1989

A Statistical approach to formant tracking

Robert T. Gayvert

A Statistical Approach to Formant Tracking

by

Robert T. Gayvert

Submitted to the Graduate Department of Computer Science of the Rochester Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science

Approved by:

Dr. James M. Hillenbrand

John A. Biles

Dr. Peter G. Anderson

Title of Thesis: **A Statistical Approach to Formant Tracking**

I, Robert Gayvert _____ hereby grant permission to the Wallace Memorial Library, of R.I.T., to reproduce my thesis in whole or in part. Any reproduction will not be used for commercial use or profit.

Date 1-31-89

ABSTRACT

This thesis investigates a statistical approach to tracking formant trajectories in continuous speech. In this approach a probability measure is applied to a set of features extracted from each analysis frame of the speech signal, and a conditional mean estimate is used to determine formant frequency values. The features used can be vector quantization symbols, spectrum levels, or other sets of features related to formant frequencies. Continuity constraints can be applied via either simple smoothing algorithms or hidden Markov models. An example of this technique using a multivariate probability measure on LPC spectral values is examined in detail. A second example using vector quantization is also examined for comparison. The performance of these trackers under a variety of conditions is discussed.

Table of Contents

I. Introduction

Formant tracking has been a topic of interest since the invention of the sound spectrograph, and a variety of techniques and features have been examined. Among the features used have been zero-crossings (Peterson, 1951; Niederjohn and Lahat, 1985), spectral peaks (Flanagan, 1956; Rabiner and Schafer, 1970; Markel, 1972; McCandless, 1974), spectral moments (Suzuki et. al, 1963), and vector quantization symbols (Kopec, 1986). Most of these attempts have suffered from three main deficiencies. First, most of the algorithms devised have made use of ad hoc rules which were typically based on results from a few sample utterances. Second, few of these algorithms were quantitatively analyzed in a methodical fashion on a large set of data. Finally, most of these algorithms are based on rigid frameworks which do not allow for additional features to be incorporated.

The most common approach to formant tracking, typified by the algorithms of Markel (1972) and McCandless (1974), has been to extract spectral peaks and then assign formant labels to these peaks according to a set of rules. There are two main problems in using peak information. First, a peak may occur where there is no formant. Second, there may be no peak where there is in fact a formant, as is often the case when two formants merge. To overcome these difficulties, a set of rules must be derived which take into account the natural continuity of formant movements together with empirically derived knowledge of formant characteristics such as the range of F1 or the maximum bandwidth of F2. These rules can be tuned to work well for a few utterances, but tend to break down

when used on a large database. The net result is that while accurate estimates can be obtained most of the time, when errors occur they can be quite large.

A recent study by Kopec (1986) used a statistical approach that did not rely on explicit peak picking. He described a family of formant trackers based on hidden Markov models and vector quantization. Instead of attempting to extract precise formant frequency values analytically, Kopec used a large set of hand-marked utterances to determine the likelihoods that a formant would be in particular frequency ranges when particular vector quantization symbols were observed. These likelihoods were used as the observation probabilities of the hidden Markov models. The states of these models represented either discrete formant frequency values or formant frequency vectors, depending on whether the formants were being tracked individually or jointly. The transition probabilities for the models were derived in a similar fashion from the hand-marked data. Formant frequency estimates were then obtained by applying the forward-backward algorithm. Kopec showed that by the use of this probabilistic formulation, ad hoc rules and tunable parameters could be avoided. In addition, he provided the first extensive quantitative analysis of a formant tracker using a large multispeaker database.

This paper proposes a statistical approach which generalizes the method introduced by Kopec (1986). A probability measure is applied to a set of features extracted from each analysis frame of the speech signal, and a conditional mean estimate is used to determine formant frequency values. The features used can be vector

quantization symbols, as in Kopec's method, or more general sets of parameters from either the time or frequency domain. Continuity constraints can be introduced via either simple smoothing algorithms or hidden Markov models. For the purposes of this paper, one primary example which uses a multivariate probability measure on LPC spectral values will be examined in detail. A second example using vector quantization will also be considered for comparison.

II. Statistical Estimation of Formant Frequencies

In the present approach, each of the formants F1, F2 and F3 is tracked individually. Thus in the following discussion we will consider a single formant, denoted by F. The speech waveform is subjected to short-time analysis which produces a set of features O_t for the analysis frame at time t. The frequency range of a formant F is partitioned into finite set of disjoint frequency intervals q_i, where q_0 is a special interval which represents a null formant. For $i \neq 0$, the midpoint of q_i, denoted by $M(q_i)$, is taken to be the formant frequency value associated with the interval q_i.

Suppose we have a probability measure which allows us to determine the probability that the value of F at time t, denoted by F(t), is in the interval q_i:

$$p_t(i) = \Pr\{ F(t) \in q_i \mid O_t \} \qquad (1)$$

Then the probability that F is present at time t is given by

$$\gamma(t) = \sum_{i \neq 0} p_t(i) \qquad (2)$$

and the expected value of F(t) is

$$\tilde{F}(t) = \sum_{i \neq 0} \frac{M(q_i) \times p_t(i)}{\gamma(t)} \qquad (3)$$

Following Kopec (1986), a formant track $\mathcal{F}(t)$ can be found by choosing a threshold value γ_{min} and setting

$$\mathcal{F}(t) = \tilde{F}(t) \qquad \text{if } \gamma(t) > \gamma_{min}. \qquad (4)$$

The selection of the threshold value γ_{min} allows some control over the relative frequency of false alarms and missed formants.

In this paper, two methods will be used to compute the probabilities $p_i(t)$. In the first method, which will be denoted by FSD, a multivariate statistical distance measure is used. Suppose that for each interval q_i the feature vector $O_t = \{o_1, ..., o_n\}$ comes from a multivariate Gaussian distribution with mean $\bar{O}_i$ and covariance matrix S_i. For any feature vector O_t, a generalized squared distance from O_t to the class representing q_i can be computed as

$$D_i^2(O) = (O_t - \bar{O}_i)^T S_i^{-1} (O_t - \bar{O}_i) + \ln |S_i| \qquad (5)$$

(Johnson and Wichern, 1982). The probability $p_i(t)$ is then given by

$$p_i(t) = \frac{\exp(-0.5 \times D_i^2(O_t))}{\sum_j \exp(-0.5 \times D_j^2(O_t))} \qquad (6)$$

The parameters $\bar{O}_i$ and S_i are determined by training data, and for a test utterance the computations in (5) and (6) are performed for each analysis frame.

In the second method, which will be denoted by FVQ, the feature vector O_t consists of a single vector quantization (VQ) symbol. This family of formant trackers has been studied extensively by Kopec, and is included here primarily for comparison. The probabilities $p_i(t)$ can be computed once and stored in a table as follows. If N_{ij} is the number of times VQ symbol v_j is associated with interval q_i in the training data for F, then the probability that F lies in q_i when v_j is observed can be estimated by

$$b_{ij} = \frac{N_{ij}}{\sum_m N_{im}} \qquad (7)$$

Thus, given an observation sequence $O = (O_1, O_2, ..., O_T)$ of VQ symbols, we can choose

$$p_i(t) = b_{ij} \qquad \text{if } O_t = v_j \qquad (8)$$

For practical reasons a small bias term should be introduced into (7) to prevent probabilities of zero (Kopec, 1986).

Since formant frequency trajectories are inherently continuous, it should be beneficial to introduce continuity constraints of some kind. One simple technique which can be used is median smoothing (Rabiner and Schafer, 1976). Passing a formant track $F(t)$ through a median smoother should remove any sharp discontinuities. Another way to introduce continuity constraints is to treat the problem as one of decoding the states of a hidden Markov model (Kopec, 1986).

Consider the intervals q_i to be the states of a Markov model and the probabilities $p_i(t)$ to be the associated observation probabilities. The transition probabilities of the model can be estimated from training data in a fashion similar to the way in which the probabilities b_{ij} were found for the FVQ tracker above. The forward-backward algorithm can then be applied to adjust the probabilities $p_i(t)$, yielding

$$\tilde{p}_t(i) = \Pr\{ F(t) \in q_i \mid O \} \qquad (9)$$

Using these adjusted values in (2) and (3) then produces new estimates for $\gamma(t)$ and $\tilde{F}(t)$.

III. Database and Training

A. Database

The corpus used in this paper consisted of 78 utterances from the vowel-dense portion of a multi-speaker, continuous speech database collected at Carnegie-Mellon University (see Appendix A). There were four male speakers and four females speakers represented, each of whom produced either nine or ten utterances. The utterances ranged in duration from 0.99 to 3.99 seconds, with a mean duration of 2.58 seconds. These utterances were divided into separate training and testing groups of 39 utterances each. An analysis interval of 5 milliseconds was used throughout, which resulted in approximately 19,800 training tokens and 20,600 test tokens.

B. Data Analysis

The utterances in the database were originally digitized at 16 kHz. For the FSD family of trackers, the signals were low-pass filtered at 6 kHz and downsampled to 12.8 kHz. Fourteenth order LPC analysis was performed every 5 msec using a 20 msec Hamming window. An LPC spectrum was generated for each analysis frame using a 128-point FFT. An appropriate portion of this spectrum was then isolated and interpolated to yield the desired number of features.

For the FVQ family of trackers, the signals were low-pass filtered at 3.5 kHz and downsampled to 8 kHz[1]. Tenth order LPC analysis was performed every 5 msec using a 20 msec Hamming window. LPC vector quantization codebooks of size 256 were generated using the Lloyd algorithm (Lloyd, 1957; Linde, Buzo, and Gray, 1980) and the Itakura-Saito distortion measure (Itakura, 1975; Gray, 1984). This size was chosen to facilitate comparison with Kopec's (1986) results and to minimize computation time.

For FSD, the mean feature vectors and covariance matrices were computed using straightforward statistics. The features used in this paper were LPC spectral values, but could in general be any features which are reasonably normally distributed. Informal inspection of these spectral values revealed that the univariate distributions of individual features were reasonably normal. The frequency ranges over which these spectral values were extracted were taken to be 0-1500, 500-3200 and 700-4000 for F1, F2 and F3,

[1]The sample frequency and LPC analysis were changed for FVQ to duplicate Kopec's (1986) conditions.

respectively. These ranges were determined by sampling a subset of the database.

C. Hand-edited Formant Tracks

A mouse-oriented tool was used to produce hand-edited formant trajectories. For each utterance in the database, a plot of spectral peaks was generated using fourteenth order LPC analysis. An LPC spectrogram, phonetic transcription and waterfall display were aligned with this plot for reference. Tracing over the plot of peaks with the mouse yielded tracks which were exactly at the peak values when the mouse was within 50 Hz of a peak. Regions with missing values were filled in either by inserting points with the mouse or by automatic interpolation between existing peaks. The results of the tracings were then smoothed with a median smoother and running average with a window size of 10 frames. Finally, the phonetic transcription was used to truncate the formant tracks so that only vowels, liquids, glides and syllabic resonants remained traced. No particular rules were used in performing the hand-editing other than general acoustic-phonetic principles. In difficult cases, visual evidence usually took precedence over other considerations.

Table 1 shows the minimum, maximum, mean and standard deviations of the hand-edited formant tracks obtained from the database used in this study. Overall, the mean values of the formants were 533 Hz, 1531 Hz and 2602 Hz for F1, F2 and F3, respectively. These statistics are in general agreement with the results obtained by Kopec (1986).

IV. Examples

The displays used for hand-editing are illustrated in Figure 1 for the utterance "The auctioneer accepted the bid" spoken by a female speaker. Beginning with raw LPC peaks, extraneous peaks were removed and missing peaks inserted. These traces were then smoothed and truncated to produce the final hand-edited formant tracks.

The mean feature vectors $\bar{O}_i$ used by one F1 FSD tracker are illustrated in Figure 2, along with a sample test vector. In Table 2, the computations in equations (2), (3), (5), (6) and (9) are shown for this test vector. The test vector is most similar to the means of q_2 and q_3, although due to its somewhat low energy there is also a 17.7% probability that F1 is not present (before applying continuity constraints). The probabilities $p_i(t)$ in column 3 were calculated using the statistical distances $D_i(t)$ in column 2. The influence of surrounding test tokens was then accounted for by the application of the forward-backward algorithm, which yielded the adjusted probabilities $\tilde{p}_i(t)$. In this example the resulting formant value $\tilde{F}(t)$ was 470 Hz and the probability that F1 was present, $\gamma(t)$, was 0.702.

Figure 3 shows the formant tracks which resulted from applying an FSD tracker to the same utterance as in Figure 1, and allowing the threshold value γ_{min} to vary from 0.0 to 0.9999. Notice that as γ_{min} is increased, the number of false alarms decreases while the number of missed formants increases.

V. Results

Quantitative results for several members of the FSD family of trackers will be presented in this section, along with a few results for the FVQ trackers. For the FSD trackers, the independent variables considered in this paper consist of quantization size, number of features, type of continuity constraint, and composition of the training and testing sets. Due to the large number of possible configurations, the effects of varying each of these variables in a baseline FSD system will be considered. Following Kopec, the principle quantitative results presented will be RMS error, percentage of large errors, percentage of missed formants, and percentage of false alarms. A large error was defined by Kopec as a frame in which the absolute difference between the hand-marked value and the tracked value is 250 Hz or greater in the case of F1 and 500 Hz or greater in the cases of F2 and F3[2] . A missed formant is defined as a frame in which there was a hand-marked value but no tracked value. Similarly, a false alarm is a frame in which there was a tracked value but no hand-marked value. For large errors and missed formants, the percentage given is a percentage of the number of frames which had a hand-marked value. Similarly, for false alarms the percentage given is a percentage of the number of frames which did not have a hand-marked value.

[2] These values are admittedly rather large, but they are useful for comparison with Kopec's (1986) results.

A. The Baseline FSD Tracker

The most extensive set of tests were performed on an FSD tracker which had a quantization size of 100 Hz and 8 features for all three formants. In other words, the intervals q_i were 0-100, 100-200, 200-300, and so on, and the portion of the spectrum used for each formant was interpolated to yield 8 spectral values. The full training set of both male and female speakers was used to generate the model statistics, and the full test set was used in the analysis.

Figure 4(a) shows the RMS error for each of the three formants as a function of the threshold value γ_{min}. With the exception of the endpoints, all three curves are rather flat. For γ_{min} = 0.5, the RMS errors were 67.6 Hz, 122.4 Hz and 214.3 Hz, for F1, F2 and F3, respectively.

The percentage of frames with large errors, given as a function of γ_{min}, is shown in Figure 4(b). Again the resulting curves are rather flat. The values for F2, typically around 0.4%, were less than those of F1, which were mostly around 0.7%, in part due to the way in which large errors were defined. F3, with around 4% large errors, performed much worse than F1 and F2 in this measure.

Missed formants and false alarms are treated in Figure 5. As can be seen, a low threshold produces few missed formants and many false alarms, while a high threshold has the opposite effect. All three formants followed the same general trends, with no great differences in magnitude.

Results across a number of phonetic classes are presented in Figure 6. In these graphs a threshold value of 0.5 was used throughout. In Figure 6(a) the RMS error is shown for several classes

which were always labeled. The worst performance for all three formants belonged to syllabic resonants, which were composed primarily of nasals. Among the rest, back vowels did the worst for F1 but the best for F2. This is likely due to the fact that for back vowels F1 and F2 are often close together or even merged, hence making F1 relatively difficult to locate. F2, on the other hand, has a small variance in back vowels, making it relatively easy to locate. For F3, the central vowels performed significantly better than any other group.

Figure 6(b) illustrates the false alarm rate for several classes of phonemes which were never labeled in the hand-marked data. An interesting result here is that the false alarm rates were fairly equal across all three formants. Flaps and trills, although not common, almost always generated formant values. About 30 percent of the frames labeled as stops yielded false alarms, while for fricatives and affricates the rate was about 15% and 10%, respectively. Non-speech regions, which consisted of silence, breath noise, and a few other miscellaneous sounds, produced false alarms only about 3% of the time.

B. Quantization Size

Figure 7 shows the effect of varying the size of the frequency intervals q_i from 50 Hz up to 300 Hz while holding the number of features constant at 8. In general, the RMS error is smaller for smaller quantization sizes. However, with smaller quantization sizes there are more frequency intervals q_i. Hence, as the quantization size is decreased there is also an accompanying increase in computa-

tion time required for both the distance measures and for the forward-backward algorithm.

C. Number of Features

The effect of varying the number of features, i.e., spectral values, is shown in Figure 8. The results shown are for F2, with a constant quantization size of 100 Hz. Note that the curves for RMS error and percentage of large errors are nearly identical in shape. This may indicate that there is little difference in the accuracy of the formant estimates produced by different numbers of features when there is not a large error. The acceptable performance of 8 features in this test led to its selection as the baseline value. The slight decrease in performance for more than 10 features was somewhat surprising, but may be due to either insufficient training data or to the possibility that the Gaussian assumption becomes less viable.

D. Continuity Constraints

Figure 9 reveals the importance of continuity constraints. The top curves in Figures 9(a) and 9(b) show the RMS error and percentage of large errors for F1 when a track is produced using only the midpoint value of the most likely interval q_i. Median smoothing this track reduces the RMS error somewhat by removing about half of the large errors, but is still not as good as using a mean conditional estimate without any smoothing. Median smoothing the mean conditional estimates removes a few large errors, but not nearly as many as the forward-backward algorithm, which removes more than half.

Moreover, the effect of the forward-backward algorithm is nearly constant across threshold values.

E. Testing and Training Sets

As shown in Figure 10, there does not appear to be a clear advantage to using separate models for males and females to train the FSD tracker. When the baseline FSD system was trained and tested only on males speakers, the RMS error for F1 was about 56 Hz. When tested and trained on females, the RMS error for F1 was about 76 Hz. However, as seen previously, the F1 RMS error for the entire database was about 68 Hz. For F2 and F3 the situation is reversed, with females performing better. Only in the case of F3 did the separate models yield better results.

F. FVQ Trackers

The FVQ trackers implemented for this paper did not perform as well as the FSD trackers. When using the entire database, the RMS errors were around 100 Hz, 155 Hz and 225 Hz for F1, F2 and F3, respectively (Figure 11(b)). As with the FSD trackers, the RMS errors tend to change very gradually as the threshold is varied, with the exception of the threshold extremes. Unlike the FSD trackers, there was an advantage to using separate models for males and females. For example, when using males only (Figure 11(a)), the RMS errors for F1 and F2 were around 75 Hz and 125 Hz, respectively, which is similar to the results for the baseline FSD tracker.

The FVQ trackers might be improved in two ways. First, the codebook size of 256 may be too small. The FVQ results on the entire

database are very similar to Kopec's (1986) results with a codebook of size 64. Since Kopec found performance to be strongly related to codebook size, a larger codebook should certainly help. Further improvement might also be made by increasing the amount of training data. An informal inspection of the codebook probability distributions suggests that some of the codebook symbols were not adequately represented in the training data.

Figure 12 illustrates an undesirable situation that can occur in the FVQ tracker. The spectrum of one codebook entry is shown in Figure 12(a). In this spectrum the second peak appears to be around 1600 Hz. In Figure 12(b) is the probability distribution for F2 when this symbol appears in the training data. There is indeed a high probability that F2 is around 1600 Hz, but there is also a significant probability that F2 is around 1200 Hz, which could easily cause a large error.

VI. Discussion

The only approach to formant tracking in the literature which has been quantitatively analyzed in an adequate fashion is that of Kopec (1986). All of Kopec's trackers used vector quantization and hidden Markov models as in the FVQ tracker described above. The speech material used in Kopec's study was a large connected digits database from Texas Instruments. Approximately 142,000 frames of this database were hand-marked, which is about four times as much as was used in the present paper. The codebooks used by Kopec were trained on about 250,000 frames from the TI database, or

about twelve times as much data as was used to train the FVQ tracker.

Table 3 contains a summary of the performance of the FSD and FVQ trackers along with two of Kopec's (1986) trackers and a simple peak-picking algorithm due to Markel (1972). The peak-picking algorithm required no training, and was tested on all 78 utterances from the CMU database. This algorithm scored fairly well on F1, but poorly on F2 and F3. The high percentage of large errors, particularly for F3, reveals the difficulty in accurately assigning formant labels to spectral peaks. The other four trackers were all evaluated using a threshold value of 0.5. Kopec's $S_{40}(1024)$ tracker used a codebook of size 1024 and quantization sizes of 50 Hz for F1 and 100 Hz for F2 and F3. The $S_{40}(64)$ tracker used the same quantization sizes with a codebook size of 64. The results for the FVQ and $S_{40}(64)$ trackers were quite similar for all three formants. Since the codebook for the FVQ tracker was trained on only about one-twelfth as much data as the $S_{40}(64)$ tracker, this may indicate that the FVQ codebooks were insufficiently trained. Overall, the best performance among the trackers listed in Table 3 belonged to the $S_{40}(1024)$ tracker. The FSD tracker was slightly better than $S_{40}(1024)$ for F1, but significantly worse for F3.

One major difference between the present study and that of Kopec (1986) lies in the phonetic composition of the databases used. The TI connected digits database was dialectically mixed, but contained only the set of eleven digits (one-nine, oh, and zero). Thus there were a number of vocalic phonemes, including l, y, ae, and oy, which were not represented in this data. In addition, the number of

different phonetic combinations was rather small. The CMU database used to train and test the FSD and FVQ trackers was phonetically more diverse than the TI database used by Kopec, and hence contained many more types of formant trajectories. On the other hand, nasals, which are in general rather difficult to track, occur with much greater frequency in the digits (about 12%) than in the CMU database (4.4%). Hence it is hard to say which task was more difficult.

Kopec (1986) found that separate models for males and females produced better results for the S_{40} trackers. This was also seen to be the case for the FVQ trackers. As Kopec pointed out, one possible explanation for this difference is that when males and females are combined, the composite distributions for some codebook entries may be roughly bimodal, as was illustrated in Figure 12. On the other hand, the FSD tracker did not give uniformly better results for separate models. Thus, the probability measure used by the FSD tracker appears to be more robust in that it is less likely to produce bimodal distributions, but it may also be more difficult to tune to a particular speaker or group.

The advantage of using a hidden Markov model to provide continuity constraints was demonstrated clearly in Figure 9. There are three basic issues involved with HMMs: evaluation, decoding, and training (Rabiner and Juang, 1986). For formant tracking, training can be handled by estimating probabilities using hand-marked tracks. In both the present paper and Kopec (1986), the forward-backward algorithm, which is a solution to the evaluation problem, was used to produce formant estimates. That is, the

problem was treated as one of evaluating an observation sequence to a HMM. Another approach is to consider the problem to be one of decoding the optimal state sequence of a HMM, for which the Viterbi algorithm can be used. Kopec (1985) found this method to be valid but not as flexible as the forward-backward method.

The performance of the baseline FSD tracker used in this study could be improved by allowing a more flexible configuration. The quantization size and number of features for each of the formants could be set independently, rather than using the same configuration for F1, F2 and F3. In particular, F3 would probably benefit from an increase in the number of features since it has the largest frequency range and the worst performance.

Although only LPC spectral values were used in this paper, there is nothing to prevent other features from being tried in the FSD approach. For example, some early experiments were performed using smoothed FFT values and spectral moments. The results were consistent but not as good as LPC values, so later tests used only LPC values. Another set of features which could be used in the FSD tracker is cepstral coefficients (Rabiner and Juang, 1986), which are commonly used in recognition systems. Another interesting idea is to try to augment the LPC spectral values with information from the time domain such as zero crossings. This might prove to be especially useful for building formant trackers intended for noisy applications, as in Niederjohn and Lahat (1985).

VII. Summary

The statistical approach to formant tracking proposed in this paper has been shown to perform well on a large database. This approach generalizes the work of Kopec, and provides a framework which will allow for additional features to be introduced.

References

Flanagan, J. L. (1956). "Automatic Extraction of Formant Frequencies from Continuous Speech," J. Acoust. Soc. Am., 28, 110-118.

Gray, R. M. (1984). "Vector Quantization," IEEE ASSP Magazine, 4, 4-29.

Itakura, F. (1975). "Minimum Prediction Residual Principle Applied to Speech Recognition," IEEE Trans., ASSP-23, 67-72.

Johnson, R. A. and Wichern, D. W. (1982). *Applied Multivariate Statistical Analysis,* Prentice-Hall, Englewood Cliffs, N. J.

Kopec, G. (1985). "Formant Tracking using hidden Markov models," IEEE Int. Conf. Acoust., Speech, Signal Processing, Tampa, FL, 1113-1116.

Kopec, G. (1986). "Formant Tracking Using Hidden Markov Models and Vector Quantization," IEEE Trans., ASSP-34, 709-729.

Linde, Y., Buzo, A., and Gray, R. (1980). "An Algorithm for Vector Quantizer Design,", IEEE Trans., Com-28, 84-95.

Lloyd, S. P. (1957). "Least Squares Quantization in PCM," Bell Laboratories Technical Note; reprinted in IEEE Trans., IT-28, 129-137, 1982.

Markel, J. D. (1972). "Digital Inverse Filtering A New Tool for Formant Trajectory Estimation," IEEE Trans., AU-20, 129-137.

McCandless, S. S. (1974). "An Algorithm for Automatic Formant Extraction Using Linear Prediction Spectra," IEEE Trans., ASSP-22, 135-141.

Niederjohn, R. J. and Lahat, J. (1985). "A Zero-Crossing Consistency Method for Formant Tracking of Voiced Speech in High Noise Levels," IEEE Trans., ASSP-33, 349-355.

Rabiner, L. R. and Juang, B. H. (1986). "An Introduction to Hidden Markov Models," IEEE ASSP Magazine, 1, 4-16.

Rabiner, L. R. and Schafer, R. W. (1976). *Digital Processing of Speech Signals,* Prentice Hall, Englewood Cliffs, N. J..

Schafer, R. W. and Rabiner, L. R. (1970). "System for Automatic Formant Analysis of Voiced Speech," J. Acoust. Soc. Am., 47, 634-648.

Suzuki, J., Kadokawa, Y., Nakata, K. (1963). "Formant Frequency Extraction by the Method of Moment Calculations," J. Acoust. Soc. Am., 35, 1345-1353.

	Males			Females		
	F1	F2	F3	F1	F2	F3
Minimum	176	630	1396	245	816	1684
Maximum	942	2479	3427	1139	2978	3839
Mean	496	1398	2453	564	1640	2724
Std. Dev.	128	382	400	184	457	373

Table 1. Statistics for hand-edited formant tracks.

Interval	Midpoint	$D_i(0)$	$p_i(t)$	$\tilde{p}_i(t)$
q_0		113.6	0.177	0.298
q_1	100	1501.3	0.000	0.000
q_2	300	112.3	0.330	0.105
q_3	500	112.0	0.395	0.597
q_4	700	114.8	0.097	0.000
q_5	900	142.3	0.000	0.000
q_6	1100	266.6	0.000	0.000

$$\gamma(t) = 0.105 + 0.597 = 0.702$$

$$\tilde{F}(t) = [(0.105)(300) + (0.597)(500)] / 0.702 = 470$$

Table 2. Example of computations performed in FSD tracker using the mean feature vectors and sample token in Figure 2. $D_i(0)$ is the distance from the sample token to q_i; $p_i(t)$ is the probability that the sample token came from q_i; $\tilde{p}_i(t)$ is the probability that the sample token came from q_i after applying the forward-backward algorithm as a continuity constraint; $\gamma(t)$ is the probability that F1 is present in the sample token; $\tilde{F}(t)$ is the final estimate for F1 for the sample token.

	RMS Error			% Large Errors		
	F1	F2	F3	F1	F2	F3
Markel	92	274	503	1.2	5.4	21.3
$S_{40}(1024)$	72	93	150	1.6	0.4	1.5
$S_{40}(64)$	98	155	235	3.2	1.3	4.4
FSD	68	122	214	0.7	0.4	3.9
FVQ	99	154	224	3.7	1.1	4.8

Table 3. Comparison of FSD and FVQ trackers to other formant trackers. Markel is a peak-picking algorithm tested on the CMU database; $S_{40}(1024)$ and $S_{40}(64)$ are trackers used by Kopec with codebook sizes of 1024 and 64, respectively, and tested on the TI connected digits database.

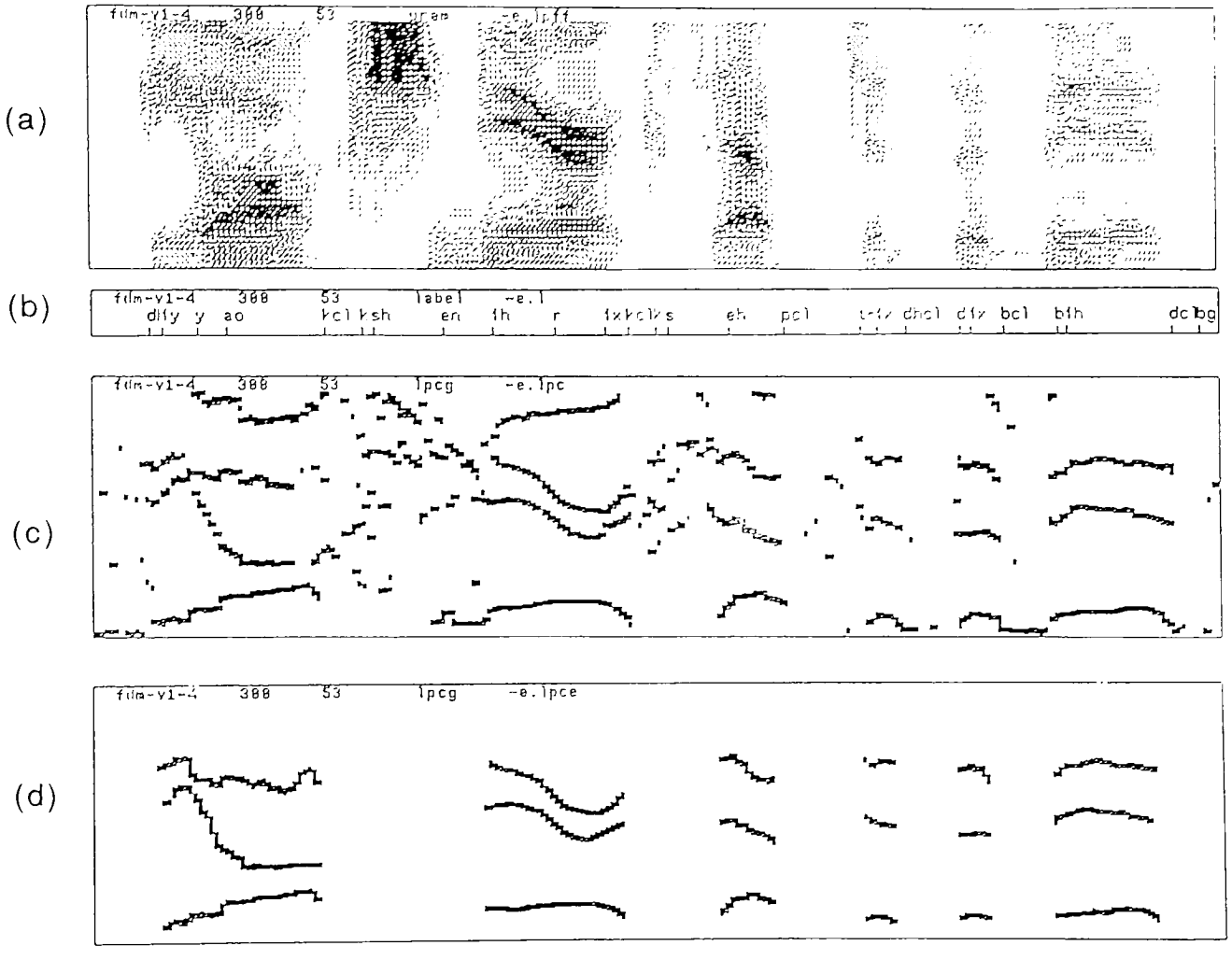


Figure 1. Example of the process of hand-editing formant tracks of the utterance "The auctioneer accepted the bid". (a) LPC spectrogram; (b) phonetic transcription; (c) plot of raw LPC peaks; (d) final hand-edited formant tracks.

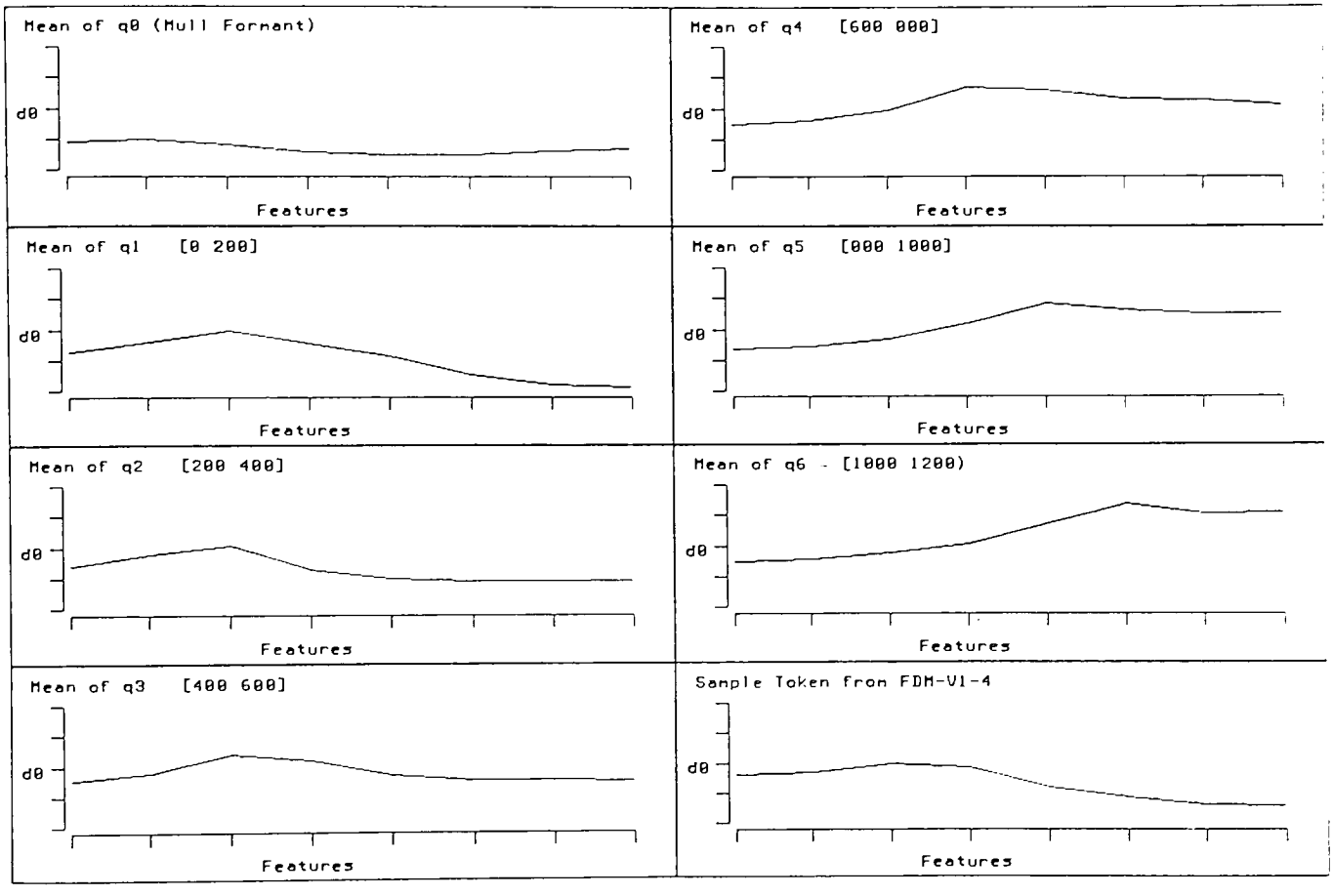


Figure 2. Mean feature vectors of intervals q_0 q_6 for an FSD tracker, together with a sample token.

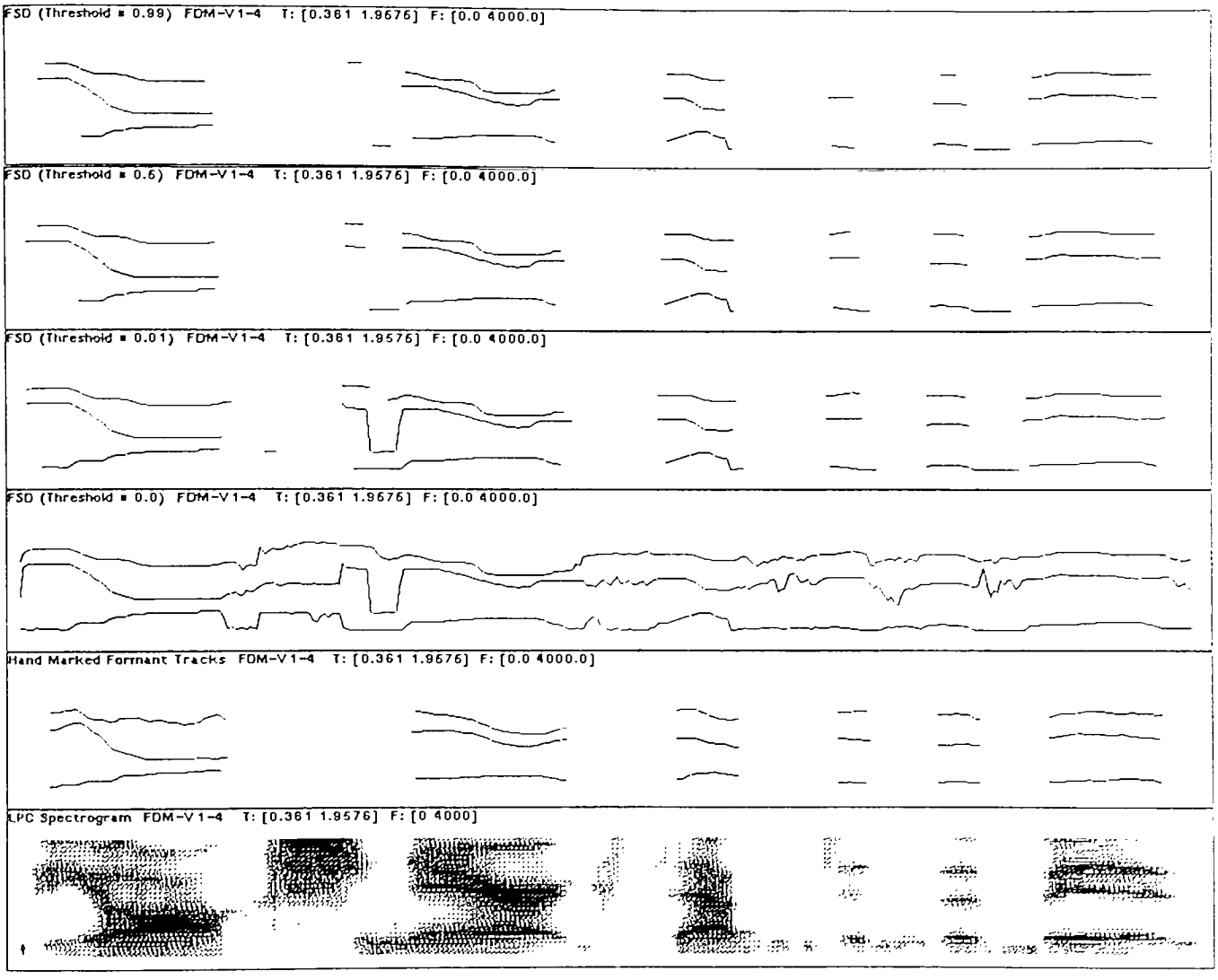
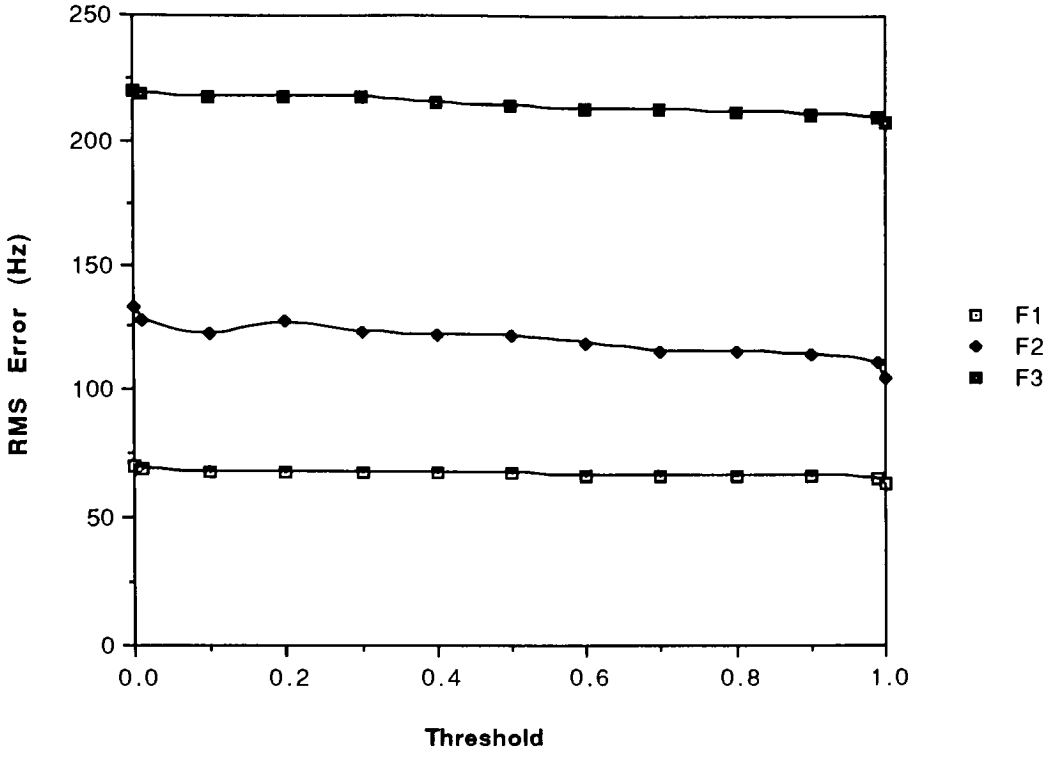
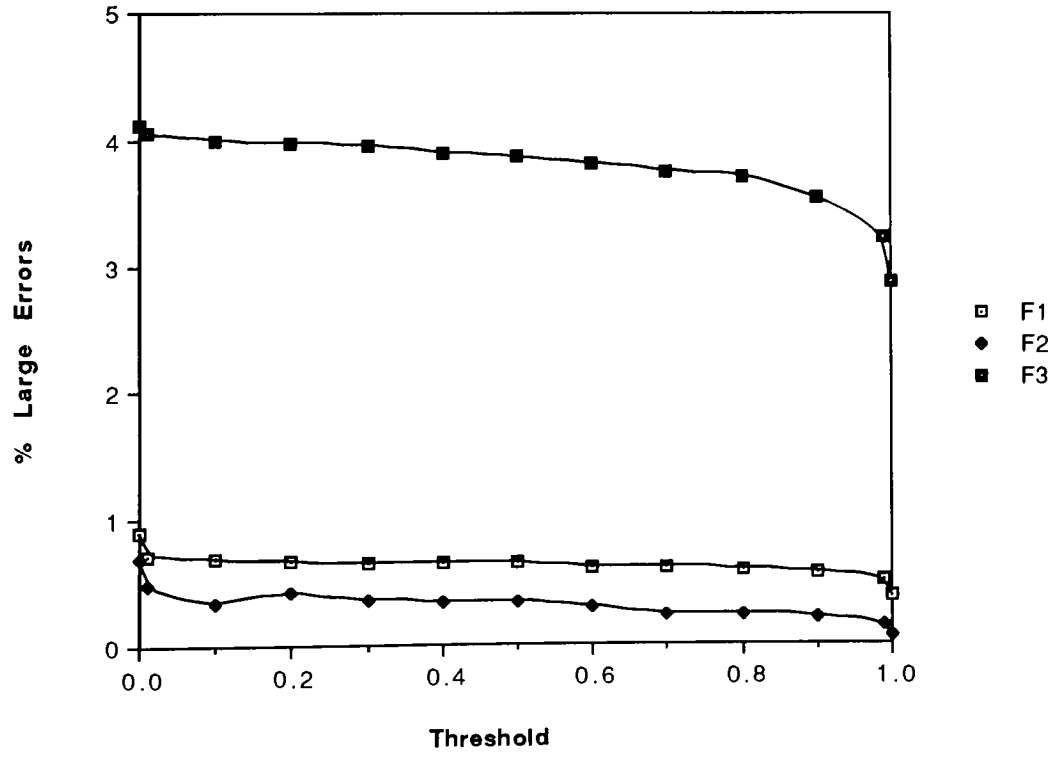


Figure 3. Example of FSD results using threshold values of 0.0, 0.01, 0.5 and 0.99.

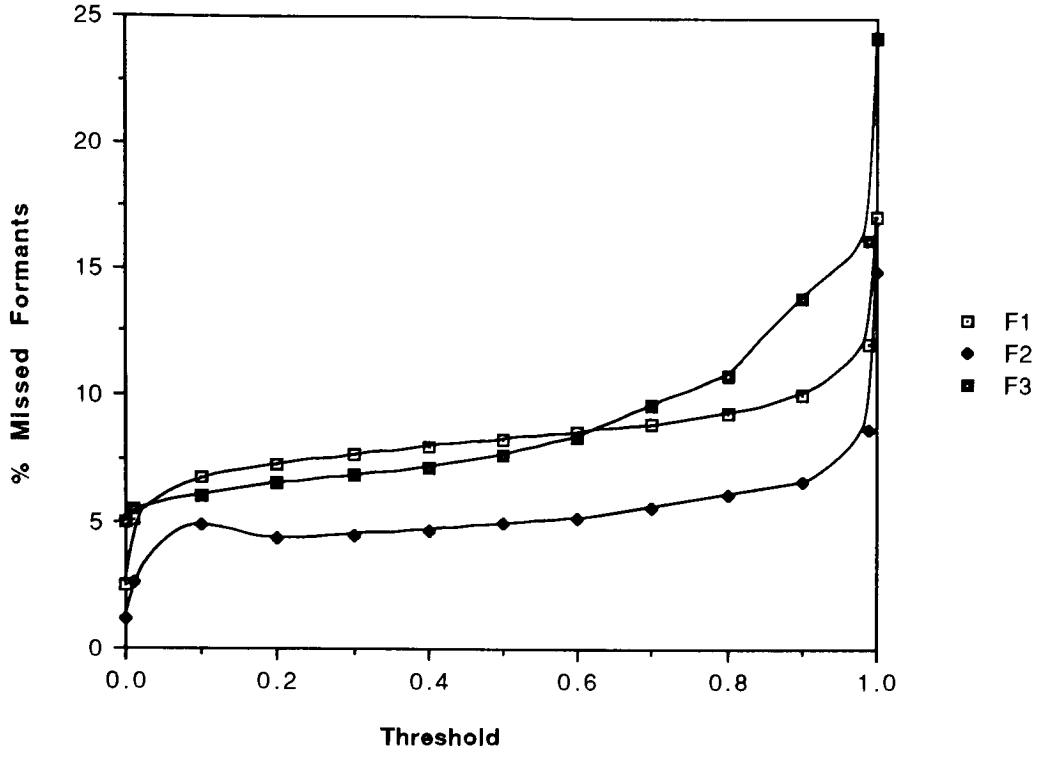


(a)

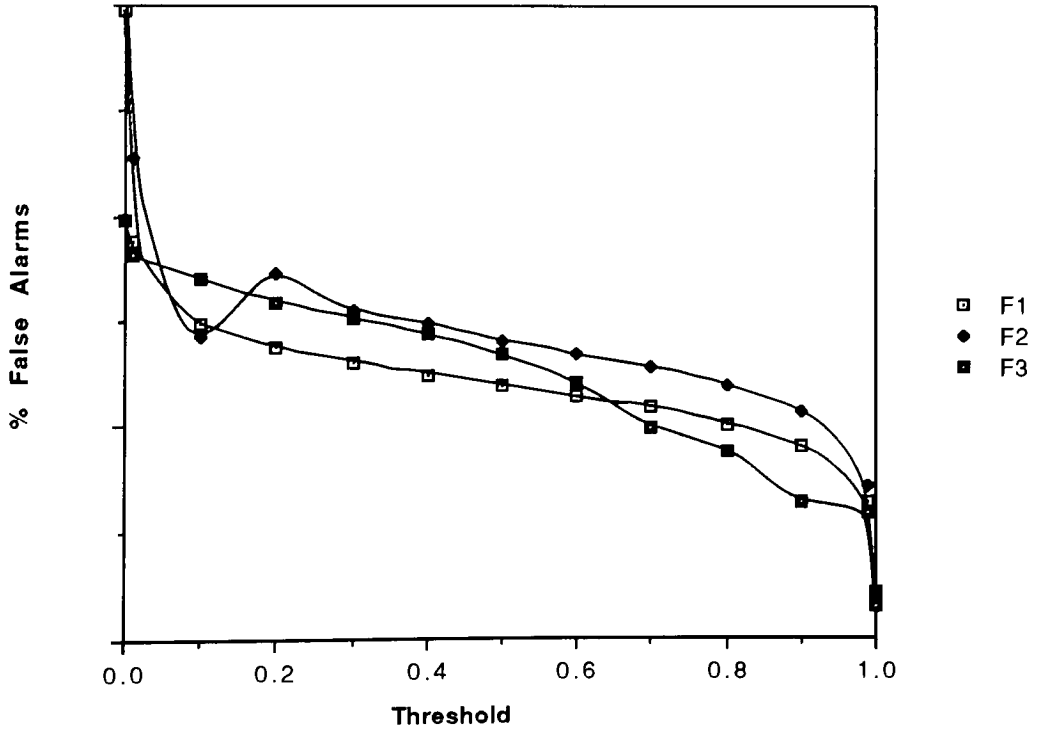


(b)

Figure 4. Quantitative results for baseline FSD tracker. (a) RMS error as a function of the threshold value; (b) Percentage of frames with large errors.

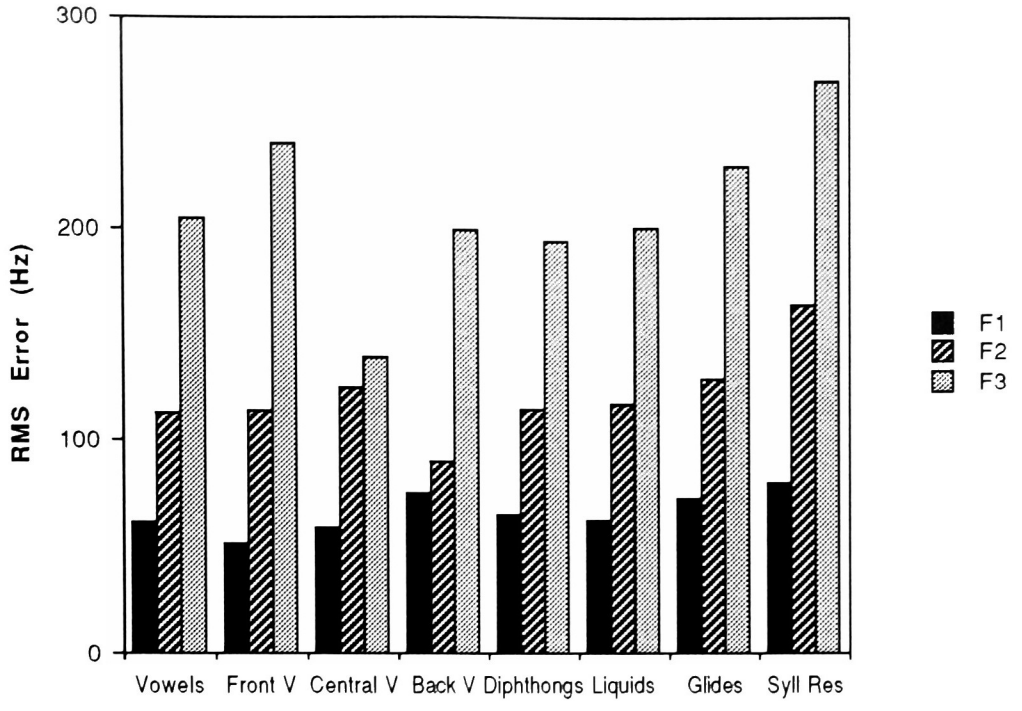


(a)

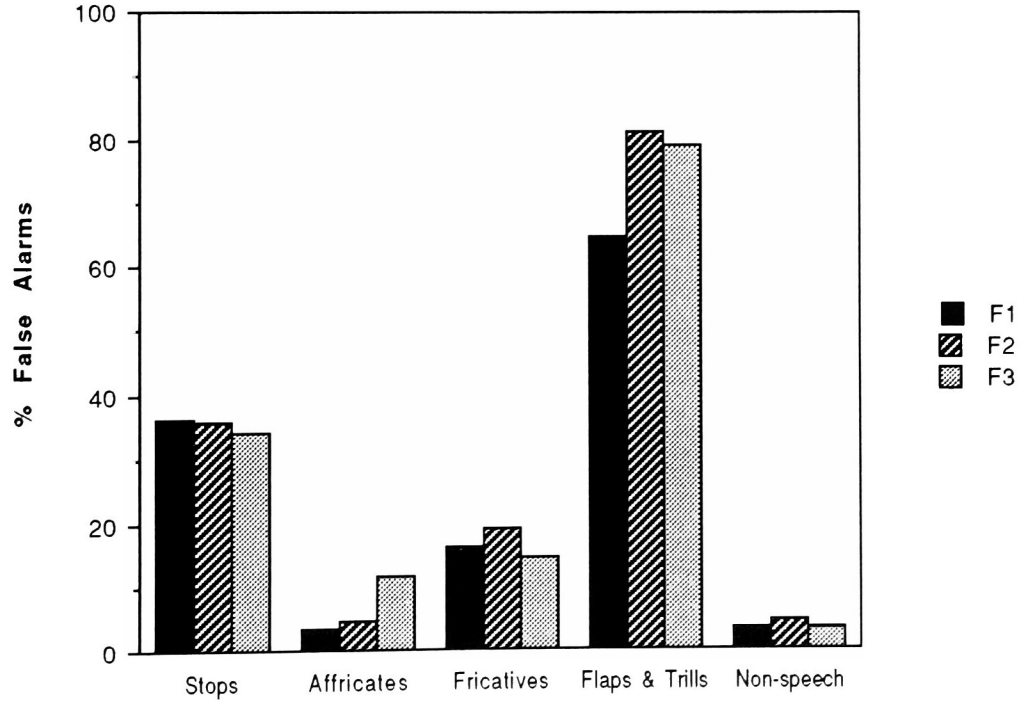


(b)

Figure 5. (a) Percentage of missed formants in baseline FSD tracker; (b) Percentage of false alarms in baseline FSD tracker.



(a)



(b)

Figure 6. (a) RMS error for baseline FSD tracker in labeled phonetic classes; (b) Percentage of false alarms in unlabeled phonetic classes.

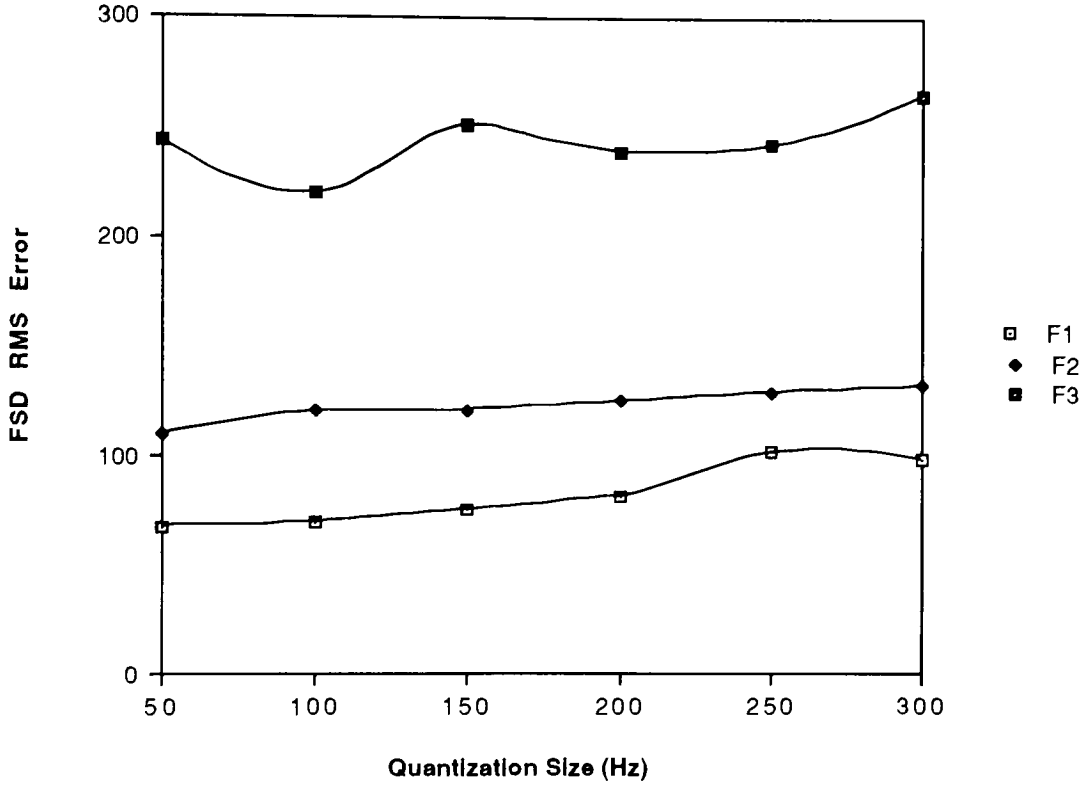
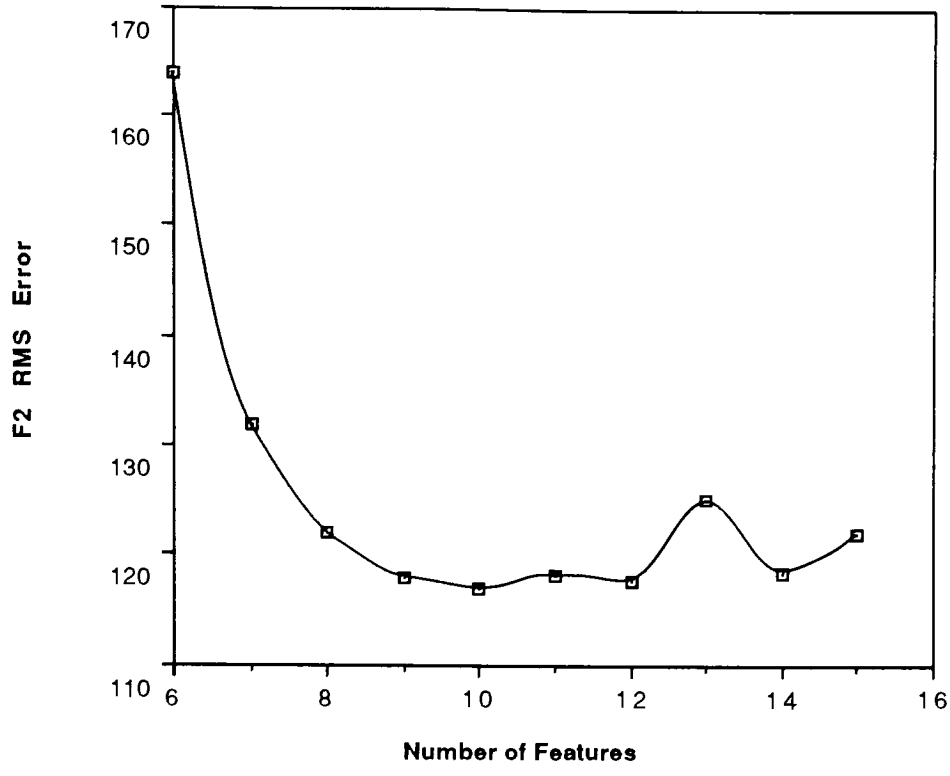
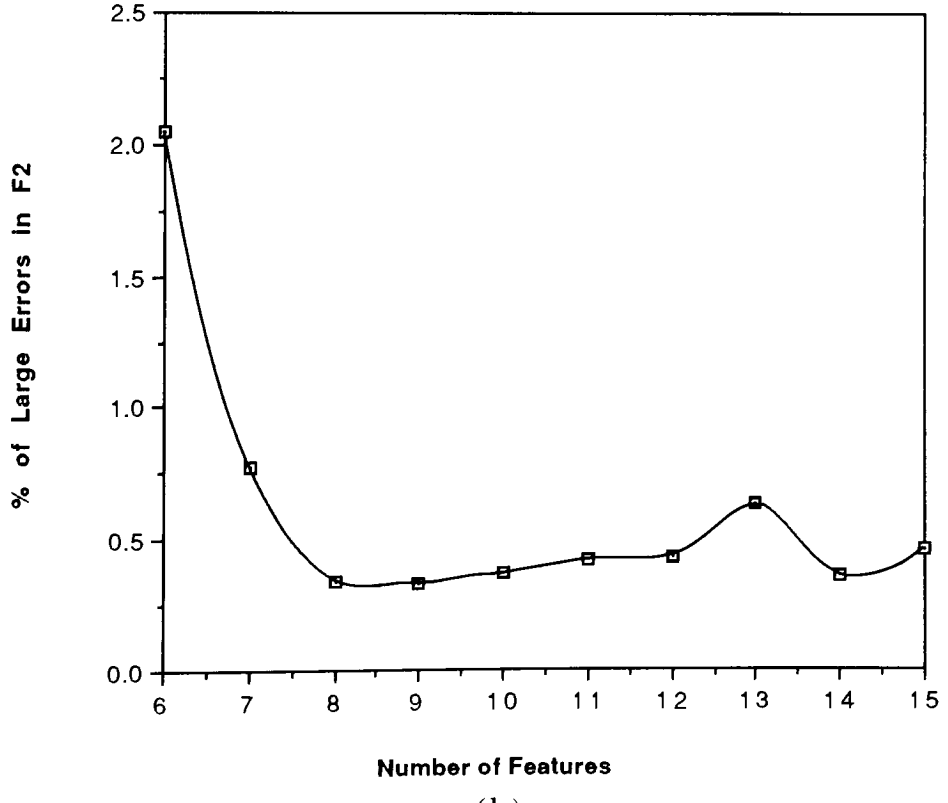


Figure 7. Role of quantization size in FSD trackers.

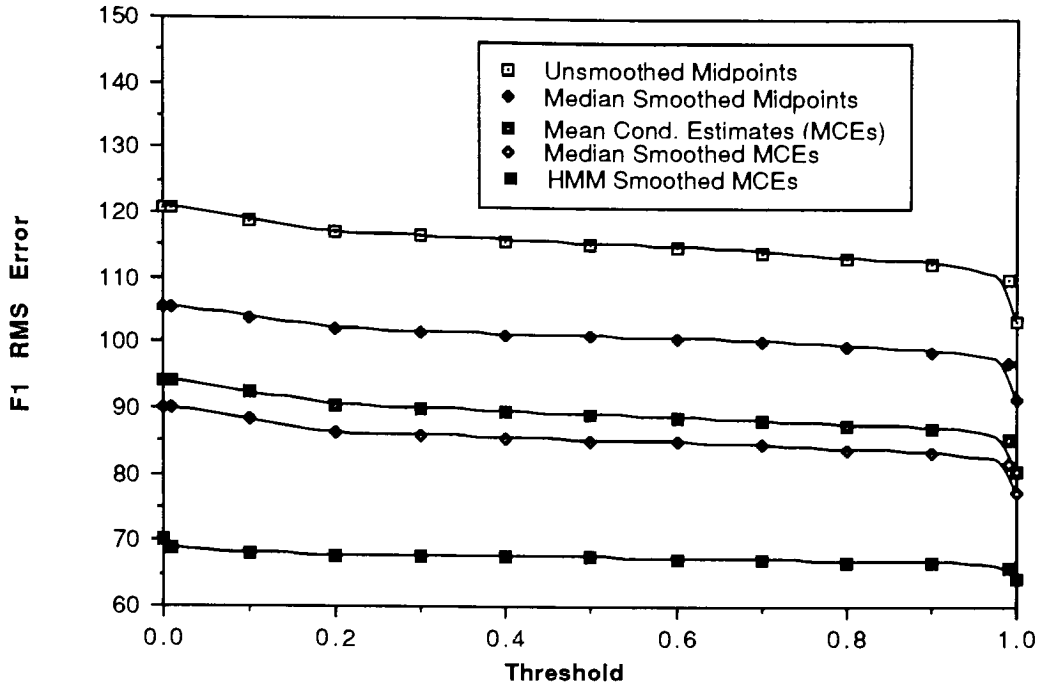


(a)

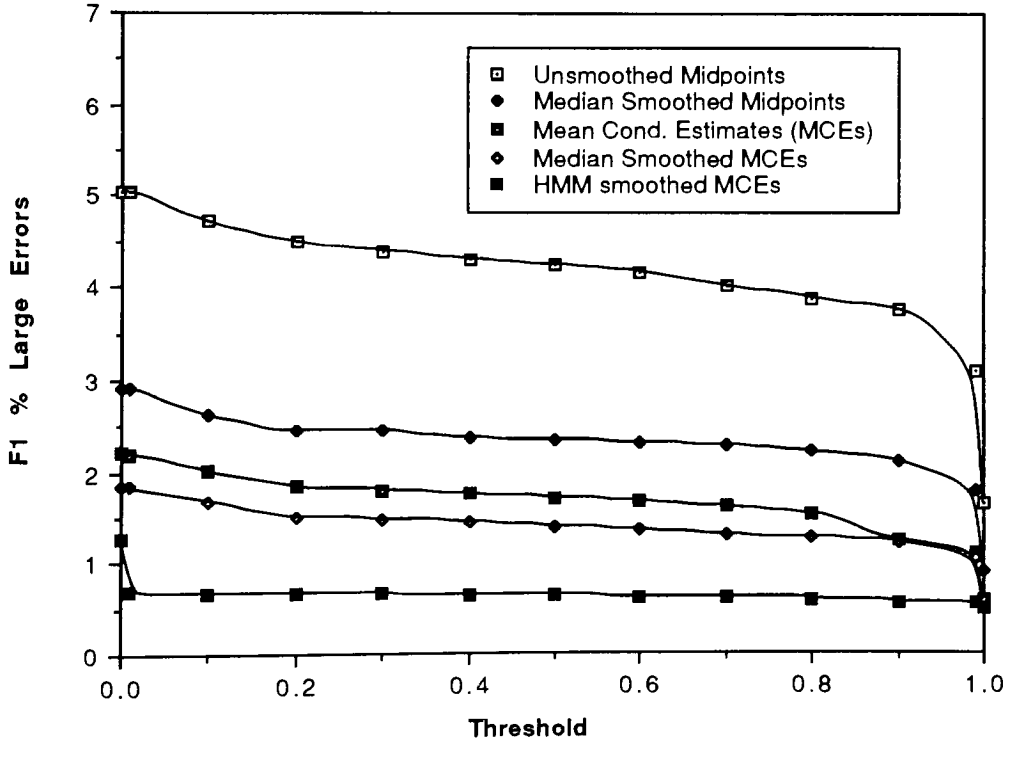


(b)

Figure 8. Role of number of features in FSD trackers. (a) RMS error in F2; (b) Percentage of large errors in F2.



(a)



(b)

Figure 9. Role of continuity constraints in FSD trackers. (a) RMS error in F1; (b) Percentage of large errors in F1.

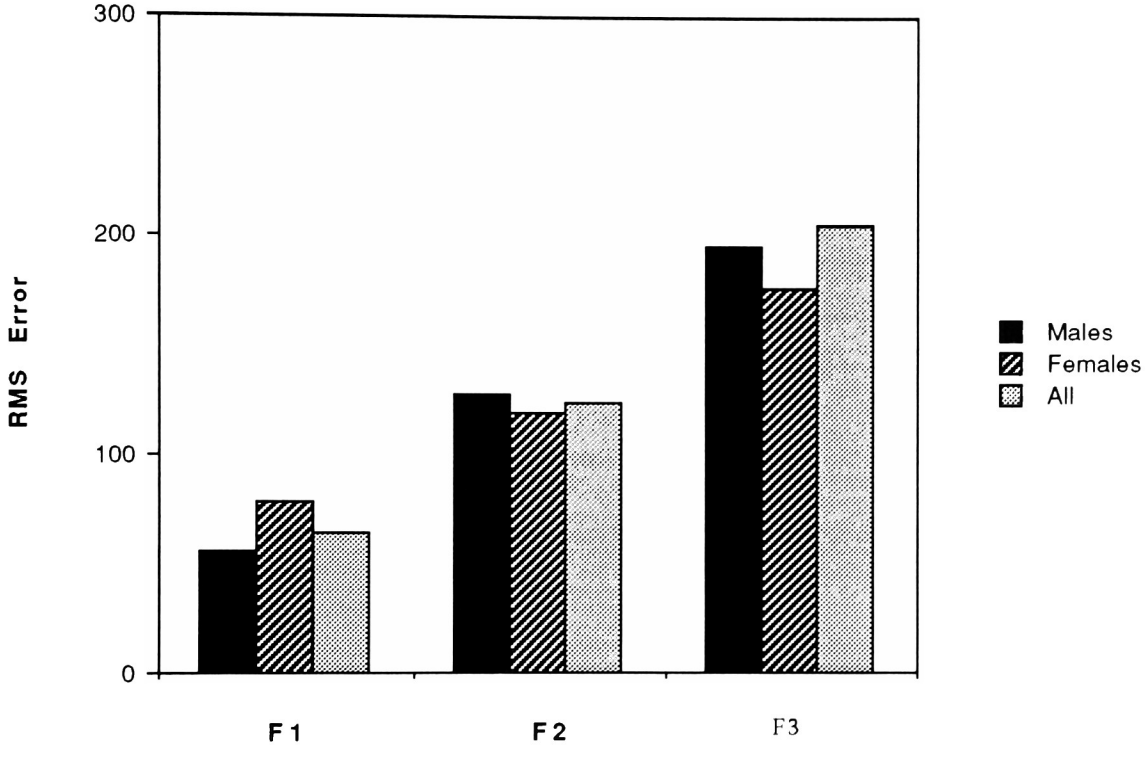
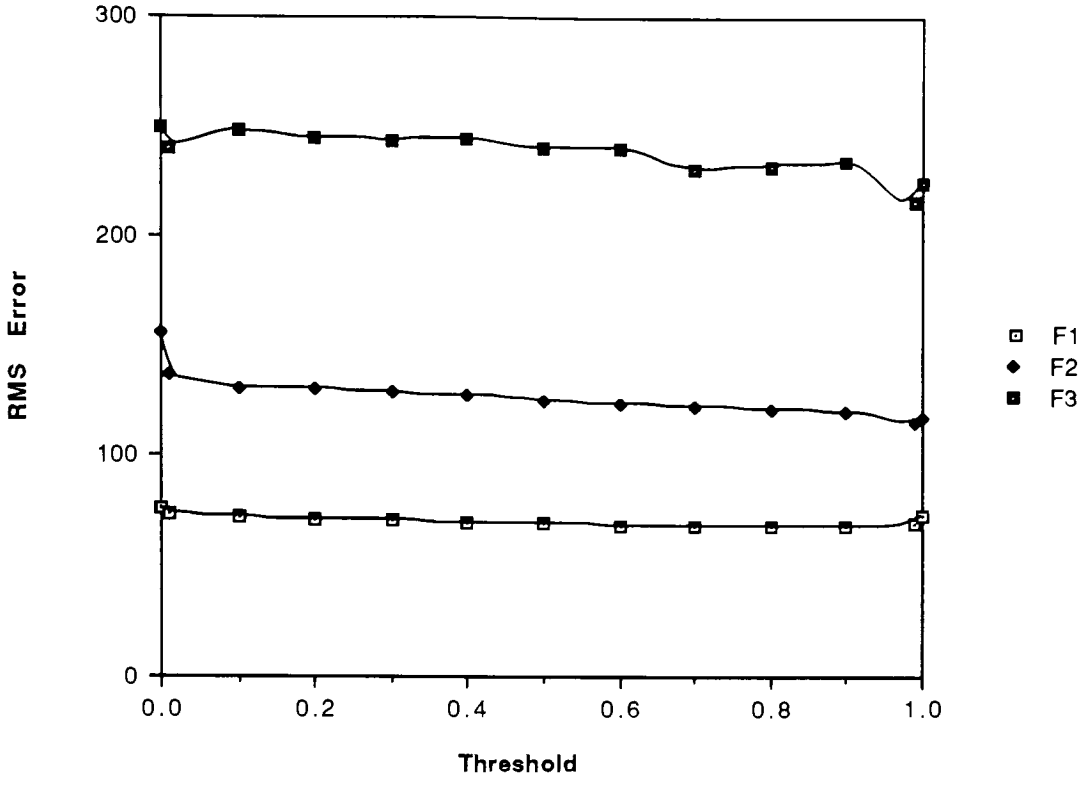
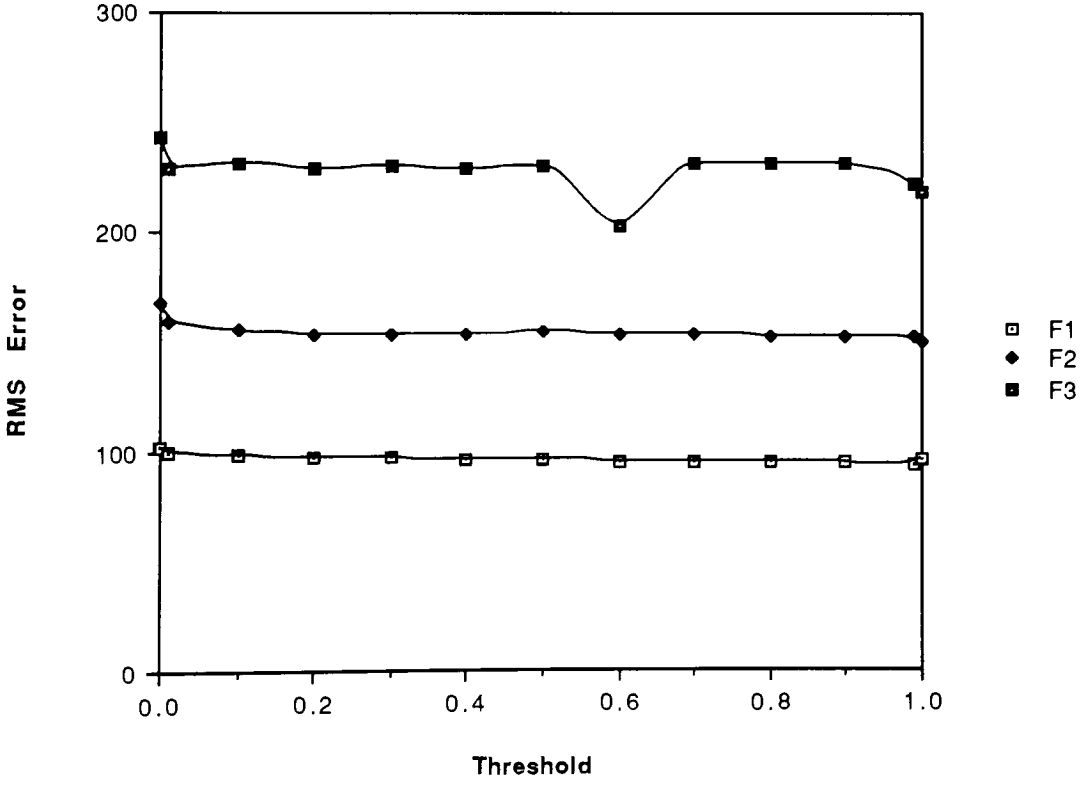


Figure 10. Results of using separate models for males and females in the baseline FSD tracker.

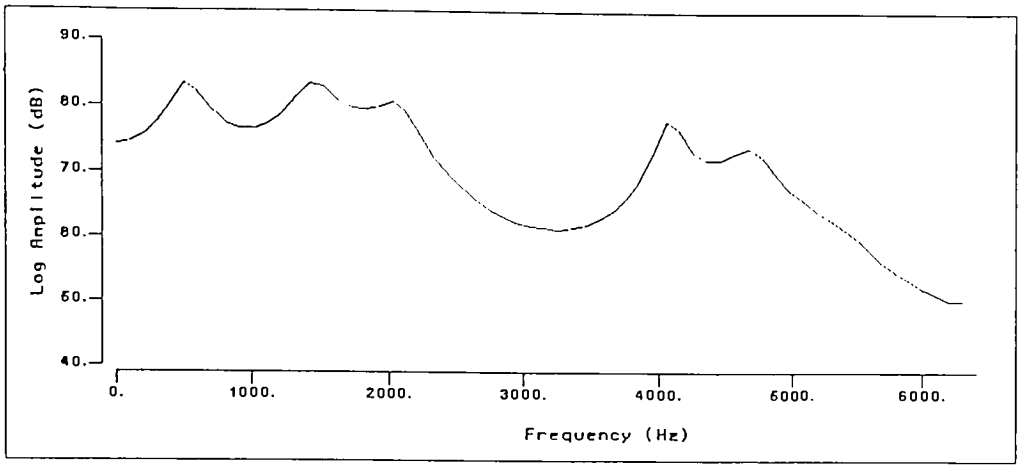


(a)

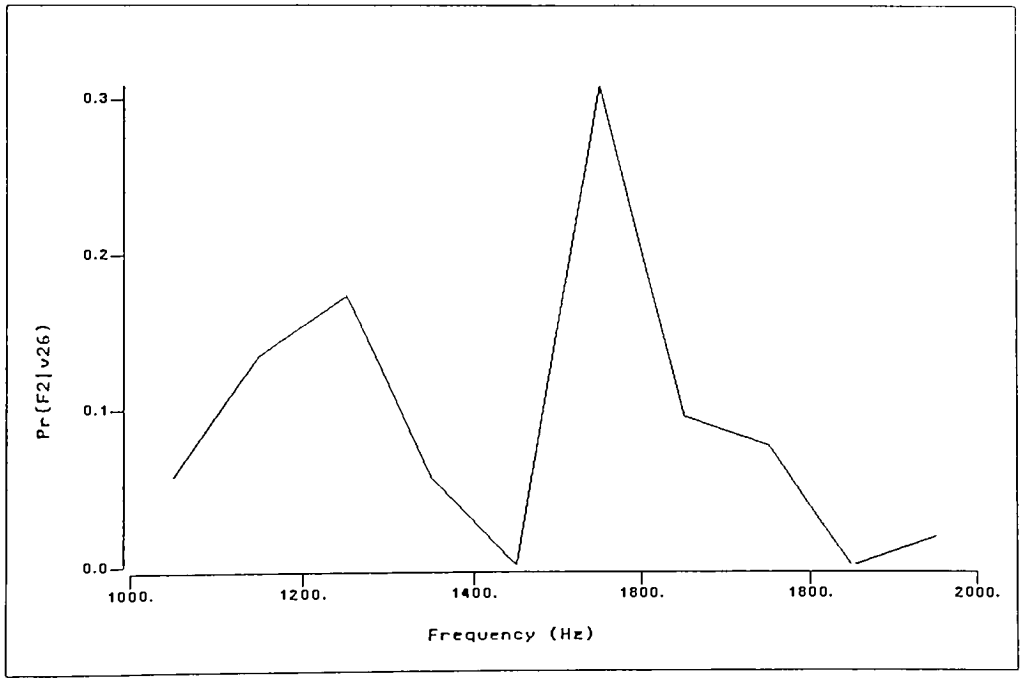


(b)

Figure 11. RMS error of FVQ tracker trained and tested on: (a) males only; (b) entire database.



(a)



(b)

Figure 12. (a) Spectrum of one codeword from FVQ codebook; (b) Probability distribution for F2 given this codeword.

Appendix A. Database Details

The database used in this thesis consisted of 78 utterances from a large continuous speech database obtained from Carnegie-Mellon University. The following is a list of the specific utterances used.

Training Set (39 utterances):

fdm-v1-4	fhg-v3-1	fjm-v4-2	fpr-v5-1
fdm-v1-5	fhg-v3-2	fjm-v4-5	fpr-v5-3
fdm-v1-6	fhg-v3-5	fjm-v4-8	fpr-v5-6
fdm-v1-10	fhg-v3-7	fjm-v4-9	fpr-v5-7
	fhg-v3-9		fpr-v5-9
	fhg-v3-10		fpr-v5-10
mdc-v1-2	mec-v5-1	mjg-v4-2	mjm-v2-2
mdc-v1 3	mec-v5-2	mjg-v4-4	mjm-v2-4
mdc-v1-5	mec-v5-4	mjg-v4-10	mjm-v2-7
mdc-v1-6	mec-v5-6		mjm-v2-9
mdc-v1-9	mec-v5-7		
	mec-v5-9		
	mec-v5-10		

Test Set (39 Utterances):

fdm-v1 1	fhg-v3-3	fjm-v4-1	fpr-v5-2
fdm-v1-2	fhg-v3-4	fjm-v4-3	fpr-v5-4
fdm-v1 3	fhg-v3-6	fjm-v4-4	fpr-v5-5
fdm-v1 7	fhg-v3-8	fjm-v4-6	fpr-v5-8
fdm-v1-8		fjm-v4-7	
fdm-v1-9		fjm-v4-10	
mdc-v1-4	mec-v5-3	mjg-v4-3	mjm-v2-1
mdc-v1 7	mec-v5-5	mjg-v4-5	mjm-v2-3
mdc-v1-8	mec-v5-8	mjg-v4-6	mjm-v2-5
mdc-v1 10		mjg-v4-7	mjm-v2-6
		mjg-v4-8	mjm-v2-8
		mjg-v4-9	mjm-v2-10

The orthographic transcriptions of these utterances are as follows.

v1-1. They toiled in the fields all day long.
v1-2. The angry crowd pushed open the door.
v1-3. He left them with a reason to believe in themselves.
v1-4. The auctioneer accepted the bid.
v1-5. The yellow rose is the most beautiful of all flowers.
v1-6. The child lured the rabbit into the cage.
v1-7. Put the damp towel over your head for protection.
v1-8. Always look before you leap.
v1-9. While you were away, we opened the package.
v1-10. The acrobat walked the tightrope.

v2-1. He bought a new clock at the Tick-Tock Shop.
v2-2. She has a twenty percent hearing loss.
v2-3. She allowd the boy to eat the cookie.
v2-4. The old hound was unenthused at the sight of the cat.
v2-5. The owl swooped down upon the mouse.
v2-6. The photograph proved he was guilty.
v2-7. A youth has many lessons to learn.
v2-8. If it never rained, we'd never grow.
v2-9. Where in the world is the Fountain of Youth?
v2-10. The handsome wool jacket was an oxford gray.

v3-1. A cooked yam is a tasty sweet potato.
v3-2. He recorded a new album with his younger partner.
v3-3. Take Cloey to the show.
v3-4. We fell for it hook, line, and sinker.
v3-5. She won a blue ribbon at the county fair.
v3-6. He was covered with soot from head to foot.
v3-7. Are you aware of the good things in life?
v3-8. Outside, the nights are only colder.
v3-9. The old woman rocked away the hours.
v3-10. He had a deep gouge over his left eye.

v4-1. No one aroused his curiosity like Eunice.
v4-2. Annoying a wild boar is insane.
v4-3. Get out before it's too late.
v4-4. The girl had a collection of wooden dolls.
v4-5. One is the loneliest number.
v4-6. The bull chased the clown from the arena.
v4-7. The thirsty girl rehearsed her lines.

v4-8. The little pooch wagged his tail.
v4-9. The hoodlum was full of malice.
v4-10. Why not make a white oak chair?

v5-1. You rang?
v5-2. A loud alarm can be an eye-opener.
v5-3. We arranged to look at the young animal.
v5-4. They stashed the loot in the pumpkin patch.
v5-5. Toast and jam tastes good for breakfast.
v5-6. The robot was programmed to clean house.
v5-7. The sauerkraut boiled till it burned.
v5-8. I am amused at the cowboy's style.
v5-9. Awhile ago, we knew very little.
v5-10. Try to remember the joyous occasions.

The phonetic classes used in the analysis of the FSD tracker (Figure 6) were as follows.

Front Vowels: iy, ih, eh, ae, e

Central Vowels: ix, ax, ah

Back Vowels: ux, uw, uh, ao, aa, o, oe

Diphthongs: ey, ay, oy, aw, ow

Retroflex Vowels: er, axr

Liquids: l, r

Glides: y, w

Syllabic Resonants: el, em, en, eng

Stops: p, b, t, d, k, g, m, n, ng, q

Affricates: ch, jh

Fricatives: f, v, th, dh, s, z, sh, zh, hh, hv

Flaps and Trills: dx, nx, lx, rx

Non-speech: bg, pau, sil, ns, h#, #h

Missing Page

Missing Page

APPENDIX B. Program Summaries

I. Fourier Analysis

II. LPC Analysis

III. Vector Quantization

IV. Hidden Markov Models

V. Hand-editing Formant Tracks

VI. Formant Trackers

VII. Quantitative Analysis of Formant Trackers

VII. Miscellaneous Routines

I. Fourier Analysis

The Discrete Fourier Transform of a sequence {x(0), x(1), ..., x(N-1)} is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{2\pi i k n/N}$$

Similarly, the inverse Discrete Fourier Transform is defined as

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{-2\pi i k n/N}$$

The Fast Fourier Transform (FFT) is a family of algorithms which compute the DFT in less than $O(N^2)$ steps. The basic idea behind all FFT algorithms is to remap the indices in such a way as to reduce the number of operations required. One commonly used FFT algorithm is the radix 2 decimation-in-time FFT algorithm, which applies to sequences of length 2^N. This algorithm consists of $\log_2 N$ stages, each of which contains O(N) "butterflies" of the form

$$x1' = x1 + w*x2$$
$$x2' = x2 \quad w*x1$$

where w is a "twiddle factor" $e^{j\omega}$. Thus the complexity of this algorithm is O(NlogN).

Floating-point FFTs: *fft*(), *inverse_fft*()

These routines compute the FFT using the radix 2 decimation-in-time algorithm described above. *Fft*() takes a sequence of floats x[0],...,x[n-1] and returns a sequence y[0],...,y[n-1] of complex values. The input sequence is left unchanged. *Inverse_fft*() performs the inverse FFT on the sequence x[0],...,x[n-1] in-place.

Usage:

```
void fft (x, y, n)
        float       *x;         /* input sequence */
        complex     *y;         /* transform sequence */
        int          n;         /* length of sequence */
```

```
void inverse_fft (x, n)
     complex     *x;          /* sequence */
     int          n;          /* length of sequence */
```

Integer FFTs: *int_fft*(), *int_complex_fft*(), *int_inverse_fft*()

These functions compute the FFT and inverse FFT using an integer version of the radix 2 decimation-in-time algorithm. A table of 16-bit cosine values is used. The transformed values are scaled down by a factor of 2 during each stage to prevent overflow. *Int_complex_fft*() can be used to compute the FFTs of two (real) sequences simultaneously by placing one sequence in the real part of the input array and the second sequence in the imaginary part.

Usage:

```
int_fft (x, y, n)
     short          *x;      /* input sequence */
     complex_int    *y;      /* transform */
     int            n;       /* length of the sequence */

void int_complex_fft (x, n)
     complex_int    *x;      /* sequence */
     int             n;      /* length of the sequence */

void int_inverse_fft (x, n)
     complex_int    *x;      /* sequence */
     int             n;      /* length of the sequence */
```

b2f

B2f read a raw speech file and produces an output file which contains the short-time FFT spectra of the signal. Each frame of the input signal is Hamming windowed and processed using *int_complex_fft*(). The window size is typically 128 for wide-band spectra and 256 or 512 for narrow-band spectra. The output file is a binary file with window-size/2 shorts per spectrum.

Usage: *b2f* {arguments}

```
arguments:
          Filename (required)
     -i   InputExtension [.b]
     -o   OutputExtension [.f]
     -w   WindowSize [128]
```

Number of points in an analysis window; at a sample rate of 12.8 kHz, 128 points is 10 msec.

-s ShiftSize [64]

Determines the frame period; at 12.8 kHz, 64 points is 5 msec.

l LogOutput

When this flag is set, spectral values will be given as $1024*\log_2|X(e^{j\omega}|$

h Pre-emphasize

When this flag is set, the signal will be pre-emphasized using a filter of the form y[n] = x[n] 0.95 * x[n-1].

II. Linear Predictive Coding

The basic all-pole model of speech production can be expressed using z-transforms as

$$X(z) = \frac{E(z)}{A(z)}$$

where X(z) is the z-transform of the speech signal, E(z) is the z-transform of the source (impulsive or noise), and A(z) is an all-zero filter. In the time domain this can be written as

$$e(n) = x(n) * a(n) = x(n) + \sum_{i=1}^{N} a_i x(n-i)$$

If we define

$$\tilde{x}(n) = x(n) \quad e(n) = \sum_{i=1}^{N} a_i x(n-i)$$

we obtain a predictor $\tilde{x}(n)$ of x(n) expressed as a linear combination of the p previous samples. In this interpretation e(n) can be thought of as the error between the actual sample x(n) and the predicted sample $\tilde{x}(n)$. The predictor coefficients a_i thus can be derived by minimizing the mean-squared error $\sum e^2(n)$. A number of different

techniques have been developed to solve this problem. The autocorrelation method leads to the system of equations

$$\sum_{i=1}^{N} a_i R_{n-i} \quad j=1,2,\ldots,p$$

where

$$R_i = \sum_{i=1}^{N} x(n)x(n-i)$$

If this system is written in matrix form as $\mathbf{R}a = \mathbf{0}$, it can be seen that the matrix $\mathbf{R}$ is symmetric and that the diagonal elements are all the same. This type of matrix is known as a symmetric Toeplitz matrix, and admits a very efficient recursive solution known as the Durbin-Levinson algorithm. This procedure can be stated as follows:

1. Initialize $a_0 = 1$, $E_0 = R_0$ and $i = 1$.
2. Compute the reflection coefficient k_i.

$$k_i = (R_i \quad \sum_{j=1}^{N} a_j(i-1)R_{i-j}\ ) \ / \ E_{i-j}$$

3. Update the predictor coefficients and error.

$$a_i(i) = k_i$$
$$a_j(i) = a_j(i-1) \quad k_i a_{i-j}(i-1)$$
$$E_i = (1-k_i^2)E_{i-j}$$

4. Repeat steps 2 and 3 until $i = p$.

5. Set $a_j = a_j(p)$ for $1 \leq j \leq p$, and $\sigma = E_p$.

Data Structures for LPC Analysis

Most of the information involved in analyzing a single frame using floating-point linear prediction routines is encapsulated in the structure *lpc_model*.

```
typedef  struct
{
        int             order;                  /*  p  */
        int             window_length;          /*  N  */
        float           gain;                   /*  σ  */
        float           *lpc_coeff;             /*  a0,....,ap  */
        float           *autocorr_coeff;        /*  R0,....,Rp  */
}   lpc_model;
```

For integer routines the structure int_lpc_model is used. This differs from lpc_model only in that the floating-point values for the gain and coefficients are replaced by scaled integer values.

Durbin_Levinson()

This function implements the Durbin-Levinson method described above.

Usage:

```
void  Durbin_Levinson  (x,  model)
        short           *x;
        lpc_model       *model;
```

lpc_spectrum()

This routine computes the spectrum for an LPC model,

$$A(e^{j\omega}) = \sum_{k=0}^{p} a(k)e^{kj\omega}$$

by taking the FFT of the zero-padded sequence

$$1, a_1, a_2, ..., a_p, 0, 0, ..., 0$$

Note that the spectrum length is independent of the window length used to compute the LPC model.

Usage:

```
void lpc_spectrum (model, N, spec, scale, logged)
      lpc_model   *model;
      int         n;
      float       *spec;
      double      scale;
      boolean     logged;
```

int_Durbin_Levinson()

This is an integer implementation of the Durbin-Levinson algorithm. The autocorrelation coefficients are normalized so that R[0] is 32767 (1b15 form). The LPC coefficients have been empirically observed to be less than 8 in magnitude, so they are expressed in 4b12 form (12 bits to the right of the binary point).

Usage:

```
void int_Durbin_Levinson (x, model)
      short           *x;
      int_lpc_model   *model;
```

int_lpc_spectrum()

This routine computes an LPC spectrum from an integer LPC model using an integer FFT of the sequence

$$1, a_1, a_2, ..., a_p, 0, 0, ..., 0$$

where the coefficients a(i) are in 4b12 form. A value of 1 is returned if overflow is encountered.

Usage:

```
void int_lpc_spectrum (model, N, spec, scale, logged)
      int_lpc_model   *model;
      int             n;
      short           *spec;
      double          scale;
      boolean         logged;
```

b2lpc

B2lpc reads a raw speech file and produces a sequence of records which contain the frequencies and amplitudes of the peaks in the

short-time LPC spectra. The frequency and amplitude values are found using a three point quadratic interpolation wherever a peak occurs. The output is an LPC peak file, which is a binary file in which the first long (32-bit integer) contains the record size in bytes, and each record contains a fixed number (typically 7) of [frequency amplitude] pairs, both elements of which are 16-bit integers.

Usage: *b2lpc* {arguments}
arguments:

	Filename (required)
-i	InputExtension [.b]
-o	OutputExtension [.lpc]
-w	WindowSize [256]
-s	ShiftSize [64]
-z	SpectrumLength [128]
-p	NumberOfPoles [14]
n	MaxNumberOfPeaks [7]
l	LinearAmplitudes
h	NoPre-emphasis

b2lps

B2lps computes a sequence of LPC spectra from a raw speech file, using integer LPC routines. The output is a binary file containing SpectrumLength shorts per spectrum.

Usage: *b2lps* {arguments}
arguments:

	Filename (required)
-i	InputExtension [.b]
-o	OutputExtension [.lps]
-w	WindowSize [256]
-s	ShiftSize [64]
-p	NumberOfPoles [14]
-z	SpectrumLength [64]
-l	LinearAmplitudes
h	NoPre-emphasis

III. Vector Quantization

According to information theory, the best way to encode a signal is to jointly quantize a block of values. Given a codebook {C(i)} of reference patterns and a vector **x**, the codebook is searched to find the codebook entry $C(i_{min})$ for which a distortion measure d(**x**, C(i)) is minimized.

For speech signals, the input vector is usually derived from linear prediction coefficients. One commonly used measure is the Itakura-Saito measure,

$$d(\mathbf{x}, \mathbf{a}) = \frac{E}{\sigma^2} + \ln \frac{\sigma^2}{E_\infty} \qquad 1,$$

where

x is the input signal;
a is the LPC model derived from **x**;
E is the prediction error when **a** is applied to **x**;
σ is the gain of **a**; and
E_∞ is the limit of E_p as p->∞

If the input vectors are put in the form

$$[\ln \sigma^2, R(0)/\sigma^2, R(1)/\sigma^2, ..., R(p)/\sigma^2],$$

where R(0),...,R(p) are the autocorrelation coefficients, it suffices to minimize

$$D_y = \frac{E_y}{\sigma_y^2} + \ln \sigma_y^2$$

over all entries y in the codebook, where

$$E_y = \sum_{n=0}^{p} R_x(n) R_y(n)$$

The codebook used in the process above should be designed so that the overall average distortion is minimized. The standard technique for building a codebook is known as the Lloyd-Buzo-Gray (LBG) algorithm. The steps in this process are as follows:

1. Start with an initial codebook of random vectors $\{y_i\}$ and a set of training vectors $\{x_n\}$.
2. Classify the training vectors into the cells C_j corresponding to the vectors $\{y_i\}$ according to the nearest neighbor rule defined by the distortion measure.
3. Update the codebook vectors by computing the centroid of the cells C_j.
4. Repeat steps 2 and 3 until the decrease in total distortion falls below some threshold.

It can be shown that step 3 will always reduce the total distortion, so this process is guaranteed to converge to a local minimum. However, the solution obtained may not be a global minimum. Hence, this procedure is usually repeated using several different initial codebook.

quantize()

This function quantizes an autocorrelation vector using a computational VQ codebook (see Gray). A straight linear search through the codebook is performed to find the VQ symbol for which the distortion is minimized. The index and distortion of the closest codebook vector are returned.

Usage:

```
vq_result *quantize (rx, cb)
      float              *rx;    /* autocorrelation vector to be quantized */
      comp_codebook      *cb;    /* codebook */
```

create_comp_codebook()

This function converts a simple VQ codebook consisting of a sequence of LPC models into a computational form for use in *quantize*(). The entries are of the form $ra[i]/\sigma^2$, where ra[i] is the i-th autocorrelation coefficient of the sequence of predictor coefficients a[i], and σ is the gain of the model.

Usage:

```
comp_codebook  *create_comp_codebook  (cb)
      vq_codebook        *cb;  /* sequence of LPC models */
```

vq

Vq vector quantizes a speech signal using a codebook file. The codebook read from a file is converted to a computational form with *create_comp_codebook()* to speed the quantization process. The output file contains one short per frame.

Usage: *vq* {arguments}
arguments:

```
            InputFile (required)
      -i    InputExt [.b]
      -c    CodebookFile
      -o    OutputExt [.vq]
      -w    WindowSize [256]
      -s    ShiftSize [64]
      -v    Verbose
                  Prints quantization indices and distortions to stdout as they
                  are computed.
```

vqinit

Vqinit creates an initial VQ codebook by computing LPC models for randomly selected frames from the input training signal.

Usage: *vqinit* {arguments}
arguments:

```
            Filename (required)
      -i    InputExtension [.b]
      -o    OutputExtension [.cb]
        n   Size [256]
      -p    NumberOfPoles [14]
      -w    WindowSize [256]
      -s    ShiftSize [128]
```

kmeans

Kmeans trains a VQ codebook using the Lloyd-Buzo-Gray (LBG) algorithm, which iteratively refines the codebook in order to reduce the total distortion over a training signal. During each iteration, a codebook entry is replaced by the centroid of all of the models in the training set which were classified as this codebook entry. The centroid is computed by averaging the autocorrelation vectors and then computing an LPC model. The process terminates when the

reduction in the total distortion falls below the specified threshold. This algorithm is guaranteed to converge (i.e., the total distortion is monotonically decreasing), but it may not converge to a global minimum.

Usage: *kmeans* {arguments}
arguments:

	Filename (required)
-i	InitialCodebookExt [.cb]
-o	NewCodebookExten [.cb]
-t	TrainingFile (required)
-d	DistortionThreshold [0.1]
-w	WindowSize [256]
-s	ShiftSize [128]

vq2lps

Vq2lps computes a sequence of LPC spectra from a vector-quantized signal file and a codebook file. The output is a binary file which contains SpectrumLength shorts per frame. This is useful for visually inspecting the quality of the vector quantization.

Usage: *vq2lps* {arguments}
arguments:

	InputFile (required)
-i	InputExt [.vq]
-c	CodebookFile
-o	OutputExt [.lps]
n	SpectrumLength [64]
-l	LoggedOutput

cb2lps

Cb2lps computes the spectra of the entries in a VQ codebook. The output is a file of spectra as in *vq2lps*, which can be displayed as a spectrogram in order to inspect the individual codeword spectra.

Usage: *cb2lps* {arguments}
arguments:

	InputFile (required)
-i	InputExt [.cb]
-o	OutputExt [.lps]
n	SpectrumLength [64]
-l	LoggedOutput

IV. Hidden Markov Models

A Hidden Markov Model (HMM) is a stochastic modeling technique which has been applied to a variety of speech recognition problems. As a simple example, consider the following situation. Suppose that each of two urns contains a large number of red, blue and yellow balls in different proportions. A sequence of balls is generated in the following manner. A coin is flipped to determine a starting urn. A ball is then selected, its color noted, and returned to the urn. The coin is then flipped again to determine which urn to use next. If the distributions of the balls were known, then the probability of generating a particular sequence could be computed. Conversely, if a large number of sequences were observed, the probability distributions of the balls could be estimated.

Formally speaking, an HMM is a 5-tuple (Q, V, A, B, π), where

$Q = \{q_1,\ldots,q_N\}$ is a set of states

$V = \{v_1,\ldots,v_M\}$ is a set of output symbols

$A = \{a_{ij}\}$ is a matrix of transition probabilities;

$$a_{ij} = \Pr\{q_j \text{ at } t+1 \mid q_i \text{ at } t\}$$

$B = \{b_j(k)\}$ is a matrix of output probabilities;

$$b_j(k) = \Pr\{v_k \text{ at } t \mid q_j \text{ at } t\}$$

$\pi = \{\pi_i\}$ is a vector of initial probabilities;

$$\pi_i = \Pr\{q_i \text{ at } t=1\}$$

There are three basic problems for HMMs:

1. Evaluation. Given a model and an input sequence, what is the probability that the model generated the sequence?

2. Decoding. Given a model and an input sequence, what was the most likely sequence of states used?

3. Training. Given a model and a set of input sequences, how should the model be adjusted to make these sequences more probable?

Data Structures for HMMs:

```
typedef  struct
{
        int             nstates;
        int             nsymbols;
        double          *alpha;      /*  transition  probabilities  */
        double          *beta;       /*  output  probabilities  */
        double          *pi;
}   discrete_HMM;
```

forward-backward( )

The forward-backward algorithm is an efficient solution to the evaluation problem. Let $\{O_i\}$ denote an input observation sequence, γ an HMM, and $\{i_t\}$ a sequence of states. Define the forward variable $\alpha_t(i)$ as

$$\alpha_t(i) = \Pr(O_1,\ldots,O_T, i_T=q_i \mid \lambda)$$

These values can be solved using a lattice structure built according to the following inductive equations:

$$\alpha_1(i) = \pi_i b_i(O_1), \qquad 1\le i\le N$$

$$\alpha_{t+1}(j) = \left(\sum_{i=1}^{N} \alpha_t(i)a_{ij}\right) b_j(O_{t+1}) \qquad t=1,2,\ldots,T-1;\ 1\le j\le N$$

$$\Pr(O \mid \lambda) = \sum \alpha_T(i)$$

The backward variables $\beta_t(i)$ can be defined in a similar fashion as

$$\beta_t(i) = \Pr(O_{t+1},O_{t+2},\ldots,O_T \mid i_t=q_i, \lambda)$$

These variables satisfy the equations

$$\beta_T(i) = 1, \qquad 1\le i\le N$$

$$\beta_t(i) = \left(\sum_{i=1}^{N} a_{ij} b_j(O_{t+1}) \right) \beta_{t+1}(j) \qquad t= T-1,T-2,...,1;\ 1 \le i \le N$$

and can thus be found using the same type of lattice structure as that used for the forward variables.

The function *forward_backward()* takes an HMM γ together with an observation sequence O and returns the probability $Pr(O \mid \gamma)$. Since the computations above involve repeated multiplications by small probabilities, scaling must be performed to prevent underflow. At each stage of the lattice, scale factors c_t and d_t are introduced so that

$$\sum_{i=1}^{N} c_t \alpha_t(i) = 1 \quad \text{and} \quad \sum_{i=1}^{N} d_t \beta_t(i) = 1$$

Usage:

```
double forward_backward (hmm, O, T)
       discrete_hmm      *hmm;
       short             *O;     /* observation sequence */
       int               T;      /* length of the observation sequence */
```

viterbi ()

The Viterbi algorithm is a solution to the decoding problem. This algorithm finds the state path which maximizes $Pr(O,I \mid \gamma)$. The solution is very similar to the lattice technique of the forward-backward algorithm, except that the multiplication at each stage is replaced by maximization. The steps to this algorithm are as follows:

1. Initialize $\delta_1(i) = \pi_i b_i(O_1)$ and $\psi_1(i) = 0$ for $1 \le i \le N$.

2. For $2 \le t \le T$ and $1 \le j \le N$, set

$$\delta_t(j) = \max [\delta_{t-1}(i) a_{ij}]\ b_j(O_t)$$

$$\psi_i = \text{argmax} [\delta_{t-1}(i) a_{ij}]$$

3. Set $i_T^* = \text{argmax}\ [\delta_t(i)]$

4. Backtrack to find the most likely state path:

$$i_T^* = \psi_{t+1}(i_{T-1}^*) \qquad t = T\text{-}1,T\text{-}2,\ldots,1$$

The function *viterbi()* takes an HMM γ together with an observation sequence O and returns the optimal state path $s_1,\ldots,s_T$ along with the probability $\Pr(O,s \mid \gamma)$.

Usage:

```
double viterbi (hmm, O, T, s)
        discrete_hmm    *hmm;
        short           *O;     /* observation sequence */
        int             T;      /* length of the observation sequence */
        short           *s;     /* optimal state path */
```

baum-welch()

This routine solves the training problem. The probabilities π_i, a_{ij} and b_{jk} are updated according to the following reestimation formulas:

$$\bar{\Pi}_i = \gamma_1(i)$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\bar{b}_j(k) = \frac{\sum_{t=1;\, O_t=k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

where

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{Pr(O|\lambda)}$$

and

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(j)}{Pr(O|\lambda)}$$

The function *baum_welch()* takes an HMM γ together with an observation sequence O and adjusts the HMM parameters according to the equations given above.

Usage:

```
double baum_welch (hmm, O, T)
       discrete_hmm      *hmm;
       short             *O;     /* observation sequence */
       int               T;      /* length of the observation sequence */
```

V. Hand-editing Formant Tracks

lpc_tool

Lpctool is a graphics tool written by Eric Luce which allows formant tracks to be edited using a mouse. Starting with an LPC peak file produced by b2lpc, formant values can be deleted, interpolated, or

traced to yield the desired tracks. The output file is again an LPC peak file.

lclean

Lclean compares a LPC peak file, which contains formant frequencies and amplitudes, with the corresponding label file. All frames in the LPC peak file which do not correspond to "good" labels (vowels, liquids, glides and syllabic resonants) are filled with zeros.

Usage: *lclean* {arguments}
arguments:

	Filename (required)
-i	LpceExtension [.lpce]
-l	LabelExtension [.l]
-o	OutputExtension [.lpcv]

slp

Slp extracts and smooths a single formant track (F1, F2 or F3) from an LPC peak file. The contents of the LPC peak file are initially filtered to remove any values which are too close (within 50 Hz). This is necessary because *lpc_tool* sometimes leaves peaks which are too close together to separate visually. The extraction process works forwards and backwards to handle the cases where F2 or F3 is in the wrong slot. The smoothing performed is a simple running average, with small gaps being filled in by interpolation.

Usage: *slp* {arguments}
arguments:

	Filename (required)
-i	InputExtension [.lpce]
-f	Formant [2]
-t	FtOutputExtension [.ft#3]
-l	LpcOutputExtension [.lpc#3]
-a	AveragingLength [10]

wftool

Wftool is a graphics tool which produces and manipulates waterfall displays. This is a useful tool to use in conjunction with *lpc_tool*, since formant trajectories are often more clearly defined in waterfall displays than in spectrograms. *Wftool* is written in SunView, and

contains scrollbars to move through the display along with other settings which control the portion of the signal being viewed and the manner in which the spectra are drawn.

VI. Formant Trackers

markel

Markel is a simple implementation of the peak-picking algorithm due to Markel. The peaks of LPC spectra are found using quadratic interpolation as in b2lpc, and formant labels are assigned by a simple left-to-right slot-filling scheme which makes use only of the decisions from the previous frame.

Usage: *markel* {arguments}
arguments:

	Filename (required)
-i	InputExtension [.b]
-o	OutputExtension [.for]
-w	WindowSize [256]
-s	ShiftSize [128]
n	NumberOfPoles [15]

fsd

Fsd was the primary formant tracker examined in this thesis. Statistics for the multivariate distributions of spectral values together with HMM model parameters are read in from a file produced by *mksdstats*. A stackfile containing a list of signal files to be tracked is also read in. Each signal is processed by computing or reading LPC spectra, finding the observation probabilities for these features using the multivariate distance measure on each frame, and then smoothing by some combination of mean conditional estimates, HMMs and median smoothing.

Usage: *fsd* {arguments}
arguments:

	Filename (required)
-i	InputExtension [.stack]
-f	StatsFile (required)
-o	OutputExtension [.d]
-w	WindowSize [256]
-s	ShiftSize [64]

```
-m    MeanConditionalEstimates
 h    HMM
-a    MedianSmoothing
```

mksdstats

Mksdstats produces the statistics needed to run *fsd*. LPC spectra are computed for each of the signals listed in a stackfile. The specified portions of these spectra are resampled to yield the desired number of features. These features are used along with the hand-edited formant tracks to generate the multivariate statistics for each formant frequency interval. The hand-edited tracks are also used to estimate the HMM transition probabilities.

Usage: *mksdstats* {arguments}
arguments:

```
        Filename (required)
-i      InputExtension [.stack]
-o      OutputFile (required)
-w      WindowSize [256]
-s      ShiftSize [64]
-a      FirstComponent [4]
 b      SecondComponent [31]
 n      NumberOfFeatures [6]
 q      QuantizationSize [200]
```

fvq

Fvq is an implementation of the Kopec formant tracker using vector quantization and hidden Markov models. The VQ observation probabilities and HMM transition probabilities are read in from a file produced by *mkvqstats*. A stackfile containing a list of signals to be tracked is also read in. Each signal is processed by computing or reading VQ codebook symbols, finding the observation probabilities by a simple table lookup, and then smoothing with the forward-backward algorithm.

Usage: *fvq* {arguments}
arguments:

```
        Filename (required)
-i      InputExtension [.stack]
-f      StatsFile (required)
-c      CodebookFile (required)
-o      OutputExt (required)
-w      WindowSize [256]
-s      ShiftSize [64]
-v      Verbose
```

mkvqstats

Mkvqstats produces the statistics needed to run *fvq*. Each of the signals in a stackfile is vector quantized and used together with the hand-edited formant tracks to generate the HMM observation and transition probabilities.

Usage: *mkvqstats* {arguments}
arguments:

	Filename (required)
-i	InputExtension [.stack]
-c	CodebookFile (required)
-o	OutputFile (required)
-w	WindowSize [256]
-s	ShiftSize [64]
-q	QuantizationSize [200]

VII. Quantitative Analysis of Formant Trackers

The results of the formant trackers *fsd* and *fvq* are written out to files which contain one 16-bit (integer) formant frequency value and one 32-bit (floating-point) detection probability per frame (5 msec). The program *fscore* produces statistics for a single signal, and *comfsc* combines these across a collection of signals. Finally, *prfsc* prints the output of *fscore* or *comfsc* in a readable (and rather verbose) style.

fscore

Fscore compares the results of *fsd* or *fvq* for a single signal at a given threshold with the hand-labeled track generated by *slp* and the associated label file to produce a collection of statistics for each class of labels. These statistics include the RMS error, number of errors of various sizes, number of false alarms, and number of missed formants. The output is a text file, but is difficult to interpret without *prfsc*.

Usage: *fscore* {arguments}
arguments:

	Testfile (required)
-i	InputExt [.d]

```
-c      HandLabelledExt  [#1.F2]
-l      LabelExt  [#1.l]
-o      OutputExt  [.fsc]
-t      Threshold  (required)
-e      ErrorStepSize  [250]
-v      Verbose
```

comfsc

Comfsc combines the scores produced by fscore on a collection of signals, and writes the results out in the same format.

Usage: *comfsc* outfile fscfile1 ... fscfilen

prfsc

Prfsc prints out the results of either *fscore* or *comfsc* in a readable fashion. The RMS error, missed formants, etc. are printed to stdout for each of the 14 label classes.

Usage: *prfsc* {arguments}
arguments:

```
        InputFile  (required)
-i      InputExt  [.fsc]
```

VIII. Miscellaneous Routines

argset()

This routine is used to handle the command line argument processing in most of the programs in this work. *Argset()* rearranges the arguments in argv[] and performs some housekeeping on these arguments. All but one of the arguments must be preceded by a switch -c, where c is a character. An argument not preceded by a switch is assumed to be the primary filename. The manner in which the arguments are processed is described in an associated argument file. For example, consider the program 'sample' with the following usage:

```
Usage: sample {arguments}
arguments:
        Filename
 p      Parameters  (required)
-i      InputExtension  [.in]
-o      OutputExtension  [.out]
 r      ResultsFile  (required)
 n      NumberOfTimes  [1]
-m      Multiplier  [3.14]
-l      LongOutput
```

When called with

```
argset (argc, argv, &params, &infile, &outfile, &resfile,
                        &ntimes, &mult, &longoutput)
```

argset() will do the following:

(1) Display the above usage message if the usage is incorrect.
(2) Prompt the user for any required arguments which were omitted.
(3) Fill in the default values for any optional arguments which were not specified.
(5) If an argument specifies a file, check to see if this file can be opened.
(6) Check integer and float types.
(7) Set any simple switches which are specified to a value of 1.
(8) Accept the arguments in any order, and rearrange them in the order given in the argument file. If two or more arguments with the same switch are given, only the last one is used.
(9) If the primary filename contains an extension, a base name is extracted from this to use with any other extensions, the extension is assigned to the first input extension argument, if there is one. All extension arguments will be returned with the full name (base name plus the extension).

The argument file contains the description of the arguments. The recommended name for this file is 'command_args.z'. This file must be compiled with *mkargs*, and the resulting object file linked with the other object files for this program. For the sample command above it might look like this:

```
#     SWITCH    NAME              OPT/REQ     TYPE        DEFAULT
#
                Filename          r           c
      p         parameters r            c
      i         inputextension    o           r           .in
      r         resultsfile       r           W
      n         numberoftimes     o           d           1
      m         multiplier        o           f           3.14
      l         longoutput        o           s
```

Each line of the argument must contain the switch, name, optional/ required specifier, type and default value of an argument (except lines starting with # which are taken as comments). These are described below.

SWITCH is assumed to be a single character.
A switch of '-' is assumed to designate a filename.
NAME will be displayed in prompts and usage messages.
It may be up to 50 characters but may not contain spaces.
OPT/REQ specifies whether the argument is optional or required. Its value must be either 'r' or 'o'
TYPE must be one of the following:

a	extension for file to be appended
A	filename for file to be appended
c	character string
d	integer
f	float
s	simple switch (no argument)
r	extension for file to be read
R	filename for file to be read
w	extension for file to be written
W	filename for file to be written

DEFAULT is the value given to optional arguments which are not specified. Extension defaults should include the ' ' and simple switches should have a default of '-' A '-' default should be given for required arguments. The default may contain '#n' , where n is the number of an argument higher on the list. '#n' will then be replaced by the value of that argument.

mkargs

Mkargs compiles an argument file into an object file for use with *argset*().

Usage: *mkargs* {arguments}
arguments:
Filename (required)

```
-i      InputExtension  [.z]
-o      OutputExtension [.c]
-c      LeaveCfile
```

sd

Sd is a general purpose Gaussian classification program. Given a set of feature vectors of the form $\mathbf{x} = (x_1, x_2, ..., x_N)$ drawn from M different groups $G_1, G_2, ..., G_M$, the goal is to derive a decision rule which can be used to classify a feature vector as one of the groups G_i. The feature vectors can be thought of as lying in a multidimensional space in which the groups occupy certain regions. In Gaussian classification, the feature vectors for each group are assumed to have multivariate normal distributions; that is, the groups occupy ellipsoidal regions. Each group then can be represented by the centroid of its feature vectors together with a covariance matrix which describes the shape of the distribution. The probability density function for group G_i can be expressed in the form

$$f_i(\mathbf{x}) = (2\pi)^{-m/2} |\mathbf{R}_i|^{-1/2} \exp [(\mathbf{x} - \bar{\mathbf{x}}_i)^T \mathbf{R}_i^{-1} (\mathbf{x} - \bar{\mathbf{x}}_i)]$$

where

$\bar{\mathbf{x}}_i$ is the mean of all feature vectors from G_i; and

$\mathbf{R}_i$ is the covariance matrix for G_i.

A feature vector $\mathbf{x}$ is classified as belonging to the group for which $f_i(\mathbf{x})$ is largest. This is equivalent to minimizing the 'distance'

$$D_i^2(\mathbf{x}) = (\mathbf{x} - \bar{\mathbf{x}}_i)^T \mathbf{R}_i^{-1} (\mathbf{x} - \bar{\mathbf{x}}_i) + \ln |\mathbf{R}_i|$$

This is the basic maximum-likelihood distance measure (D1). Another distance measure (D2) can be used which ignores the term $\ln |\mathbf{R}_i|$,

which is often insignificant. If the covariance matrices $\mathbf{R}_i$ are assumed to be equal, the Mahalanobis distance (D3)

$$D_i^2(\mathbf{x}) = (\mathbf{x} - \bar{\mathbf{x}}_i)^T \mathbf{R}^{-1} (\mathbf{x} - \bar{\mathbf{x}}_i)$$

is obtained. This measure turns out to be much easier to compute since it can be reduced to a linear calculation.

From the distance measures $D_1, \ldots, D_M$, corresponding probabilities can also be obtained. The probability $p_i(\mathbf{x})$ that a feature vector x came from group G_i is given by

$$p_i(\mathbf{x}) = \frac{\exp(-0.5 \times D_i^2(\mathbf{x}))}{\sum_k \exp(-0.5 \times D_k^2(\mathbf{x}))}$$

Sd allows a set of feature vectors to be classified using any of the above three distance measures. The input data must be in a file which has the following format. The first long contains the record length in bytes. Each record contains the group index followed by the feature values. Each value in a record must be a 32 bit floating point number. Two accompanying files, with extensions .pnames and .gnames, should contain the feature names and the group names, respectively. *Sd* outputs a text result file which contains a confusion matrix and general information about the classification performance. A binary decision file which contains the group numbers for each of the feature vectors is also written out. The statistics generated from a training set can be saved and used later for testing.

Usage: *sd* {arguments}
arguments:

TestFile

-i InputExtension [.sd]
Input file of test vectors.

-p Parameters (required)
Features to use from the test and training files. This should be a list of integers such as 1-4,6.

-t TrainingFile [#1.sd]
Input file of training vectors.

-d DistanceMeasure [1]
D1, D2 or D3 as described above.

r ResultsFile [#1.#3.#5.*7.*8.*14.*15]
Basic output file.

-s StatsFile [#1.#3.*7 *8.*15.stats]
File to use when testing.

-o DecisionFile [#1.#3.#5.*7.*8.*15.d]
Binary decision file, 1 short per token.

-m GroupNameFile [#1.gnames]
A file which contains the names of the groups.

n ParameterNameFile [#1.pnames]
A file which contains the names of the features.

-l LongForm
Prints out the classification of each token together with its distances and probabilities.

-x ExtraStats
Writes out a file with additional statistics including interclass distances and various separation measures.

-j Jackknifed
Performs jackknifing, in which each token is removed from the training set before classifying. This is extremely slow but useful if the data is too small to split into separate training and testing sets.

-z SaveStats
Saves the group statistics gathered from a set of training vectors.

-k TerminalInput
Initiates an interactive mode in which feature vectors can be entered through the keyboard.

-v Verbose
Prints the classifications to stdout as they are performed.