RSD-TR-1-82

Speech Recognition Using LPC Analysis

John L. Ostrander
Timothy D. Hopmann
Edward J. Delp

January 1982

Center for Robotics and Integrated Manufacturing

Robot System Division

COLLEGE OF ENGINEERING
THE UNIVERSITY OF MICHIGAN
ANN ARBOR, MICHIGAN 48109

TABLE OF CONTENTS

ABSTRACT

Speech recognition using LPC analysis is briefly discussed and and some experiments using these techniques are outlined. The results of these experiments are discussed and suggestions are made for the improvement of the algorithms. The authors would like to thank Professor Williams for the use of the A/D converter in the Bioengineering Laboratory and CIPRNET for the use of the computer facilities.

1. Introduction

The object of the project was to write a system of software capable of recognizing a word out of a vocabulary of 100 words, assuming all utterances were from the same speaker. A number of approaches to this problem have been proposed. Most have been based on some sort of spectral analysis of the unknown followed by comparison with a number of stored templates representing the words in the vocabulary. In the case of short-time Fourier analysis, the results are a time varying Fourier spectrum which represents the entire speech signal. The disadvantage of this method is that varying pitch period, not uncommon even with one speaker, can significantly alter the spectrum and hence affect the recognition process. Homomorphic analysis overcomes this problem by removing the effects of the glottal pulse through low-pass cepstral filtering which effectively removes the quasi-periodic portion of the speech wave. The resulting spectrum should be an approximation to the vocal tract transfer function, which is fairly consistent from utterance to utterance. Here the major disadvantage is the inaccuracy of the approximation. Linear Predictive Coding (LPC), on the other hand, generally characterizes the vocal tract quite well. LPC analysis results in a set of coefficients for the denominator polynomial of an all-pole model of the vocal tract transfer function. Since this set of coefficients is quite small (generally on the order of 10 to 15 coefficients), LPC analysis has the added advantage of reducing the amount of data which must be retained as a template and thus simplifies the comparison procedure. Because of these superior characteristics, we chose to use LPC analysis in our recognition scheme (also, we simply wanted some experience with LPC analysis). The comparison between unknown and template was done using a statistical distance measure based on the probability distribution function of the estimates of the true LPC coefficients [1]. The distance measure itself was developed by Rabiner and Schafer [2]. This choice was due to its relative

simplicity and its high probability for success as demonstrated by the use of similar distance measures in past experiments [3].

The project consisted of the following basic steps:

1. digitization and segmentation of the vocabulary,
2. analysis.
   a) LPC analysis including spectrum calculations,
   b) homomorphic analysis (for comparison to a),
3. distance calculations,
   a) template generation,
   b) actual distance calculation,
4. recognition.

Each of these steps is discussed in detail in this report.

In the working of this project, we found the numerical analysis package to be quite useful for distance calculations where matrix inversion was necessary. Also, we found the VAX plotting package to be absolutely indispensable for examination of the results of word segmentation and spectral analyses.

2. The Vocabulary

The vocabulary was chosen with the intent to obtain a representative sample or group of samples from each one of several categories. A list of the vocabulary used may be found in Appendix A. The categories were designed to be representative of each of the major problem causing types. The categories were:

1) Words starting and/or ending in fricatives such as /f/ or /s/. These words present a problem in segmentation because of the relatively low

energy of the fricatives themselves, thus they may easily be lost in background noise.

2) Words ending in nasals such as /m/ or /n/. These words present a problem in segmentation due to the slow tailing off of energy at the end of the word. LPC analysis tends to have problems with nasals because they require zeros in the Z-transform domain for accurate characterization due to the coupling of the nasal cavity and concomitant loss.

3) Words ending in plosives such as /p/ or /t/. These words present a problem in segmentation due to the long period of silence prior to the plosive burst and the short duration of the burst itself.

4) Words with low energy sections in their centers such as the middle fricative in "fifteen". Such words are a problem in segmentation due to the possibility of the low energy section being mistaken for end of word.

The uttering of the vocabulary was done by Mr. Hopmann in the relative quiet of his living room. In an effort to minimize the total amount of data required to digitize the speech prior to segmentation, the words were uttered with their beginnings spaced approximately one second apart. This was probably a mistake both because of the difficulty in speaking a large number of words at such a fast pace and because of the subsequent difficulties introduced into the segmentation process as will be discussed later. Four utterances were obtained of each word (although a few were lost as the reader will soon see). Because prior utterances can affect the speaking of a word, the words were read in four different directions from the list in Appendix A. First the list was read by columns going from top to bottom. The second utterance was by

columns from bottom to top. The third utterance was by rows from left to right. The fourth utterance was by rows from right to left.

These utterances were recorded on an ordinary, garden variety, home cassette tape deck. We used a magnetic coil (dynamic) microphone with nominal frequency response from 100 to 10,000 Hz. The recording was then digitized using the equipment in the bioengineering laboratories located on the fifth floor of East Engineering. The vocabulary was played back into the A/D converter through a bandpass filter cutting off at 100 Hz to eliminate dc and 60 Hz hum and at 4500 Hz to avoid aliasing as we sampled at 10,000 Hz. The data was then placed on magnetic tape and hand carried down to the VAX computer. Several words from the end of the vocabulary were lost during this process due to the break down of the Digi-Data tape drive in the bioengineering laboratories. This also prevented us from obtaining a set of utterances in noise as we had originally planned.

The A/D converter in the bioengineering laboratories has ten bits of accuracy. Quantization is linear and the peak to peak maximum voltage is 2V. The data numbers are two's complement and thus may have values from -512 to +511. The data is in the following format:

first 4 bits -- unused, always one,

next 10 bits -- data,

next 2 bits -- channel information, ignored.

Hence it was necessary to reformat the data into a usable form. This was accomplished by the program "convert" which lopped off the first 4 bits and the trailing two bits then sign extended to make each data point into a two- byte short integer.

Following this process, we made a few plots of the initial portions of our data files and found that there was a significant dc offset probably introduced by the A/D. This was removed, at least for the most part, by the program "nodc" (read "no dc"). This program assumed that the noise surrounding the words was zero mean and that the dc offset remained fairly constant. Dc was then removed by calculating the mean of the first 100 ms of noise then subtracting this mean from each data point.

3. Segmentation

Segmentation of words out of background noise presents a variety of problems due to the statistical nonstationarity of the speech signal. Unfortunately, no rigorous method has been devised to distinguish speech from noise so we must rely on basically heuristic, ad hoc approaches. It is assumed, in this process, that the voiced sections of speech are significantly higher energy than the background and that unvoiced sections of speech, while they may not be distinguishable from noise by their energy, do cross zero more often than noise due to their high concentration of energy at high frequencies.

The algorithm we used was approximately the same as that of Rabiner and Sambur [4] with minor changes due to our slightly different situation. The algorithm was as follows:

1) The first 100 ms (1,000 points) of data was read and used to calculate three thresholds, a high energy threshold, a low energy threshold, and a zero crossing threshold. These first 100 ms are assumed to be pure noise. "Energy" is calculated by the short time average magnitude (STAM). These thresholds came from the equations:

$$lothresh = MIN\left[0.03 \times \left[13000 - \mu_{mag}\right] + \mu_{mag},\ 4 \times \mu_{mag}\right]$$

$$hithresh = N \times lothresh$$

$$zcthresh = MIN\left[\mu_{zc} + 2 \times \sigma_{zc}, minimum\right]$$

where μ_{mag} is the STAM, μ_{zc} is the mean of the zero crossing rate, σ_{zc} is the standard deviation. "Minimum" (in the zcthresh equation) is a relatively arbitrary constant meant to insure that zcthresh is a reasonably high value.

2) The next 2 seconds (20,000 points) of data were read in. The STAM was calculated over a 200 sample window. If the STAM exceeded the high threshold, then the starting point of the word was tentatively set to this point. The window was then advanced by 100 samples and the sum taken again. End of word was tentatively set at a point past which the STAM fell below the threshold for some preset number of windows. If start of word was not found in the first 1,250 ms of data, then it was assumed that the first second was all noise and the second second was moved down to the bottom half of the data array and the second half was refilled. In this way, the approximate boundaries of the voiced sections of the word were found. Using the high threshold hopefully avoided classifying background taps, clicks, etc. as speech. The allowance for a number of segments below the high threshold avoids premature marking of end of word due to center fricatives, etc. The requirement that the beginning of the word be found in the first 1.250 ms avoided a word being only partially within the data array when found (assuming, of course, that no utterance is longer than 750 ms).

3) The tentative beginning and ending points of the words were

extended by comparing the STAM to the low threshold. This should have definitely established the ends of the voiced section of the word and given new tentative beginning and ending points.

4) The tentative beginning and ending points were again extended by comparing the zero crossing rate over 100 sample intervals to the zero crossing threshold. The endpoints established here were considered to be the actual endpoints of the word. In this way, the endpoints should be established to within 50 samples of the actual location.

6) The program then informed the user of start, end, and length of the word segmented and requested that a filename be given to which the program then wrote the data (note that this means that words are not constant length).

7) If another word is expected (number of words expected is input on the command line), the program moved the data in the remainder of the data array (after the word) down to the bottom and the array was refilled back to 20,000 points. The process then continues at step 2.

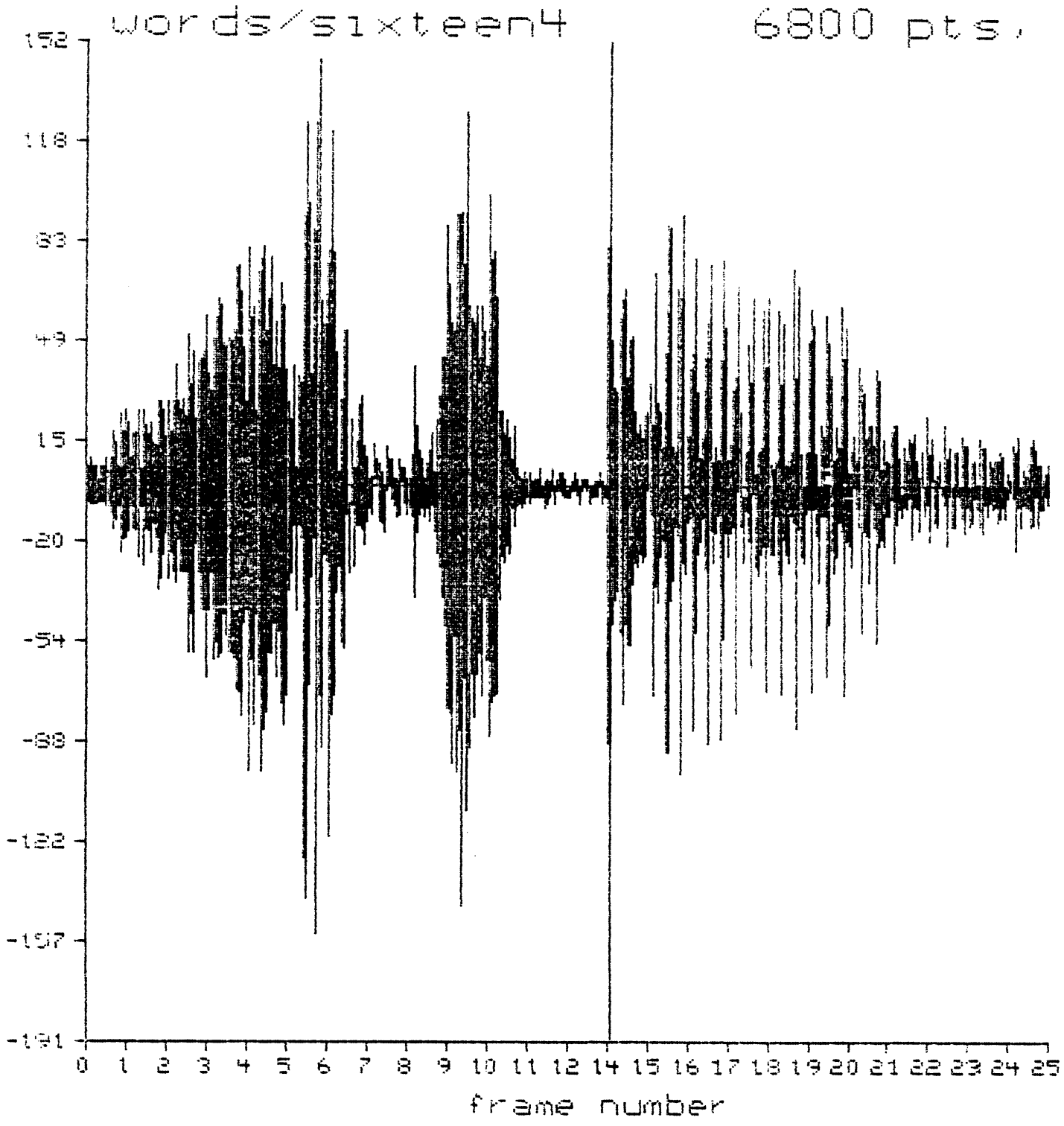
This algorithm was implemented in the program "segment."

As might be expected due to the ad hoc nature of this approach, some problems were encountered in the implementation of this algorithm. First of all, it was noted that the zero crossing checks rarely if ever actually extended the word. This, we deduced, was not due to a fault in the algorithm, but was simply due to the extremely low noise levels in our recording. This meant that the STAM of fricatives was still significantly higher than the noise and thus even fricatives were included by the low energy threshold extension. Second, it was noted that since a small dc offset was present in the segment where the zero

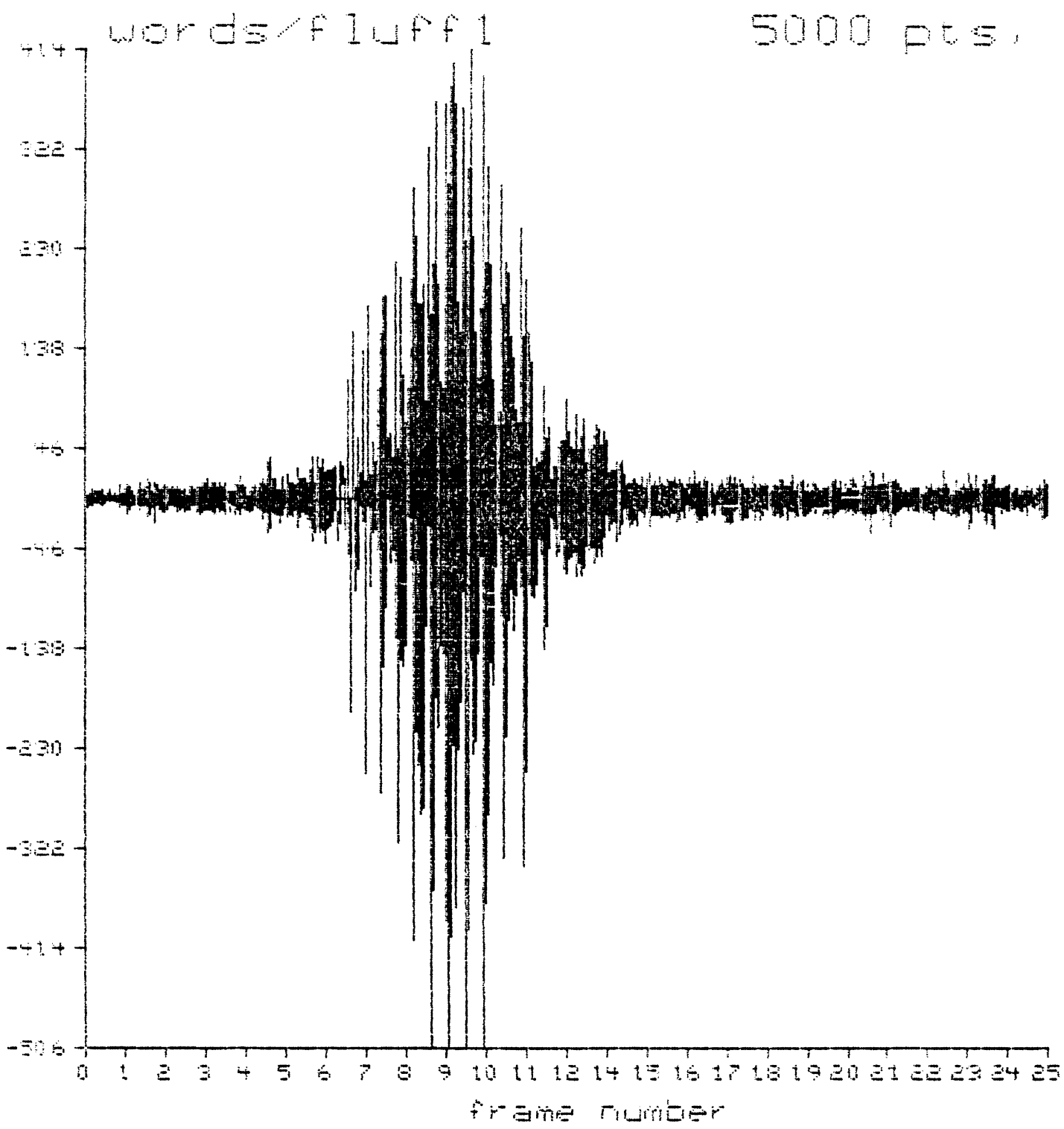
crossing threshold was set, and since the noise levels were quite small, an artificially low zero crossing threshold was found. This would not be a problem in data containing higher noise levels. Third, it was noted that the ends of the word seemed to be occasionally chopped and other times slightly too liberal. Careful examination of the data before and after segmentation showed, however, that this was basically the result of the resolution of the segmenter. The fourth and most insidious problem was that of occasional premature end of word marking because of the surprisingly long segment of dead space prior to middle and ending plosives. This meant that the high threshold had to be lowered or the amount of dead space allowed had to be lengthened. This is where the unfortunately small spacing between some words became a problem. This small spacing was sometimes on the same order as the dead space before a middle or ending plosive! After playing around with the parameters we were able to solve this problem in most cases, but several utterations were lost. The final problem was that of extraneous loud taps, etc., being segmented off. For the most part, these were simply recognized as junk and thrown away. In several cases, however, these noises were included at the beginning and ending of words and had to be removed. This was done by "clean" which simply deletes a portion of the data file as specified by the user.

The final format of the words. then, was variable length files of short integers in 2's complement format, magnitude -511 to +512. The appendix shows which of the utterations were lost in the segmentation process.

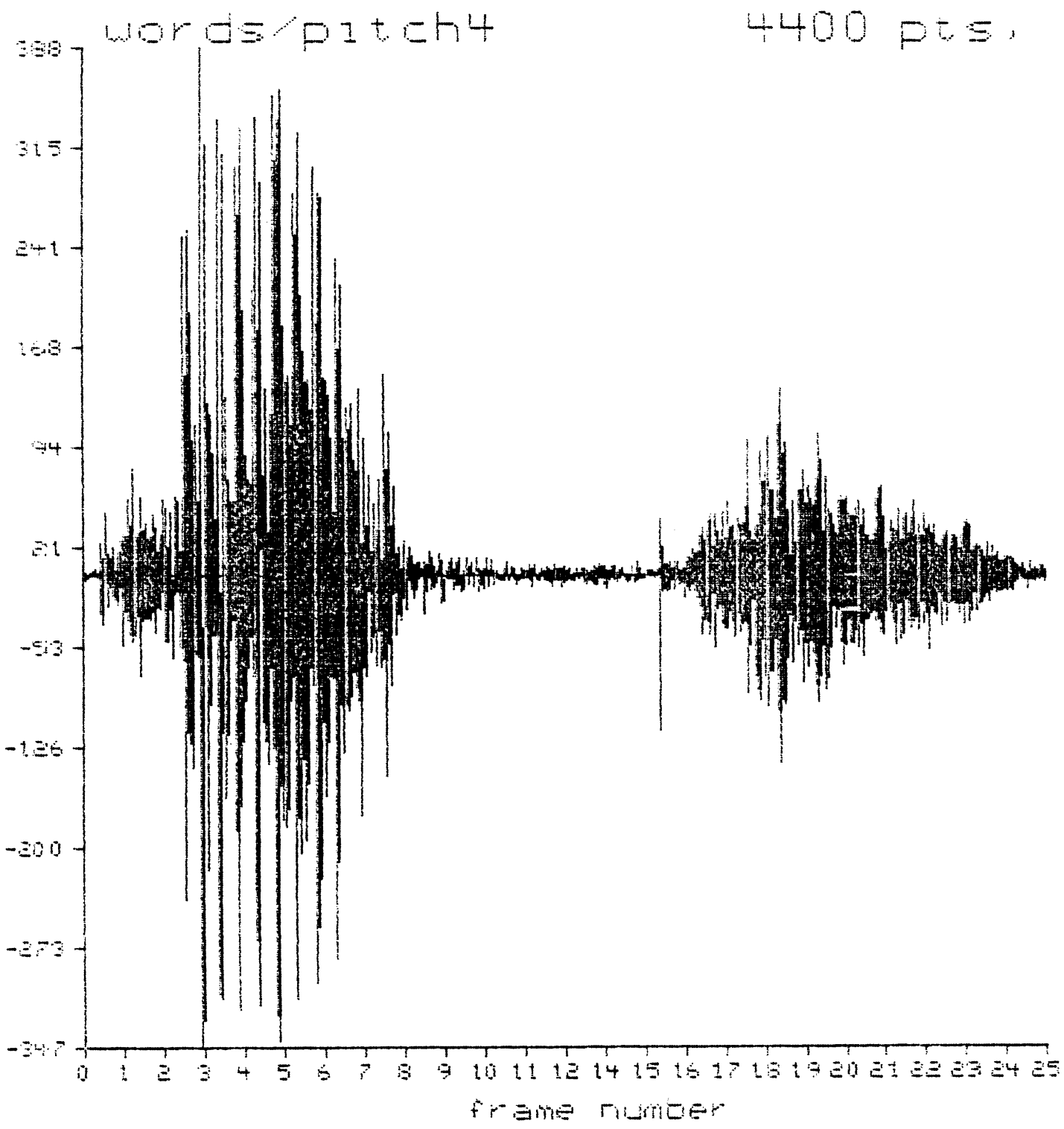
Some results of this segmentation can be seen on the following few pages.



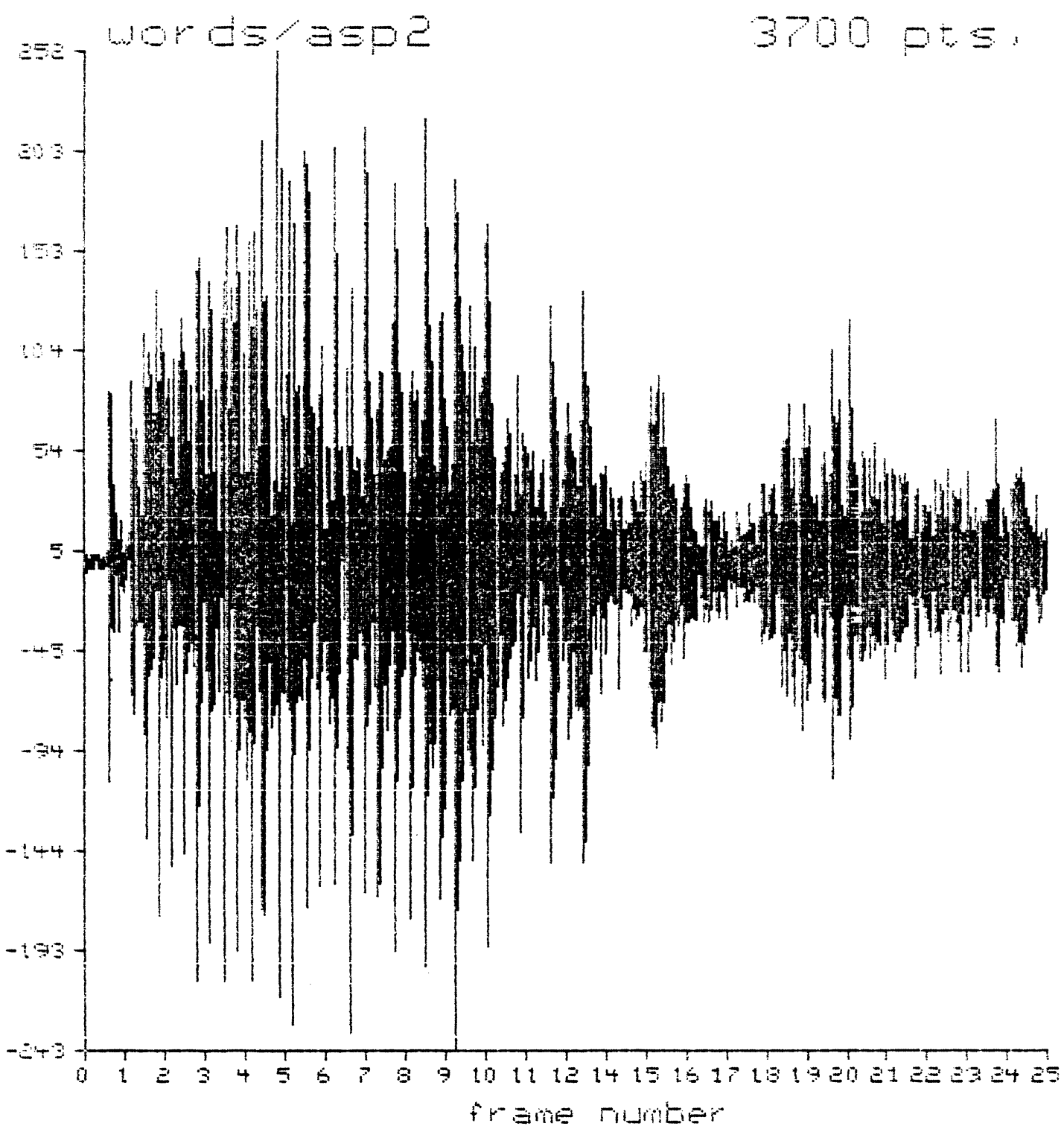
Time domain plot of "sixteen": Note the chopped appearance at the end of the word. This is due to the slow tailing off of energy in the final "n."



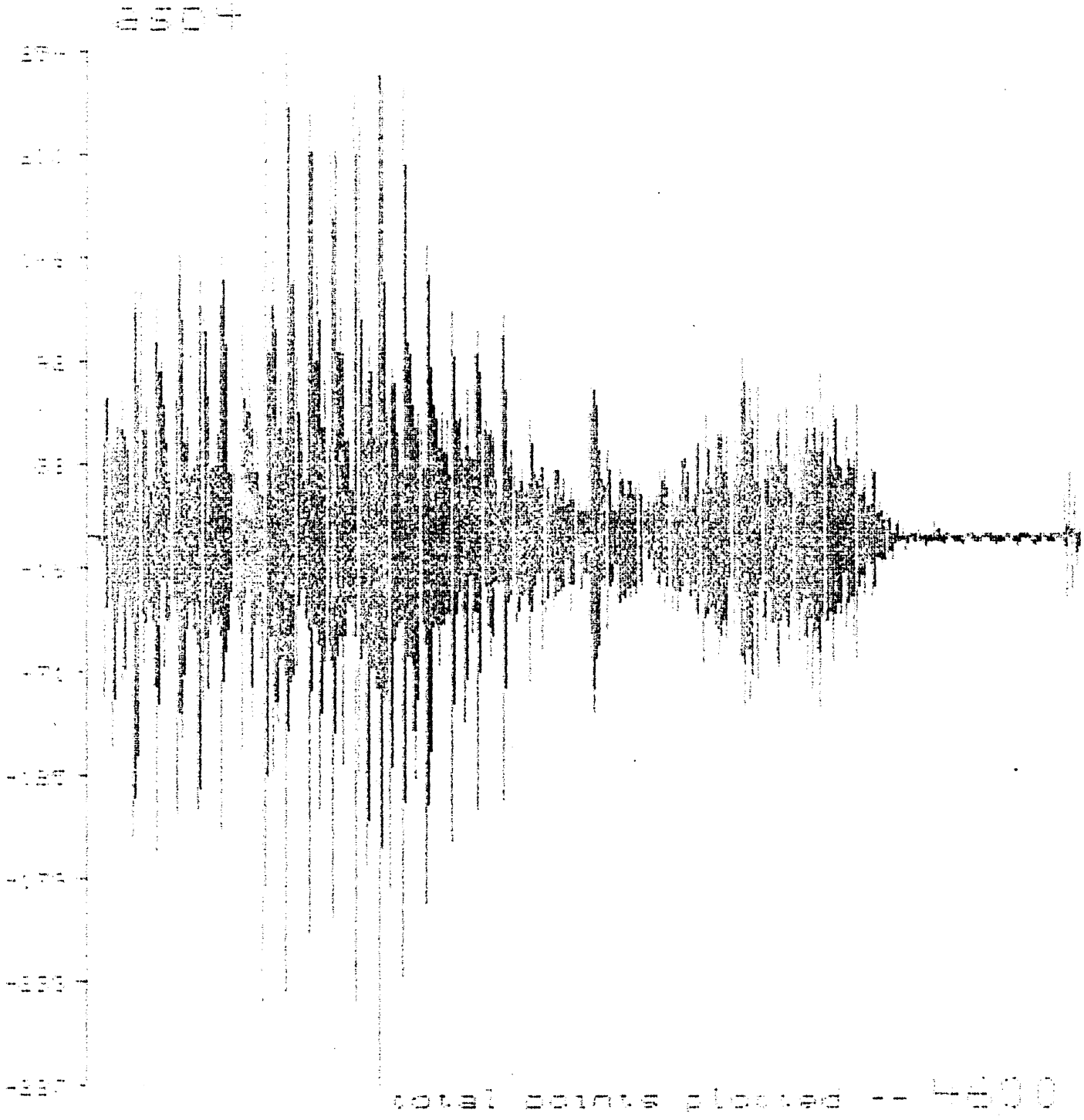
Time domain plot of "fluff": Note the low energy of the fricatives on both ends of the word. Chopped appearance at the end of the word is due to the segmenter resolution.



Time domain plot of "pitch": This is the utterance which caused the greatest problems in segmentation. Note the long period of deadspace prior to the "t" plosive.



Time domain plot of "asp" after clean-up: Note the short segment of noise at the beginning due to the segmenter resolution. The endpoint was gotten from trial and error.



Time domain plot of "asp" before clean-up: Note the long segment of noise at the end followed by a short but strong click.

4. Analysis Techniques

We wanted to use LPC analysis for the actual recognition of words, in this section.hat is one of the methods discussed here. We desired, however, a means of verifying the correctness of our LPC parameter calculations. In order to do this, we undertook to find the log magnitude frequency domain response of the vocal tract through homomorphic processing. The vocal tract responses from these methods were then compared.

4.1. LPC Analysis

LPC analysis is based on the assumption that the speech signal can be characterized by a predictor model which looks at past values of the output alone, hence it is an all pole model in the Z-transform domain. This assumption is good for most speech signals, which contain strong resonances and weak anti-resonances. The assumption breaks down somewhat for nasal sounds such as /m/ or /n/ due to the strong anti-resonances introduced through nasal coupling. Another assumption in this kind of analysis is the short time stationarity of the speech signal. This also breaks down, most obviously at phoneme boundaries, but also within certain phonemes such as the glides (/r/ and /w/). Thus one should expect generally good results using such an all pole model except in these cases where the signal will not, in general, be well characterized.

A further simplification of the analysis is introduced by the assumption that the vocal tract is excited either by random noise (unvoiced speech) or an impulse train (voiced speech). This assumption places the burden for characterizing the glottal pulse shape on the analysis technique.

Studies of the speech signal have shown [2] that one can expect one resonant (or formant) frequency per 1,000 Hz bandwidth. Since our signal was bandlimited to 4,500 Hz, we expected approximately 5 formants to be in

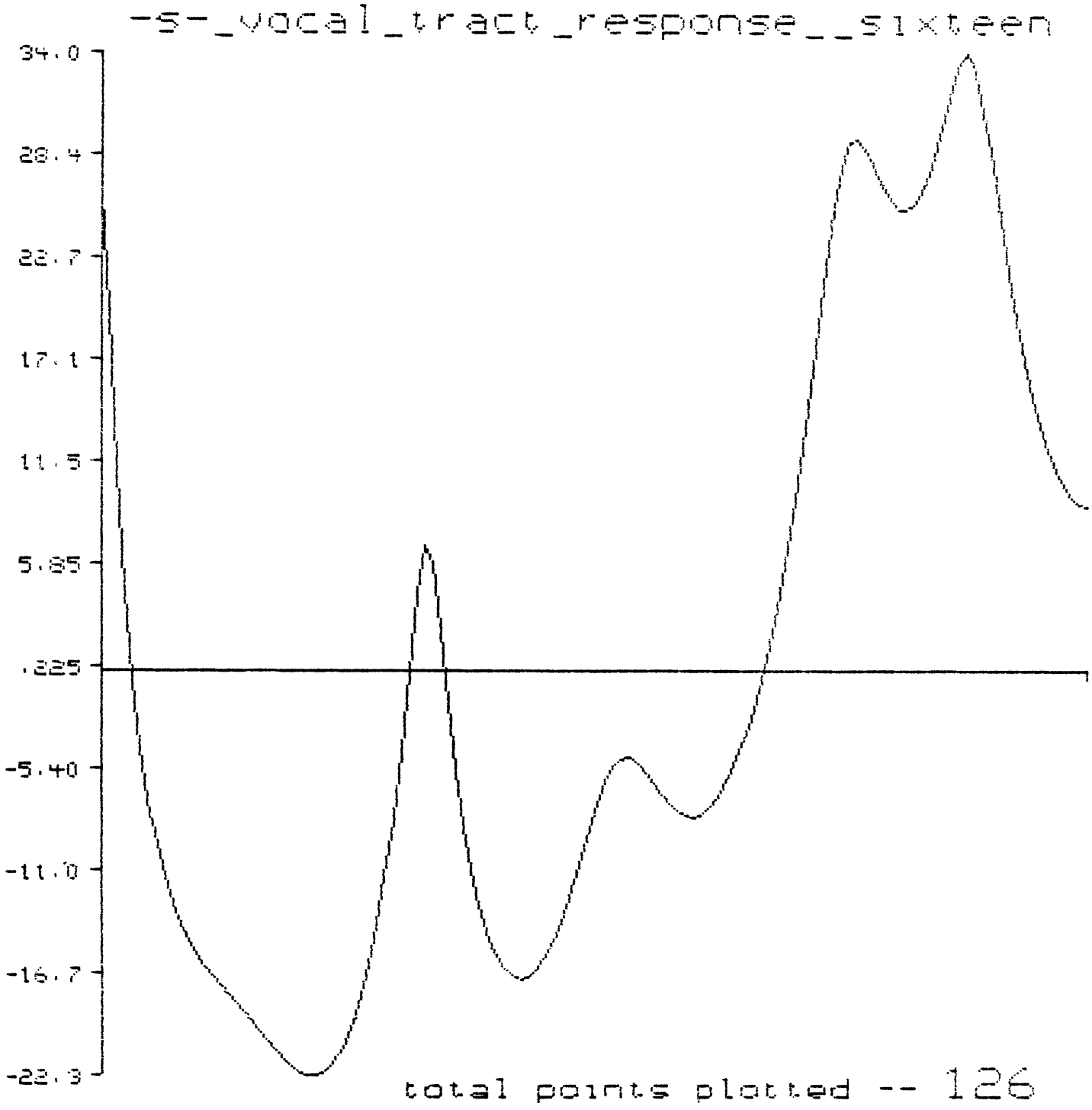
evidence. Since LPC analysis produces N complex poles (where N is the order of the predictor) and characterization of the formants requires a complex conjugate pole pair per formant, twice as many LPC coefficients as formants are required. This would imply that a tenth order predictor would be appropriate for our application. The situation is complicated, however, by the fact that the glottal pulse is not an ideal impulse train and there are radiation losses at the lips. These effects add further information to the system which must be taken into account in the analysis. Again, it has be found [2] that this additional information may be characterized using three to four poles. Thus we chose our predictor to be 14th order.

The actual implementation of the LPC analysis is somewhat dependent upon the time-warping method chosen. The purpose of time-warping is to attempt to get the various portions of two utterances to match up thus eliminating, in as far as possible, the effects of varying speed in the utterance of a word. Since speed can vary even within a word, a non-linear time warping algorithm would seem desirable. In recent years, attempts have been made to use various non-linear time-warping techniques and the effects have been promising. In order to simplify our implementation, however, we chose to use a simpler, linear time warping scheme. This scheme used a constant analysis window size and simply varied the starting points of the windows so that a constant number of evenly spaced windows were used. It is desirable that analysis windows overlap to some extent, so we chose to use 25 300-point windows so that at a wordlength of 5100 samples (the approximate average) the windows would have 1/3 overlap.

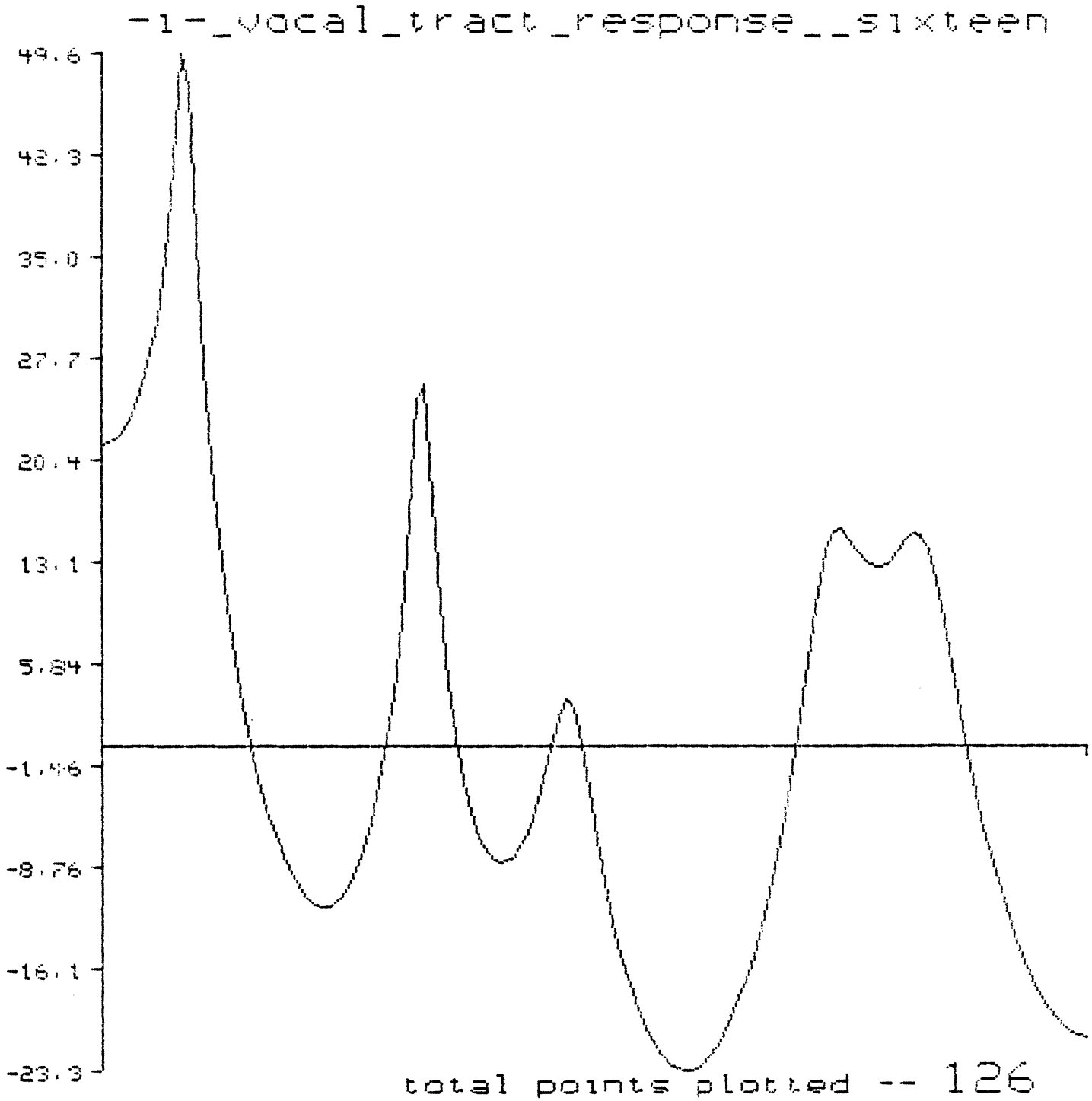
The LPC analysis algorithm itself was based on Burg's algorithm [2]. The windows were multiplied by a 300 point Hamming window in order to reduce sidelobe effects in the frequency domain. The LPC coefficients were then calculated for the window and written out sequentially to a file, thus the file

contained 25 sets of 14 LPC coefficients.

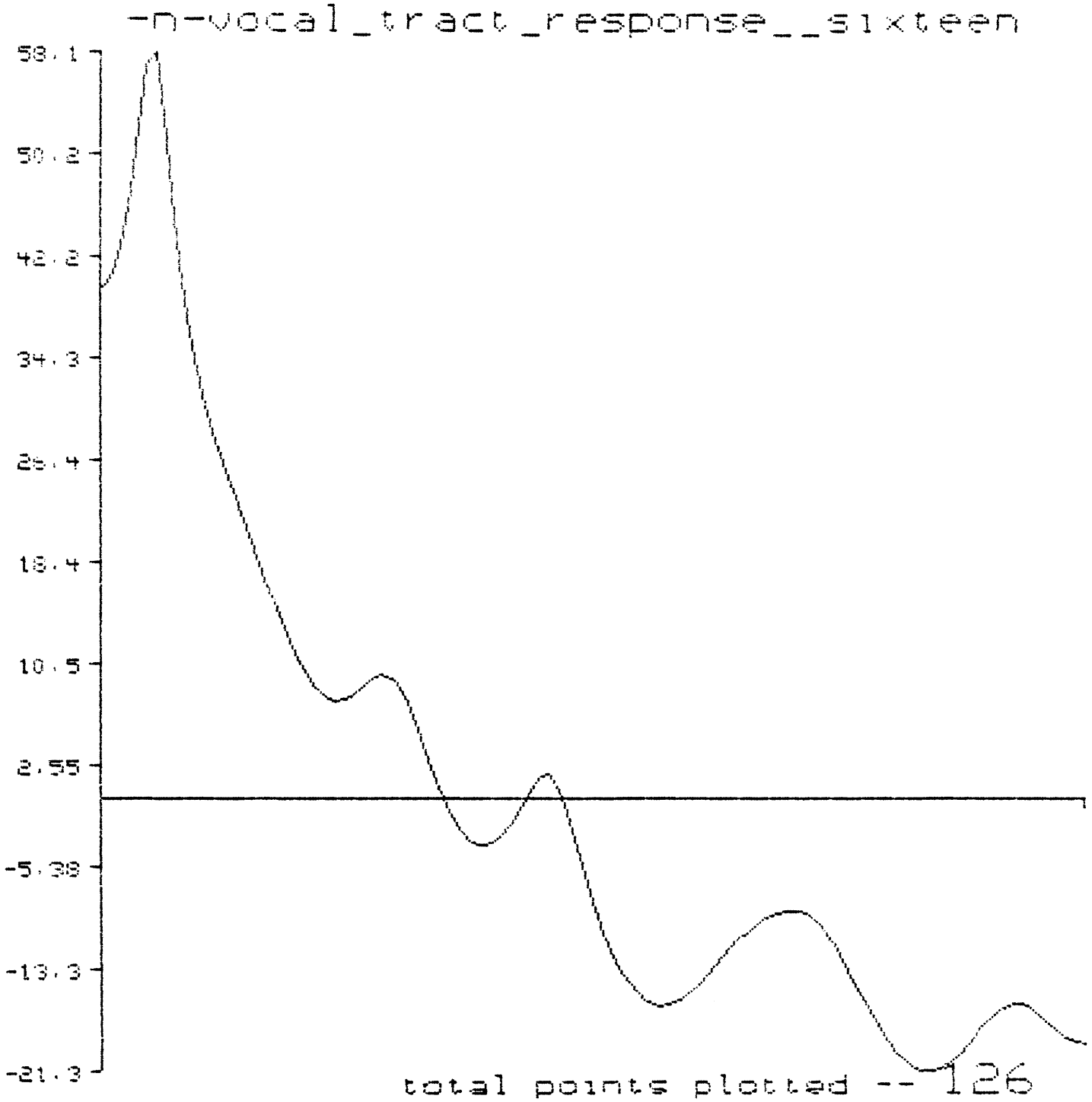
In order to examine the results of the LPC analysis, we wrote a program which calculated and plotted the vocal tract response from the LPC coefficient information. These results can be seen in the plots of the next few pages.



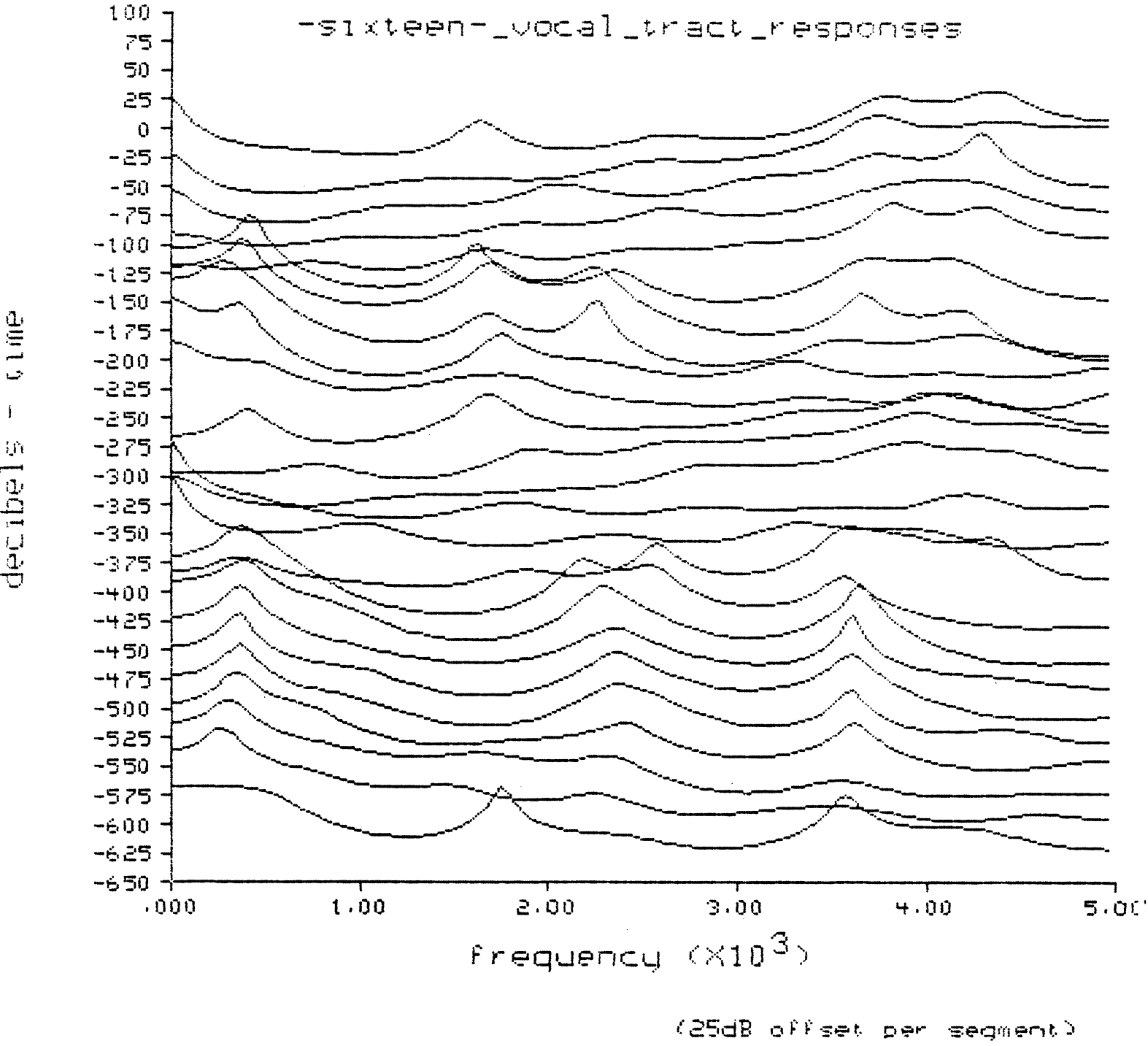
Vocal tract response for the letter "s" in "sixteen"



Vocal tract response for the letter "i" in "sixteen"



Vocal tract response for the letter "n" in "sixteen"



Pseudo three dimensional plot of vocal tract responses throughout the word "sixteen"

4.2. Homomorphic Analysis

A second technique which is sometimes used in the analysis of speech signals is that of homomorphic signal processing. It is suitable for use with systems satisfying a generalized principle of superposition, which are known as homomorphic systems. For speech signals, we encounter what is called a "multiplicative" homomorphic system, [2] in which the generalized "+" operation is multiplication, and the generalized "*" operation is exponentiation.

4.2.1. Application to Speech Processing

Homomorphic processing was applied to the speech signal in order to obtain an estimate of the vocal tract response as follows. We again consider the speech production model to be a slowly time-varying linear system excited by either a semi-periodic impulse train in the case of voiced speech, or by random noise in the case of unvoiced speech. Through homomorphic processing, we are able to "de-convolve" these two signal components (excitation source and system response). This is accomplished as follows.

First, using a DFT on a segment of speech, we transform the time domain input into the frequency domain. In so doing, the operation of convolution is transformed into multiplication. Next, a logarithm of the frequency domain signal is calculated. Since, in general, the frequency domain values will be complex, this must be a complex logarithm. However, we will see later that for our purposes the log of the magnitude is sufficient. The logarithm transforms the operation of multiplication into addition. Finally, an inverse DFT is performed to return the data to a quasi-time domain known as the "quefrency" domain. Since the Fourier transform as well as the logarithm of an impulse train is still an impulse train, what was initially the convolution of an impulse train with a linear system is now the addition of an impulse train to the complex log of the transform of the linear system response. The

inverse Fourier transform of the latter is known as the complex "cepstrum".

4.2.2. Information Obtained

The cepstral peaks resulting from the impulse train are easily found by a "peak picker", and can then be removed from the cepstral signal. The spacing of these peaks corresponds to that of the impulses in the time domain. Obviously, if the excitation source is random noise, these peaks will not be present. Thus the cepstral domain information can be used to indicate first of all whether a segment of speech is voiced or unvoiced, and if it was voiced, to give an estimate of the speaker's pitch period.

We can take the processing one step further, and remove the excitation components by zeroing them out, then perform the inverse operations of DFT and exponentiation to return to the frequency domain. The result is an approximate vocal tract response for that segment of speech. Since we are generally interested only in determining the formant frequencies of the speech, phase information is not required. Similarly, phase information is not required to estimate the pitch period in the cepstral domain. Thus, we may avoid the use of a complex logarithm and the computational problems involved in phase calculation, and simply take the logarithm of the magnitude of the frequency domain signal during the calculation of the cepstrum.

Note that it is also possible to retain only the excitation components of the cepstrally processed speech signal rather than the vocal tract components. Then through a sequence of inverse operations on this information, we can obtain an approximation to the frequency domain and/or time domain signal corresponding to the excitation alone. Further, we can also IDFT the previously mentioned vocal tract response frequency domain information and obtain the impulse response of the vocal tract. These pieces of information would be of some value in a consideration of speech synthesis using a

homomorphic system, but in the case of speech recognition systems, are of little practical use.

4.2.3. Implementation

The homomorphic speech analysis was performed in the following manner. A window length of 300 samples or 30 msec was chosen to insure that more than one period of the signal was included in the segment, yet still be short enough so that the vocal tract response was relatively constant. A Hamming window was applied to the data to reduce side lobes in the frequency domain. A 256 point DFT was performed, and magnitudes calculated.

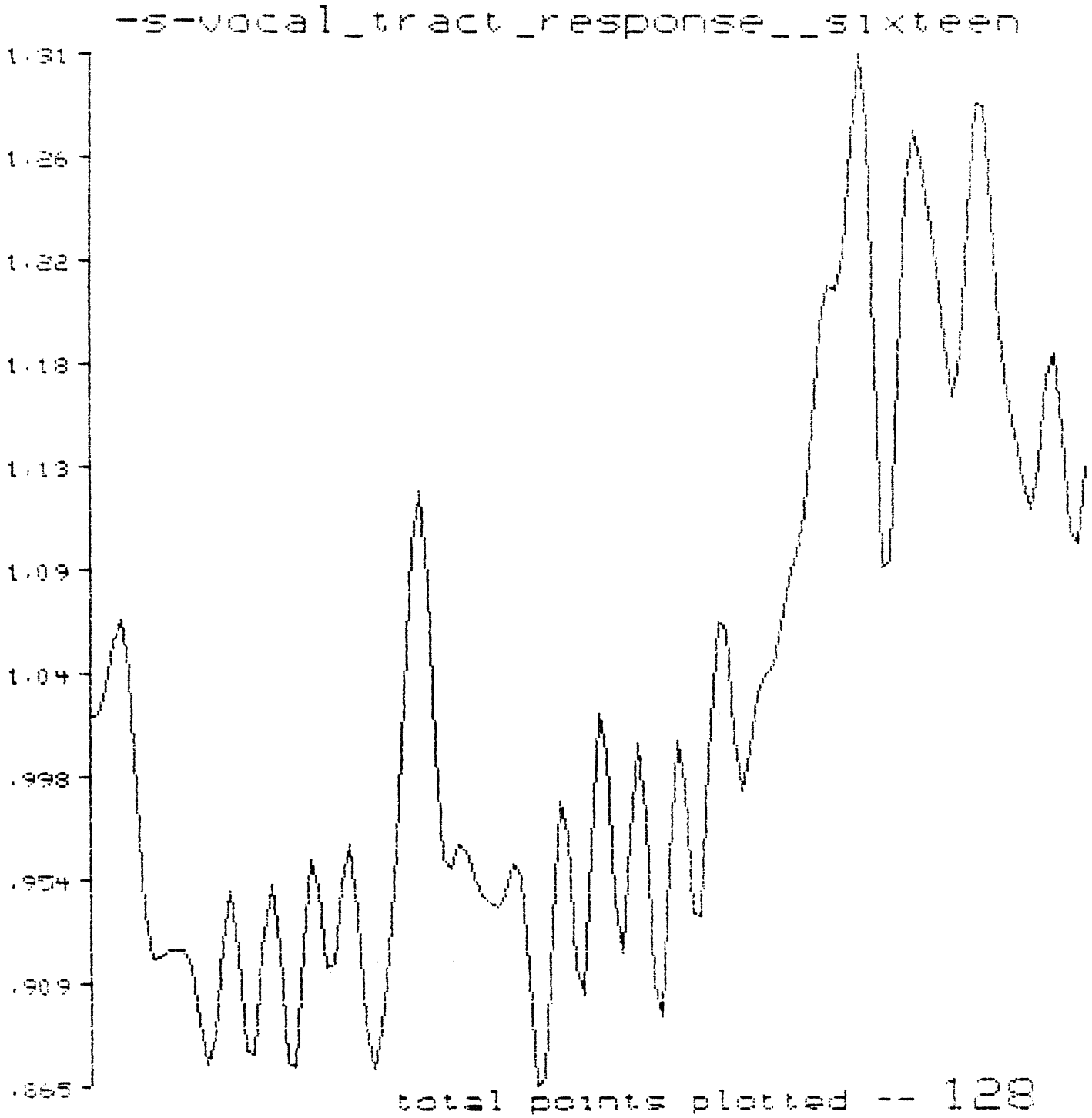
Following this, a logarithm and IDFT were performed. Taking advantage of the symmetry properties of the DFT of a real sequence, the latter was performed only to obtain values of positive "time" or quefrency. Low time filtering was performed by searching for the cepstral impulse occurring at the first pitch period and zeroing out all values included in this impulse and at higher quefrencies. At a later time, this cepstral filtering was changed so that only the low-time data half-way to the pitch period impulse was retained, rather than low-time data all the way up to the impulse.

A magnitude-DFT on the remaining points was performed, and the resulting data plotted out as an approximation to the vocal tract response. Since phase information was not preserved, no attempt at reconstructing the time domain response was made.

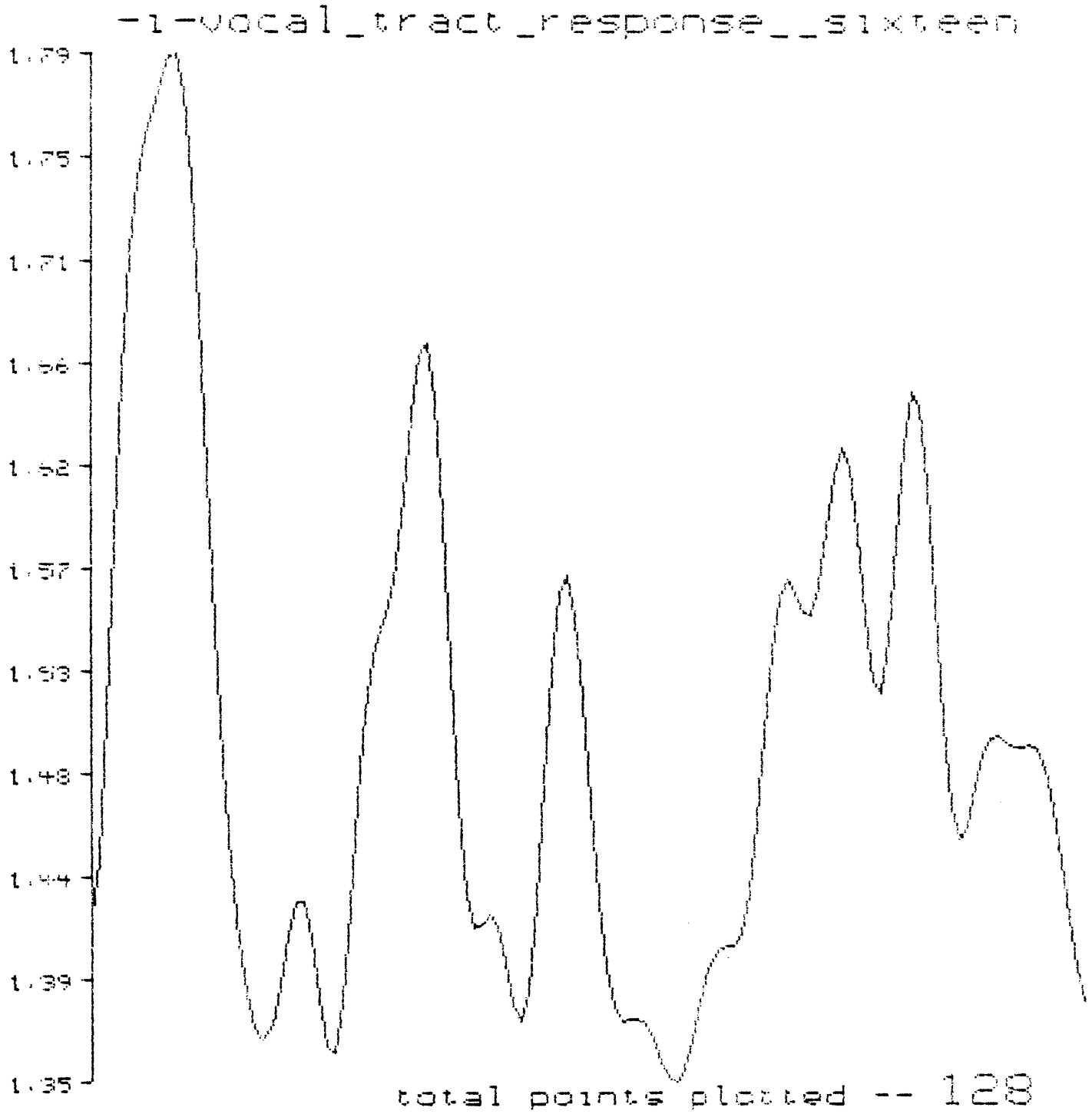
4.2.4. Results

Shown are the results of cepstral analysis on the words ***nine*** and ***sixteen***. The former was chosen because it is entirely voiced, and thus a good candidate for cepstral analysis since a well-defined pitch period should be present throughout the word. The latter was chosen because it contains a mixture of

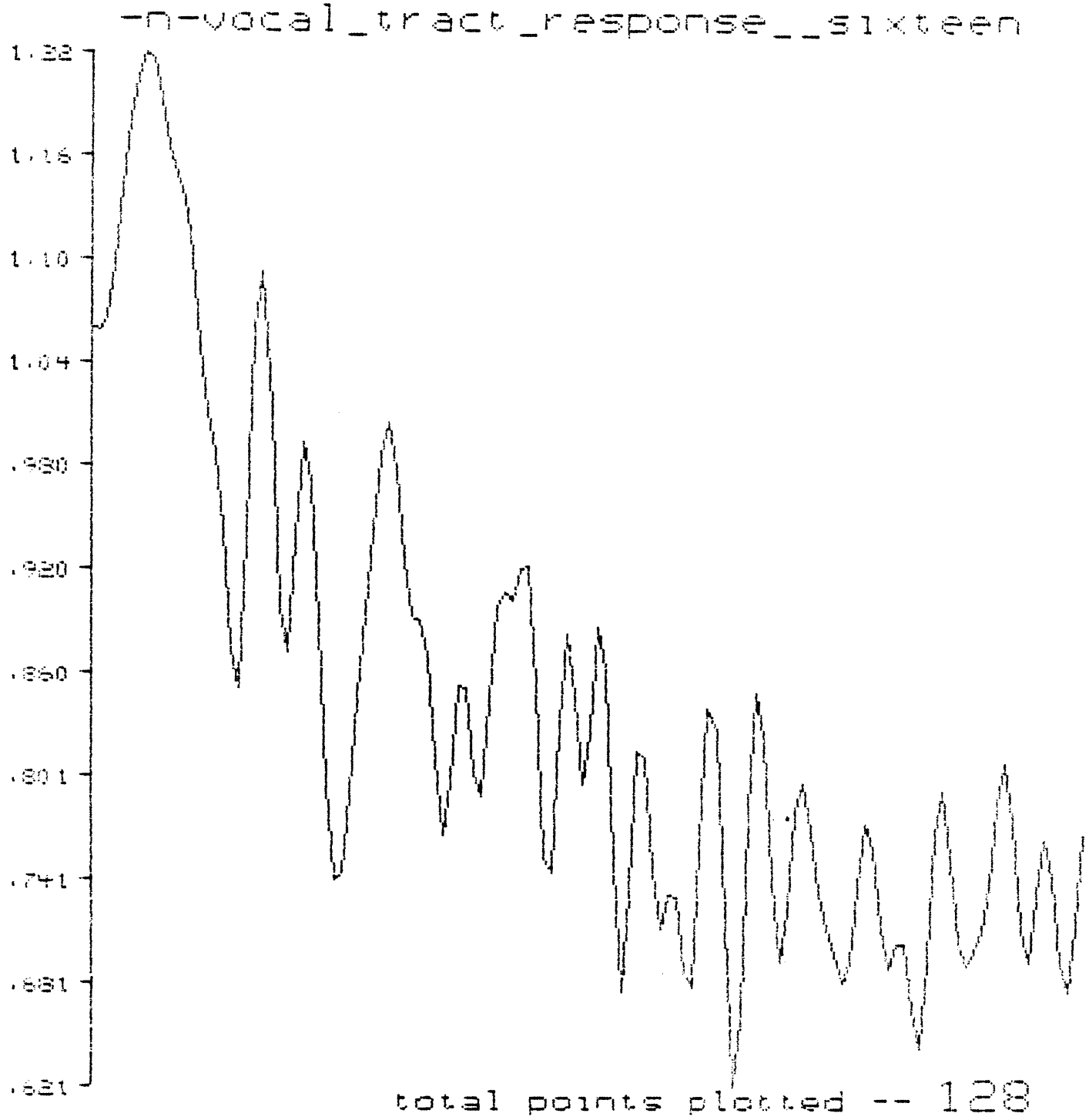
voiced and unvoiced sections. Note that, as expected, the low time portion of the cepstrum contains vocal tract response information for both voiced *and* unvoiced sections, and there are no cepstral domain impulses for the latter. When fewer points were retained in the cepstral domain, the resulting vocal tract response was smoother and retained its peaks in approximately the same locations. However, the overall shape became distorted from the loss of information. Plots of the results appear on the following pages.



Vocal tract response for the letter "s" in "sixteen"



Vocal tract response for the letter "i" in "sixteen"

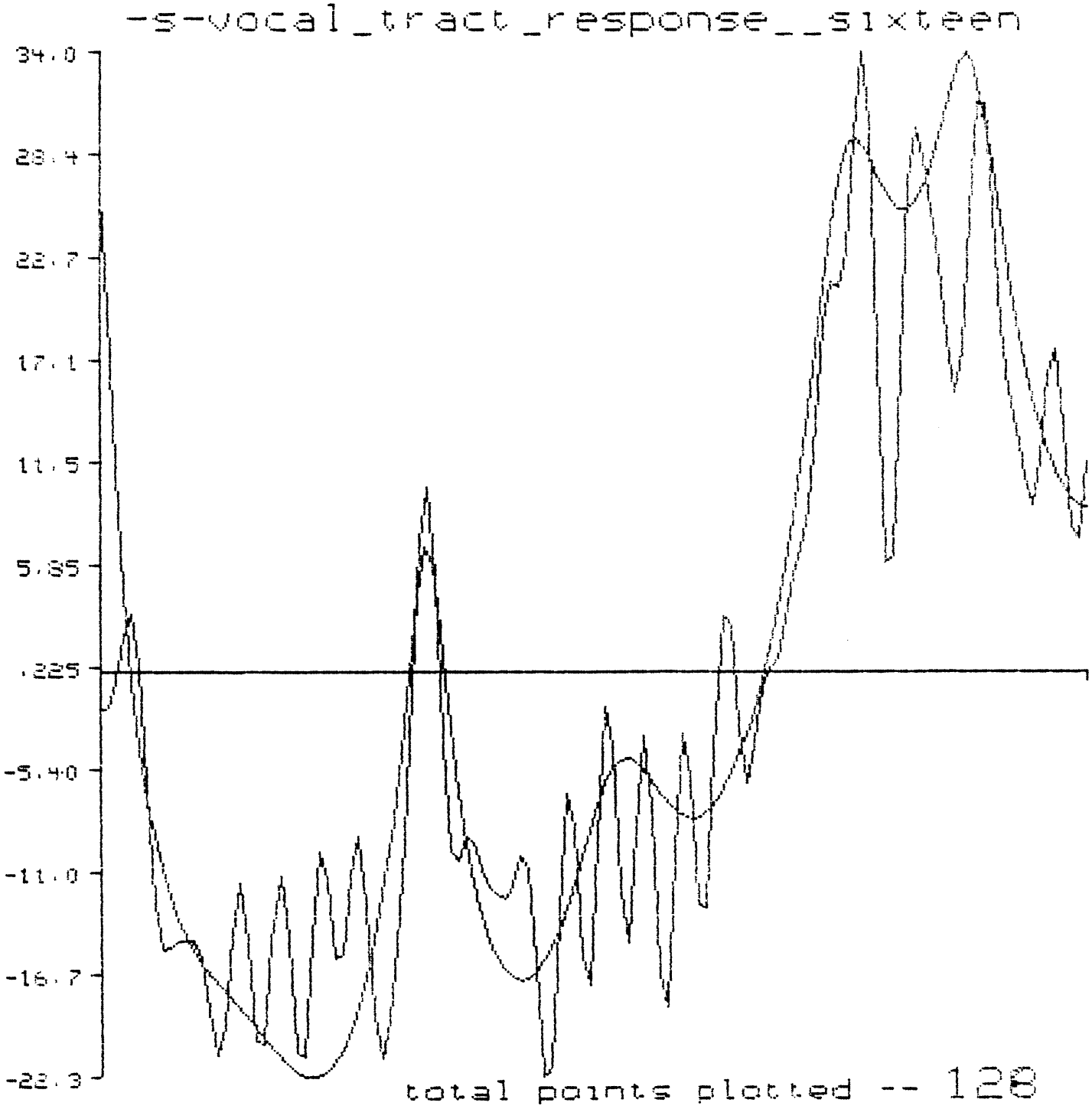


Vocal tract response for the letter "n" in "sixteen"

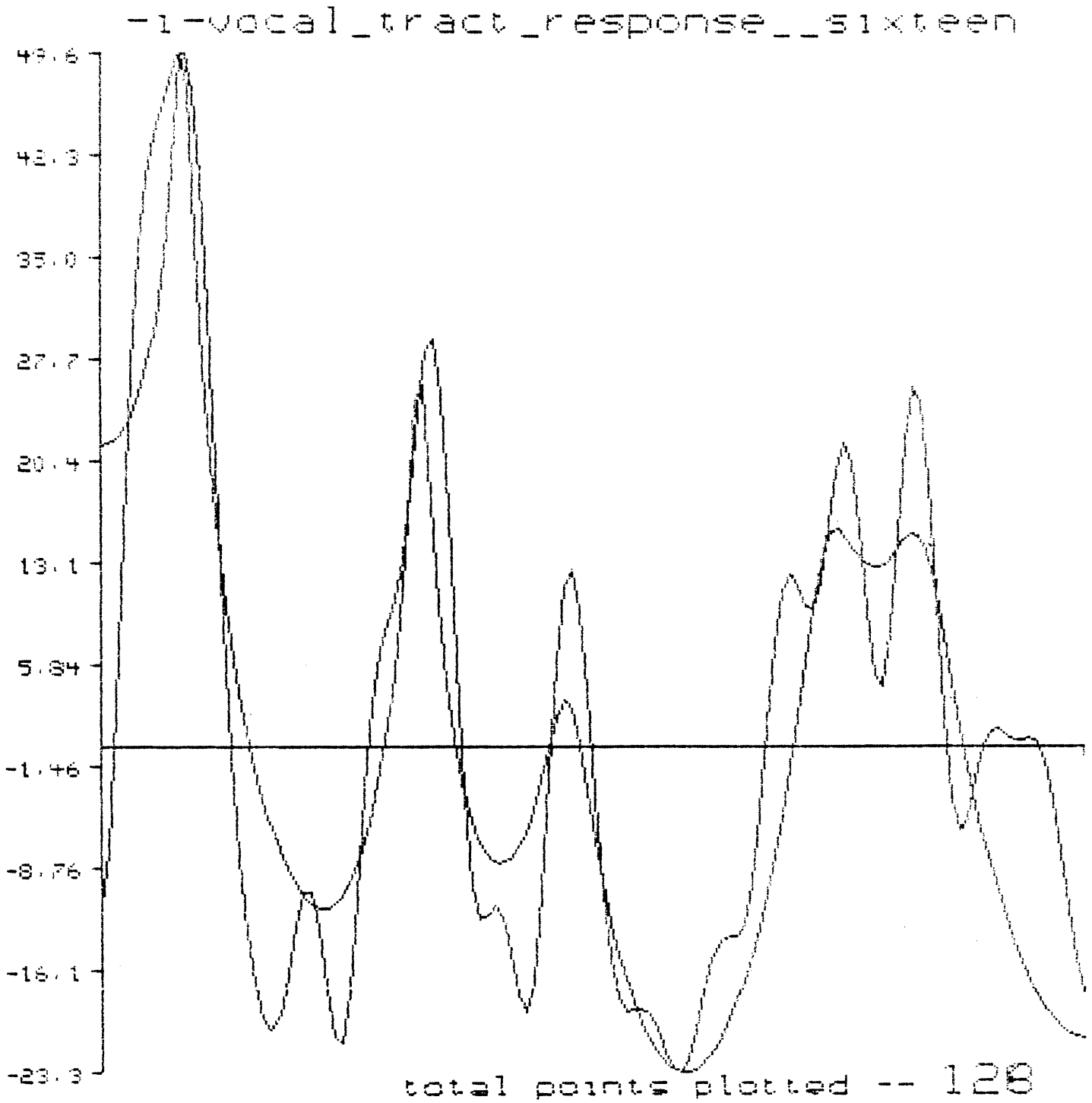
4.3. Comparison of Analysis Techniques

It can be seen from looking at the plots obtained from the two analysis methods presented in this report that estimates of the vocal tract response from them are similar. Indeed, one of the reasons for implementing both methods was that they would allow us to check results against each other to insure that they are reasonable and correct. Overlayed plots presenting these results simultaneously are shown on the following pages.

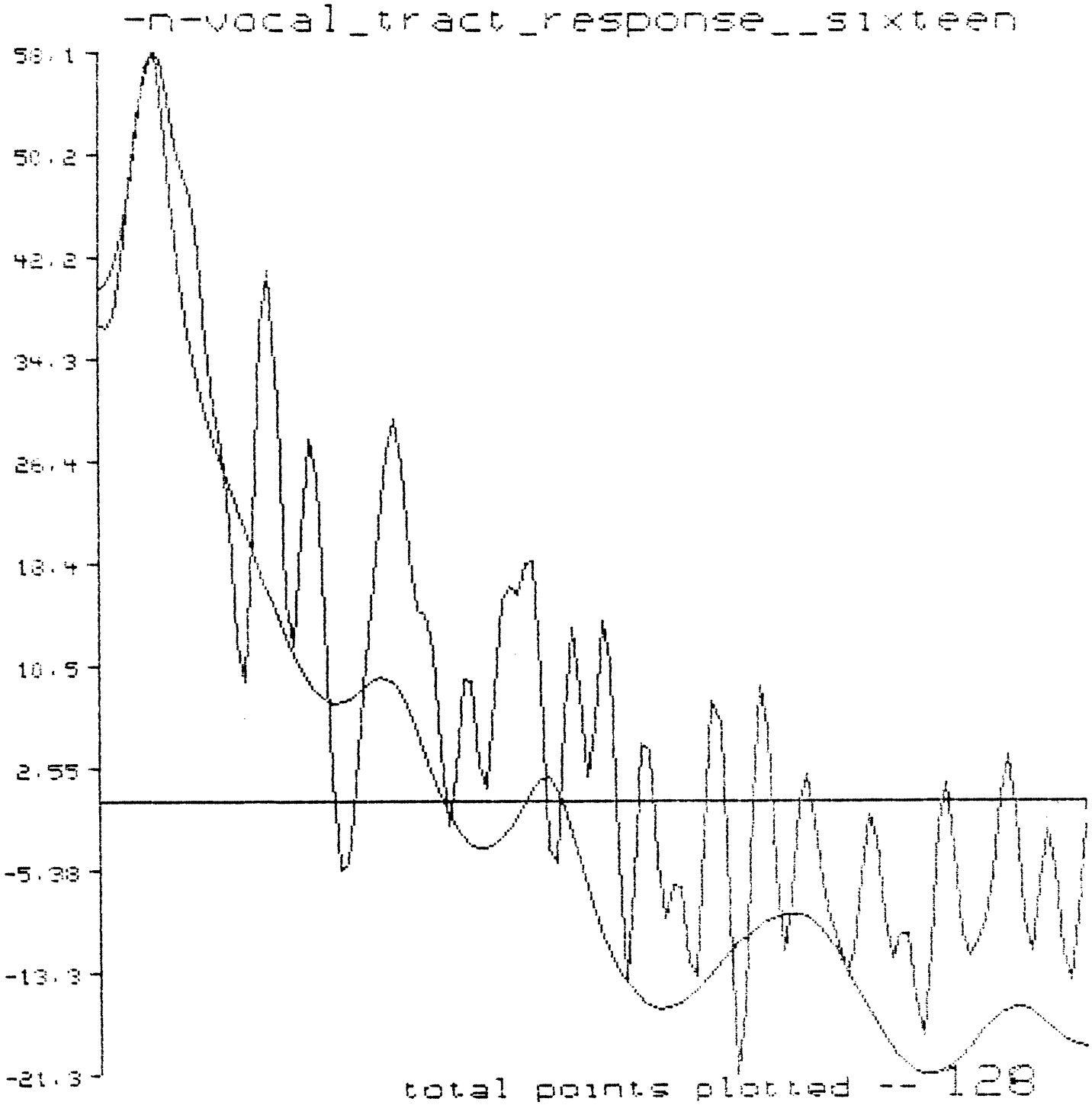
The LPC analysis is better able to characterize a sample of speech with a smaller amount of storage space than is required with homomorphic analysis. Pitch period and voiced/unvoiced information must be stored for either. However, LPC analysis requires the storage of 14 predictor coefficients for each of the 25 frames of the word, whereas storage of the time domain vocal tract response obtained through the cepstrum takes much more storage space. Computationally, it is believed that homomorphic analysis implemented with FFT's is more efficient than LPC analysis. The final decision regarding which analysis technique would be used for the recognition implementation was based on the availability of a relatively simple and robust decision technique based on distance measures for the LPC parameters. This topic will be the subject of the next section.



Combined homomorphic and lpc vocal tract responses. Smooth line is lpc.



Combined homomorphic and lpc vocal tract responses. Smooth line is lpc.



Combined homomorphic and lpc vocal tract responses. Smooth line is lpc.

5. Distance Measurement

In order to perform recognition of an unknown word, some measure of similarity is needed to compare it to each of the words known to be in the vocabulary. If this measure of similarity takes the form of a probability that the unknown word is the same as a given word of the vocabulary, the obvious rule is to decide that the unknown word is actually that which for which the probability measure is greatest. In the case of LPC analysis, the probability measure reduces to calculating a distance measure between the unknown word's LPC parameters and those stored as "templates" for each of the 100 words in the entire vocabulary.

5.1. Derivation

The origin of the distance measure is as follows [2]. We maintain the assumption of an all pole speech production model excited by white noise or by a quasi-periodic impulse signal. In this case, the probability distribution of the LPC estimates $\boldsymbol{a}'$ is a multi-dimensional Gaussian distribution whose mean is $\boldsymbol{a}$ and covariance matrix is $\mathbf{\Lambda}$ so that the probability of obtaining $\boldsymbol{a}'$ when the actual coefficients are $\boldsymbol{a}$ is:

$$p(a'/a) = \frac{1}{\sqrt{(2\pi)^p[\Lambda]}} \exp \frac{(a'-a)\Lambda^{-1}(a'-a)^t}{2}$$

where[$\mathbf{\Lambda}$] is the determinant of the matrix $\mathbf{\Lambda}$. The log probability (neglecting the constant factor due to the leading term) is:

$$(a'-a)\Lambda^{-1}(a'-a)^t = D(a',a)$$

That this is a reasonable distance measure is easily seen, because the closer the $\boldsymbol{a}'$ values are to the $\boldsymbol{a}$ values, the smaller the above calculated dis-

tance, and the greater the probability that they came from an utterance with true LPC parameters $\boldsymbol{a}$. The covariance matrix $\boldsymbol{\Lambda}$ is defined as;

$$\Lambda = \frac{R^{-1}}{N} \times (a\, Ra^t)$$

so that:

$$D(a',a) = (a'-a)\left[N\frac{R}{a'Ra'^t}\right](a'-a)^t$$

where $\boldsymbol{R}$ is the (p+1) by (p+1) correlation matrix of a speech segment of length N.

Since the true underlying LPC coefficients $\boldsymbol{a}$ are not known, they are estimated by performing LPC analysis over several repetitions of the word. It is then assumed that the distribution of the $\boldsymbol{a}$ parameters is Gaussian with mean $\boldsymbol{m}$ and covariance matrix $\boldsymbol{S}$, which are estimated from the J multiple repetitions as:

$$m(n) = \frac{1}{J}\sum_{1}^{J} a'_j(n) \qquad n = 1,2,\ldots p$$

and

$$s(n,p) = \frac{1}{J}\sum_{1}^{J} a'_j(n)a'_j(p) - m(n)m(p)$$

This information is used to form a total covariance matrix $\boldsymbol{C}$ replacing Λ where:

$$C = S + \Lambda = S + \frac{R^{-1}}{N}(a'Ra'^t)$$

then the distance measure becomes:

$$D(a',a) = (a' - m)C^{-1}(a' - m)^t.$$

A simpler distance measurement has been proposed by Itakura. This measurement takes the form:

$$D'(a',a) = \log\left[\frac{aRa^t}{a'Ra'^t}\right]$$

This can be seen to be closely related to the first distance measurement presented above. This measure was considered for use in the recognizer because of its increased simplicity and greater speed. However, further investigation yielded several sources [2,5,6] commenting on incorrect assumptions made in the derivation of the formula, and a loss in recognition accuracy when it is used.

5.2. Implementation

5.2.1. Template Generation

Close investigation of the distance measure which was used reveals that, if the only information stored for the vocabulary templates are the sets of LPC coefficients, a great deal of computing is required for distance measurement during recognition. Thus, the elements of recognition which depend on the known utterances, namely the mean vector $\boldsymbol{m}$ and covariance matrix $\boldsymbol{S}$, were precomputed and stored as the templates to save time during recognition.

Additionally, a "label" was appended to the end of each template file. In our case, the label simply contained a character string to enable printing out the word that was recognized during program execution. In an actual system,

this label may be a word of data to be entered into the system, or possibly an instruction or command to be executed.

The template files were generated with the program "tesplate" and were a total of 24010 bytes in length. This data represented the following:

14 words x 25 frames x 4 bytes for $\boldsymbol{m}$'s

15 x 15 words x 25 frames x 4 bytes for $\boldsymbol{S}$'s

10 bytes for label

Obviously, such storage requirement make this method impractical for large vocabularies on small systems. For example, template storage for a simple 100 word recognizer is 2.4 Mbytes (!).

The inputs to the "tesplate" program are the known utterances of a word. The program performs the LPC analysis previously described in section 3.1. After this analysis has been performed on all repetitions of the word, these results are combined to form the $\boldsymbol{m}$ vector and $\boldsymbol{S}$ matrix of the template.

5.2.2. Initial Distance Measurements

In order to gain a "feel" for how the distance measurement would work for comparing words to both correct and incorrect templates, the program "testdis" was written. This program generates a frame by frame account of the present distance and cumulative distance it has calculated for specified word/template pair. This information was required for establishing stepsize and offset parameters to be discussed in the recognition section. Also, it provided us with a little encouragement that the whole system would work when we finally got all the pieces put together.

Distance calculation (both here and in the case of actual recognition) requires matrix multiplication and inversion. Since these operations are always performed on the same size matrices, the former calculation was easily included in the distance program itself. However, the latter operation was dependent on the use of the NAAS (Numerical Analysis and Statistics) package on the VAX, in particular the subroutines SLUD (for matrix decomposition) and SILU (for the actual inversion). Had these not been available, distance calculation programming would have been (to say the least) nontrivial.

5.3. Results

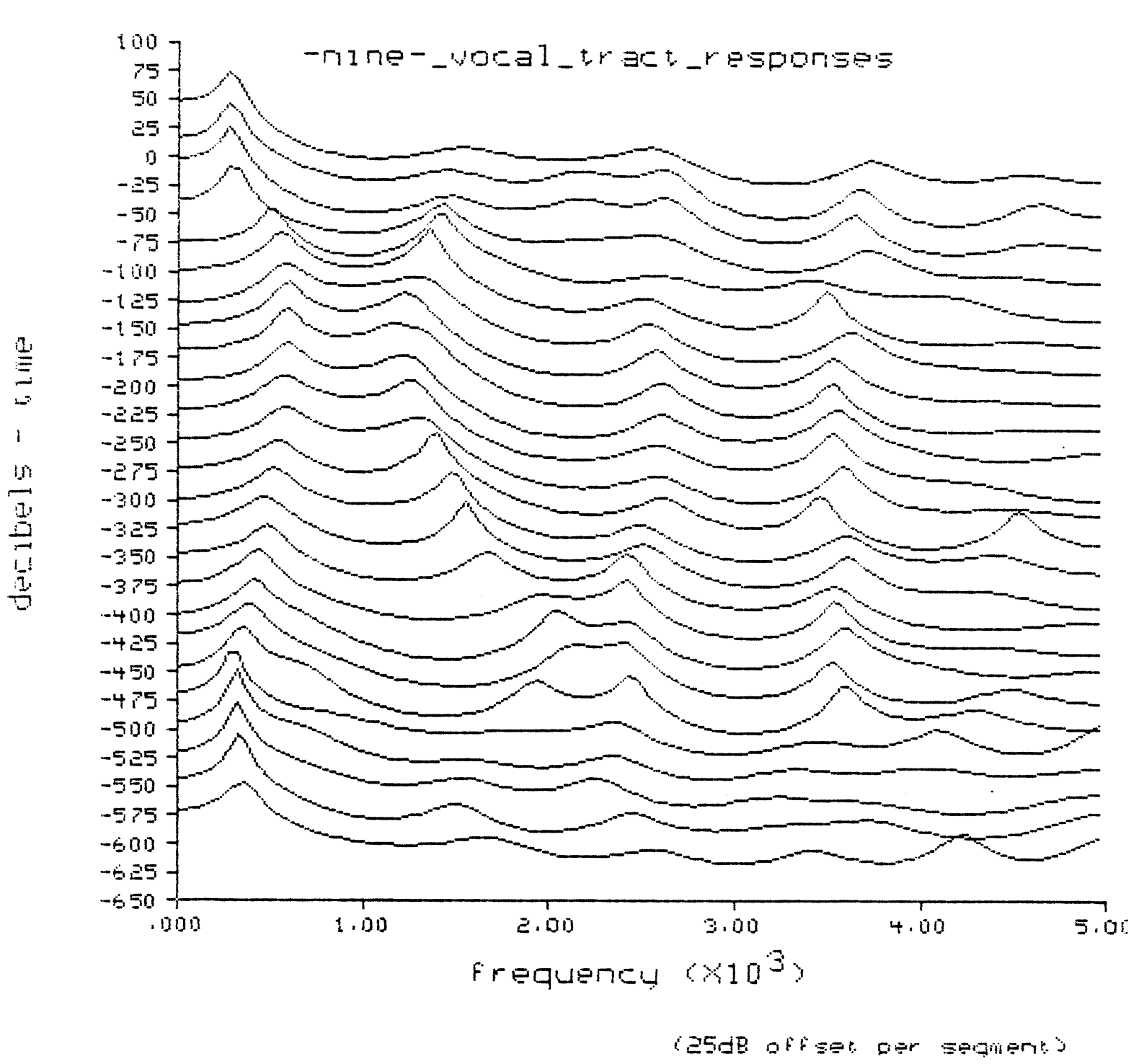
The results of the testdis program were received with mixed feelings. On the one hand they were encouraging because most often when the word was compared to the wrong template, relatively large distances were calculated. On the other hand, there were several instances noted for which a word would in general yield small distances when compared to the proper template,. yet for a few frames out of the 25 large distances would be calculated, so that the overall distance was larger than was expected. This was attributed to two primary causes:

1) The use of the linear time warp instead of a nonlinear time warp, and

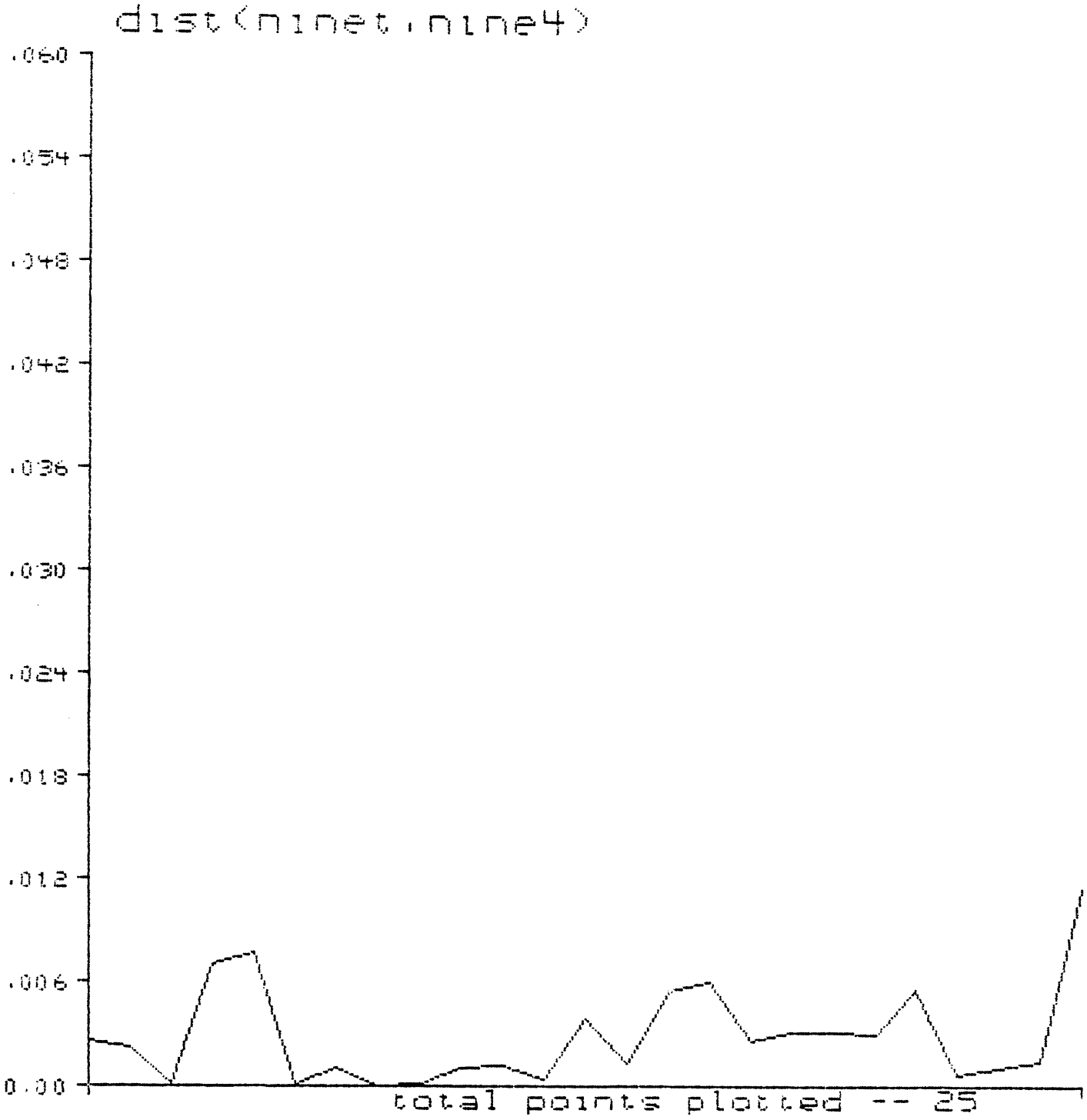
2) Inclusion of artifacts such as breath noise, tongue clicks, or microphone contact noise either in the unknown speech or in the template utterances.

The former could be corrected at the cost of increased processing time, whereas the latter would require use of more utterances in the template (to average out random artifacts) or a different recording technique (speak more slowly, use a microphone stand, etc.). However, insufficient time was available to pursue either of these avenues of investigation.

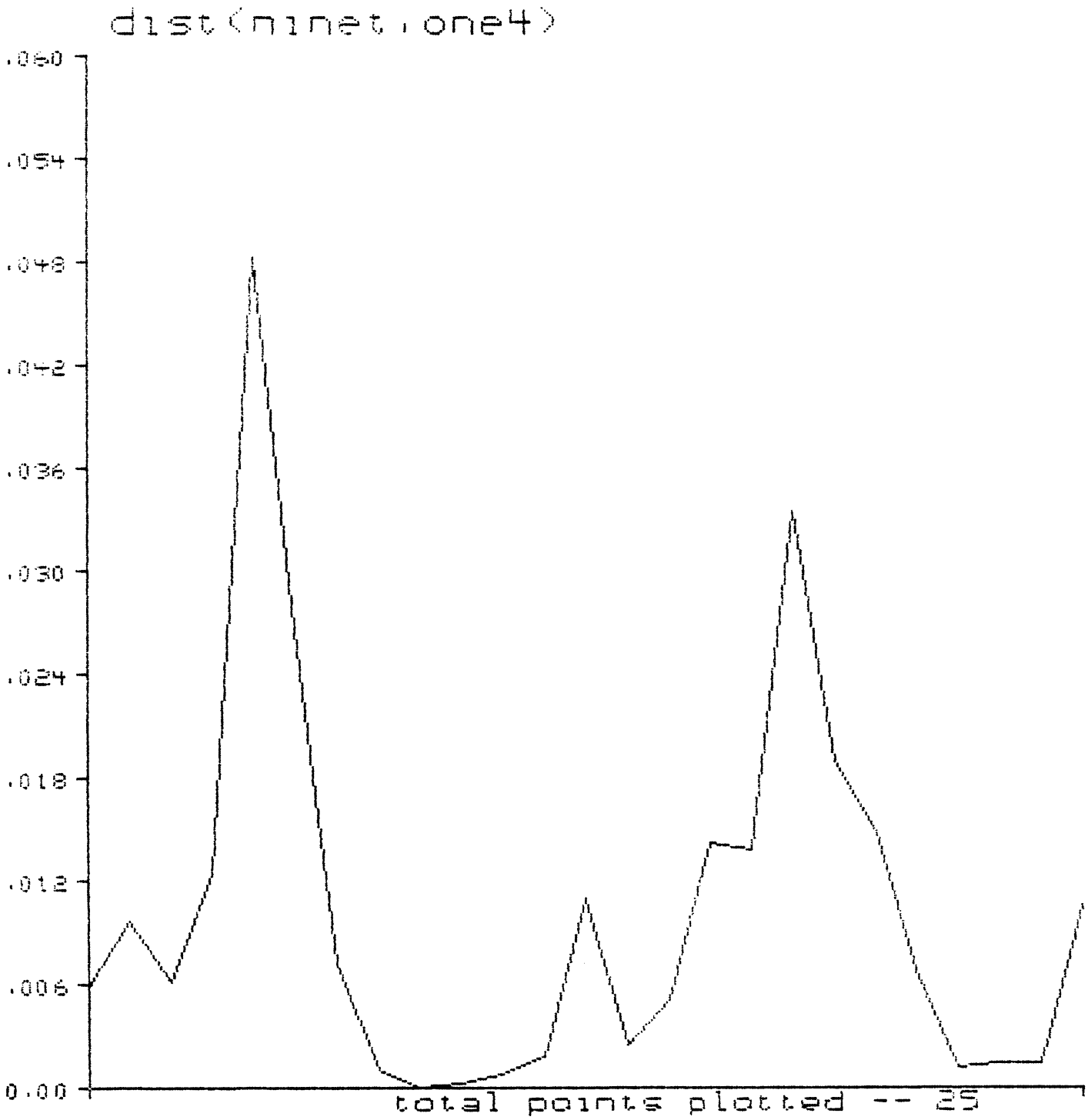
Several examples of distance calculation using "testdis" are shown on the following pages.



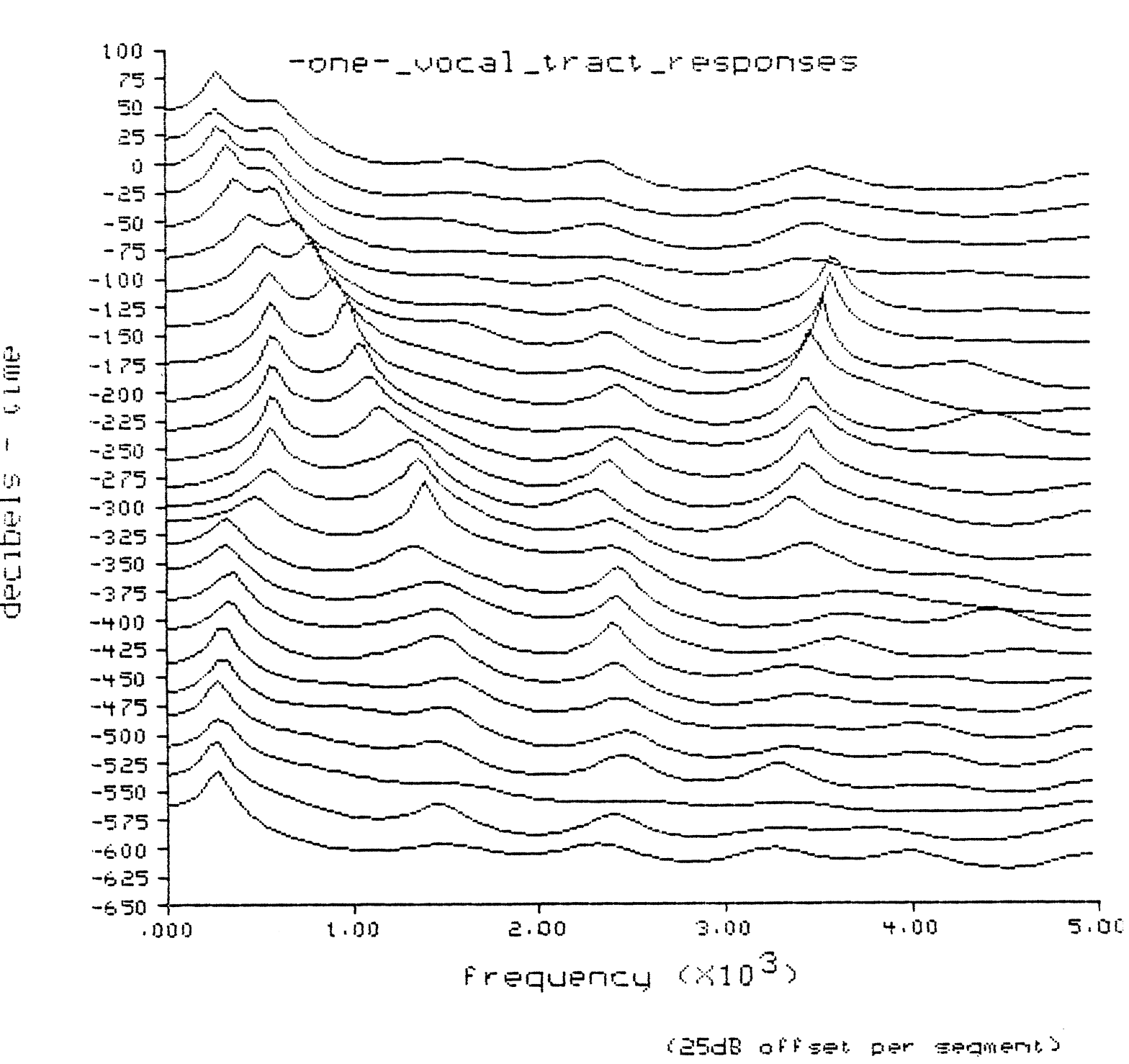
Pseudo three dimensional plot of vocal tract responses throughout the word "nine"



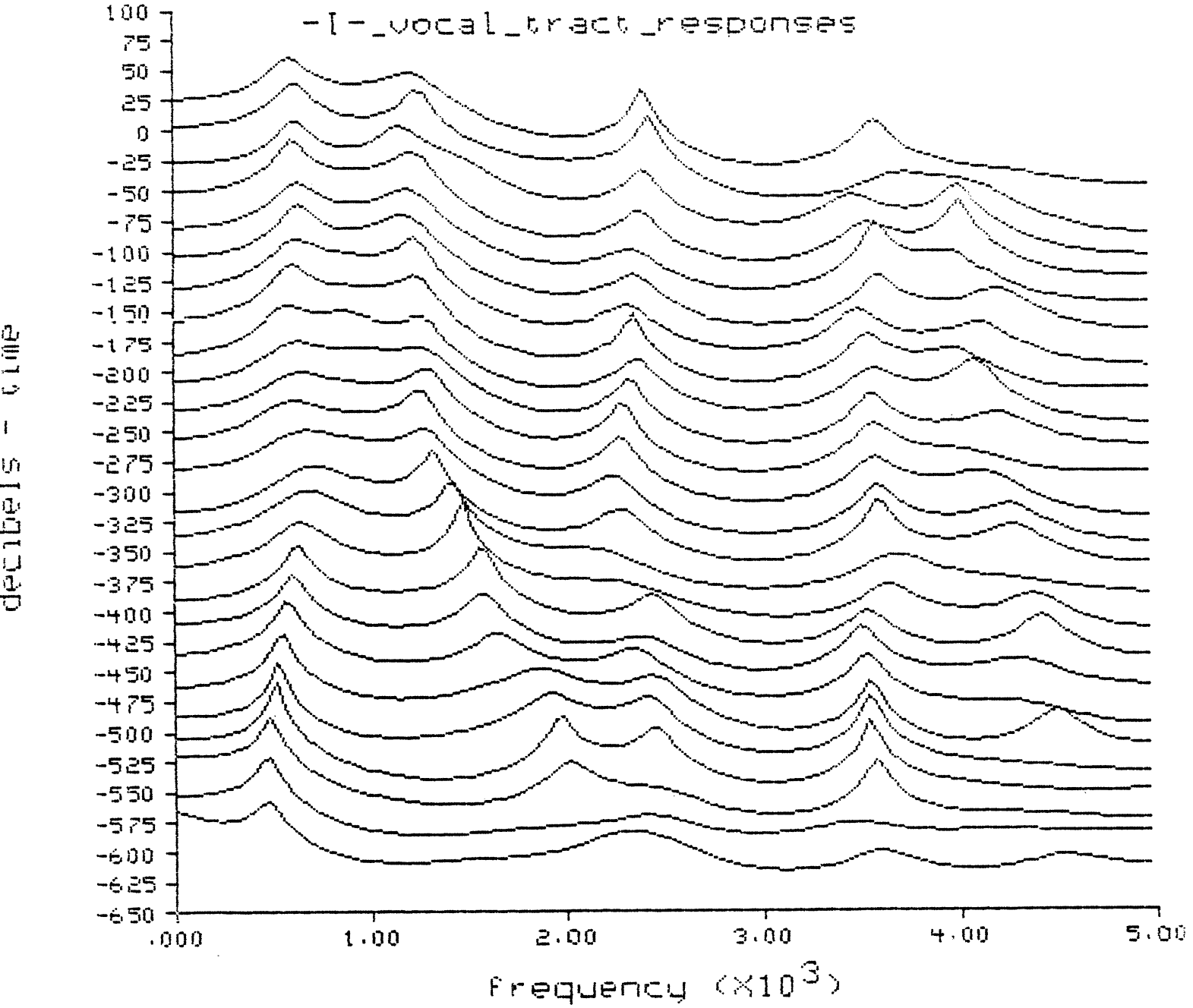
Frame distance versus frame number for the distance between the word "nine" and the template for "nine"



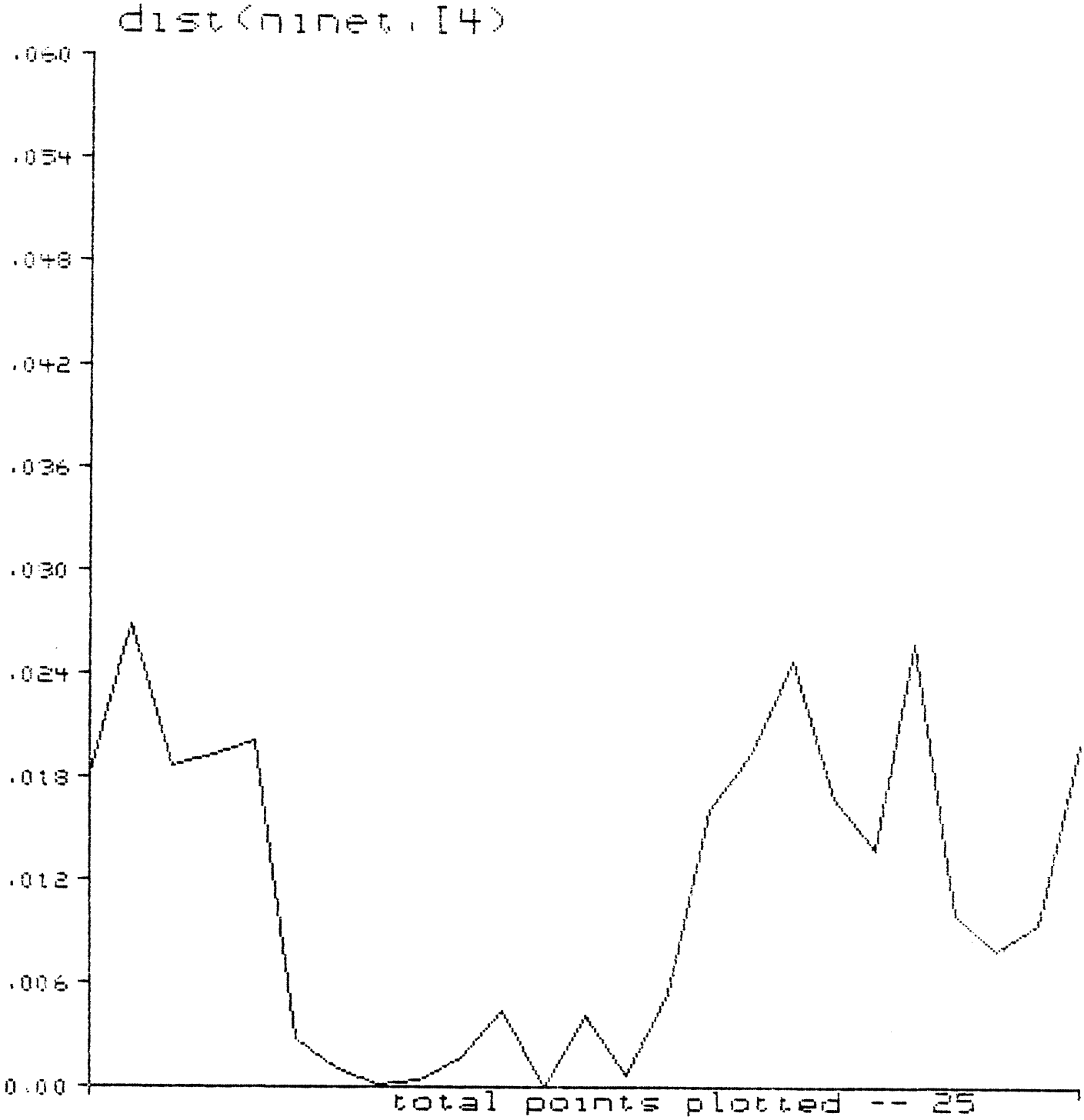
Frame distance versus frame number for the distance between the word "nine" and the template for "one"



Pseudo three dimensional plot of vocal tract responses throughout the word "one"



Pseudo three dimensional plot of vocal tract responses throughout the word "I"



Frame distance versus frame number for the distance between the word "nine" and the template for "I"

6. Recognition

Once all the preceding work had been performed, the actual recognition was a matter of combining the templates generated for all the words into a large template "vocabulary," and writing a recognition program which could compare input words against the vocabulary and inform the user of the most likely candidate for recognition.

6.1. Implementation

The final recognition program was called "recognize," and performed as follows:

1) The unknown word was read in, linearly time warped into the standard 25 frame format.

2) On each frame, LPC analysis was performed, and the autocorrelation matrix R was calculated, then inverted (as required for distance calculation).

3) The algorithm of testdis was performed for all 100 combinations of the unknown word and the stored templates. This involved calculating a cumulative distance, then reading the template label.

4) The calculated distance was compared to those already calculated and, if small enough, the distance and label was stored for possible later output.

5) The process was repeated from step 3 until all combinations had been examined.

6.2. Improvements to Basic Program

In order to increase the run-time efficiency of the program, the following changes were made. First, if the total distance calculated at any point within a word exceeded the best distance calculated up to that point, no further processing on that word was performed. Secondly a moving threshold or stairstep threshold was set. If the cumulative distance exceeded this threshold, no further processing was performed. The threshold was of the form:

$$thresh = offset + (frame\#) \times (stepsize)$$

where offset and stepsize were heuristically determined from information obtained during execution of the testdis program. There is, of course, a trade-off involved in setting these values. If they are too small, processing may be erroneously terminated right at the beginning of a word. Too large, and they do not increase the efficiency.

No direct comparison of recognition time was made between the basic program and the modified program. However, through information obtained about the expected cumulative distances for incorrect word/template pairings, it is known that the moving threshold eliminated a significant number of words as candidates.

7. Results and Conclusions

Once the recognition routine was up and running, we undertook to recognize some words in order to verify the algorithm. As mentioned earlier, one utterance of each word was not used in making the template for that word. These remaining utterances were used for recognition. Recognizing each of these words is far from an exhaustive test of the recognizer. Indeed an

interesting experiment would have been to use just one utterance of each word for the template and then to recognize the much larger group of words that would have been left over. Unfortunately, due to limitations of time, this has not been done.

In order to get a better idea of the performance of the recognition algorithm, a second, slightly altered, form was compiled which informed the user of the top three candidates for recognition. The results discussed here were obtained using this program, called "toprec."

Upon obtaining the recognition results, we found that 11 errors had been made. In eight of those cases, however, the correct word was the second choice. Most of these errors were made in confusing highly similar pairs of words such as "filter" and "theta," "goes" and "cords," "sine" and "shine," and "if" and "with." One of the errors was made with the correct word coming in third, confusing "seven" with "sir" and "sine." In the other two errors, the correct word did not make the top three. In almost all of these cases, errors were mainly due to three to five relatively large frame distances in comparing the word to the correct template. This can be attributed to the poor matching of word parts due to the use of linear time warping.

A significant pattern in examination of the results is that those words containing significant portions of voiced sounds with strong nasal coupling or glides were, in general, much closer to being confused (i.e. the first and second candidates were much less widely separated according to our distance measure) than were words in general. This should not be surprising since, as was pointed out earlier, LPC analysis has a much more difficult time characterizing nasals, which have significant anti-resonances, and glides, which are relatively non-stationary.

Despite its problems, this initial experiment shows some promise for this type of word recognizer. The success rate of 89% in this case is less than spectacular (although, as stated before, this success rate is not necessarily a good characterization of the system) but a number of relatively straightforward techniques may be used to improve upon it. One of the most obvious improvements would be to use a non-linear time warping scheme wherein, based upon incremental distance measures, a decision would be made to compress or expand the next segment of a word as one steps through it in the time warping process. Another improvement might be to use a two-pass recognizer which would first classify a word as belonging to some easily confusing set then to distinguish among the members of the set by emphasizing, in the distance measurements, the portions of the words which differ most obviously (such as the beginning fricative in "sine" and "shine"). This approach has been used with some success by Rabiner and Wilpon [7]. Finally, a simple improvement would be to use a matrix inversion routine which takes advantage of the symmetry of the autocorrelation matrices used in the distance measures thus providing a significant improvement in efficiency.

One of the major disadvantages of this recognition scheme is the large amount of real time required to do the recognition. Each set of LPC coefficients takes a significant amount of time to calculate (although they are only calculated 25 times for each input word). Also, the distance calculation requires the inversion of 25 15 by 15 matrices for the input word and one 15 by 15 matrix each time a distance measure is made between a frame of the input and a frame of a template. Special purpose hardware can help in this regard, but is probably not a panacea. This lack of efficiency can only become worse as the vocabulary size is increased.

The problem of speed is one of the most difficult to overcome in modern speech recognition algorithms. This problem has been largely resolved in

systems based on short-time Fourier analysis where, through use of analog filter banks and modulators along with high speed special purpose digital hardware, the digital processing time has been reduced to a minimum. Much progress must be made in systems of the type discussed here before they will be able to compete effectively with these systems.

8. References

[1] F. Itakura, "Minimum prediction Residual Principle Applied to Speech Recognition," *IEEE Trans. ASSP*, Vol. ASSP-23, pp.169-176, 1975.

[2] L.R. Rabiner and R.W. Schafer, *Digital Processing of Speech Signals*, Prentice-Hall, 1978.

[3] G.M. White and R.B. Neely, "Speech Recognition Experiments with Linear Prediction, Bandpass Filtering, and Dynamic Programming," *IEEE Trans. ASSP*, Vol ASSP-24, pp. 183-188, April 1976.

[4] L.R. Rabiner and M.R. Sambur, "An Algorithm for Determining the Endpoints of Isolated Utterances," *BSTJ*, Vol. 54, No. 1, pp. 81-102, January 1975.

[5] P.V. deSouza, "Statistical Tests and Distance Measures for LPC Coefficients," *IEEE Trans. ASSP*, Vol. ASSP-25, No. 6, pp. 554-559, December 1977.

[6] V.N. Gupta, J.N. Gowdy, and J.K. Bryan, "Evaluation of Some Distance Measures for Computerized Voice Recognition," *IEEE SECON '77*, pp. 423-426, 1977.

[7] L.R. Rabiner and J.G. Wilpon, "A Two-Pass Pattern-Recognition Approach to Isolated Word Recognition," *BSTJ*, Vol. 60, No. 5, May-June 1981.

9. Appendix A: Vocabulary List

one	are	fluff	pitch	these
two	I	how	which	buzz
three	see	after	tape	switch
four	a	kite	sine	will
five	can	plum	viscous	asp
six	you	goes	duck	tau
seven	read	shine	hertz	noise
eight	it	round	fishy	low
nine	so	king	cords	pierce
ten	sir	food	glottis	Bell
eleven	point	theta	tongue	Labs
twelve	stop	with	best	Dan
thirteen	minus	Delp	common	George
fourteen	wrong	Tim	tract	Mary
fifteen	begin	John	pressure	Pat
sixteen	times	damped	window	square
seventeen	do	vocal	pharynx	lack
eighteen	power	still	filter	thats
nineteen	if	oral	each	all
zero	equals	flow	valid	folks

10. Appendix B: Median Filtering

Median filtering is an ad hoc procedure designed to remove sharp discontinuities due to shot noise, incorrect data transmission, etc. The median filter is a non-linear system which replaces each point in the file with the median of the points in some window oriented around the point in question. In this way, most slopes should be preserved (although perhaps somewhat distorted) and large, single point discontinuities would be removed.

The median filter can often be optimized for certain situations by the following technique. A threshold can be set such that the point will be replaced only if it differs from the median by more than the threshold. In this way, only large discontinuities are removed.

Our algorithm used a window size which could be set on the command line. Each point was replaced by the median of the N points prior to that point. No attempt was made to set thresholds, etc.

The results can be seen in the plots of the next few pages. As can be seen, the filter performs quite well in voiced regions, preserving the waveform. In the fricative portions, performance is somewhat poorer, particularly with larger window sizes. The plosive, however, was severely attenuated regardless of the window size. While this is not surprising due to the sharp discontinuity introduced by a plosive, it is nonetheless distressing. The nature of plosives is such that even the optimizing techniques discussed above cannot be expected to significantly improve performance. For this reason, we do not recommend the use of a median filter for speech processing in general, although it may be useful for certain situations.

11. Appendix C: Description of Software

The following is a list of all the major programs which were used for our speech recognition project. Programs are listed along with their command lines. The user may invoke these programs without command line arguments, in which case a description of the command line will be returned.

append infile1 infile2 ... infileN outfile

This program is used to append template files into one large vocabulary template file. This large file is used by recognize.c as the dictionary.

cct.sub

This subroutine concatenates strings in order to make command line entries simpler.

cepstrum infile

This program homomorphically processes word input, cepstrally filtering to remove the periodic portion of the speech wave. The processing is done on the same 25 equally spaced frames that the lpc analysis uses. The outputs are the files "dftout" which contains the 25 dft's of the input word, "cepout" which contains the 25 cepstra of the input word (before filtering), and "respout" which contains the 25 log magnitude vocal tract response curves for the input word. The algorithm does not use an FFT (although some optimization has been done by taking advantage of the special properties of the information we were processing) and is thus rather slow. The filtering is done by finding the largest peak in the cepstrum then removing it and all nonzero values in higher "time."

cfilter.sub

This subroutine does the cepstral filtering for cepstrum.c.

clean infile outfile start end

This program is used to clean up poorly segmented words. The inputs are the input word (infile) the output file and the segment of the word the user wishes removed. The segment is specified by giving the first and last points of the segment. The start of the file may be specified by "s" and the end by "e."

convert infile outfile nbytes

This program takes a file of numbers in the format gotten from the bioengineering A/D converter and converts it into a file of 2's complement short integers. The input file, output file, and number of bytes in the input file must be specified.

dbmag infile startword outfile

This program takes the lpc coefficients for a segment of a word and computes the log magnitude spectrum from them. The spectrum is then placed in outfile in the form of 4 byte floating point numbers.

These may then be plotted. The inputs are input data file, position of segment (each segment contributes 14 lpc coefficients, thus position is given as a multiple of 14) and output file.

dft.sub

This subroutine is called by cepstrum.c and is used to calculate the log magnitude of the dft of a segment of a word.

distance.sub

This subroutine calculates the distance between a word frame and a template frame. It is called by the programs testdis.c, recognize.c, and toprec.c.

idft.sub

This subroutine is called by cepstrum.c and is used to calculate the inverse dft of a real even sequence.

invert.sub

This subroutine calculates the inverse of a 15x15 matrix and returns a pointer to the result. It is called by the programs template.c, testdis.c, recognize.c, and toprec.c.

lpanalyze infile outfile

This program calculates the 14th order lpc coefficients for the 25 segments of a word. The coefficients are calculated from the data in infile then output to outfile. Outfile, then, contains the 350 coefficients for all 25 segments formatted as 4 byte floating point numbers and concatenated together.

lpc.sub

This subroutine does the actual lpc coefficient calculation on one segment of a word. It is called from lpanalyze.c, template.c, recognize.c, and toprec.c.

makewindow

This program calculates the Hamming window coefficients to be used by those programs that do lpc analysis. In this way, those programs are made more efficient. The output is 300 floating point numbers placed in the file "Hamming."

median infile outfile windowlength

This program does median filtering on an input word. The inputs are an input file, an output file, and the length of the window desired.

mfilter.sub

This subroutine does the actual median filtering on one window of the input word. It is called by "median."

nodc infile outfile nbytes

This program removes dc offset from data files coming from the A/D Its inputs are input file, output file, and number of bytes in the input file.

parcor.sub

This subroutine calculates the PARCOR coefficients for lpc analysis by

the Burg algorithm. It is called by the subroutine "lpc.sub."

rdshort infile numwords

This program reads numwords from a file of short integers and prints their values on the standard output.

rdfloat infile numwords

This program reads numwords from a file of 4 byte floating point numbers and prints their values on the standard output.

recognize word.

This program does the actual recognition of words. $Word_i$ is the name of the word to be recognized with the utterance number appended to it. Recognize must be run in the HOME directory as it accesses $word_i$ from the "words" directory and accesses the vocabulary file from the "templates" directory.

segment infile numwords

This program takes files of short integers containing continuous (but discretely uttered) speech and segments them into single words. The inputs on the command line are the input file and the number of discrete utterances expected. Utterances should not exceed 750 ms in length (although this could be easily altered) and should be separated by at least 300 ms. The program will segment these words and prompt the user for a filename for each word, informing him of the starting point, ending point, and length of the word segmented.

spectime infile label

This program produces a pseudo three dimensional plot of the vocal tract response throughout the utterance of a word. The input file is the file of lpc coefficients for that word produced by lpanalyze. The output consists of 25 lines representing the 25 vocal tract responses for the frames of the word. The beginning of the word is at the top of the plot. "Label" is the tag which will be printed at the heading of the plot. The output of this program should be piped to an appropriate output device.

xplotx

These programs are used to obtain plots in various formats. The programs splotx give plots which are shortened in the x direction to allow them to be copied on 8 1/2x11 paper. The programs xplots plot short numbers while the programs xplotf plot floating point numbers. If the programs are invoked with no arguments, they will return a description of the command line. The outputs of these programs should be piped to an appropriate output device.

template inword1 inword2 ... inwordJ outfile label

This program takes a number of utterances of the same word and produces a template from them which may be used for a recognition dictionary. The "label" is the name of the word which will be indicated when that word is chosen by the recognition routines.

tesplate wordname index1 index2 ... indexJ outfile label

This program is a variant of "template" which allows for a simpler command line. If the word files are named by wordnameX where X is the

utterance number, then this program can be more easily used because of the simplification of the command line.

testdis templates/infile words/infile

This program calculates the frame distances and cumulative distance for the template/word pair given on the command line. These distances are printed on the standard output as the program runs. The frame distances are also output to the file "testdisout" so that they may be plotted. The outputs are all floating point numbers. This program is useful in setting the offset and stepsize in the recognition programs. It is also useful for verifying results.

toprec wordname

This program works in exactly the same way as does recognize.c, but its output is a list of the top three candidates for recognition rather than just the top candidate.

window.sub

This subroutine multiplies the Hamming window by frame of a word during lpc analysis. It is called by lpc.sub which in turn is called by all programs which do lpc analysis. Note that the file "Hamming" must be present in all directories where this subroutine (and hence lpc analysis) is called.