged data) has scores above this threshold (according to the distribution of scores).

```
likelihoods = scoreGMM_shm(Xtrain, dModel);
```

learn a normal distribution over the scores

```
model_p = mle(likelihoods,'distribution', 'normal');
```

pick the threshold

```
confidence = 0.9;
threshold = icdf('normal', 1-confidence, model_p(1), model_p(2));
```

Test the detector

Now the detector consists simply of getting the distribution of scores over the test data, under the distribution learned on the undamaged training data (dModel). We simply flag a test point as "damaged" whenever it falls below our threshold.

Test scores

```
scores = scoreGMMSemiParametricModel_shm(Xtest, dModel);
```

Results contains a 1 whenever we think the point is damaged, a 0 otherwise.

```
results = scores <= threshold;
```

Report the detector's performance.

Various error rates

```
TotalErr = sum(results ~= testLabels)/nTest;
falsePositiveErr = sum(results(1:nTest_0) ~= 0)/nTest_0;
falseNegativeErr = sum(results(nTest_0+1:end)~=1)/(nTest - nTest_0);
fprintf('\n Total error: %2.2f\n False Positive rate: %2.2f\n False Negative rate: %2.2f\n',
TotalErr, falsePositiveErr, falseNegativeErr);
```

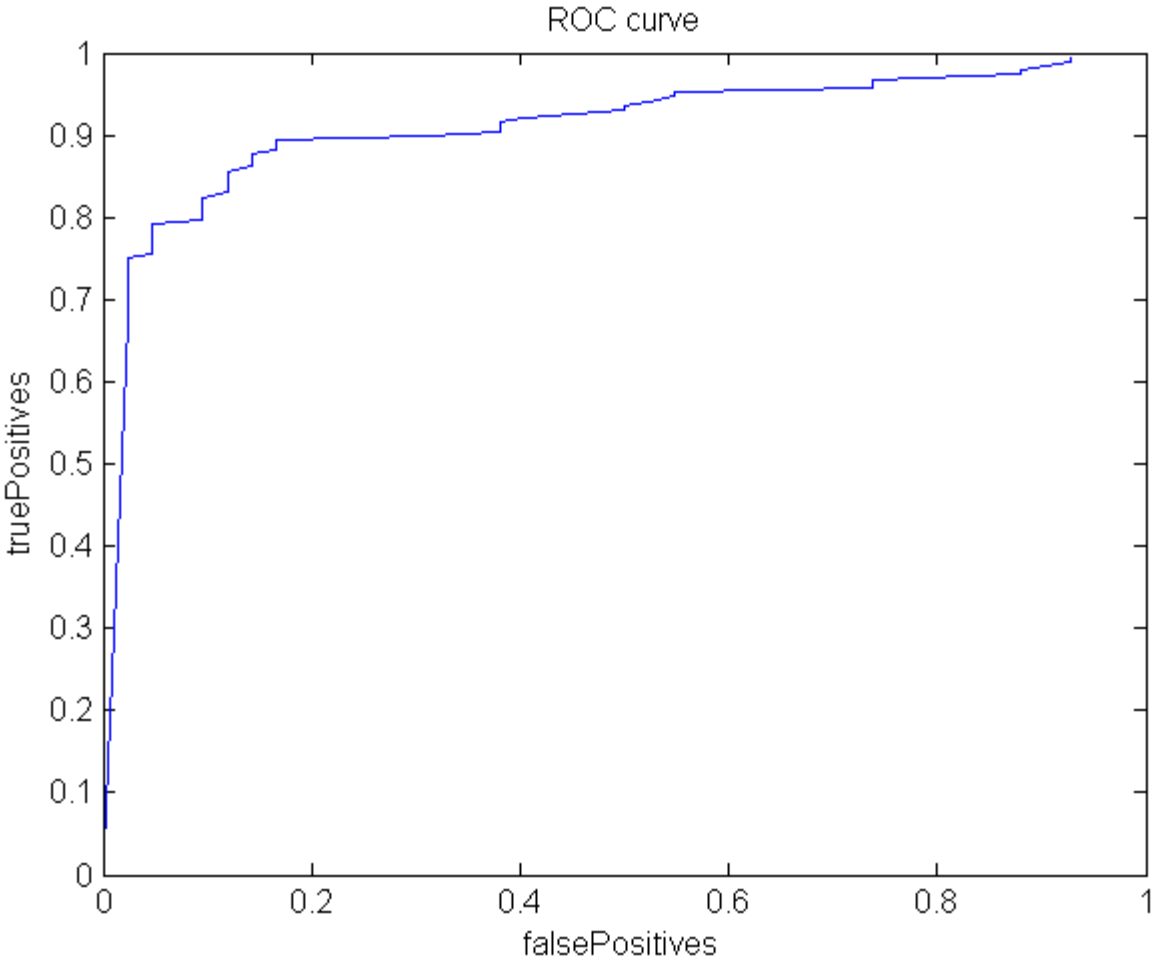
```
 Total error: 0.14
 False Positive rate: 0.57
 False Negative rate: 0.04
```

ROC curve

```
[truePositives,falsePositives] = ROC_shm(scores,testLabels);
```

Now plot the curve

```
figure;
plot(falsePositives, truePositives);
xlabel('falsePositives');
ylabel('truePositives');
title('ROC curve');
```



Published with MATLAB® R2013a

Example Usage: Direct Use of Non-Parametric Routines

Contents

Introduction

Here we show how to directly use the Nonparametric routines while bypassing the "trainOutlierDetector" routine.

The data used in this example is from the 3-story structure. More details about the data sets can be found in 3-Story Data Sets.

Requires data3SS4.mat dataset.

SHMTools functions called:

```
arModel_shm
learnKernelDensity_shm
scoreKernelDensity_shm
```

Author: Samory Kpotufe

Date Created: August 19, 2009

Load data

The data here is in the form of time series in a 3 dimensional matrix (time, sensors, instances) and also a state vector representing the various environmental conditions under which the data is collected.

```
load('data3SS.mat');
```

For this example, we will break each 8192 point time series into 4, 2048 point time series.

```
timeData = zeros(2048,5,680);
timeDataStates = zeros(1,680);
for i=1:4
    timeData(:,:,i:4:680)=dataset((1+2048*(i-1)):(2048*i),:,:);
    timeDataStates(:,i:4:680)=states(:,:);
end
```

Extract some features using your favorite function, but first pick N of the instances (each time series reading over all sensors). Each instance is then transformed into a feature vector: the returned matrix has the form (instances, features).

```
N = 400;
Idx = randperm(size(timeData, 3));
Idx = Idx(1:N);
Xdata = arModel_shm(timeData(:, :, Idx));
Xstates = timeDataStates(Idx);
```

Now set 80% of states 1:9 aside as the training data, these states correspond to undamaged readings. We'll then test on the remaining 20% of 1:9 and on the "damaged" states 10:17.

```
Idx = logical(ismember(Xstates, [1:9]));
Xundamaged = Xdata(Idx, :);
nUndamaged = size(Xundamaged, 1);
nTrain = round(0.8*nUndamaged);
Xtrain = Xundamaged(1:nTrain, :);
Xtest = [Xundamaged(nTrain+1:nUndamaged, :); Xdata(~Idx, :)];
nTest = size(Xtest, 1);
```

Now set labels for the test data, 0 corresponds to undamaged, and 1 to damaged.

number of undamaged in test.

```
nTest_0 = nUndamaged -nTrain;
```

test labels

```
testLabels = [zeros(nTest_0, 1); ones(nTest -nTest_0, 1)];
```

Train a model over the undamaged data

The next call learns a nonparametric distribution over the data. We supply a kernel function to use, from one in "NonParametricDetectors/Kernels", or any other function with similar behavior and signature. Next we instruct the learner to do bandwidth selection through cross-validation by setting bs_method to 2 and H to []. If an actual bandwidth matrix H is passed in, this will be used instead.

```
kernelFun = @epanechnikovKernel_shm;
H = [];
bs_method = 2;
dModel = learnKernelDensity_shm(Xtrain, H, kernelFun, bs_method);
```

Pick a threshold from the training data.

We will first obtain the "scores" over the training data, that is the log-likelihoods that are given by the learned distribution. (Note that every 'learn' function has a corresponding 'score' function, see the example in direct use of semi-parametric).Then we learn a distribution of these scores, and pick a threshold so that 90% of the training data (undamaged data) has scores above this threshold (according to the distribution of scores).

```
likelihoods = scoreKernelDensity_shm(Xtrain, dModel);
```

learn a normal distribution over the scores

```
model_p = mle(likelihoods,'distribution', 'normal');
```

pick the threshold

```
confidence = 0.9;
threshold = icdf('normal', 1-confidence, model_p(1), model_p(2));
```

Test the detector

Now the detector consists simply of getting the distribution of scores over the test data, under the distribution learned on the undamaged training data (dModel). We simply flag a test point as "damaged" whenever it falls below our threshold.

Test scores

```
scores = scoreKernelDensity_shm(Xtest, dModel);
```

Results contains a 1 whenever we think the point is damaged, a 0 otherwise.

```
results = scores <= threshold;
```

Report the detector's performance.

Various error rates

```
TotalErr = sum(results ~= testLabels)/nTest;
falsePositiveErr = sum(results(1:nTest_0) ~= 0)/nTest_0;
falseNegativeErr = sum(results(nTest_0+1:end)~=1)/(nTest - nTest_0);
fprintf('\n Total error: %2.2f\n False Positive rate: %2.2f\n False Negative rate: %2.2f\n',
TotalErr, falsePositiveErr, falseNegativeErr);
```

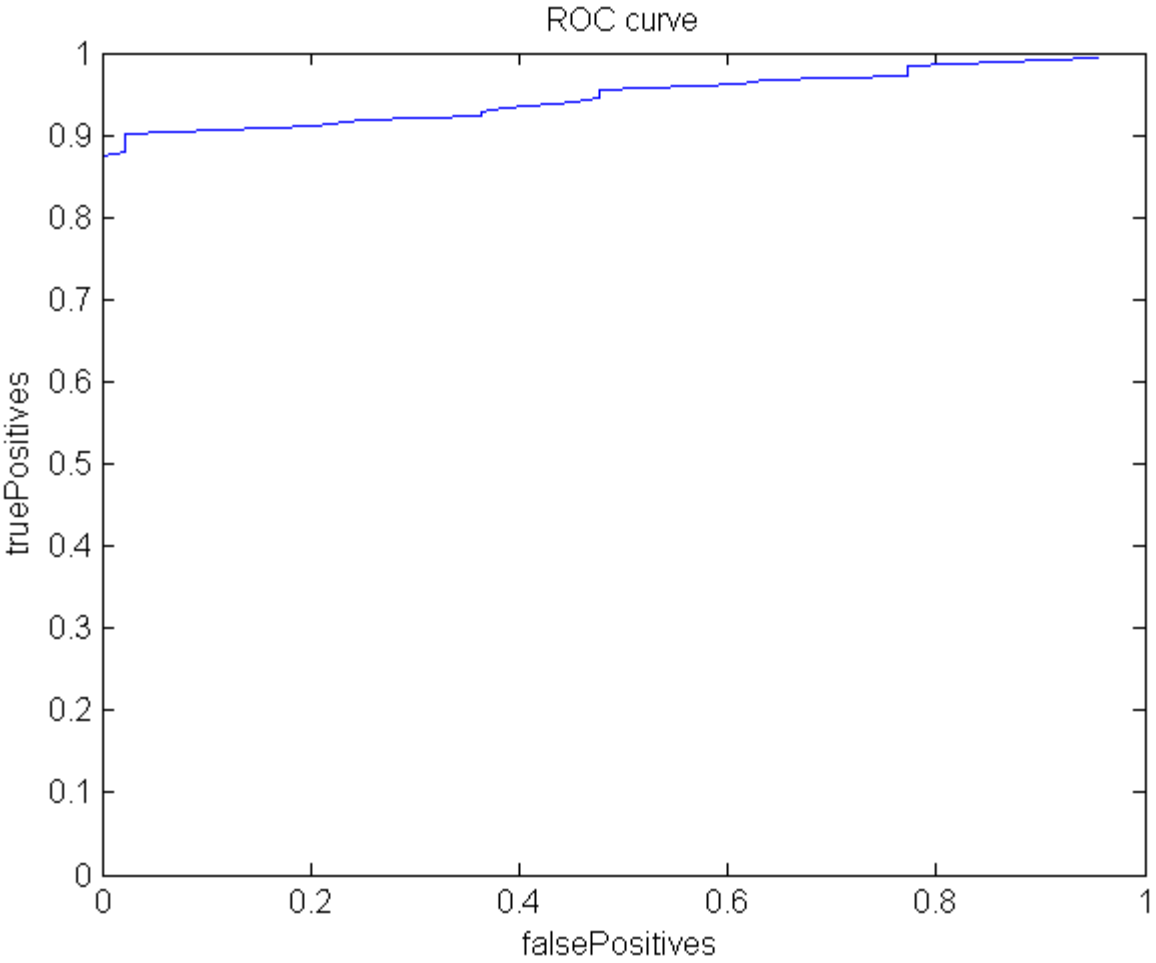
```
 Total error: 0.13
 False Positive rate: 0.48
 False Negative rate: 0.04
```

ROC curve

```
[truePositives,falsePositives] = ROC_shm(scores,testLabels);
```

Now plot the curve

```
figure;
plot(falsePositives, truePositives);
xlabel('falsePositives');
ylabel('truePositives');
title('ROC curve');
```



Published with MATLAB® R2013a

Example Usage: Fast Metric Kernel Density Estimation

Contents

Introduction

Distance metrics play an important role in nonparametric density estimation. In kernel density estimation, the notion of distance used is the Euclidean l2 norm: the density at a point x is roughly estimated as the proportion of points that fall in a Euclidean ball B(x, h) divided by the volume of this ball. Instead of using a Euclidean ball, we might use a different metric which we deem more "natural" for our particular application. Here we show how to use the "Fast Metric Kernel Estimation" modules to accomplish this. These modules are implemented using a so-called "cover-tree" data structure which enables fast estimation of the density at x. The datastructure allows us to avoid computing the distance from x to every training point (in order to identify the points that fall in the ball B(x, h)), and instead arranges the training points hierarchically in covers so that one only need to check the distances to some of the points. Note that this is a large datastructure and needs to reside in memory if we want to obtain speedups, otherwise loading time becomes the bottleneck.

We note however that in Matlab, matrix operations are usually faster that loops, and this fact can be used to compute L2 distances faster by taking advantage of its relation to inner products. The modules in "Fast Metric Kernel Estimation" therefore are mainly useful when using a distance metric that cannot be implemented fast through matrix operations.

To learn more on cover tree type datastructures, we refer the user to the following references on fast proximity search methods.

References: Samory Kpotufe. Fast, smooth and adaptive regression in metric spaces. NIPS 2009. A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbors. ICML, 2006. R. Krauthgamer and J. Lee. Navigating nets: Simple algorithms for proximity search. SODA, 2004.

SHMTools functions called:

```
learnFastMetricKernelDensity_shm
scoreFastMetricKernelDensity_shm
lkDist_shm
metricKernel_shm
```


Author: Samory Kpotufe

Date Created: January 7, 2010

Training data

The data will be drawn out of a mixture of two gaussians in R^2.

```
D = 2;
m = 700;
```

```
Xtrain = [4*randn(m/2, D); 5*randn(m/2, D) - repmat([15 0], m/2, 1)];
```

Build the model

Here we build two models, one using an l1 metric, and the other using an l2 Euclidean metric and later compare the results.

not supplying h forces learner to pick bandwidth through cross validation

```
h = [];
```

by default kernelType = 1

```
kernelType = 2;
```

kernel function to use

```
distMetric = @lkDist_shm;
```

The model contains the cover tree datastructure, the learned h and a copy of Xtrain.

```
fastL1Model = learnFastMetricKernelDensity_shm(Xtrain, h, kernelType, distMetric);
```

Now the l2 model

```
distMetric = @l2Dist_shm;
fastL2Model = learnFastMetricKernelDensity_shm(Xtrain, h, kernelType, distMetric);
```

Test data

We pick uniformly from a grid over the R^2 plane

```
[x y] = meshgrid([-40:40]);
n = size(x, 1);
t = 1;
for i=1:n
    for j =1:n
        X(t, :) = [x(i, j), y(i, j)] ;
        t = t+1;
    end
end
```

Score on the test points

The scoring functions below will return the log of the density at each test point. We record the time to compare it against the naive way of doing kernel density estimation, i.e. by computing the distance to all training points.

```
tic;
fastL1Z = scoreFastMetricKernelDensity_shm(X, fastL1Model);
```

```
fastL1Time = toc;

fastL2Z = scoreFastMetricKernelDensity_shm(X, fastL2Model);
```

Naive kernel density estimation

Now we do the estimation by the "naive" implementation where we just compute the distance to all points and use a kernel, using a bandwidth parameter h = 6 (pre-picked as a good one over this particular data). Here again we use the l1 distance of earlier.

```
h = 6;
naiveL1Z = zeros(size(X, 1), 1);
tic;
for i=1:size(X, 1)
    dist = lkDist_shm(X(i, :), Xtrain);
    weigths = metricKernel_shm(dist/h);
    naiveL1Z(i) = (1/(n*h^D))*sum(weigths);
end
naiveL1Time = toc;

fprintf('\nfast L1 Time: %2.2f s, naive L1 Time: %2.2f s\n', fastL1Time, naiveL1Time);
```

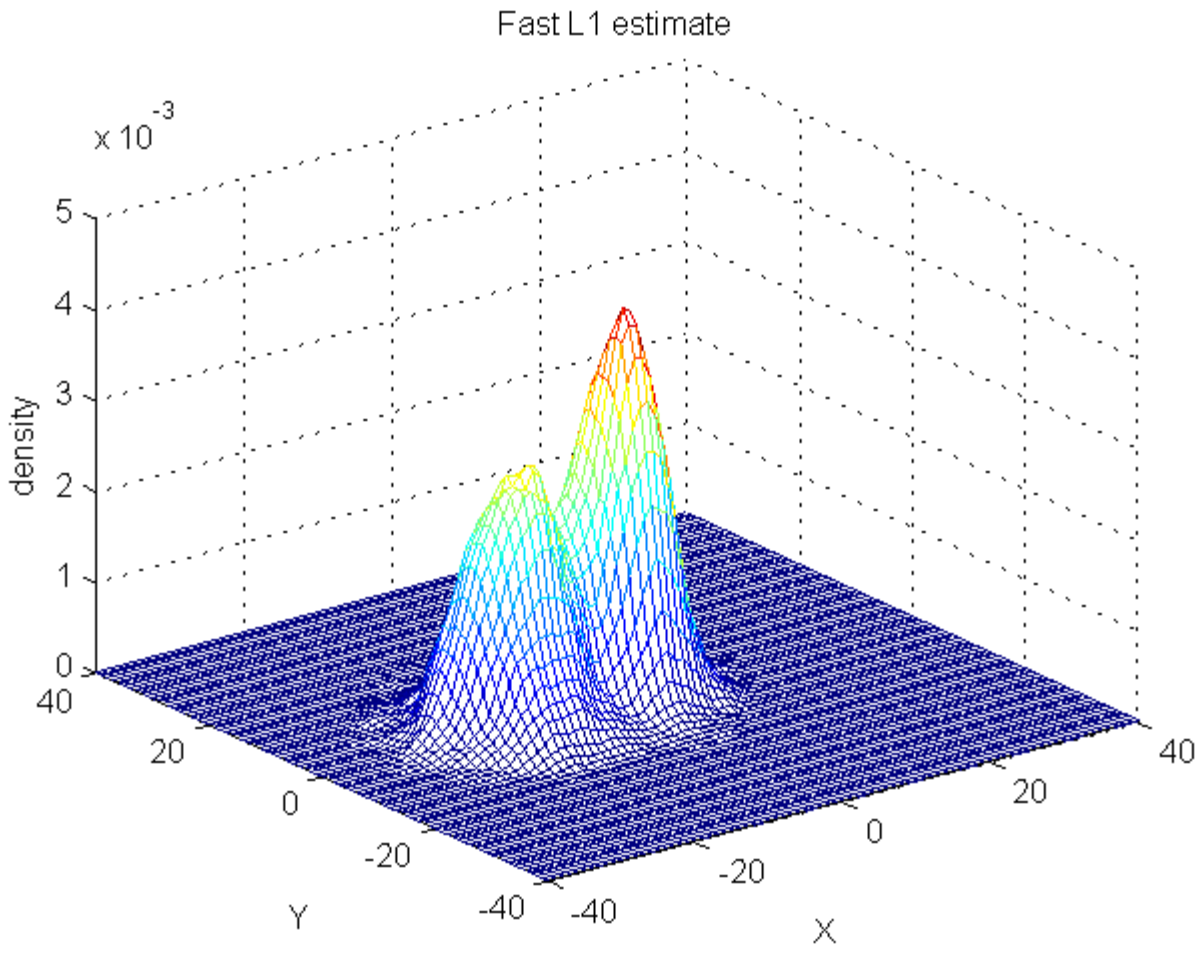
```
fast L1 Time: 2.53 s, naive L1 Time: 11.83 s
```

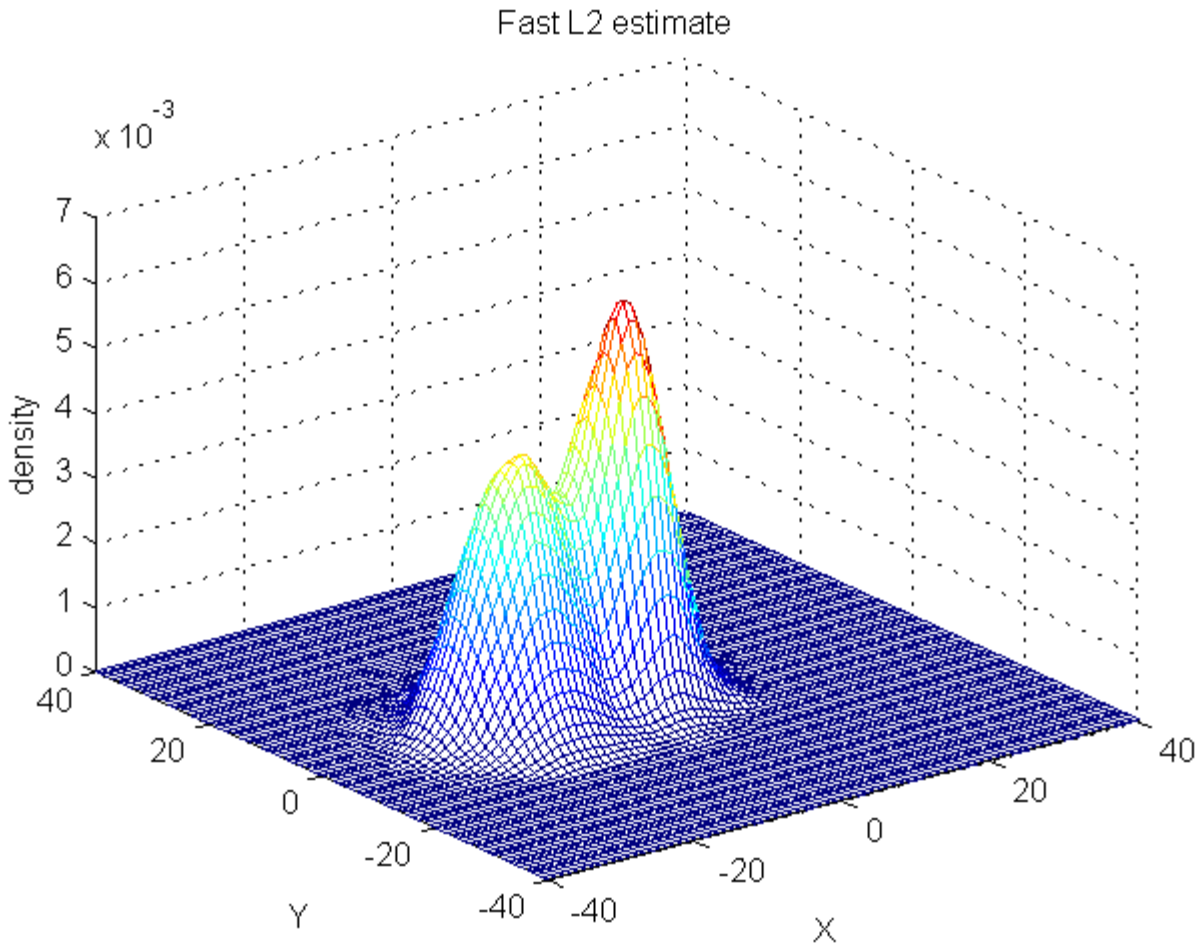
Plot the estimated densities

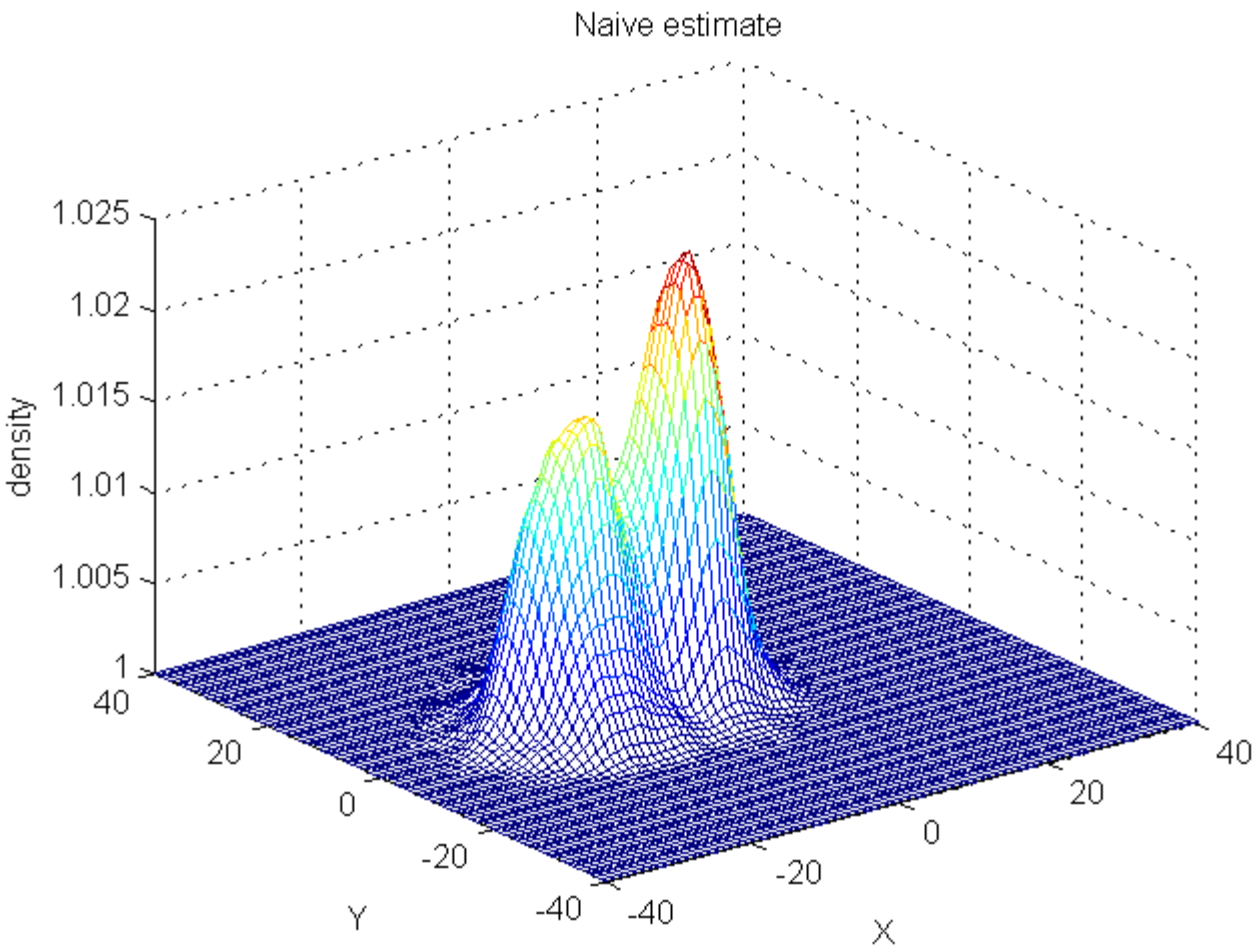
Put Z in the right format for meshing, and plot.

```
t = 1;
for i=1:n
    for j =1:n
        l1z(i, j) = exp(fastL1Z(t)) ;
        t = t+1;
    end
end

t = 1;
for i=1:n
    for j =1:n
        l2z(i, j) = exp(fastL2Z(t)) ;
        t = t+1;
    end
end

t = 1;
for i=1:n
    for j =1:n
        naivez(i, j) = exp(naiveL1Z(t)) ;
        t = t+1;
    end
end

figure;
mesh(x, y, l1z);
xlabel('X');
ylabel('Y');
zlabel('density');
```

```
title('Fast L1 estimate');

figure;
mesh(x, y, l2z);
xlabel('X');
ylabel('Y');
zlabel('density');
title('Fast L2 estimate');

figure;
mesh(x, y, naivez);
xlabel('X');
ylabel('Y');
zlabel('density');
title('Naive estimate');
```




Fast L2 estimate
x 10^-3
density
7
6
5
4
3
2
1
0
40
20
0
-20
-40
Y
-40
-20
0
20
40
X



Published with MATLAB® R2013a