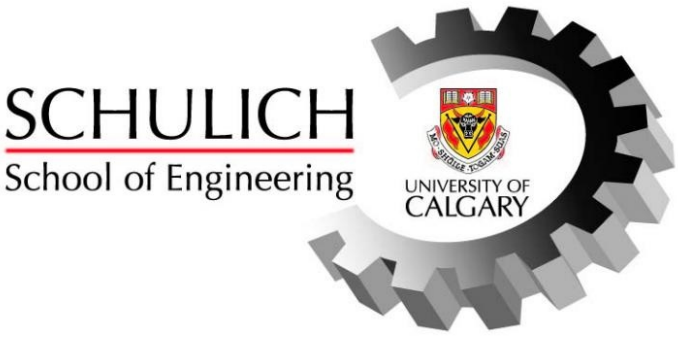
Speaker Recognition



DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING

ENCM 509 – Fundamentals of Biometric Systems Design

Group Members:

Justin Nguyen - 30042258

Lukas Morrison - 30038720

Professor	Professor	Teaching Assistant
Svetlana Yanushkevich	Helder Oliveira	Illia Yankovyi

Group Number: 9

Table of Contents

Introduction & Objective

The objective of this exercise is to create a Gaussian Mixture Model (GMM) training and classification process to successfully determine the author of a spoken speech sample. The GMM approach used for this project requires training and testing data samples, with the training samples being used to train the model. The GMM classifier will then be able to determine if the author of a speech sample exists in the trained database, even with the test sample never being exposed to the database for training. The method to achieve this functionality begins with feature extraction from the training and testing speech samples. This is done to extract the relevant information to be used in the model; for the scope of this project, the relevant feature extracted is the cepstrum of the signal (a form of frequency domain representation). The extracted features of the training samples are used in the classifier training. From these extracted features, the GMM training function calculates a mean, variance, and weight associated with each data sample. These metrics make up a statistical model for the given training sample, and collectively these statistical models make up the trained database. The GMM classifier then compares the features of a test sample against all known models in the database, outputting a similarity score for each. These similarity scores can be used to determine authorship of the test sample, with a higher similarity score corresponding to a better match. With appropriate choice of a threshold, the classifier can determine if the author of a test sample exists in the trained database.

Procedure & Analysis

1. Overview

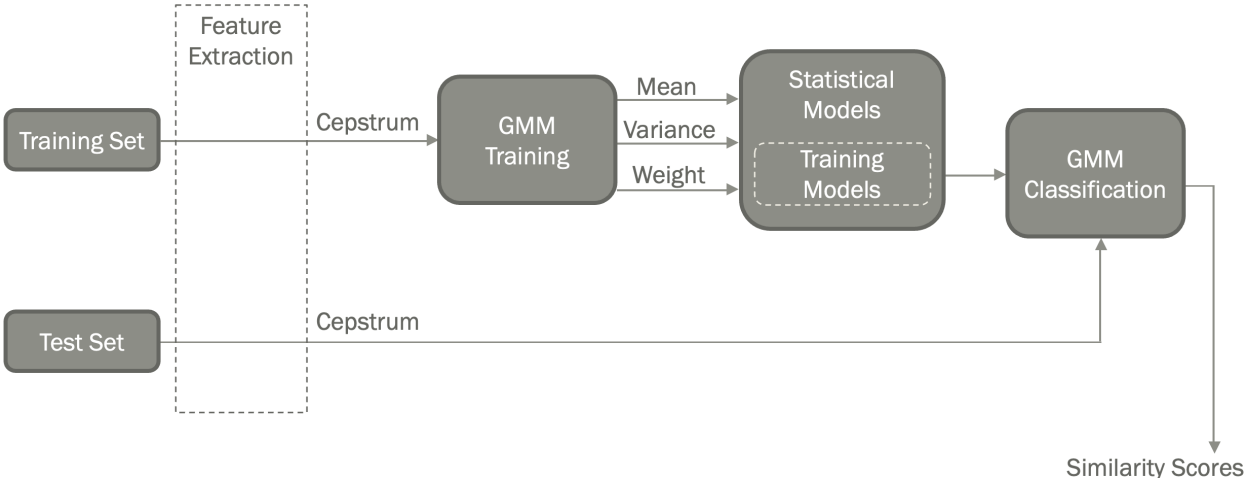


Figure 1-1: Project Flow Block Diagram

Figure 1-1 depicts the project flow diagram, depicting the main steps in the speech recognition process: feature extraction, model training, and classification.

The training sample set consisted of approximately 40-second-long speech samples collected from four subjects. The testing sample set consisted of approximately 12-second-long speech samples from six subjects, four of which were the same as the training sample set and known to the database, two of which were unknown to the database. The speech samples collected by each sample consisted of the same paragraph script within the training and testing samples. This was done to ensure consistency for the training of the model, and to ensure speaker recognition was based solely on the speaker, not the similarity of the words spoken.

The speech samples for both the training and testing sets were run through feature extraction. For this, the `melcepst` function was used (see Appendix A-2.10: melcepst.m). This function outputs the cepstrum, given inputs of a speech sample and sample rate. For the purpose of this project, all speech samples were .wav files recorded at 8000 Hz. MATLAB's `audioread` function was used to load the samples and determine the encoded sample rate.

The GMM training was performed by the `gmm_estimate` function (see Appendix A-2.3: gmm_estimate.m). This function takes the extracted features of a speech sample (cepstrum) as an input, and given the number of Gaussian components, it generates parameters to form a statistical model of the sample (mean, variance, weight). For the case of this project, the number of Gaussian components used is 30; the reasoning for choosing this number is explored in Section 2.

The GMM classification is done by the `lmultigauss` function (see Appendix A-2.6: lmultigauss.m). This function takes the features of a test speech sample and a trained statistical model as inputs. It then outputs a similarity score for the test sample compared against the statistical model. For the purpose of this project, each test sample is tested against every trained statistical model in the database. This yields a collection of similarity scores, which are then compared against a chosen threshold to determine if a match exists. For the purpose of this project, the threshold chosen is -15.7.

2. Results

Initially, the GMM speech recognizer was run using 5 Gaussian components and the results were analyzed to determine the ideal number of components. The confusion matrix of the test sample subjects tested against the database for this number of components is shown in Figure 2-1.

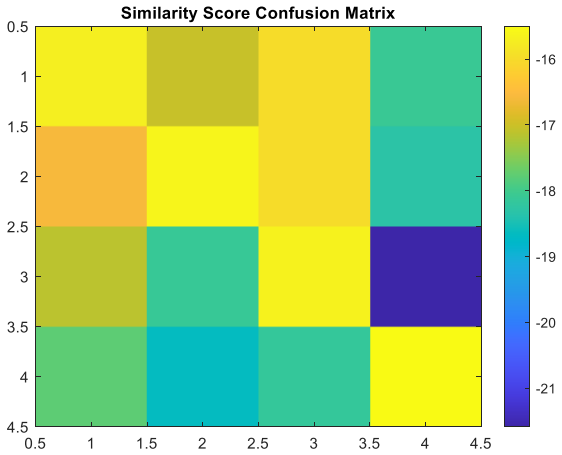


Figure 2-1: Known Sample Confusion Matrix – 5 Gaussian Components

Ideally, this plot would have perfect scores along the diagonal, and very low scores in every other section. It can be seen from this plot, however, that the diagonal sections do not have an especially large separation from other sections. This means that the classifier is obtaining similar results for a sample that is testing against itself, and a sample testing against a sample that is not by the same author. Increasing the number of Gaussian components will increase the differential between expected true scores and expected false scores, improving the performance of the classifier. For this reason, it was determined that the number of Gaussian components must be increased. The same plot for 30 Gaussian components is shown in Figure 2-2. This plot has a much better separation between the similarity scores for an expected match and an expected non-match (differences lie in the scale). The number of Gaussian components was not increased further due to diminishing returns in the spread of similarity scores. Thus, 30 was decided upon as the ideal number of Gaussian components to yield good performance without unnecessary classifier complexity.

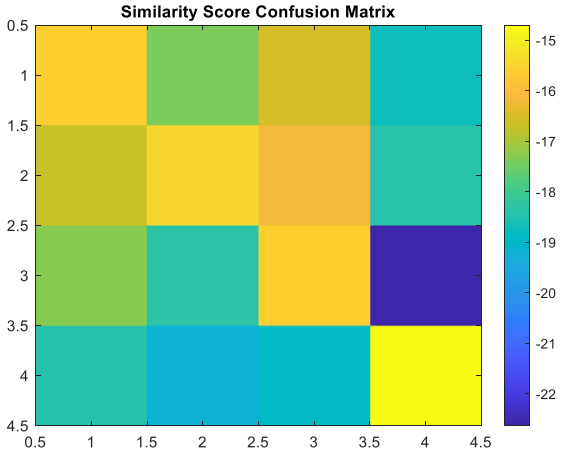


Figure 2-2: Known Sample Confusion Matrix – 30 Gaussian Components

After the ideal number of Gaussian components was determined, the testing speech samples form subjects not included in the dataset were classified against the GMM model. Each test sample not known to the database was tested against each model held within the database. For comparison purposes, the test samples for the same individuals known to the database were also included to have a reference of a true match similarity score. The confusion matrices for the test samples not known to the database are depicted in Figure 2-3 and Figure 2-4.

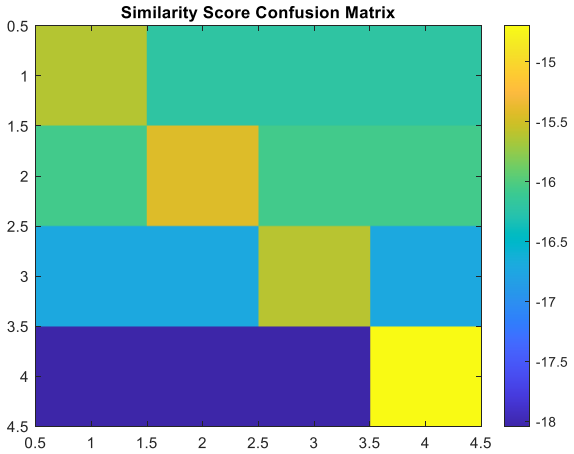


Figure 2-3: Test Subject Unknown to Database – 1

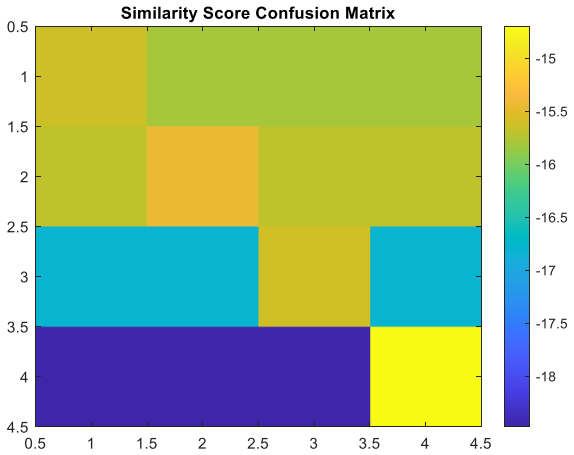


Figure 2-4: Test Subject Unknown to Database – 2

From these figures, it can be seen that the classifier always has the expected known individual as the highest score for each statistical model in the database. Additionally, the chosen threshold of -15.7 is such that all expected true individuals match with a score higher than the threshold, and all the tests with the samples not known to the database result in a similarity score lesser than the threshold. This allows the classifier to successfully recognize all test samples with a match in the database, and reject all samples not included in the database.

According to the previously mentioned trails, the error rates of the classifier are as follows:

Table 2-1: Error Rates with Varying Gaussian Components

# Gaussian Components	5	30
Error Rate (FAR)	33%	0%

The results for the classifier with the finalized choice of 30 Gaussian components are further analysed in Section 3 and Section 4.

3. ROC – Receiver Operating Characteristic

Table 3-1 and Figure 3-1 depicts the Receiver Operating Characteristic (ROC) curve for the classifier. The ROC curve is a plot of the rate of genuine attempts accepted versus the rate of imposter attempts accepted[1]. The rate of imposter attempts is displayed on the logarithmic scale. It can be seen that this curve resembles that of a perfect classifier. This can be due partially to the limited number of testing samples, as with more samples that are not included in the database there is a greater chance for misclassification by the model.

Table 3-1: GMM Classifier ROC Curve of Known Samples

Threshold	-25	-23	-21	-20	-19	-18	-17	-16	-15.5	-15.4	-15.3	-14.7
FAR (%)	100	100	91.7	83.3	41.7	25.0	0.01	0.01	0.01	0.01	0.01	0.01
TAR (%)	100	100	100	100	100	100	100	100	75.0	50.0	25.0	0.01

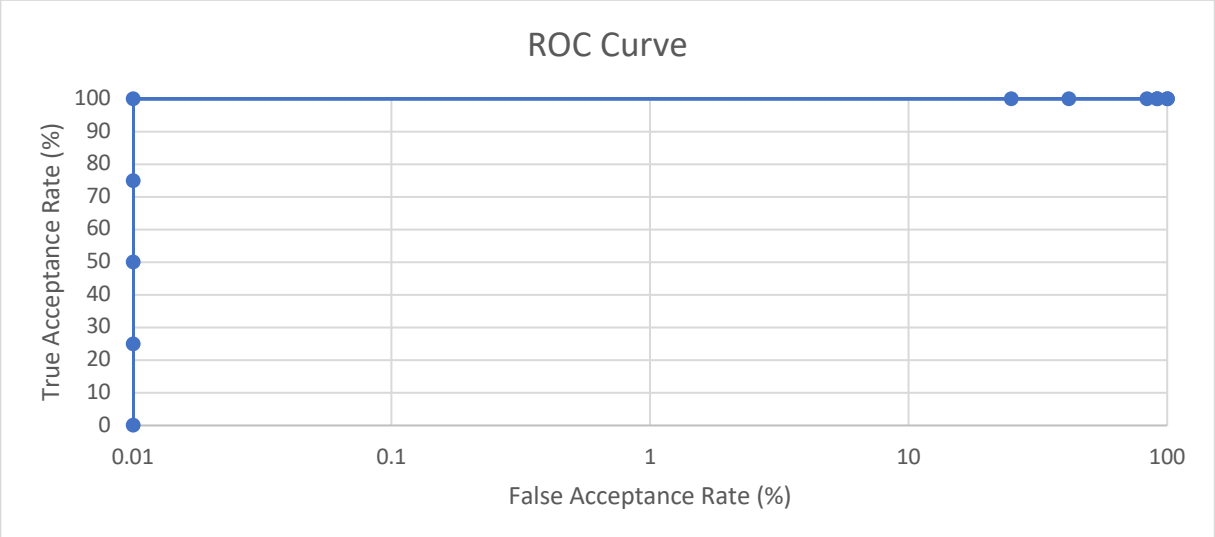


Figure 3-1: GMM Classifier ROC Curve of Known Samples

Another reason for the resemblance of a perfect classifier is the variance of scores between the subjects. The subjects within the training dataset are seen to have a significant variance between scores indicating large differences between voices as seen in Figure 3-2.

```
A =

  -15.4862  -17.4580  -16.2981  -18.6066
  -16.5963  -15.3681  -16.0975  -18.6406
  -17.2134  -18.2503  -15.4706  -22.5005
  -18.3870  -19.1643  -18.6429  -14.6421
```

Figure 3-2: Known Sample Confusion Matrix Scores

Subjects within the database varied in ethnicity[2] and gender[3], indicating clear differences in acoustics of the voices. These factors all contribute to the guise of a perfect classifier, when in reality, this classifier is capable of producing varying error rates as the scores change through each run of the code.

4. DET – Detection Error Trade-Off

The Detection Error Trade-Off (DET) curve for the classifier can be seen in Table 4-1 and Figure 4-1. The DET curve is a modified ROC curve which plots the rate of genuine attempts rejected versus the rate of imposter attempts accepted[1]. Both axes are displayed on the logarithmic scale.

Table 4-1: GMM Classifier DET Curve of Known Samples

Threshold	**-25**	**-23**	**-21**	**-20**	**-19**	**-18**	**-17**	**-16**	**-15.5**	**-15.4**	**-15.3**	**-14.7**
FAR (%)	100	100	91.7	83.3	41.7	25.0	0.01	0.01	0.01	0.01	0.01	0.01
FRR (%)	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	25.0	50.0	75.0	100

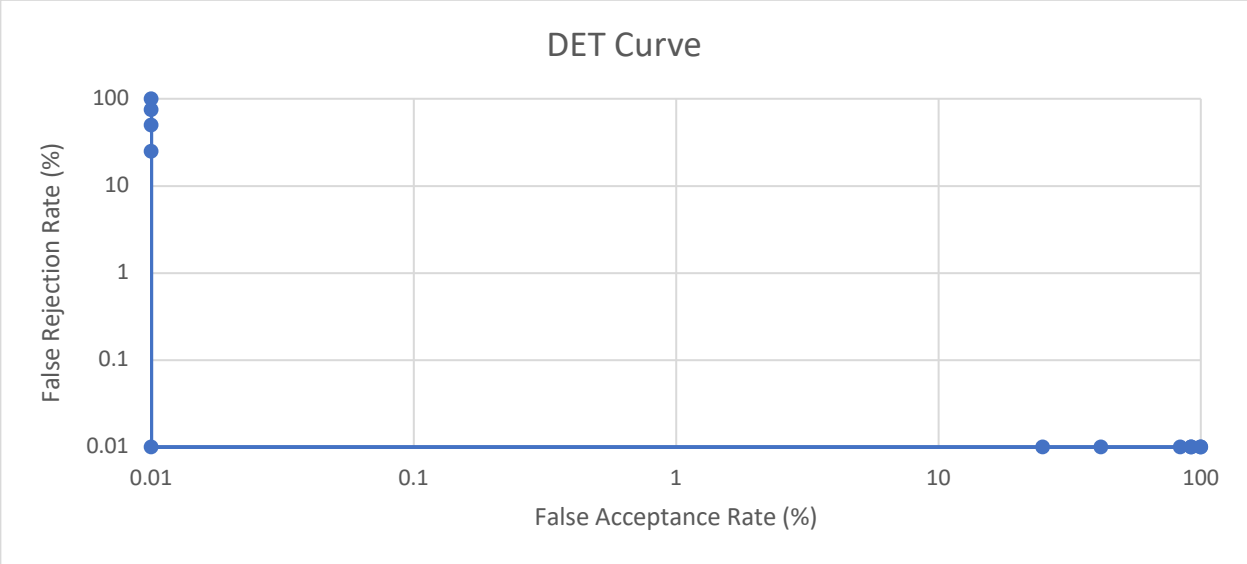


Figure 4-1: GMM Classifier DET Curve of Known Samples

Once again, this curve resembles a perfect classifier due to the limited number of subjects and the variances between those subjects. The DET curve is intended to be a more linear representation of the ROC curve, however, because of the resemblance of the perfect classifier, we are left with the inverse of the ROC curve.

Conclusion

This exercise explored the capabilities of the GMM classification process to determine the author of a spoken speech sample. This process involved extracting the cepstrum features from all training and testing data samples. This was then used for the training a GMM model with training data samples in order to create a statistical model representation of each speech author. These statistical models were compared with the features from test samples using the GMM classifier to obtain similarity scores, and finally make a decision on the validity of a speech test sample belonging to an author in the database.

From the results, the classifier is able to successfully determine the difference between subjects presented to the model through similarity scores. Genuine attempts were scored the highest while imposter attempts were scored the lowest. This difference was due to multiple factors, being the increase in the number of Gaussian components to 30, resulting in a greater differential between scores, and the variance between the limited number of subjects being tested. Subjects varied in both ethnicity[2] and gender[3], contributing to the greater divide between scores and thus in the ease in choosing a threshold to distinguish between the genuine and imposter speech samples. Using the similarity scores, confusion matrices were created as visual representation of the difference between scores through a colour-gradient. From the confusion matrices, there is a clear separation between genuine and imposter attempts as the genuine attempts were always lighter in colour than the imposter attempts. The large contrast between colours also indicates the large variance in

acoustics between subjects. These factors contributed to the resemblance of a perfect classifier when evaluating the ROC and DET curves of the known samples.

The ROC curve and the DET curve both serve the same basic purpose: evaluating the performance of a classifier. The two visualization methods, however, have strengths and weaknesses pertaining to the classifier in question. For this reason, both the ROC and DET curves for this classifier were calculated in order to capture the most meaningful metrics. The ROC curve gives the user direct feedback on the sacrifice to true positive rate when reducing the false positive rate (or vice versa). Comparatively, the DET curve gives the user direct feedback on the improvement to the false negative rate with an improvement in the false positive rate (or vice versa). The DET curve may prove more useful in distinguishing classifier performance due to the larger visual deviations, however, both serve specific use depending on the optimizations the user desires for the classifier. In the case of this speaker recognition classifier, however, the ROC curve and DET curve displayed minimal differences. Both curves communicated the conclusion that, in its current state, albeit with limited testing data, the classifier stood to have little to no room for improvement.

References

[1] ESE Department, Schulich School of Engineering, "Week 4: Jan 31 – Feb 4 Part 3: Classification decision: Threshold and Ranking," *Biometric Systems Design*, pp. 3-4, 2022.

[2] A. B. Wassink, C. Gansen, and I. Bartholomew, "Uneven success: Automatic speech recognition and ethnicity-related dialects," *Speech Communication*, vol. 140, pp. 50–70, 2022.

[3] J. R. Dubno, F.-S. Lee, L. J. Matthews, and J. H. Mills, "Age-related and gender-related changes in monaural speech recognition," *Journal of Speech, Language, and Hearing Research*, vol. 40, no. 2, pp. 444–452, 1997.

Appendix

A-1: Code Instructions

A-1.1: Initial Setup

1. Ensure `speaker_recognition.m` (see A-2.1: speaker_recognition.m) is placed in the same folder as the following files:
   A. `enframe.m` (see A-2.2: enframe.m)
   B. `gmm_estimate.m` (see A-2.3: gmm_estimate.m)
   C. `graph_gmm.m` (see A-2.4: graph_gmm.m)
   D. `histn.m` (see A-2.5: histn.m)
   E. `lmultigauss.m` (see A-2.6: lmultigauss.m)
   F. `lsum.m` (see A-2.7: lsum.m)
   G. `mel2frq.m` (see A-2.8: mel2frq.m)
   H. `melbankm.m` (see A-2.9: melbankm.m)
   I. `melcepst.m` (see A-2.10: melcepst.m)
   J. `multigauss.m` (see A-2.11: multigauss.m)
   K. `rdct.m` (see A-2.12: rdct.m)
   L. `rfft.m` (see A-2.13: rfft.m)
2. Gather voice samples and place within the main folder or a sub-folder.
   A. Recommended to obtain or convert voice samples to `.wav` file extension with a sample rate of 8000 Hz.
   B. Obtain 30-60 second voice samples of different subjects for training.
   C. Obtain 10 second voice samples of the same subjects for testing.
   D. Obtain 10 second voice samples of 1-2 subjects (probe samples) not in the training set for testing.

A-1.2: Inputting Voice Samples

1. Within `speaker_recognition.m` under `reading in the training data`, enter each gathered training voice sample into `audioread()` functions.
    A. Ensure proper path format if voice samples are within a sub-folder.
    B. Remove or add lines appropriately to the number of subjects.
2. Repeat the steps outlined above for each gathered testing voice sample under `reading in the test data`.

A-1.3: Testing Voice Samples

1. After inputting the voice samples, modify lines appropriately under `feature extraction`, `training the input data using GMM` and `testing against the input data`.
    A. For testing the probe samples with the confusion matrix, ensure the diagonal remains as true matches while every other cell will use the probe sample as input.
2. Under `testing against the input data`, adjust `truematch` to the expected number of true matches, and `truenonmatch` to the expected number of true non-matches.
3. Still under `testing against the input data`, adjust the range of `x` and `y` appropriately to the number of training subjects.
4. Run the code to obtain preliminary scores.
5. Based on all the scores obtained in the matrices, determine the appropriate threshold to differentiate between true matches and true non-matches.
6. Under `testing against the input data`, adjust the value of `threshold` as necessary.
7. Run the code again and observe the error rates obtained. If the error rates are too high, increase the number of Gaussian components (10-80) until there is a greater accuracy in the results.

A-2: Code

A-2.1: speaker_recognition.m

```
%define the number of Gaussian invariants - could be modified
No_of_Gaussians=30;
%Reading in the data
%Use wavread from matlab
disp('--------------------------------------------------------------------');
disp('                    Speaker recognition Demo');
```

```
disp('                    using GMM');
disp('-------------------------------------------------------------------');

%----------reading in the training data---------------------------------
training_data1=audioread('voice memos/wav/Lukas - 40 sec.wav');
training_data2=audioread('voice memos/wav/Josh - 38 sec.wav');
training_data3=audioread('voice memos/wav/Justin - 40 sec.wav');
training_data4=audioread('voice memos/wav/Aliah - 38 sec.wav');

%-----------reading in the test data-------------------------------------
[testing_data1,Fs]=audioread('voice memos/wav/Lukas - 13 sec.wav');
testing_data2=audioread('voice memos/wav/Josh - 13 sec.wav');
testing_data3=audioread('voice memos/wav/Justin - 12 sec.wav');
testing_data4=audioread('voice memos/wav/Aliah - 13 sec.wav');
%not in training set (probe samples)
testing_data_p1=audioread('voice memos/wav/Adam (not in set) - 11 sec.wav');
testing_data_p2=audioread('voice memos/wav/Dan (not in set) - 14 sec.wav');

disp('Completed reading taining and testing data');

%Fs=8000;   %uncoment if you cannot obtain the feature number from wavread
above

%-------------feature extraction-------------------------------------------
training_features1=melcepst(training_data1,Fs);
training_features2=melcepst(training_data2,Fs);
training_features3=melcepst(training_data3,Fs);
training_features4=melcepst(training_data4,Fs);

disp('Completed feature extraction for the training data');

testing_features1=melcepst(testing_data1,Fs);
testing_features2=melcepst(testing_data2,Fs);
testing_features3=melcepst(testing_data3,Fs);
testing_features4=melcepst(testing_data4,Fs);
%not in training set (probe samples)
testing_features_p1=melcepst(testing_data_p1,Fs);
testing_features_p2=melcepst(testing_data_p2,Fs);

disp('Completed feature extraction for the testing data');

%-------------training the input data using GMM-------------------------
%training input data, and creating the models required
disp('Training models with the input data');

[mu_train1,sigma_train1,c_train1]=gmm_estimate(training_features1',No_of_Gaus
sians);
disp('Completed Training Speaker 1 model');

[mu_train2,sigma_train2,c_train2]=gmm_estimate(training_features2',No_of_Gaus
sians);
disp('Completed Training Speaker 2 model');

[mu_train3,sigma_train3,c_train3]=gmm_estimate(training_features3',No_of_Gaus
sians);
```

```
disp('Completed Training Speaker 3 model');

[mu_train4,sigma_train4,c_train4]=gmm_estimate(training_features4',No_of_Gaus
sians);
disp('Completed Training Speaker 4 model');

disp('Completed Training ALL Models');

%-------------------------testing against the input data--------------
%testing against the first model
[lYM,lY]=lmultigauss(testing_features1', mu_train1,sigma_train1,c_train1);
A(1,1)=mean(lY);
[lYM,lY]=lmultigauss(testing_features2', mu_train1,sigma_train1,c_train1);
A(1,2)=mean(lY);
[lYM,lY]=lmultigauss(testing_features3', mu_train1,sigma_train1,c_train1);
A(1,3)=mean(lY);
[lYM,lY]=lmultigauss(testing_features4', mu_train1,sigma_train1,c_train1);
A(1,4)=mean(lY);

%testing against the second model
[lYM,lY]=lmultigauss(testing_features1', mu_train2,sigma_train2,c_train2);
A(2,1)=mean(lY);
[lYM,lY]=lmultigauss(testing_features2', mu_train2,sigma_train2,c_train2);
A(2,2)=mean(lY);
[lYM,lY]=lmultigauss(testing_features3', mu_train2,sigma_train2,c_train2);
A(2,3)=mean(lY);
[lYM,lY]=lmultigauss(testing_features4', mu_train2,sigma_train2,c_train2);
A(2,4)=mean(lY);

%testing against the third model
[lYM,lY]=lmultigauss(testing_features1', mu_train3,sigma_train3,c_train3);
A(3,1)=mean(lY);
[lYM,lY]=lmultigauss(testing_features2', mu_train3,sigma_train3,c_train3);
A(3,2)=mean(lY);
[lYM,lY]=lmultigauss(testing_features3', mu_train3,sigma_train3,c_train3);
A(3,3)=mean(lY);
[lYM,lY]=lmultigauss(testing_features4', mu_train3,sigma_train3,c_train3);
A(3,4)=mean(lY);

%testing against the fourth model
[lYM,lY]=lmultigauss(testing_features1', mu_train4,sigma_train4,c_train4);
A(4,1)=mean(lY);
[lYM,lY]=lmultigauss(testing_features2', mu_train4,sigma_train4,c_train4);
A(4,2)=mean(lY);
[lYM,lY]=lmultigauss(testing_features3', mu_train4,sigma_train4,c_train4);
A(4,3)=mean(lY);
[lYM,lY]=lmultigauss(testing_features4', mu_train4,sigma_train4,c_train4);
A(4,4)=mean(lY);

disp('Results in the form of confusion matrix for comparison');
disp('Each column i represents the test recording of Speaker i');
disp('Each row i represents the training recording of Speaker i');
disp('The diagonal elements corresponding to the same speaker');
disp('-----------------------------------------------------------------');
A
```

```
disp('-------------------------------------------------------------------------');
threshold = -15.7; %threshold based on scores
%determining match error rate
match = 0;
truematch = 4; %expected number of true matches
for x = 1:4 %cycle through matrix
    for y = 1:4
        if (A(x,y) > threshold)
            match = match + 1;
        end
    end
end
disp('Match Error Rate:');
errorrate = (truematch - match)/(match)*100;
fprintf('%.2f%%\r\n',abs(errorrate))

%determining non-match error rate
nonmatch = 0;
truenonmatch = 12; %expected number of true non-matches
for x = 1:4 %cycle through matrix
    for y = 1:4
        if (A(x,y) < threshold)
            nonmatch = nonmatch + 1;
        end
    end
end
disp('Non-Match Error Rate:');
errorrate = (truenonmatch - nonmatch)/(nonmatch)*100;
fprintf('%.2f%%\r\n',abs(errorrate))

%true/false match rate
fmatch = 0;
tmatch = 0;
falsematch = 12; %expected number of false matches
truematch = 4; %expected number of true matches
for x = 1:4 %cycle through matrix
    for y = 1:4
        if (A(x,y) > threshold)
            if (x==y)
                tmatch = tmatch + 1;
            end
            if (x~=y)
                fmatch = fmatch + 1;
            end
        end
    end
end
disp('False Match Rate:');
falserate = (fmatch)/(falsematch)*100;
fprintf('%.2f%%\r\n',abs(falserate))
disp('True Match Rate:');
truerate = (tmatch)/(truematch)*100;
fprintf('%.2f%%\r\n',abs(truerate))

%false non-match rate
```

```
fnmatch = 0;
falsenonmatch = 4; %expected number of false non-matches
for x = 1:4 %cycle through matrix
    for y = 1:4
        if (A(x,y) < threshold)
            if (x==y)
                fnmatch = fnmatch + 1;
            end
        end
    end
end
disp('False Non-Match Rate:');
fnmrate = (fnmatch)/(falsenonmatch)*100;
fprintf('%.2f%%\r\n',abs(fnmrate))


% confusion matrix in color
figure; imagesc(A); colorbar;
title("Similarity Score Confusion Matrix")
disp('-------------------------------------------------------------------');

%-------------testing first probe sample against the input data--------------
%testing against the first model
[lYM,lY]=lmultigauss(testing_features1', mu_train1,sigma_train1,c_train1);
A1(1,1)=mean(lY);
[lYM,lY]=lmultigauss(testing_features_p1', mu_train1,sigma_train1,c_train1);
A1(1,2)=mean(lY);
[lYM,lY]=lmultigauss(testing_features_p1', mu_train1,sigma_train1,c_train1);
A1(1,3)=mean(lY);
[lYM,lY]=lmultigauss(testing_features_p1', mu_train1,sigma_train1,c_train1);
A1(1,4)=mean(lY);

%testing against the second model
[lYM,lY]=lmultigauss(testing_features_p1', mu_train2,sigma_train2,c_train2);
A1(2,1)=mean(lY);
[lYM,lY]=lmultigauss(testing_features2', mu_train2,sigma_train2,c_train2);
A1(2,2)=mean(lY);
[lYM,lY]=lmultigauss(testing_features_p1', mu_train2,sigma_train2,c_train2);
A1(2,3)=mean(lY);
[lYM,lY]=lmultigauss(testing_features_p1', mu_train2,sigma_train2,c_train2);
A1(2,4)=mean(lY);

%testing against the third model
[lYM,lY]=lmultigauss(testing_features_p1', mu_train3,sigma_train3,c_train3);
A1(3,1)=mean(lY);
[lYM,lY]=lmultigauss(testing_features_p1', mu_train3,sigma_train3,c_train3);
A1(3,2)=mean(lY);
[lYM,lY]=lmultigauss(testing_features3', mu_train3,sigma_train3,c_train3);
A1(3,3)=mean(lY);
[lYM,lY]=lmultigauss(testing_features_p1', mu_train3,sigma_train3,c_train3);
A1(3,4)=mean(lY);

%testing against the fourth model
[lYM,lY]=lmultigauss(testing_features_p1', mu_train4,sigma_train4,c_train4);
A1(4,1)=mean(lY);
```

```
[lYM,lY]=lmultigauss(testing_features_p1', mu_train4,sigma_train4,c_train4);
A1(4,2)=mean(lY);
[lYM,lY]=lmultigauss(testing_features_p1', mu_train4,sigma_train4,c_train4);
A1(4,3)=mean(lY);
[lYM,lY]=lmultigauss(testing_features4', mu_train4,sigma_train4,c_train4);
A1(4,4)=mean(lY);

disp('Results in the form of confusion matrix for comparison');
disp('Each column i represents the test recording of Speaker i');
disp('Each row i represents the training recording of Speaker i');
disp('The diagonal elements corresponding to the same speaker');
disp('-------------------------------------------------------------------');
A1
disp('-------------------------------------------------------------------');
%determining match error rate
match = 0;
truematch = 4; %expected number of true matches
for x = 1:4 %cycle through matrix
    for y = 1:4
        if (A1(x,y) > threshold)
            match = match + 1;
        end
    end
end
disp('Match Error Rate:');
errorrate = (truematch - match)/(match)*100;
fprintf('%.2f%%\r\n',abs(errorrate))

%determining non-match error rate
nonmatch = 0;
truenonmatch = 12; %expected number of true non-matches
for x = 1:4 %cycle through matrix
    for y = 1:4
        if (A1(x,y) < threshold)
            nonmatch = nonmatch + 1;
        end
    end
end
disp('Non-Match Error Rate:');
errorrate = (truenonmatch - nonmatch)/(nonmatch)*100;
fprintf('%.2f%%\r\n',abs(errorrate))

% confusion matrix in color
figure; imagesc(A1); colorbar;
title("Similarity Score Confusion Matrix")
disp('-------------------------------------------------------------------');

%------------testing second probe sample against the input data-------------
%testing against the first model
[lYM,lY]=lmultigauss(testing_features1', mu_train1,sigma_train1,c_train1);
A2(1,1)=mean(lY);
[lYM,lY]=lmultigauss(testing_features_p2', mu_train1,sigma_train1,c_train1);
A2(1,2)=mean(lY);
[lYM,lY]=lmultigauss(testing_features_p2', mu_train1,sigma_train1,c_train1);
A2(1,3)=mean(lY);
```

```
[lYM,lY]=lmultigauss(testing_features_p2', mu_train1,sigma_train1,c_train1);
A2(1,4)=mean(lY);

%testing against the second model
[lYM,lY]=lmultigauss(testing_features_p2', mu_train2,sigma_train2,c_train2);
A2(2,1)=mean(lY);
[lYM,lY]=lmultigauss(testing_features2', mu_train2,sigma_train2,c_train2);
A2(2,2)=mean(lY);
[lYM,lY]=lmultigauss(testing_features_p2', mu_train2,sigma_train2,c_train2);
A2(2,3)=mean(lY);
[lYM,lY]=lmultigauss(testing_features_p2', mu_train2,sigma_train2,c_train2);
A2(2,4)=mean(lY);

%testing against the third model
[lYM,lY]=lmultigauss(testing_features_p2', mu_train3,sigma_train3,c_train3);
A2(3,1)=mean(lY);
[lYM,lY]=lmultigauss(testing_features_p2', mu_train3,sigma_train3,c_train3);
A2(3,2)=mean(lY);
[lYM,lY]=lmultigauss(testing_features3', mu_train3,sigma_train3,c_train3);
A2(3,3)=mean(lY);
[lYM,lY]=lmultigauss(testing_features_p2', mu_train3,sigma_train3,c_train3);
A2(3,4)=mean(lY);

%testing against the fourth model
[lYM,lY]=lmultigauss(testing_features_p2', mu_train4,sigma_train4,c_train4);
A2(4,1)=mean(lY);
[lYM,lY]=lmultigauss(testing_features_p2', mu_train4,sigma_train4,c_train4);
A2(4,2)=mean(lY);
[lYM,lY]=lmultigauss(testing_features_p2', mu_train4,sigma_train4,c_train4);
A2(4,3)=mean(lY);
[lYM,lY]=lmultigauss(testing_features4', mu_train4,sigma_train4,c_train4);
A2(4,4)=mean(lY);

disp('Results in the form of confusion matrix for comparison');
disp('Each column i represents the test recording of Speaker i');
disp('Each row i represents the training recording of Speaker i');
disp('The diagonal elements corresponding to the same speaker');
disp('-------------------------------------------------------------------');
A2
disp('-------------------------------------------------------------------');
%determining match error rate
match = 0;
truematch = 4; %expected number of true matches
for x = 1:4 %cycle through matrix
    for y = 1:4
        if (A2(x,y) > threshold)
            match = match + 1;
        end
    end
end
disp('Match Error Rate:');
errorrate = (truematch - match)/(match)*100;
fprintf('%.2f%%\r\n',abs(errorrate))

%determining non-match error rate
```

```
nonmatch = 0;
truenonmatch = 12; %expected number of true non-matches
for x = 1:4 %cycle through matrix
    for y = 1:4
        if (A2(x,y) < threshold)
            nonmatch = nonmatch + 1;
        end
    end
end
disp('Non-Match Error Rate:');
errorrate = (truenonmatch - nonmatch)/(nonmatch)*100;
fprintf('%.2f%%\r\n',abs(errorrate))

% confusion matrix in color
figure; imagesc(A2); colorbar;
title("Similarity Score Confusion Matrix")
disp('----------------------------------------------------------------------');
```

A-2.2: enframe.m

```
function f=enframe(x,win,inc)
%ENFRAME split signal up into (overlapping) frames: one per row.

%     Copyright (C) Mike Brookes 1997

nx=length(x);
nwin=length(win);
if (nwin == 1)
   len = win;
else
   len = nwin;
end
if (nargin < 3)
   inc = len;
end
nf = fix((nx-len+inc)/inc);
f=zeros(nf,len);
indf= inc*(0:(nf-1)).';
inds = (1:len);
f(:) = x(indf(:,ones(1,len))+inds(ones(nf,1),:));
if (nwin > 1)
    w = win(:)';
    f = f .* w(ones(nf,1),:);
end
```

A-2.3: gmm_estimate.m

```
function [mu,sigm,c]=gmm_estimate(X,M,iT,mu,sigm,c,Vm)

  DEBUG=0;
  GRAPH=0;

  % ********************************************************
  % GENERAL PARAMETERS
  [L,T]=size(X);        % data length
```

```
  varL=var(X')';    % variance for each row data;

  min_diff_LLH=0.001;   % convergence criteria

  % DEFAULTS
  if nargin<3  iT=10; end   % number of iterations, by defaut 10
  if nargin<4  mu=X(:,[fix((T-1).*rand(1,M))+1]); end % mu def: M rand vect.
  if nargin<5  sigm=repmat(varL./(M.^2),[1,M]); end % sigm def: same variance
  if nargin<6  c=ones(M,1)./M; end  % c def: same weight
  if nargin<7  Vm=4; end   % minimum variance factor

  min_sigm=repmat(varL./(Vm.^2*M.^2),[1,M]);   % MINIMUM sigma!

  if DEBUG sqrt(devs),sqrt(sigm),pause;end

  % VARIABLES
  lgam_m=zeros(T,M);    % prob of each (X:,t) to belong to the kth mixture
  lB=zeros(T,1);        % log-likelihood
  lBM=zeros(T,M);       % log-likelihhod for separate mixtures


  old_LLH=-9e99;        % initial log-likelihood

  % START ITERATATIONS
  for iter=1:iT
    if GRAPH graph_gmm(X,mu,sigm,c),pause,end
    if DEBUG disp(['************ ',num2str(iter),'
**********************']);end

    % ESTIMATION STEP ***************************************************
    [lBM,lB]=lmultigauss(X,mu,sigm,c);

    if DEBUG lB,B=exp(lB),pause; end

    LLH=mean(lB);

    disp(sprintf('log-likelihood :  %f',LLH));

    lgam_m=lBM-repmat(lB,[1,M]);  % logarithmic version
    gam_m=exp(lgam_m);            % linear version           -Equation(1)


    % MAXIMIZATION STEP *************************************************
    sgam_m=sum(gam_m);            % sum of gam_m for all X(:,t)


    % gaussian weights **************************************
    new_c=mean(gam_m)';      %                                  -Equation(4)

    % means    ****************************************
    % (convert gam_m and X to (L,M,T) and .* and then sum over T)
    mu_numerator=sum(permute(repmat(gam_m,[1,1,L]),[3,2,1]).*...
              permute(repmat(X,[1,1,M]),[1,3,2]),3);
    % convert  sgam_m(1,M,N) -> (L,M,N) and then ./
    new_mu=mu_numerator./repmat(sgam_m,[L,1]);              % -Equation(2)
```

```
    % variances ****************************************
    sig_numerator=sum(permute(repmat(gam_m,[1,1,L]),[3,2,1]).*...
                permute(repmat(X.*X,[1,1,M]),[1,3,2]),3);

    new_sigm=sig_numerator./repmat(sgam_m,[L,1])-new_mu.^2; % -Equation(3)

    % the variance is limited to a minimum
    new_sigm=max(new_sigm,min_sigm);

           %*******
    % UPDATE

    if old_LLH>=LLH-min_diff_LLH
      disp('converge');
      break;
    else

        old_LLH=LLH;

      mu=new_mu;
      sigm=new_sigm;
      c=new_c;

  end

    %*****************************************************************
  end

 graph_gmm(X,mu,sigm,c);
```

A-2.4: graph_gmm.m

```
function graph_gmm(X,mi,sig,c,coefs,ft)

DEBUG=0;
PRINT=0;
[L,T]=size(X);

if (nargin<5), coefs=1:L; end
if (nargin<6), ft=0; end

LL=length(coefs);

li=fix(sqrt(LL));
co=ceil(LL/li);

figure;
clf;


for ll=1:LL
l=coefs(ll);
  xm=min(X(l,:));
```

```
xM=max(X(l,:));
x=(-ft*(xM-xm)+xm):((ft+1)*(xM-xm)./100):(xM+ft*(xM-xm));

subplot(li,co,ll);

histn(X(l,:),300);
hold on;

if DEBUG size(x),end

[laux,lmulti]=lmultigauss(x,mi(l,:),sig(l,:),c);
aux=exp(laux);
multi=exp(lmulti);

if DEBUG size(x),size(multi'),pause,end

hp=plot(x,multi','r','Linewidth',3);
%xlim([ -xM xM ]);

ha=get(gca,'Children');
%it seem that the bars are children number 4

set(ha(2),'FaceColor',[ 0.8 0.8 0.8 ]);
set(ha(2),'EdgeColor',[ 0.8 0.8 0.8 ]);%*
plot(x,aux);
end
```

A-2.5: histn.m

```
function H=histn(x,ncont)

 [y,bc]=hist(x,ncont);
nc=sum(y).*(bc(2)-bc(1));
H=bar(bc,y./nc,'hist');
```

A-2.6: lmultigauss.m

```
function [YM,Y]=lmultigauss(x,mus,sigm,c)

DEBUG=0;
DEBUG1=0;

[L,T]=size(x);
M=size(c,1);

if DEBUG [ size(x), size(mus), size(sigm), size(c)], end

% repeating, changing dimensions:
X=permute(repmat(x',[1,1,M]),[1,3,2]);      % (T,L) -> (T,M,L) one per
mixture

Sigm=permute(repmat(sigm,[1,1,T]),[3,2,1]); % (L,M) -> (T,M,L)

Mu=permute(repmat(mus,[1,1,T]),[3,2,1]);     % (L,M) -> (T,M,L)
```

```
if DEBUG size(X), size(Sigm), size(Mu), end

%Y=squeeze(exp( 0.5.*dot(X-Mu,(X-Mu)./Sigm))) % L dissapears: (L,T,M) ->
(T,M)
lY=-0.5.*dot(X-Mu,(X-Mu)./Sigm,3);
% c,const -> (T,M) and then multiply by old Y
lcoi=log(2.*pi).*(L./2)+0.5.*sum(log(sigm),1); % c,const -> (T,M)
lcoef=repmat(log(c')-lcoi,[T,1]);

if DEBUG1 lcoi,lcoef,lY,pause;end
YM=lcoef+lY;            % ( T,M ) one mixture per column
Y=lsum(YM,2);                % add mixtures
if DEBUG [ size(YM) NaN size(Y) ], end
```

A-2.7: lsum.m

```
function lz=lsum(X,DIM);

  if nargin==1
    DIM=1;
  end

  s=size(X);


  if DIM == 1
    % formula is:
    % lz=log(bigger)+log(1+sum(exp(log(others)-log(bigger))))

    % ************************************************************
    X=sort(X,1);   % just for find bigger in all dimensions
    lz=X(end,:,:,:,:)+...
       log(1+sum(exp(X(1:end-1,:,:,:,:)-...
                     repmat(X(end,:,:,:,:),[size(X,1)-1,1,1,1,1])),1));
    % ************************************************************
  else
    % we put DIM to first dimension and do the same as before
    X=permute(X,[ DIM, 1:DIM-1 , DIM+1:length(s)]);

    % ************************************************************
        X=sort(X,1);
    lz=X(end,:,:,:,:)+...
       log(1+sum(exp(X(1:end-1,:,:,:,:)-...
                     repmat(X(end,:,:,:,:),[size(X,1)-1,1,1,1,1])),1));
    % ************************************************************

    lz=permute(lz,[2:DIM, 1, DIM+1:length(s)]);
    % we bring back dimesions
  end
```

A-2.8: mel2frq.m

```
function frq = mel2frq(mel)
%MEL2FRQ  Convert Mel frequency scale to Hertz FRQ=(MEL)
```

```
%      Copyright (C) Mike Brookes 1998

frq=700*(exp(mel/1127.01048)-1);
```

A-2.9: melbankm.m

```
function [x,mn,mx]=melbankm(p,n,fs,fl,fh,w)
%MELBANKM determine matrix for a mel-spaced filterbank

%      Copyright (C) Mike Brookes 1997

if nargin < 6
  w='tz';
  if nargin < 5
    fh=0.5;
    if nargin < 4
      fl=0;
    end
  end
end
f0=700/fs;
fn2=floor(n/2);
lr=log((f0+fh)/(f0+fl))/(p+1);
% convert to fft bin numbers with 0 for DC term
bl=n*((f0+fl)*exp([0 1 p p+1]*lr)-f0);
b2=ceil(bl(2));
b3=floor(bl(3));
if any(w=='y')
  pf=log((f0+(b2:b3)/n)/(f0+fl))/lr;
  fp=floor(pf);
  r=[ones(1,b2) fp fp+1 p*ones(1,fn2-b3)];
  c=[1:b3+1 b2+1:fn2+1];
  v=2*[0.5 ones(1,b2-1) 1-pf+fp pf-fp ones(1,fn2-b3-1) 0.5];
  mn=1;
  mx=fn2+1;
else
  b1=floor(bl(1))+1;
  b4=min(fn2,ceil(bl(4)))-1;
  pf=log((f0+(b1:b4)/n)/(f0+fl))/lr;
  fp=floor(pf);
  pm=pf-fp;
  k2=b2-b1+1;
  k3=b3-b1+1;
  k4=b4-b1+1;
  r=[fp(k2:k4) 1+fp(1:k3)];
  c=[k2:k4 1:k3];
  v=2*[1-pm(k2:k4) pm(1:k3)];
  mn=b1+1;
  mx=b4+1;
end
if any(w=='n')
  v=1-cos(v*pi/2);
elseif any(w=='m')
```

```
  v=1-0.92/1.08*cos(v*pi/2);
end
if nargout > 1
  x=sparse(r,c,v);
else
  x=sparse(r,c+mn-1,v,p,1+fn2);
end
```

A-2.10: melcepst.m

```
function c=melcepst(s,fs,w,nc,p,n,inc,fl,fh)
%MELCEPST Calculate the mel cepstrum of a signal C=(S,FS,W,NC,P,N,INC,FL,FH)

%      Copyright (C) Mike Brookes 1997

if nargin<2 fs=11025; end
if nargin<3 w='M'; end
if nargin<4 nc=12; end
if nargin<5 p=floor(3*log(fs)); end
if nargin<6 n=pow2(floor(log2(0.03*fs))); end
if nargin<9
   fh=0.5;
   if nargin<8
     fl=0;
     if nargin<7
        inc=floor(n/2);
     end
  end
end

if length(w)==0
   w='M';
end
if any(w=='R')
   z=enframe(s,n,inc);
elseif any (w=='N')
   z=enframe(s,hanning(n),inc);
else
   z=enframe(s,hamming(n),inc);
end
f=rfft(z.');
[m,a,b]=melbankm(p,n,fs,fl,fh,w);
pw=f(a:b,:).*conj(f(a:b,:));
pth=max(pw(:))*1E-6;
if any(w=='p')
   y=log(max(m*pw,pth));
else
   ath=sqrt(pth);
   y=log(max(m*abs(f(a:b,:)),ath));
end
c=rdct(y).';
nf=size(c,1);
nc=nc+1;
if p>nc
```

```
   c(:,nc+1:end)=[];
elseif p<nc
   c=[c zeros(nf,nc-p)];
end
if ~any(w=='0')
   c(:,1)=[];
   nc=nc-1;
end
if any(w=='e')
   c=[log(sum(pw)).' c];
   nc=nc+1;
end

% calculate derivative

if any(w=='D')
  vf=(4:-1:-4)/60;
  af=(1:-1:-1)/2;
  ww=ones(5,1);
  cx=[c(ww,:); c; c(nf*ww,:)];
  vx=reshape(filter(vf,1,cx(:)),nf+10,nc);
  vx(1:8,:)=[];
  ax=reshape(filter(af,1,vx(:)),nf+2,nc);
  ax(1:2,:)=[];
  vx([1 nf+2],:)=[];
  if any(w=='d')
     c=[c vx ax];
  else
     c=[c ax];
  end
elseif any(w=='d')
  vf=(4:-1:-4)/60;
  ww=ones(4,1);
  cx=[c(ww,:); c; c(nf*ww,:)];
  vx=reshape(filter(vf,1,cx(:)),nf+8,nc);
  vx(1:8,:)=[];
  c=[c vx];
end
if nargout<1
   [nf,nc]=size(c);
   t=((0:nf-1)*inc+(n-1)/2)/fs;
   ci=(1:nc)-any(w=='0')-any(w=='e');
   imh = imagesc(t,ci,c.');
   axis('xy');
   xlabel('Time (s)');
   ylabel('Mel-cepstrum coefficient');
      map = (0:63)'/63;
      colormap([map map map]);
      colorbar;
end
```

A-2.11: multigauss.m

```
function [YM,Y]=multigauss(x,mi,sigm,c)
```

```
DEBUG=0;

[L,T]=size(x);

if DEBUG L,T,end

M=size(c,1);

if DEBUG M,end

% repeating, changing dimensions:
X=permute(repmat(x',[1,1,M]),[1,3,2]);      % (T,L) -> (T,M,L) one per
mixture

Sigm=permute(repmat(sigm,[1,1,T]),[3,2,1]); % (L,M) -> (T,M,L)

Mu=permute(repmat(mi,[1,1,T]),[3,2,1]);     % (L,M) -> (T,M,L)

if DEBUG size(X),size(Mu),size(Sigm),pause;end


%Y=squeeze(exp( 0.5.*dot(X-Mu,(X-Mu)./Sigm))) % L dissapears: (L,T,M) ->
(T,M)
lY=-0.5.*dot(X-Mu,(X-Mu)./Sigm,3);
% c,const -> (T,M) and then multiply by old Y
coef=(2.*pi).^(L./2).*sqrt(prod(sigm,1)); % c,const -> (T,M)
lcoef=repmat(log(c')-log(coef),[T,1]);

if DEBUG log(coef),lcoef,lY,pause;end

YM=exp(lcoef+lY);            % ( T,M ) one mixture per column
Y=sum(YM,2);                 % add mixtures
```

A-2.12: rdct.m

```
function y=rdct(x,n,a,b)
%RDCT     Discrete cosine transform of real data Y=(X,N,A,B)

%      Copyright (C) Mike Brookes 1998

fl=size(x,1)==1;
if fl x=x(:); end
[m,k]=size(x);
if nargin<4 b=1;
    if nargin<3 a=sqrt(2*m);
        if nargin<2 n=m;
        end
    end
end
if n>m x=[x; zeros(n-m,k)];
elseif n<m x(n+1:m,:)=[];
end

x=[x(1:2:n,:); x(2*fix(n/2):-2:2,:)];
```

```
z=[sqrt(2) 2*exp((-0.5i*pi/n)*(1:n-1))].';
y=real(fft(x).*z(:,ones(1,k)))/a;
y(1,:)=y(1,:)*b;
if fl y=y.'; end
```

A-2.13: rfft.m

```
function y=rfft(x,n,d)
%RFFT     FFT of real data Y=(X,N)

%      Copyright (C) Mike Brookes 1998

s=size(x);
if prod(s)==1
    y=x
else
    if nargin <3
        d=find(s>1);
        d=d(1);
        if nargin<2
            n=s(d);
        end
    end
    if isempty(n)
        n=s(d);
    end
    y=fft(x,n,d);
    y=reshape(y,prod(s(1:d-1)),n,prod(s(d+1:end)));
    s(d)=1+fix(n/2);
    y(:,s(d)+1:end,:)=[];
    y=reshape(y,s);
end
```