University of Sheffield

Multisensor and Decision Systems Part-I Report

ACS6124

Sourojit Goswami

REGISTRATION NO: 210158764

Contents

Abstract

The main idea behind this report is to bridge the gap between theory and practical application of multi-sensor and decision systems.

In first part of the report, due to complicated structure of rotating machines and high dimensions and correlation between process features, a multiple fault classification method is discussed based on Principal Component Analysis (PCA) and K-Nearest Neighbour (KNN). After extracting features from the given signals and combining different fault types, PCA model is used for dimensionality reduction with KNN classifier to identify faults in abnormal data. At last accuracy of the algorithm is calculated which showed that the proposed method is very effective in multiple faults identification.

In the second part, common estimation techniques are discussed along with examples that have found applicability in robotics. Sensor estimation is an essential aspect to design any system, from state of the robot to any task-oriented interpretation of sensor information across space and time to facilitate planning. For this report two different type of prior knowledge of data is incorporated, Uniformly distributed prior and Gaussian distributed prior for sensor estimation. Estimators are mathematically formed for both collected and sequential measurement. Lastly a CUSUM test is performed to detect faults in a strain gauge sensor.

Introduction

In industrial process rotating machines are widely used. Due to its complex process and structure, the equipment often fails in case of improper or long operation. This may cause decline in quality and in some cases complete shutdown, which will cause immeasurable economic loss. Which is why fault diagnosis plays a crucial part in rotating machines **[1]**.

In Problem I of this report, vibration signals from five different types of faults from a rotating machine is discussed. Using Power Spectral Density (PSD) the time domain data is converted into frequency domain to extract features defined by Root Mean Square (RMS) value, after passing it through different Butterworth filters of varying orders. Then Principal Component Analysis (PCA) is used to reduce the dimensions for visualising the overall fault data. Then 1-Nearest Neighbour algorithm is applied to classify each type of fault after splitting the data into training and testing dataset. Then, accuracy of the classifier is calculated to validate the effectiveness of the algorithm used. Lastly a comparison in extracting different features is made and alternative approaches to KNN is discussed.

In recent years wind energy industry has seen a rapid growth due to environmental issues and demand for sustainable energies. Various studies to improve the reliability of health monitoring systems for wind turbines have been conducted based on the different types of sensor estimation **[2]**.

For Problem II in this report, sensor estimation for blade pitching mechanism of wind turbine along with a fault detection test is discussed. Prior knowledge of sensor data is very important for estimating sensor values, which is why two types of prior knowledge is incorporated in this report-Uniform distribution and Gaussian distribution. To optimize the estimator Bayesian Optimization is used along with Minimum Mean Squared Error (MMSE) approach. To have a better understanding of the techniques used, both collected measurement and sequential measurement is used, which is then extended over small and large size of data and how the estimator balances between prior knowledge and current data. At last, for fault detection in sensor gauge measuring the angle vertical to the rotating axis, a two-sided CUSUM Test is performed under a pre-defined threshold.

1. Problem I

For this problem vibration signals of different fault types have been plotted in both time and frequency to extract features indicating fault. After collection of different features, individual fault types are visualized and later classified using Nearest-Neighbour algorithm.

1.1 Lab A.I — Data Acquisition and Frequency Analysis.

The main idea is to obtain a feature extractor to extract features from a set of vibration signals for fault analysis pf rotating machinery. The vibration signals consist of five main faults- **Bearing defect, Gear Mesh, Resonance, Imbalance and Misalignment.** These features are then used for pattern classification to classify machine condition based on 1-nearest neighbour classifier.

1.1.1 Task I – Vibration Signal Analysis

Five different fault types are loaded and plotted to perform signal analysis in time domain to decide the features distinguishing different faults.

The signals are converted to frequency domain for spectral analysis through energy distributions using Power Spectral Density (PSD).

1.1.1.1 Result

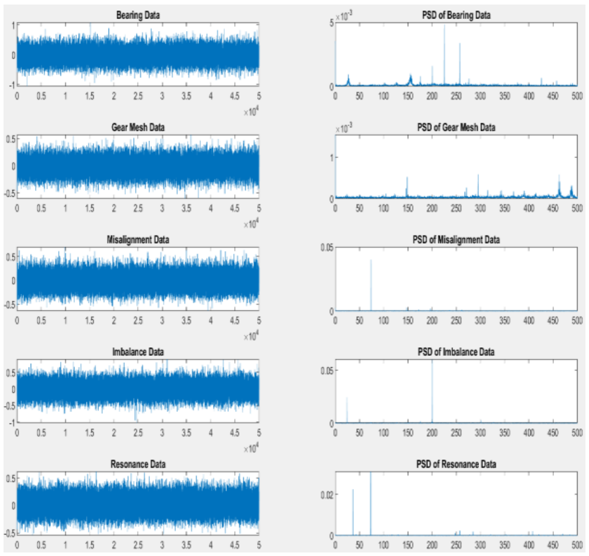


Figure 1- Different Fault Signals in Time and Frequency Domain

Left side of Figure 1 is time domain whereas right side is frequency domain for each fault type.

1.1.1.2 Analysis

After analysing Figure 1 it became difficult to decide which features to extract from each fault type like- RMS, Variance, Skewness, Entropy, etc **[3]**. The main reason behind it is the presence of noise inside the signals, which made the signals hard to read in time domain. Which is why the signals are then converted into frequency domain using PSD.

To convert the signals in frequency domain Fourier-Transform of the signals is done using Power Spectral Density (PSD). The term **power** represents the magnitude of the PSD function corresponds to the **RMS** value of the signal, whereas **spectral** indicates that it is a frequency-based function and lastly **density** states that the function has been normalised to a bandwidth of 1hz. This indicates that the RMS value of each signal can be used to extract different features **[4]**.

1.1.2 Task II-Feature Extraction

Different features have been extracted indicating fault condition based on RMS value of the signal as because it gradually increases as fault developed.

Each fault data is divided into 50 blocks of length 1000, and then normalised (**$x-x_{mean}$**).

For the first feature no filter is used, for the next three features three different Butterworth filters **(Low-pass [0-50Hz], Band-pass [50-200Hz] and High-pass [200-500Hz]) are used with filter order [11, 13, 18] respectively).**

Then RMS value of each feature is calculated, RMS = $\frac{\|PSD_i\|_2}{\sqrt{N}}$, N = length of each PSD column.

1.1.2.1 Result

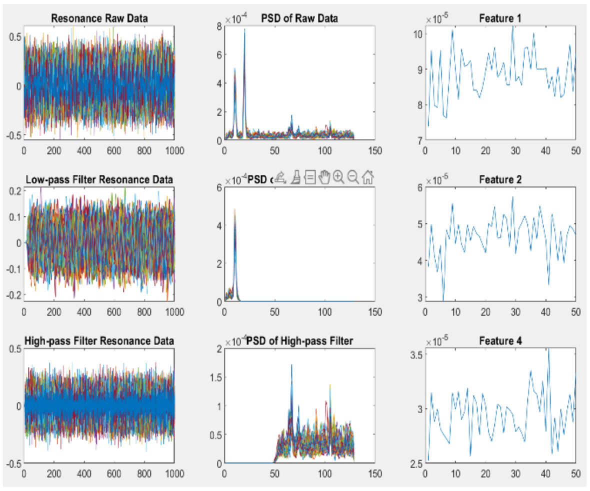


Figure 2- Feature Extraction of Resonance Data

Figure 2 shows time domain, frequency domain and RMS value(feature) for three different type of **resonance data** in three separate columns respectively.

1.1.2.2 Analysis

The first row of Figure 2 contains the raw resonance signal with its power spectral plot and RMS value.

For second row the signal is passed through a low pass filter of 11th order, which made the signal very less dense in time domain due to presence of low frequencies only, then the PSD is used to individualize faults present at lower frequency which made the calculation of RMS values much precise.

In row three a 18th order high filter is used which ultimately helped in precise calculation of RMS values at higher frequencies.

[* Butterworth filters are used because those are maximally flat filters, for a given order they have the sharpest roll-off without inducing any peak. The higher the order the flatter the response.] **[5]**

1.1.3 Data Visualisation

A feature matrix is created consisting of five fault signals each composed of four features of energy levels in different frequency bands.

Correlation co-efficient of the feature matrix is calculated along with eigen decomposition to determine ordered eigen values and eigen vectors.

Then a Transformation matrix is created for the first two principal components.

Lastly the dimension is reduced by creating a two-dimensional feature vector from the first two components to visualize the data in a scatter plot.

1.1.3.1 Result

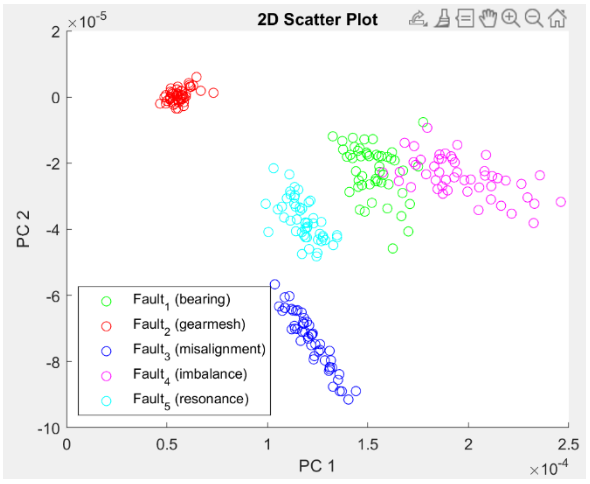


Figure 3- Scatter plot of the Faults

Figure 3 shows scatter plot of five different fault types with separate colour coding.

1.1.3.2 Analysis

It is very clear from the scatter plot in Figure 3 that each fault is separated from each other in the form of a cluster.

Although most of the clusters are separated, two of the faults (1 and 4) are showing correlation among themselves, meaning if one occurred, there is a possibility of getting the other.

1.2 Lab A.II – Pattern Classification

In pattern classification machine conditions are classified using 1 nearest neighbour algorithm.

The whole feature matrix is divided into training and testing dataset in a70:30 ratio with labels assigned to each of them.

After that a classifier is applied based on K-Nearest Neighbour Algorithm, which calculates the shortest Euclidian distance between a single testing data with whole training set for each element of testing set. Then they are labelled accordingly.

Lastly the accuracy of the classifier is calculated by matching the shortest distance labels with testing labels that have been created at start of the program.

1.2.2 Result and Evaluation

The accuracy of 1-Nearest Neighbour algorithm is→ $\boxed{\text{ACC = 98.6667}}$

Evident from Figure 3 that each fault formed a separate cluster in the scatter plot except two with little correlation, it can be said that the nearest neighbour to any of those testing data should be the one belonging to same fault type.

Thus, the classifier that has been applied worked properly classifying most of the testing data correctly with little uncertainty due to presence of correlation.

1.3 a) Extracting different features from Lab A.I to compare

Here, instead of extracting four features mentioned in section **[1.1.2]**, six different features have been extracted and visualized using same methods to establish a comprehensive relationship on effects of different features in the data. The features consist of one **Low-pass filter (25Hz), four Band-pass filters (20-50Hz, 50-100Hz, 100-200Hz, 200-350Hz) and one High-pass filter (350Hz) with 'Filter Orders' = [7 ,6, 9, 8, 9, 16] respectively.**

1.3.1 Result

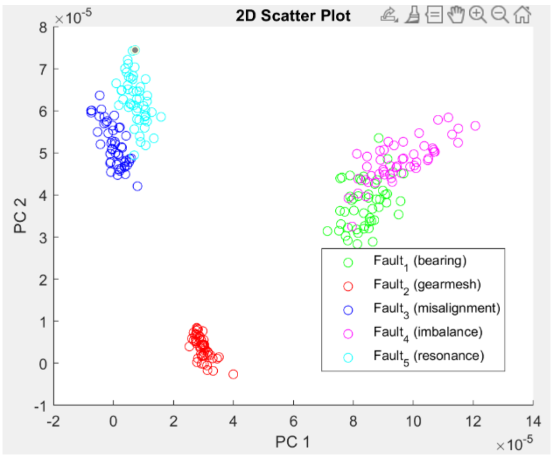


Figure 4- Scatter plot of five faults

Figure 4 shows the scatter plot of five fault signals consisting of six different features for each fault.

1.3.2 Critical Comparison

The clusters formed in Figure 4 (six features) is different compared to clusters in Figure 3 (Four features), the main reason behind it is that frequency space of six features is more scattered than that of four features.

In both cases fault 1 and fault 4 are showing correlation, more in case of six features than that of four. Not only that, fault 3 and fault 5 are also starting to be correlated in case of six features, meaning one will most likely happen in the event of other.

As, data starts to get more correlated, it becomes harder to classify distinctively one fault from other, which is why in this case taking more features might be disadvantageous.

1.4 b) K-Nearest Neighbour and Alternatives

1.4.1 PART 1

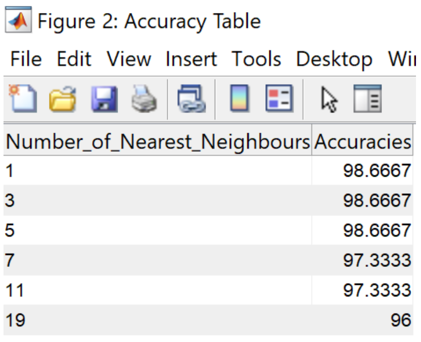
In this part the 1 nearest neighbour algorithm is extended into K-Nearest Neighbours.

To extend the 1 nearest neighbour algorithm, at first an array of k samples is developed. A distance matrix is then created to get all the Euclidian distances over a reference test data.

Then all those distances are sorted in ascending order and index of each distance is noted. After that, labels are assigned to each data by comparing the index of the data with trained labelled index.

To calculate the final label voting scheme is applied over k samples by using mode and accuracy of the algorithm is calculated by checking the number of correct assigned label.

1.4.1.2 Result



Number_of_Nearest_Neighbours	Accuracies
1	98.6667
3	98.6667
5	98.6667
7	97.3333
11	97.3333
19	96

Figure 5- Accuracy Table of KNN

Figure 5 shows accuracy of KNN algorithm over different values of ‘k’ in k-samples.

1.4.1.3 Critical Analysis

As it can be seen from Figure 5, that the accuracy of KNN doesn’t change over 1, 3 or 5 samples of nearest neighbours. This is because each fault type in our dataset formed a separate cluster with little correlation only between fault 1 and fault 4 as evident from Figure 3.

As most of the faults are separated, it doesn't matter whether 1-nearest neighbour or 3,5 nearest neighbours are taken into consideration because it's obvious that most of the nearest neighbours will belong to same cluster as because the distance between two separate clusters is bigger than distance between most of the points within same cluster.

The Accuracy changes when larger values for 'k' are take, the reason behind this is that, as its evident from data that there is a little correlation between fault 1 and fault 4, when larger values of 'k' is considered like – 7, 11 or 19 accuracy starts decreasing because of miscalculation in response to correlation and also when number of nearest neighbour starts increasing, the distance between two data points of separate cluster may get lesser than the two furthest points of the same cluster.

At last, it can be said that 5 is the highest number of nearest neighbours that can be considered to optimize the KNN algorithm for this vibration dataset.

1.4.2 PART 2- Alternative approaches and Improvements

As mentioned earlier fault diagnostic approaches can be categorized mainly into two types- model-based and data-driven approaches. Model-based techniques are mostly based on physics of the process, whereas data-driven approaches depend on the features that are extracted from the data. Here, few of the data-driven

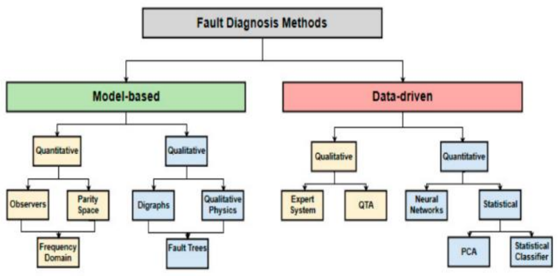


Figure 6- Fault Diagnosis

approaches are discussed **[6]**.

1.4.2.1 Support Vector Machines (SVM)

It is one of the most common algorithms used for statistical pattern recognition. Due to its effective implementation, it is used in wide range of industrial applications.

The basic principle is creation of hyperplanes to separate two groups of data with maximum margin of separation. The hyperplane that maximizes the separating distance between nearest points of both group of data is assigned as optimal hyperplane.

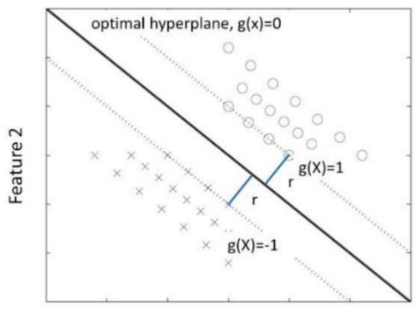


Figure 7- SVM

Figure 7 shows the idea behind SVM classification, where g(x) = 0, is the optimal hyperplane, and 'r' is the distance separating it from both group of data.

1.4.2.2 Decision Tree (DT)

In this approach a classifier represents the data in a tree scheme as a recursive partition of the data. The main idea behind decision tree is "Divide and Conquer". In simple terms a tree is formed from a dataset and divided into smaller subsets as it goes, until no further subset can be formed.

The main idea is to form a tree structure consisting of leaf nodes which further give rise to branches and the top node is called as 'root node'. Here, the hierarchy lies in top-bottom approach, which means that the root node is the best classifier.

1.4.2.3 Improved KNN / Distance-Weighted KNN (WKNN)

As discussed earlier, KNN algorithm depends on finding k-nearest neighbours for some reference data and then classified by taking a vote over those k-samples.

Distance-Weighted KNN is a modified version of the same algorithm to improve its performance and minimize misclassification. It is a variant of the KNN where a simple, yet elegant assumption is made that the impact of nearer neighbours on the test data should be more than the farther points. This means that instead of just voting based on majority, weighting of distances is also considered, where weight is inversely related to distance. Studies showed that WKNN is more efficient for multiple fault classification.

[* Other approaches that can be used are→ **Artificial Neural Network, Logistic Regression, etc.**]

2. Problem II

For this set a multisensory signal estimation and health monitoring system is designed for blade pitching mechanism of a wind turbine. To measure the pitch-angle (ω) the blade is equipped with rotatory encoder. The sensor noise is v~N (0, 9).

2.1 MMSE Estimation with uniformly distributed prior knowledge

The prior knowledge is uniformly distributed between $0^0 \leq \omega \leq 30^0$.

2.1.1 Mathematical formulation

The prior knowledge about the parameter x available before could be modelled as x ~ U [−Δ, Δ] **[7]**.

Considering a Bayesian Estimator **x*** with prior knowledge the mean squared error distribution can be defined as→ **BMSE(x*) =** $\boldsymbol{E}[(\boldsymbol{x}-\boldsymbol{x}^*)^2]$

$= \iint (\boldsymbol{x}-\boldsymbol{x}^*)^2 \boldsymbol{p}(\boldsymbol{Y}_{1:T}, \boldsymbol{x})\, \boldsymbol{dY}_{1:T}\boldsymbol{dx}$ ……. _Equation 1_where $Y_{1:T} = [y_1, y_2, \ldots\ldots\ldots, y_T]$ is the vector observation.

From Bayes' Theorem the joint PDF→ ***p(Y_{1:T} , x) = p(x/ Y_{1:T}) p(Y_{1:T})*** ……._Equation 2_

The MSE of the estimator **x*** can be written as→ **MSE(x*)**

$= \int \left(\int (\boldsymbol{x}-\boldsymbol{x}^*)^2 \boldsymbol{p}(\boldsymbol{Y}_{1:T}, \boldsymbol{x})\right) \boldsymbol{p}(\boldsymbol{dY}_{1:T})\boldsymbol{dx}$

……. _Equation 3_

To optimize the estimator **x*** the MSE is minimized, and since $p(dY_{1:T})>0$ and not a function of x, only the term inside the parenthesis is minimized→

$\frac{\partial}{\partial x^*}\int (\boldsymbol{x}-\boldsymbol{x}^*)^2 \boldsymbol{p}(\boldsymbol{Y}_{1:T}, \boldsymbol{x})\boldsymbol{dx}$ **= 0**

=> $\boldsymbol{E}[(\boldsymbol{x}|\boldsymbol{Y}_{1:T})^2]$ ……. _Equation 4_

Again, from Bayes' theorem→ $\boldsymbol{p}(\boldsymbol{x}|\boldsymbol{Y}_{1:T}) = \frac{\boldsymbol{p}(\boldsymbol{Y}_{1:T}|\boldsymbol{x})\boldsymbol{p}(\boldsymbol{x})}{\int \boldsymbol{p}(\boldsymbol{Y}_{1:T}|\boldsymbol{x})\boldsymbol{p}(\boldsymbol{x})\,\boldsymbol{dx}}$ ……. _Equation_ 5

From there assuming x and v_t are independent for all t, $\boldsymbol{p}(\boldsymbol{Y}_{1:T}|\boldsymbol{x}) = \prod_{t=1}^{T} \boldsymbol{p}(\boldsymbol{y}_t|\boldsymbol{x}) = \frac{1}{(2\pi\sigma^2)^{T/2}} \exp\left[-\frac{1}{2\sigma^2}\sum_{t=1}^{T}(\boldsymbol{y}_t-\boldsymbol{x})^2\right]$

……._Equation 6_

Lastly after some algebraic manipulation of Equation 6, the PDF→ $\boldsymbol{p}(\boldsymbol{x}|\boldsymbol{Y}_{1:T}) =$

$$\begin{cases} \frac{1}{c\sqrt{2\pi\frac{\sigma^2}{T}}} \exp\left[-\frac{1}{2\frac{\sigma^2}{T}}\left(\boldsymbol{x}-\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{y}_t\right)^2\right] & |\boldsymbol{x}| \leq \boldsymbol{\Delta} \\ \boldsymbol{0} & |\boldsymbol{x}| \geq \boldsymbol{\Delta} \end{cases}$$

……. _Equation 7_

Then final MMSE Estimator is determined by the conditional PDF. From Equation 4 and Equation 7→**x*** $= \boldsymbol{E}[(\boldsymbol{x}|\boldsymbol{Y}_{1:T})] =$

$$\int_{-\infty}^{\infty} xp(x|y_{1:T})\,dx = \frac{\int_{-\Delta}^{\Delta} \frac{x}{\sqrt{2\pi\frac{\sigma2}{T}}} \exp\left[-\frac{1}{2\frac{\sigma2}{T}}(x-\frac{1}{T}\sum_{t=1}^{T} y_t)^2\right] dx}{\int_{-\Delta}^{\Delta} \frac{1}{\sqrt{2\pi\frac{\sigma2}{T}}} \exp\left[-\frac{1}{2\frac{\sigma2}{T}}(x-\frac{1}{T}\sum_{t=1}^{T} y_t)^2\right] dx}$$ ....*Equation 8*

2.1.2 Results

Estimator table for Uniformly distributed prior.

LENGTH OF DATASET (T)	ESTIMATED VALUE
100	20.3131
0	0
5	17.3125

Table 1

2.1.3 Remarks and Evaluation

Remarks→

If there are no observations the E[x] = 0, as result the estimator x* = 0.

If $\frac{\sigma2}{T}$ -> 0, then x* = $\hat{x}$ (average of measurements) and as T increases the estimator relies more on the data and less on prior knowledge.

Lastly the estimator balances the weight of prior knowledge where x = 0 with the evidence from the data where x = $\hat{x}$.

Evaluation→

In case of the whole dataset, $\frac{\sigma2}{T} = \frac{9}{100}$ = 0.09, which is close to zero. As a result, the estimator starts to rely heavily on the data instead of prior knowledge making it equal to the mean of the measured data, i.e.,20.3131.

When no observations are there, the estimator relies on prior knowledge, E[x] = 0, as a result when only first five elements are taken it shifts its balance towards the data making it equal to the mean of those five elements, i.e., 17.3125.

2.1.4 Code

```
clear all;
a = load("encoder.mat");
A = struct2array(a);
T = length(A);
b = A(1:5,:);
% Noise
var_n = 9;
% Uniform distribution between 0-30 (Delta)
min = 0;
max = 30;
% Function for numerator
f1 = @(x)(x/sqrt(2 * pi *(var_n/T))) .* exp(-
((1/(2 * (var_n/T))) .* (x - mean(A)).^2));
```

```
% Function for denominator
f2 = @(x)(1/sqrt(2 * pi *(var_n/T))) .* exp(-
((1/(2 * (var_n/T))) .* (x - mean(A)).^2));
% MMSE
F1 = integral(f1,min,max);
F2 = integral(f2,min,max);
F = F1/F2;
disp(F)
% First five elements
n_f1 = @(x)(x/sqrt(2 * pi *(var_n/5))) .*
exp(-((1/(2 * (var_n/5))) .* (x -
mean(b)).^2));
n_f2 = @(x)(1/sqrt(2 * pi *(var_n/5))) .*
exp(-((1/(2 * (var_n/5))) .* (x -
mean(b)).^2));
n_F1 = integral(n_f1,min,max);
n_F2 = integral(n_f2,min,max);
n_F = n_F1/n_F2;
disp(n_F)
```

2.2 MMSE Estimation with Gaussian Priors

In this case the prior knowledge is Gaussian distributed with variance of 4 and mean value of 15.

2.2.1 Mathematical formulation

Let the PDF of the prior knowledge be→ $\boldsymbol{p}(\boldsymbol{x})$

$$= \frac{\mathbf{1}}{\sqrt{\mathbf{2}\pi\sigma_x^2}} \mathbf{exp}\left[-\frac{\mathbf{1}}{\mathbf{2}\sigma_x^2}(\boldsymbol{x}-\boldsymbol{\mu}_x)^2\right] \text{.......}$$

Equation 9

Recalling Equation 6 the posterior probability can be→ $p(x|Y_{1:T}) =$

$$\frac{1}{\left(2\pi\sigma_{x|Y_{1:T}}^2\right)^{1/2}} \exp\left[-\frac{1}{2\sigma_{x|Y_{1:T}}^2}\left(x-\mu_{x|Y_{1:T}}\right)^2\right]$$

.......*Equation 10* ,   Where→

$$\boldsymbol{\mu}_X|\boldsymbol{Y}_{1:T} = \frac{\frac{1}{\sigma^2}\left(\sum_{t=1}^{T} y(t)\right)+\frac{\mu_x}{\sigma_x^2}}{\frac{T}{\sigma^2}+\frac{1}{\sigma_x^2}} \text{.......} \; \textit{Equation 11}$$

As a result, from Equation 11 the estimator can be defined as→ $\boldsymbol{E}[(\boldsymbol{x}|\boldsymbol{Y}_{1:T})] =$

$$\boldsymbol{\mu}_x|\boldsymbol{Y}_{1:T} = \frac{\sigma_x^2}{\sigma_x^2+\frac{\sigma^2}{T}}\left(\frac{1}{T}\sum_{t=1}^{T} y_t\right) + \frac{\frac{\sigma^2}{T}}{\sigma_x^2+\frac{\sigma^2}{T}}\boldsymbol{\mu}_x$$

.... *Equation 12*

2.2.2 Results

Estimator table for Gaussian distributed prior.

LENGTH OF DATASET (T)	ESTIMATED VALUE
100	20.1962
5	16.5948

Table 2

2.2.3 Remarks and Evaluation

Remarks→

When large dataset is available, $\sigma_x^2 >> \frac{\sigma^2}{T}$, the value of estimator $E[(x|Y_{1:T})] \approx$ Mean of data $(\frac{1}{T}\sum_{t=1}^{T} y_t)$.

When little data is available, $\sigma_x^2 << \frac{\sigma^2}{T}$, then estimator $E[(x|Y_{1:T})] \approx$ Mean of the prior knowledge (μ_x).

Evaluation→

When the whole dataset is taken $\sigma_x^2 = 4$ becomes $>> \frac{\sigma^2}{T} = \frac{9}{100} = 0.09$. As a result, the estimator gives a result close to mean of the dataset (20.3131), i.e., 20.1962.

For the case of only first five elements $\sigma_x^2 = 4$ becomes $> \frac{\sigma^2}{T} = \frac{9}{5} = 1.8$. As because variance of prior knowledge is just a little bit bigger, the estimator became a weighted average between mean of prior knowledge (15) and mean of the data (5 elements- 17.3125), i.e., 16.5948.

2.2.4 Code

```
clear all;
a = load("encoder.mat");
A = struct2array(a);
T = length(A);
b = A(1:5,:);
% Noise
var_n = 9;
m = 15; % mean
var = 4; % variance
% Prrior Data
p = ((var_n/T)/(var + (var_n/T))) * m;
% Given data
d = ((var)/(var + (var_n/T))) * mean(A);
% MMSE
M = p + d;
disp(M)
% First five
n_p = ((var_n/5)/(var + (var_n/5))) * m;
n_d = ((var)/(var + (var_n/5))) * mean(b);
n_M = n_p + n_d;
disp(n_M)
```

2.3 Sequential MMSE Estimation with Gaussian Priors

Here also the prior knowledge is Gaussian distributed with mean of 15 and variance of 4. But the measurements are collected in a sequential manner.

2.3.1 Mathematical Formulation

From Equation 12 it can be said that for Gaussian priors the MMSE estimator is→ $E[(x|Y_{1:T})] = \mu_X|Y_{1:T}$, and as because both x_t and y_t are jointly Gaussian, the MMSE became LMMSE estimator.

As a result, when new data (y_{T+1}) becomes available the updated estimator becomes→

$$\hat{x}_{T+1} = \frac{\sigma_x^2}{(T+1)\sigma_x^2+\sigma^2}\sum_{t=1}^{T+1} y_t + \frac{\frac{\sigma^2}{T+1}}{\sigma_x^2+\frac{\sigma^2}{T+1}}\mu_x$$

……. *Equation 13*

2.3.2 Results

Estimator table for Gaussian distributed prior in sequential format.

LENGTH OF DATASET (T)	ESTIMATED VALUE
100	20.1962
5	16.5948

Table 3

2.3.3 Evaluation

The result is same as simple estimation with Gaussian Priors.

The advantage of sequential estimation is that the data can be fed to the system as soon as it comes instead of storing a sample size. This makes it useful to use in microcontrollers and in real time applications.

The main disadvantage though is that it will be more time consuming.

2.3.4 Code

```
clear all;
a = load("encoder.mat");
A = struct2array(a);
T = length(A);
b = A(1:5,:);
% Noise
var_n = 9;
m = 15; % mean
var = 4; % variance
T = length(A) - 1;
% Taking each data in sequential form
for i = 1:T+1
    s(i) = sum(A(1:i,:));
    % Prrior Data
    P = ((var_n)/(var*i + var_n)) * m;
    % Given data
    D = ((var)/(var*i + var_n)) * s(i);
    % MMSE
    s_M = P + D;
end
disp(s_M)
% First five
for i = 1:4+1
    s(i) = sum(A(1:i,:));
    n_P = ((var_n)/(var*i + var_n)) * m;
    n_D = ((var)/(var*i + var_n)) * s(i);
    n_sM = n_P + n_D;
end
disp(n_sM)
```

2.4 CUSUM Test

Sensor estimates are collected over k sample intervals from a strain gauge sensor that measures the vertical to the rotating axis bending moment of the blade. Then a two-sided CUSUM Test is performed for fault detection, under the assumption that the normal operation of the system is 3000 KNm with a variance of 1. A threshold value of ±20 has been selected, ignoring the leakage term.

$s_t = \left(\frac{y_t-\theta_0}{\sigma_0}\right)$ where, y_t = Each data element, θ_0 = Normal operation and σ_0 = Standard Deviation.

$g_t = g_{t-1} + s_t$, here g_t acts as a counter for s_t giving rise to the decision rule that states "If g_t is greater or lesser than the threshold for positive or negative fault detection respectively, then the loop will break showing that there is a fault, otherwise it will turn to zero after each loop."

2.4.2 Result

According to Figure 8 both positive and negative faults have been detected.

```
-ve_fault, take action
+ve_fault, take action
```

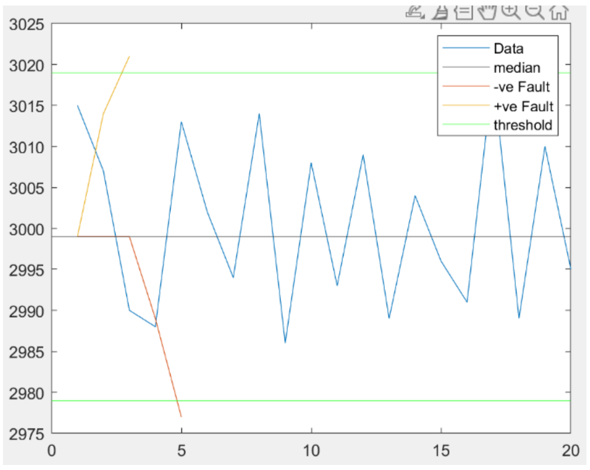


Figure 8- CUSUM Test

2.4.1 Code

```
clear all;
a = load("straingauge.mat");
A = struct2array(a);
g(1) = 0;
g1(1) = 0;
n = 3000; % Normal value
var = 1; % Variance
c = 0;
% Threshold is taken as [20, -20]
% +ve CUSUM each observation
for t = 2:length(A) + 1
    s(t-1) = (A(t-1)-n)/var;
    g(t) = g(t-1) + s(t-1);
    % -ve CUSUM each observation
    if c == 0
        for t1 = 2:length(A) + 1
            s1(t1-1) = (A(t1-1)-n)/var;
            g1(t1) = g1(t1-1) + s1(t1-1);
            if g1(t1) > 0
               g1(t1) = 0;
            end
            % Decision Rule
            if g1(t1) < -20 % Threshold
               display('-ve_fault, take action')
               c = 1;
               break
            end
         end
      end
      if g(t) < 0
         g(t) = 0;
      end
      % Decision Rule
      if g(t) > 20 % Threshold
         display('+ve_fault, take action')
         break
      end
end
% Plotting
plot(A);
hold on;
yline(median(A));
hold on;
plot((median(A)+g1))
hold on;
plot((median(A)+g));
hold on;
yline(median(A)-20, 'g');
hold on;
yline(median(A)+20, 'g')
legend('Data','median','-ve Fault','+ve
Fault','threshold')
```

Conclusion

In this report a fault classification method of five different vibration signals, along with sensor estimation for blade pitching mechanism is discussed. Features are extracted and based on high dimensionality and little correlation, PCA is applied to reduce dimension and a KNN algorithm is used to classify different faults. Accuracy is calculated which showed the effectiveness of the method used. Then comparisons are made between different features and alternative approaches are also taken into consideration. For the final part estimators are developed based on different prior knowledge of both collected and sequential data. Lastly a CUSUM test is performed over strain gauge sensor data which showed result for both positive and negative faults.

References

[1] Y. Yinghua, S. Guoqiang and S. Xiang, "Fault monitoring and classification of rotating machine based on PCA and KNN," *2018 Chinese Control And Decision Conference (CCDC)*, 2018, pp. 1795-1800, doi: 10.1109/CCDC.2018.8407418.

[2] Pandit, R.K. and Infield, D., 2018, October. Comparative analysis of binning and Gaussian Process based blade pitch angle curve of a wind turbine for the purpose of condition monitoring. In *Journal of Physics: Conference Series* (Vol. 1102, No. 1, p. 012037). IOP Publishing.

[3] Caesarendra, W. and Tjahjowidodo, T., 2017. A review of feature extraction methods in vibration-based condition monitoring and its application for degradation trend estimation of low-speed slew bearing. *Machines*, *5*(4), p.21.

[4] https://www.safeloadtesting.com/power-spectral-density/

[5] The Butterworth filter (with simple pre- and post-filters) provides a powerful and anti-aliasing function to precede the A-to-D converter of a DSP (digital system processor) system.

From: Analog Circuits Cookbook (Second Edition), 1999

[6] Jung, M., Niculita, O. and Skaf, Z., 2018. Comparison of different classification algorithms for fault detection and fault isolation in complex systems. *Procedia Manufacturing*, *19*, pp.111-118.

[7] Lecture notes produced by Dr. Inaki Esnaola and Prof. Visakan Kadirkamanatha. ACS6124 Multisensor and Decision Systems Part I: Multisensor Systems.

Appendix

Code for LAB A.I and LAB A.II

```
%% Task 1 - Data Acquisition and Frequency Analysis
clear all;
% Loading faults
load("bearing.mat")
load("gearmesh.mat")
load("misalignment.mat")
load("imbalance.mat")
load("resonance.mat")
% Time domain Plot
subplot(5,2,1)
plot(bearing);
title('Bearing Data')
subplot(5,2,3)
plot(gearmesh);
title('Gear Mesh Data')
subplot(5,2,5)
plot(misalignment);
title('Misalignment Data')
subplot(5,2,7)
plot(imbalance);
title('Imbalance Data')
subplot(5,2,9)
plot(resonance);
title('Resonance Data')
% PSD plot
subplot(5,2,2)
[P,f]= pwelch(bearing,[],[],[],1000);
plot(f,P)
title('PSD of Bearing Data')
subplot(5,2,4)
[P1,f1]= pwelch(gearmesh,[],[],[],1000);
plot(f1,P1)
title('PSD of Gear Mesh Data')
subplot(5,2,6)
[P2,f2]= pwelch(misalignment,[],[],[],1000);
plot(f2,P2)
title('PSD of Misalignment Data')
subplot(5,2,8)
[P3,f3]= pwelch(imbalance,[],[],[],1000);
plot(f3,P3)
title('PSD of Imbalance Data')
subplot(5,2,10)
[P4,f4]= pwelch(resonance,[],[],[],1000);
plot(f4,P4)
title('PSD of Resonance Data')
%% Task 2 - Feature Extraction
clear all;
% Bearing
a = load('bearing.mat');
a1 = struct2array(a);
A = reshape(a1,[1000,50]);
for j=1:50
```

```
    A1(:,j) = A(:,j)- mean(A(:,j));
end
% F1 Data
[P1,f1]= pwelch(A1,[],[],[],1000);
% F2 Data
[B1,C1]= butter(11,0.1);
y1= filter(B1,C1,A1);
[P2,f2]= pwelch(y1,[],[],[],1000);
% F3 Data
[B2,C2]= butter(13,[0.1,0.4]);
y2= filter(B2,C2,A1);
[P3,f3]= pwelch(y2,[],[],[],1000);
% F4 Data
[B3,C3]= butter(18,0.4,"high");
y3= filter(B3,C3,A1);
[P4,f4]= pwelch(y3,[],[],[],1000);
for j=1:50
    P1i(:,j)= norm(P1(:,j),2);
    N= length(P1);
    F1_b = P1i./sqrt(N); % F1
    P2i(:,j)= norm(P2(:,j),2);
    N= length(P2);
    F2_b = P2i./sqrt(N); % F2
    P3i(:,j)= norm(P3(:,j),2);
    N= length(P3);
    F3_b = P3i./sqrt(N); % F3
    P4i(:,j)= norm(P4(:,j),2);
    N= length(P4);
    F4_b = P4i./sqrt(N); % F4
end
% Gearmesh
a = load('gearmesh.mat');
a1 = struct2array(a);
A = reshape(a1,[1000,50]);
for j=1:50
    A1(:,j) = A(:,j)- mean(A(:,j));
end
% F1 Data
[P1,f1]= pwelch(A1,[],[],[],1000);
% F2 Data
[B1,C1]= butter(11,0.1);
y1= filter(B1,C1,A1);
[P2,f2]= pwelch(y1,[],[],[],1000);
% F3 Data
[B2,C2]= butter(13,[0.1,0.4]);
y2= filter(B2,C2,A1);
[P3,f3]= pwelch(y2,[],[],[],1000);
% F4 Data
[B3,C3]= butter(18,0.4,"high");
y3= filter(B3,C3,A1);
[P4,f4]= pwelch(y3,[],[],[],1000);
for j=1:50
    P1i(:,j)= norm(P1(:,j),2);
    N= length(P1);
    F1_g = P1i./sqrt(N); % F1
    P2i(:,j)= norm(P2(:,j),2);
    N= length(P2);
    F2_g = P2i./sqrt(N); % F2
    P3i(:,j)= norm(P3(:,j),2);
    N= length(P3);
    F3_g = P3i./sqrt(N); % F3
    P4i(:,j)= norm(P4(:,j),2);
    N= length(P4);
    F4_g = P4i./sqrt(N); % F4
end
% Misalignment
a = load('misalignment.mat');
a1 = struct2array(a);
A = reshape(a1,[1000,50]);
for j=1:50
    A1(:,j) = A(:,j)- mean(A(:,j));
end
% F1 Data
[P1,f1]= pwelch(A1,[],[],[],1000);
% F2 Data
[B1,C1]= butter(11,0.1);
y1= filter(B1,C1,A1);
[P2,f2]= pwelch(y1,[],[],[],1000);
% F3 Data
[B2,C2]= butter(13,[0.1,0.4]);
y2= filter(B2,C2,A1);
[P3,f3]= pwelch(y2,[],[],[],1000);
% F4 Data
[B3,C3]= butter(18,0.4,"high");
y3= filter(B3,C3,A1);
[P4,f4]= pwelch(y3,[],[],[],1000);
for j=1:50
    P1i(:,j)= norm(P1(:,j),2);
    N= length(P1);
    F1_m = P1i./sqrt(N); % F1
    P2i(:,j)= norm(P2(:,j),2);
    N= length(P2);
    F2_m = P2i./sqrt(N); % F2
    P3i(:,j)= norm(P3(:,j),2);
    N= length(P3);
    F3_m = P3i./sqrt(N); % F3
    P4i(:,j)= norm(P4(:,j),2);
    N= length(P4);
    F4_m = P4i./sqrt(N); % F4
end
% Imbalance
a = load('imbalance.mat');
a1 = struct2array(a);
A = reshape(a1,[1000,50]);
for j=1:50
    A1(:,j) = A(:,j)- mean(A(:,j));
end
% F1 Data
[P1,f1]= pwelch(A1,[],[],[],1000);
% F2 Data
[B1,C1]= butter(11,0.1);
y1= filter(B1,C1,A1);
```

```
[P2,f2]= pwelch(y1,[],[],[],1000);
% F3 Data
[B2,C2]= butter(13,[0.1,0.4]);
y2= filter(B2,C2,A1);
[P3,f3]= pwelch(y2,[],[],[],1000);
% F4 Data
[B3,C3]= butter(18,0.4,"high");
y3= filter(B3,C3,A1);
[P4,f4]= pwelch(y3,[],[],[],1000);
for j=1:50
   P1i(:,j)= norm(P1(:,j),2);
   N= length(P1);
   F1_i = P1i./sqrt(N); % F1
   P2i(:,j)= norm(P2(:,j),2);
   N= length(P2);
   F2_i = P2i./sqrt(N); % F2
   P3i(:,j)= norm(P3(:,j),2);
   N= length(P3);
   F3_i = P3i./sqrt(N); % F3
   P4i(:,j)= norm(P4(:,j),2);
   N= length(P4);
   F4_i = P4i./sqrt(N); % F4
end
% Resonance
a = load('resonance.mat');
a1 = struct2array(a);
A = reshape(a1,[1000,50]);
for j=1:50
   A1(:,j) = A(:,j)- mean(A(:,j));
end
% F1 Data
[P1,f1]= pwelch(A1,[],[],[],1000);
% F2 Data
[B1,C1]= butter(11,0.1);
y1= filter(B1,C1,A1);
[P2,f2]= pwelch(y1,[],[],[],1000);
% F3 Data
[B2,C2]= butter(13,[0.1,0.4]);
y2= filter(B2,C2,A1);
[P3,f3]= pwelch(y2,[],[],[],1000);
% F4 Data
[B3,C3]= butter(18,0.4,"high");
y3= filter(B3,C3,A1);
[P4,f4]= pwelch(y3,[],[],[],1000);
for j=1:50
   P1i(:,j)= norm(P1(:,j),2);
   N= length(P1);
   F1_r = P1i./sqrt(N); % F1
   P2i(:,j)= norm(P2(:,j),2);
   N= length(P2);
   F2_r = P2i./sqrt(N); % F2
   P3i(:,j)= norm(P3(:,j),2);
   N= length(P3);
   F3_r = P3i./sqrt(N); % F3
   P4i(:,j)= norm(P4(:,j),2);
   N= length(P4);
   F4_r = P4i./sqrt(N); % F4
end

% Plotting Resonance data
subplot(3,3,1)
plot(A1)
title('Resonance Raw Data')
subplot(3,3,2)
plot(P1)
title('PSD of Raw Data')
subplot(3,3,3)
plot(F1_r)
title('Feature 1')
subplot(3,3,4)
plot(y1)
title('Low-pass Filter Data')
subplot(3,3,5)
plot(P2)
title('PSD of Low-pass Filter')
subplot(3,3,6)
plot(F2_r)
title('Feature 2')
subplot(3,3,7)
plot(y3)
title('High-pass Filter Data')
subplot(3,3,8)
plot(P4)
title('PSD of High-pass Filter')
subplot(3,3,9)
plot(F4_r)
title('Feature 4')

%% Task 3 - Data Visualization
Fault_1= [F1_b; F2_b; F3_b; F4_b]';
Fault_2= [F1_g; F2_g; F3_g; F4_g]';
Fault_3= [F1_m; F2_m; F3_m; F4_m]';
Fault_4= [F1_i; F2_i; F3_i; F4_i]';
Fault_5= [F1_r; F2_r; F3_r; F4_r]';
G= [Fault_1; Fault_2; Fault_3; Fault_4;
Fault_5];
c=corrcoef(G); % Calculates a correlation
coefficient matrix c of G
[v,d] =eig(c); % Find the eigenvectors v and
the eigenvalues d of G
T=[ v(:,end)';v(:,end-1)']; % Create the
transformation matrix T from
% the first two principal components
z=T*G'; % Create a 2-dimensional feature
vector z
figure;
% Scatter plot of the 2-dimensional features
scatter(z(1,1:50), z(2,1:50),'green')
hold on
scatter(z(1,51:100), z(2,51:100),'red')
hold on
scatter(z(1,101:150), z(2,101:150),'blue')
```

```
hold on
scatter(z(1,151:200),
z(2,151:200),'magenta')
hold on
scatter(z(1,201:250), z(2,201:250),'cyan')
xlabel('PC 1')
ylabel ('PC 2')
title('2D Scatter Plot')
legend('Fault_1 (bearing)','Fault_2
(gearmesh)','Fault_3
(misalignment)','Fault_4
(imbalance)','Fault_5 (resonance)'
,'Location','southwest')

%% Pattern classification
numberOfTrainingCases = 35;
numberOfTestingCases = length(Fault_1) -
numberOfTrainingCases;
% Specify the number of training and testing
cases:
trainingSet = [Fault_1(1:35,:);
Fault_2(1:35,:); Fault_3(1:35,:);
Fault_4(1:35,:); Fault_5(1:35,:)];
testingSet = [Fault_1(36:end,:);
Fault_2(36:end,:);Fault_3(36:end,:);
Fault_4(36:end,:); Fault_5(36:end,:)];
% Specifying labels
trainingTarget=
[ones(1,numberOfTrainingCases)
ones(1,numberOfTrainingCases)*2
ones(1,numberOfTrainingCases)*3
ones(1,numberOfTrainingCases)*4
ones(1,numberOfTrainingCases)*5];
testingTarget=
[ones(1,numberOfTestingCases)
ones(1,numberOfTestingCases)*2
ones(1,numberOfTestingCases)*3
ones(1,numberOfTestingCases)*4
ones(1,numberOfTestingCases)*5];
% Calculate the total number of test and
train classes
totalNumberOfTestingCases =
numberOfTestingCases * 5;
totalNumberOfTrainingCases =
numberOfTrainingCases * 5;
% Create a vector to store assigned labels
inferredLabels = zeros(1,
totalNumberOfTestingCases);
% This loop cycles through each unlabelled
item:
for unlabelledCaseIdx =
1:totalNumberOfTestingCases
   unlabelledCase =
testingSet(unlabelledCaseIdx, :);
   % As any distance is shorter than infinity
   shortestDistance = inf;
```

```
    shortestDistanceLabel = 0; % Assign a
temporary label
    % This loop cycles through each labelled
item:
    for labelledCaseIdx =
1:totalNumberOfTrainingCases
        labelledCase =
trainingSet(labelledCaseIdx, :);
        % Calculate the Euclidean distance:
        currentDist = euc(unlabelledCase,
labelledCase);
        % Check the distance
        if currentDist < shortestDistance
            shortestDistance = currentDist;
            shortestDistanceLabel =
trainingTarget(labelledCaseIdx);
        end
    end
    inferredLabels(unlabelledCaseIdx) =
shortestDistanceLabel;
end
Na= totalNumberOfTestingCases;
Nc=
length(find(inferredLabels==testingTarget));
Accuracy= 100*(Nc/Na);
display(Accuracy)
```

Code for New Feature (Six Features) Extraction and Data Visualization

```
%% Feature Extraction
clear all;
% Bearing
a = load('bearing.mat');
a1 = struct2array(a);
A = reshape(a1,[1000,50]);
for j=1:50
    A1(:,j) = A(:,j)- mean(A(:,j));
end
% F1 Data
[B1,C1]= butter(7,0.05,"low");
y= filter(B1,C1,A1);
[P1,f1]= pwelch(y,[],[],[],1000);
% F2 Data
[B1,C1]= butter(6,[0.05,0.1]);
y1= filter(B1,C1,A1);
[P2,f2]= pwelch(y1,[],[],[],1000);
% F3 Data
[B2,C2]= butter(9,[0.1,0.2]);
y2= filter(B2,C2,A1);
[P3,f3]= pwelch(y2,[],[],[],1000);
% F4 Data
[B3,C3]= butter(8,[0.2,0.4]);
y3= filter(B3,C3,A1);
[P4,f4]= pwelch(y3,[],[],[],1000);
% F5 Data
```

```
[B1,C1]= butter(9,[0.4,0.7]);
y4= filter(B1,C1,A1);
[P5,f5]= pwelch(y4,[],[],[],1000);
% F6 Data
[B1,C1]= butter(16,0.7,"high");
y5= filter(B1,C1,A1);
[P6,f6]= pwelch(y5,[],[],[],1000);
for j=1:50
   P1i(:,j)= norm(P1(:,j),2);
   N= length(P1);
   F1_b = P1i./sqrt(N); % F1
   P2i(:,j)= norm(P2(:,j),2);
   N= length(P2);
   F2_b = P2i./sqrt(N); % F2
   P3i(:,j)= norm(P3(:,j),2);
   N= length(P3);
   F3_b = P3i./sqrt(N); % F3
   P4i(:,j)= norm(P4(:,j),2);
   N= length(P4);
   F4_b = P4i./sqrt(N); % F4
   P5i(:,j)= norm(P5(:,j),2);
   N= length(P5);
   F5_b = P5i./sqrt(N); % F5
   P6i(:,j)= norm(P6(:,j),2);
   N= length(P6);
   F6_b = P6i./sqrt(N); % F6
end
% Gearmesh
a = load('gearmesh.mat');
a1 = struct2array(a);
A = reshape(a1,[1000,50]);
for j=1:50
   A1(:,j) = A(:,j)- mean(A(:,j));
end
% F1 Data
[B1,C1]= butter(7,0.05,"low");
y= filter(B1,C1,A1);
[P1,f1]= pwelch(y,[],[],[],1000);
% F2 Data
[B1,C1]= butter(6,[0.05,0.1]);
y1= filter(B1,C1,A1);
[P2,f2]= pwelch(y1,[],[],[],1000);
% F3 Data
[B2,C2]= butter(9,[0.1,0.2]);
y2= filter(B2,C2,A1);
[P3,f3]= pwelch(y2,[],[],[],1000);
% F4 Data
[B3,C3]= butter(8,[0.2,0.4]);
y3= filter(B3,C3,A1);
[P4,f4]= pwelch(y3,[],[],[],1000);
% F5 Data
[B1,C1]= butter(9,[0.4,0.7]);
y4= filter(B1,C1,A1);
[P5,f5]= pwelch(y4,[],[],[],1000);
% F6 Data
[B1,C1]= butter(16,0.7,"high");
```

```
y5= filter(B1,C1,A1);
[P6,f6]= pwelch(y5,[],[],[],1000);
for j=1:50
   P1i(:,j)= norm(P1(:,j),2);
   N= length(P1);
   F1_g = P1i./sqrt(N); % F1
   P2i(:,j)= norm(P2(:,j),2);
   N= length(P2);
   F2_g = P2i./sqrt(N); % F2
   P3i(:,j)= norm(P3(:,j),2);
   N= length(P3);
   F3_g = P3i./sqrt(N); % F3
   P4i(:,j)= norm(P4(:,j),2);
   N= length(P4);
   F4_g = P4i./sqrt(N); % F4
   P5i(:,j)= norm(P5(:,j),2);
   N= length(P5);
   F5_g = P5i./sqrt(N); % F5
   P6i(:,j)= norm(P6(:,j),2);
   N= length(P6);
   F6_g = P6i./sqrt(N); % F6
end
% Misalignment
a = load('misalignment.mat');
a1 = struct2array(a);
A = reshape(a1,[1000,50]);
for j=1:50
   A1(:,j) = A(:,j)- mean(A(:,j));
end
% F1 Data
[B1,C1]= butter(7,0.05,"low");
y= filter(B1,C1,A1);
[P1,f1]= pwelch(y,[],[],[],1000);
% F2 Data
[B1,C1]= butter(6,[0.05,0.1]);
y1= filter(B1,C1,A1);
[P2,f2]= pwelch(y1,[],[],[],1000);
% F3 Data
[B2,C2]= butter(9,[0.1,0.2]);
y2= filter(B2,C2,A1);
[P3,f3]= pwelch(y2,[],[],[],1000);
% F4 Data
[B3,C3]= butter(8,[0.2,0.4]);
y3= filter(B3,C3,A1);
[P4,f4]= pwelch(y3,[],[],[],1000);
% F5 Data
[B1,C1]= butter(9,[0.4,0.7]);
y4= filter(B1,C1,A1);
[P5,f5]= pwelch(y4,[],[],[],1000);
% F6 Data
[B1,C1]= butter(16,0.7,"high");
y5= filter(B1,C1,A1);
[P6,f6]= pwelch(y5,[],[],[],1000);
for j=1:50
   P1i(:,j)= norm(P1(:,j),2);
   N= length(P1);
```

```
    F1_m = P1i./sqrt(N); % F1
    P2i(:,j)= norm(P2(:,j),2);
    N= length(P2);
    F2_m = P2i./sqrt(N); % F2
    P3i(:,j)= norm(P3(:,j),2);
    N= length(P3);
    F3_m = P3i./sqrt(N); % F3
    P4i(:,j)= norm(P4(:,j),2);
    N= length(P4);
    F4_m = P4i./sqrt(N); % F4
    P5i(:,j)= norm(P5(:,j),2);
    N= length(P5);
    F5_m = P5i./sqrt(N); % F5
    P6i(:,j)= norm(P6(:,j),2);
    N= length(P6);
    F6_m = P6i./sqrt(N); % F6
end
% Imbalance
a = load('imbalance.mat');
a1 = struct2array(a);
A = reshape(a1,[1000,50]);
for j=1:50
    A1(:,j) = A(:,j)- mean(A(:,j));
end
% F1 Data
[B1,C1]= butter(7,0.05,"low");
y= filter(B1,C1,A1);
[P1,f1]= pwelch(y,[],[],[],1000);
% F2 Data
[B1,C1]= butter(6,[0.05,0.1]);
y1= filter(B1,C1,A1);
[P2,f2]= pwelch(y1,[],[],[],1000);
% F3 Data
[B2,C2]= butter(9,[0.1,0.2]);
y2= filter(B2,C2,A1);
[P3,f3]= pwelch(y2,[],[],[],1000);
% F4 Data
[B3,C3]= butter(8,[0.2,0.4]);
y3= filter(B3,C3,A1);
[P4,f4]= pwelch(y3,[],[],[],1000);
% F5 Data
[B1,C1]= butter(9,[0.4,0.7]);
y4= filter(B1,C1,A1);
[P5,f5]= pwelch(y4,[],[],[],1000);
% F6 Data
[B1,C1]= butter(16,0.7,"high");
y5= filter(B1,C1,A1);
[P6,f6]= pwelch(y5,[],[],[],1000);
for j=1:50
    P1i(:,j)= norm(P1(:,j),2);
    N= length(P1);
    F1_i = P1i./sqrt(N); % F1
    P2i(:,j)= norm(P2(:,j),2);
    N= length(P2);
    F2_i = P2i./sqrt(N); % F2
    P3i(:,j)= norm(P3(:,j),2);
    N= length(P3);
    F3_i = P3i./sqrt(N); % F3
    P4i(:,j)= norm(P4(:,j),2);
    N= length(P4);
    F4_i = P4i./sqrt(N); % F4
    P5i(:,j)= norm(P5(:,j),2);
    N= length(P5);
    F5_i = P5i./sqrt(N); % F5
    P6i(:,j)= norm(P6(:,j),2);
    N= length(P6);
    F6_i = P6i./sqrt(N); % F6
end
% Resonance
a = load('resonance.mat');
a1 = struct2array(a);
A = reshape(a1,[1000,50]);
for j=1:50
    A1(:,j) = A(:,j)- mean(A(:,j));
end
% F1 Data
[B1,C1]= butter(7,0.05,"low");
y= filter(B1,C1,A1);
[P1,f1]= pwelch(y,[],[],[],1000);
% F2 Data
[B1,C1]= butter(6,[0.05,0.1]);
y1= filter(B1,C1,A1);
[P2,f2]= pwelch(y1,[],[],[],1000);
% F3 Data
[B2,C2]= butter(9,[0.1,0.2]);
y2= filter(B2,C2,A1);
[P3,f3]= pwelch(y2,[],[],[],1000);
% F4 Data
[B3,C3]= butter(8,[0.2,0.4]);
y3= filter(B3,C3,A1);
[P4,f4]= pwelch(y3,[],[],[],1000);
% F5 Data
[B1,C1]= butter(9,[0.4,0.7]);
y4= filter(B1,C1,A1);
[P5,f5]= pwelch(y4,[],[],[],1000);
% F6 Data
[B1,C1]= butter(16,0.7,"high");
y5= filter(B1,C1,A1);
[P6,f6]= pwelch(y5,[],[],[],1000);
for j=1:50
    P1i(:,j)= norm(P1(:,j),2);
    N= length(P1);
    F1_r = P1i./sqrt(N); % F1
    P2i(:,j)= norm(P2(:,j),2);
    N= length(P2);
    F2_r = P2i./sqrt(N); % F2
    P3i(:,j)= norm(P3(:,j),2);
    N= length(P3);
    F3_r = P3i./sqrt(N); % F3
    P4i(:,j)= norm(P4(:,j),2);
    N= length(P4);
    F4_r = P4i./sqrt(N); % F4
    P5i(:,j)= norm(P5(:,j),2);
    N= length(P5);
    F5_r = P5i./sqrt(N); % F5
    P6i(:,j)= norm(P6(:,j),2);
    N= length(P6);
    F6_r = P6i./sqrt(N); % F6
end
%% Data Visualization
Fault_1= [F1_b; F2_b; F3_b; F4_b; F5_b;
F6_b]';
Fault_2= [F1_g; F2_g; F3_g; F4_g; F5_g;
F6_g]';
Fault_3= [F1_m; F2_m; F3_m; F4_m; F5_m;
F6_m]';
Fault_4= [F1_i; F2_i; F3_i; F4_i; F5_i; F6_i]';
Fault_5= [F1_r; F2_r; F3_r; F4_r; F5_r; F6_r]';
G= [Fault_1; Fault_2; Fault_3; Fault_4;
Fault_5];
c=corrcoef(G); % Calculates a correlation
coefficient matrix c of G
[v,d] =eig(c); % Find the eigenvectors v and
the eigenvalues d of G
T=[ v(:,end)';v(:,end-1)']; % Create the
transformation matrix T from
% the first two principal components
z=T*G'; % Create a 2-dimensional feature
vector z
figure;
% Scatter plot of the 2-dimensional features
scatter(z(1,1:50), z(2,1:50),'green')
hold on
scatter(z(1,51:100), z(2,51:100),'red')
hold on
scatter(z(1,101:150), z(2,101:150),'blue')
hold on
scatter(z(1,151:200),
z(2,151:200),'magenta')
hold on
scatter(z(1,201:250), z(2,201:250),'cyan')
xlabel('PC 1')
ylabel ('PC 2')
title('2D Scatter Plot')
legend('Fault_1 (bearing)','Fault_2
(gearmesh)','Fault_3
(misalignment)','Fault_4
(imbalance)','Fault_5
(resonance)','Location','best')
```

Code for KNN

```
%% KNN algorithm
clear all;
load("Faults.mat") % Whole Fault data from
the Lab
k = [1 3 5 7 11 19]; % Nearest Neighbours
numberOfTrainingCases = 35;
```

```
numberOfTestingCases = length(Fault_1) -
numberOfTrainingCases;
% Specify the number of training and testing
cases:
trainingSet = [Fault_1(1:35,:);
Fault_2(1:35,:); Fault_3(1:35,:);
Fault_4(1:35,:); Fault_5(1:35,:)];
testingSet = [Fault_1(36:end,:);
Fault_2(36:end,:);Fault_3(36:end,:);
Fault_4(36:end,:); Fault_5(36:end,:)];
% Specifying labels
trainingTarget=
[ones(1,numberOfTrainingCases)
ones(1,numberOfTrainingCases)*2
ones(1,numberOfTrainingCases)*3
ones(1,numberOfTrainingCases)*4
ones(1,numberOfTrainingCases)*5];
testingTarget=
[ones(1,numberOfTestingCases)
ones(1,numberOfTestingCases)*2
ones(1,numberOfTestingCases)*3
ones(1,numberOfTestingCases)*4
ones(1,numberOfTestingCases)*5];
% Calculate the total number of test and
train classes
totalNumberOfTestingCases =
numberOfTestingCases * 5;
totalNumberOfTrainingCases =
numberOfTrainingCases * 5;
% This loop cycles through each unlabelled
item:
for unlabelledCaseIdx =
1:totalNumberOfTestingCases
   unlabelledCase =
testingSet(unlabelledCaseIdx, :);
   % This loop cycles through each labelled
item:
   for labelledCaseIdx =
1:totalNumberOfTrainingCases
      labelledCase =
trainingSet(labelledCaseIdx, :);
      % Calculate the Euclidean distance:
      currentDist = euc(unlabelledCase,
labelledCase);
      Dist(labelledCaseIdx, unlabelledCaseIdx)
= currentDist; % Putting the distance in an
array
   end
   [Dist_s, idx] = sort(Dist,1); % Sorting the
distance array in ascending order
end
% This loop cylcles through each choice of
neighbours
for i = 1 : length(k)
```

```
    DistanceLabel = trainingTarget(idx(1:k(i),:)); % Assigning Lable
    shortestDistanceLabel = mode(DistanceLabel,1); % Taking mode of the distance labels.
    Nc(i) = length(find(shortestDistanceLabel==testingTarget));
    Na = totalNumberOfTestingCases;
    Accuracy(i) = 100*(Nc(i)/Na); % Accuracy
end
% Accuracy Table
Number_of_Nearest_Neighbours = {'1'; '3'; '5'; '7'; '11'; '19'};
Accuracies = {Accuracy(1,1); Accuracy(1,2); Accuracy(1,3); Accuracy(1,4); Accuracy(1,5); Accuracy(1,6)};
T = table(Number_of_Nearest_Neighbours, Accuracies);
figure('Name','Accuracy Table')
uitable('Data',T{:,:},'ColumnName',T.Properties.VariableNames,...
'RowName',T.Properties.RowNames,'Units', 'Normalized', 'Position',[0, 0, 1, 1]);
```

Code for function “euc.m”

```
function distance = euc(a, b)
%Euclidean Distance
% Calculates the Euclidean distance between two cases which an equal
% number of features.
%
% Author: Andrew Hills
% Date: 23/02/07
if nargin ~= 2
  error('Two input arguments required.');
  return;
end
if ~all(size(a) == size(b))
  error('Dimensions of inputs are not equal.');
  return;
end
if min(size(a)) ~=1
  error('Input is not a vector');
  return;
end
% Calculate the Euclidean Distance using the MATLAB’s norm function
distance = norm(a - b);
```