TR-I-0013

Voice Conversion by Analysis-Synthesis Method

分析・合成方式による声質変換

Hubert Ségot and Hisao Kuwabara

ユベール・セゴ、桑原尚夫

November, 1987

Abstract

This paper deals with a way of controlling the quality of natural speech by manipulating such acoustic parameters as formant frequencies / bandwidths and pitch frequency. These parameters were modified making use of the conventional analysis-synthesis system. After extracting trajectories of the lowest three formants, their frequencies and bandwidths at each analysis frame were subjected to change. Modification of pitch frequency was performed by changing the first peak of the spectral envelope obtained from re-analysis of the residual signals.

This work was done as an internship at ATR Interpreting Telephony Research Laboratories. Software programs for the above methods were developed during the internship period. A few examples of synthesized speech by changing the acoustic parameters are included.

ATR Interpreting Telephony Research Laboratories
ATR自動翻訳電話研究所

This is an excerpt from the Internship Report

Period: from April 1, 1987 to November 30, 1987

Company: ATR Interpreting Telephony Research Laboratories
Osaka, Japan

Topic: Voice conversion by analysis-synthesis method

Name: Hubert Ségot, Student of the ENST (Ecole Nationale Supérieure des Télécommunications), Paris, France

Acknowledgments

Special thanks to

Mr. Kurematsu and Mr. Shikano,
for all the freedom and support to conduct speech research at Speech Processing Department.

Mr. Abe, Mr. Takeda and Mr. Nakamura,
for their help and clear explanations of some technical points, and for the participation in the hearing test.

All the staff of ATR,
for their unlimited kindness and neverending obligingness.

VOICE CONVERSION BY ANALYSIS-SYNTHESIS METHOD

LIST OF CONTENTS

EXECUTIVE SUMMARY

During my internship in ATR, I was a member of the Interpreting Telephony Research Laboratories, in the Speech Processing Department.

This company needs for its future automatic intepreter a high quality voice synthesis system that, if possible, would be able to add some personal information to the outputs.
In order to gain further insight in the contributions of the vocal tract characteristics and the pitch frequency to speech personality , a voice conversion by analysis-synthesis method was intended. I was to design it using the Linear Prediction method, which modelizes digitalized speech signals by a time-varying all-pole digital filter of which the coefficients are known as the Linear Prediction Coefficients (LPC), excited by a pulse train.

That task required the implementation of several software tools.
Fist of all, a voiced/unvoiced/silent decision concerning the parts of the treated speech had to be made. Then, a scheme to extract the parameters of the model was to be selected. After that, a formant tracker proved necessary. Finally, a synthesis method was to be chosen.

Hence, now, the analysis of a speech wave can be performed, the parameters manipulated, individually or not, and a new speech wave corresponding to the new parameters is obtained.

When the corresponding software worked smoothly, perceptual experiments were made. Even though the lack of time did not allow me to push them as far as I would have wished, they lead to the conclusion that, probably, the voice personality is more affected by formant than by bandwidths or pitch frequency manipulation.

Now, further attempts remain to be aimed towards this direction and towards a deeper understanding of the dynamic features of power, pitch, and formants, and their relationship with speech individuality.

ABOUT THE INTERNSHIP

Voice personality is a complex concept.
That matter is rather subjective, in the current state of the knowledge. As previouly quoted, a certain understanding of it could lead to a better speech synthesis technology, and that is one of the goals of ATR Interpreting Telephony Research Laboratories.
It probably results from the combination of several data, both dynamic, such as prosody, and instantaneous, such as vocal tract configuration.

The internship was focussed on the second type. Its aim was to generate a tool to alter them, through the use of the LPC model, which is currently a broadly used approach to digitalized signals. It consists in considering speech as the output of an all-pole time-varying digital filter excited by a pulse train. The filter provides the instantaneous spectrum of the signal. The difference between the modelized sequence and the real one is called the residual signal, and contains useful data as far as the pitch is concerned.

At the beginning of the internship, I was provided with some programs that I spent some time rewriting.

Then, I had to implement an analysis-change-synthesis system.
During that process, I encountered some problems to solve and some decisions to take.

First of all, it became quickly obvious that a kind of voiced/unvoiced/silent speech wave zone detector was necessary. Several ideas were put into real trial.
The best discovered algorithm was based on the assumptions that the spectrum of a voiced zone has several neat peaks (the formants), and that a voiced frame allows the reliable extraction of a pitch value.
So a spectral flatness measure was devised:

$$\text{Spectral Flatness} = \exp\left(\int_{-\pi}^{+\pi} \ln |S(e^{j\Omega})| \, d\Omega/2\pi\right) / \int_{-\pi}^{+\pi} |S(e^{j\Omega})| \, d\Omega/2\pi$$

S being the z-transform of the filter,
and a pitch presence probability measure was also designed:

$$\text{Pitch Presence Probability} = \underset{n=a}{\overset{b}{\text{Maximum}}} \left(r(n) / r(0) \right)$$

r being the autocorrelation sequence of the residual signal,
a and b, certain boundaries correponding to the range 50-500Hz.

An algorithm using those two measures proved extremely successful, but also extremely computer-time consumming.
That consideration and others, lead me to the conclusion that a human decision was quicker, and maybe, somewhat more reliable too.

Then the choice of a method to extract the LPC was needed. I first used the covariance one but since the poles of the filter were often lying outside the unit circle of the complex plane, thus entailing useless data and processing, I had to switch for the autocorrelation method which is always stable.

After that, a device for the three first formant tracks extraction was considered. Basically, it searches for the peaks of the spectrum (by means of the LPC roots bandwidths), and in case of several possibilities for the current zone, selects the closer one to the previously selected one. It is followed by a five-points error retrieval algorithm.
As it proved perfection-free, again, hand corrections are needed.

As far as the pitch frequency is concerned, all the methods I tried were unperfect, too.
I chose finally to use the formula:

$$\text{Pitch Period} = \underset{n=a}{\overset{b}{\text{Maximum}}} \left(r(n) / r(0) \right)$$

r being the autocorrelation sequence of the residual signal,
a and b, certain boundaries corresponding to the range 50-500Hz.

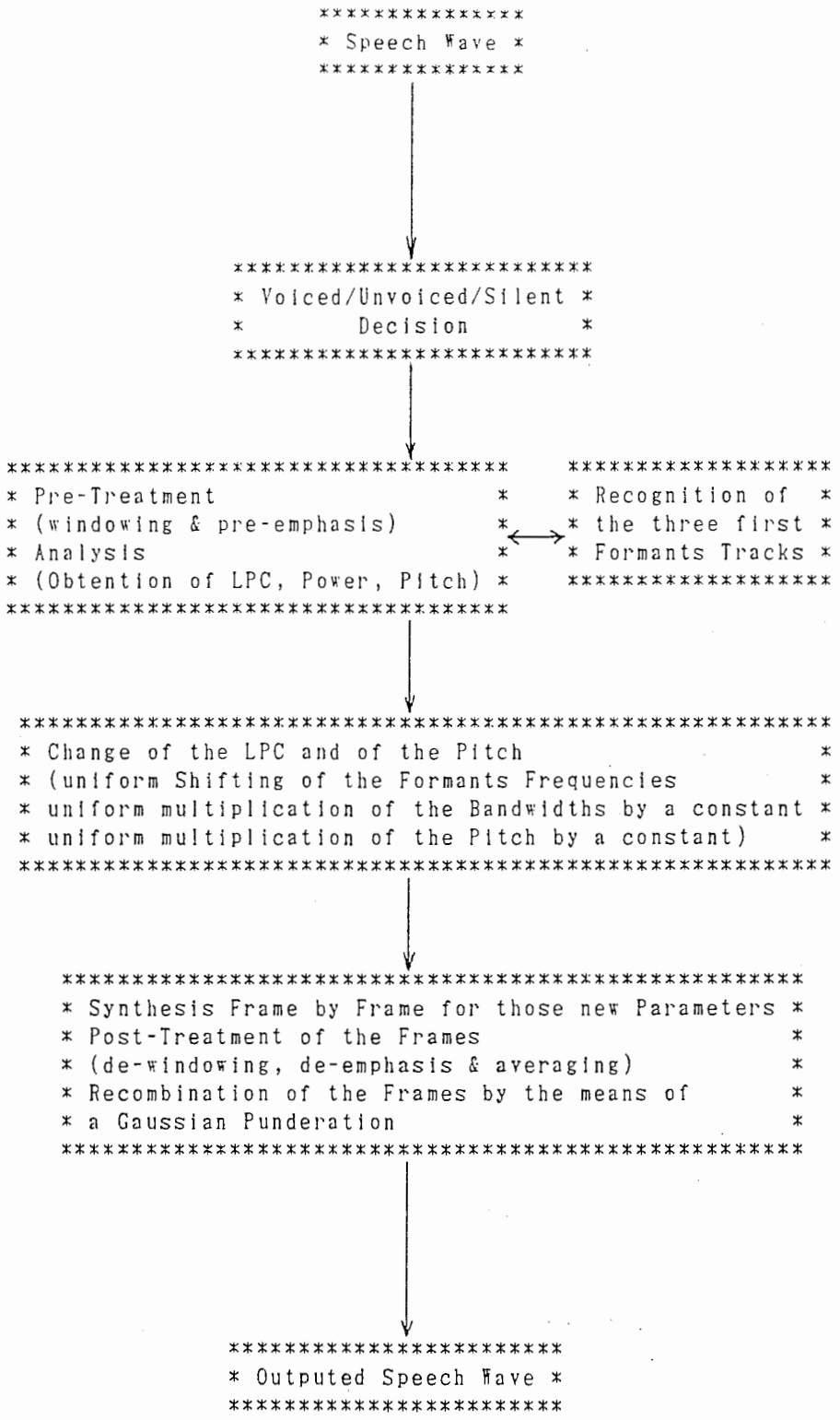
In other words, it consists in scanning the residual signal for regularly spaced peaks. Their distance will provide the pitch period.
Then, a five-points mistake removing procedure is appended.
And again, human assistance is required.

All that lead to a highly-human interactive analysis design.
To allow an easy and efficient human intervention , a previouly existing wave and spectrogram displayer software went into further developments.

The manipulation of the parameters proved a rather straightforward task.
Concerning the formant characteristics, the frequencies and bandwidths were changed by extracting the roots of the LPC, then modifying them accordingly, then finally computing the new LPC fitting those new roots.
Concerning the pitch, the distance between the pulse was lengthened or shortened, as it pleased.

The last step was to implement a synthesis device to create the new speech wave corresponding to those alterations.
The selected method was the computation on overlapping speech zones, then their addition, using a Gaussian weigth, so as to give more importance to the values of the center the zone, which are somewhat more accurate than the others.

The following block-diagram summarizes the algorithm.


Speech Wave
Voiced/Unvoiced/Silent Decision
Pre-Treatment (windowing & pre-emphasis) Analysis (Obtention of LPC, Power, Pitch)
Recognition of the three first Formants Tracks
Change of the LPC and of the Pitch (uniform Shifting of the Formants Frequencies uniform multiplication of the Bandwidths by a constant uniform multiplication of the Pitch by a constant)
Synthesis Frame by Frame for those new Parameters Post-Treatment of the Frames (de-windowing, de-emphasis & averaging) Recombination of the Frames by the means of a Gaussian Punderation
Outputed Speech Wave

Then some perceptual experiments were attempted.
They consisted in taking several people and making them listen to first the original speech wave, then the manipulated one. They had to decide whether it was the same person or not. It provided some recognition ratio curves.
As I could not carry those experiments very far due to the lack of time, they are not comprehensive and extensive. Nevertheless, it can be stated that voice personality seems more perturbated by a formant frequency than by a formant bandwidth or a pitch frequency manipulation.

ACQUIREMENTS

First and foremost, let us note that this internship was carried out in a research center, and that a liberal use of its important and sophisticated tools could be performed. Thus the milieu was peculiar, propicious and profitable.
It is hard to part all the attainments into different categories, nevertheless, three main domains can be defined rather clearly: the technical one, the methodological one, and the "human" one.

FROM A TECHNICAL VIEWPOINT

First of all, I have greatly improved my programming skills and my background in algorithmics and computer science.
This was a necessary step for the practical mastery of the LPC method which is currently a most widely used (if not one of the best) approach in the field of Speech Processing, thus leading me to a certain proficiency at, and knowledge of it.
On a broader basis, that specific point, blended with some readings, the regular talks held by researchers, their neighbourhood, and the difficulties encountered in my experiments, gave me a first-hand opportunity to get in touch with some of both the classical and the latest problems and trends of Speech Processing (and somewhat also of Machine Translation too).
Meanwhile, I became aware of how complex and challenging a world it is.

FROM A METHODOLOGICAL VIEWPOINT

Since I was not included into any team, and since my job was rather independant from any larger project, my freedom was great, both in terms of local approaches to select and in terms of general design to lay out.
During my advances in Speech Processing, I discovered that many technologies are tricky and "kotsu", and that the knack and the trial and error attitude are badly needed.
Hence, all through that formative process, I acquired a certain experience in the way to handle a research work.
I was also fortunate enough to get acquointed with the Japanese management and work system, far from an overintellectual approach, in a practical work-a-day world. Bluntly stating that the Nippon-style companies are at the antipodes of the French-style ones would be daring. Nevertheless, it can be said that in ATR, I witnessed a system much less contentious and individualistic than what I previously knew. I noticed several devices to strengthen group solidarity, boost employee unity, and company loyalty, all that leading to an excellent working atmosphere that I really appreciated. The emphasis layed on group conciousness was also noticeable in the working style through regularly held meetings (in Japanese), during which researchers explain their latest advances. It also avoids the specialist syndrome where everyone can see only his own tree, but misses the forest.

FROM AN HUMAN VIEWPOINT

The main points probably proceed from the fact that I could taste a bit of the unique flavour of Japan from a privilegied location, even though my knowledge of the Japanese language proved rather poor.
In ATR, I could meet some "real" Japanese men and women, and thus, I deepened my understanding and enjoyment of that country and got rid of many a stereotype; for it became obvious that, all too often, the overseas opinions on Japan are not based on fully accurate information, just like the Japanese opinions on France are, as I felt.
Shuch an exposure to a new world cannot but increase one's openness.

CONCLUSION

The conversion by analysis-synthesis method system that I developped works. Nevertheless, regarding the naturalness of the output, in case of pitch manipulation, further improvements could probably be attained, especially by using the original residual signal instead of a pulse train for synthesizing, but so far, my attempts on that topic did not seem decisive.
More perceptual experiments should be made, and since this was concerned only with instantaneous pitch frequency and vocal tract characteristics a new direction of investigation could focuss on their dynamics.

As a conclusion I could say that this internship proved a thrilling experience. I have learned a lot about a new civilization where amidst the zooming cans of human sardine known as subways and the serenity of the Shinto shrines, similarities in the way of life between the French and the Japanese are striking. Nevertheless, they remain definitely different, I dare not say which is better or worse. To understand that country is far else than blind teasing or admiration, and for a comprehensive and critical first look at Japan, I think the best way was a personal experience, even though a limited one.

And also, I have learned a lot in many other fields including especially science, technology, and handling a research work.
But I am very well aware that I only made a petty scratch into those areas.

A BRIEF OVERVIEW OF LPC METHOD

0] WARNINGS AND DISCLAIMERS

In this appendix, the following assumptions will be made

- We deal with digitalized signals.
- The z-transform theory is known.

Unless explicitly stated on a peculiar point, it is held true that

- z stands for a complex number.
- x(n) stands for a sequence.
- X(z) stands for its z-transform.

Also, the shorthand notation "LPC" will be employed for "Linear Prediction Coefficients".

I] LINEAR PREDICTION: THE BASIC IDEA

In first approximation, linear prediction consists in computing in advance the terms of a sequence by linearly combining a fixed number of the previous terms.
That approach to the signal may be translated into mathematical terms, thus providing the following:

DEFINITION:
A Linear Prediction of a sequence x(n) is a system whose output is

$$y(n) = \sum_{k=1}^{m} a_k * x(n-k)$$

the input being x(n).
Its order is m.

The system function of a m^{th} order linear predictor is the polynomial

$$LP(z) = \sum_{k=1}^{m} a_k * z^{-k}$$

We also define the prediction error as

$$e(n) = x(n) - y(n)$$

$$= x(n) - \sum_{k=1}^{m} a_k * x(n-k)$$

To wit, in terms of z-transform,

$$E(z) = A(z) * X(z)$$

$$A(z) = \sum_{k=1}^{m} a_k * z^{-k}, \quad a_0 = 1$$

These definitions and direct applications of the z-transform will lead to further insight in the speech signals.

II] MODEL OF THE SPEECH PRODUCTION

We modelize the composite effects of radiation, vocal tract and glottal excitation by a time varying digital filter whose system function can be written as:

$$H(z,n) = S(z,n) / U(z,n) = G(n) / \left(1 - \sum_{k=1}^{m} a_k(n) * z^{-k} \right)$$

That system gets excited by an impulse train (i.e. $x(n) / x(p) = 1$, $x(q) = 1$, $x(n) = 0$ for $p < n < q$, and $q-p$ is the instantaneous pitch period) for voiced speech, and by random noise for an unvoiced speech.

So, this model is determined by the parameters:

- Voiced / Unvoiced
- Pitch period
- G(n) (Gain parameter)
- m (the order of the model)
- $\{ a_k(n) \}_{k=1,m}$

They are assumed to all vary slowly with time.

Hence, a speech wave, s(n), is computed by the formula:

$$s(n) = \sum_{k=1}^{m} a_k(n) * s(n-k) + G(n) * u(n)$$

where u(n) is the applied excitation.

II] APPLICATION OF THE MODEL OF THE SPEECH PRODUCTION

Assuming we are provided with a real digitalized speech wave, the computing of the system function of our model is equivalent to the finding of a linear predicton filter for this given sequence.
Let s(n) be the given speech wave, LP(z) the system function of the linear predicton filter and y(n), the approximation of s(n) that the model outputs.

The best approximation will minimize the mean square of the prediction error sequence, (y-s)(n).

As the LPC vary slowly with time, in order to compute them, we will assume that they remain constant on a short duration, and we will consider a short-time average prediction error, and minimize it. The computed LPC will be valid only for that short time.

Let us take $s_n(p) = s(n+p)$ for a segment of speech selected in the vicinity of the sample n.

$$\text{Min}(E_n) = \text{Min}(\sum_{p} (s_n(p) - y_n(p))^2)$$

$$\text{Min}(E_n) = \text{Min}(\sum_{p} (s_n(p) - \sum_{k=1}^{m} a_k * s_n(p-k))^2)$$

(as $(a_k(n))_{k=1,m}$ are constant on this short-time interval, we shall drop the indication (n) of their time-dependency)

Using the Lagrange theory, we set $\partial E_n / \partial a_k = 0$ for $1 \le k \le m$.

We get $\sum_{k=1}^{m} a_k * \Omega_n(i,k) = \Omega_n(i,0)$ for $1 \le i \le m$

and $E_n = \Omega_n(0,0) + \sum_{k=1}^{m} a_k * \Omega_n(0,k)$

using $\Omega_n(i,k) = \sum_{p} s_n(p-i) * s_n(p-k)$.

In order to apply those considerations, we need to chose the short-time interval. Two basic ways of doing so emerge: the autocorrelation one, and the covariance one.

THE AUTOCORRELATION METHOD

It consists in setting $s_n(p) = s(n+p) * w(n)$
with w(n) being a finite-length window (e.g. a Hamming window)
such that $w(n) = 0$ for $n \geq N$ and $n < 0$.
In that case, $\Omega_n(i,k) = R_n(|i-k|)$ for $1 \leq i \leq m$ and for $0 \leq k \leq m$
where R_n is the short-time autocorrelation function of the sequence,

i.e $$R_n(k) = \sum_{p=0}^{N-1-k} s_n(p) * s_n(p+k) \ .$$

In a matrix formulation, it gives
$$[R_n(|i-j|)]_{i=1,m\ ;\ j=1,m} \quad [a_i]_{i=1,m} = [R_n(i)]_{i=1,m}$$

THE COVARIANCE METHOD

We first set the interval over which the mean squared error is computed and the consider the impact of such a setting on the computation of $\Omega_n(i,k)$.

Hence, as we define $E_n = \sum_{p=0}^{N-1} (s_n(p) - y_n(p))^2$

we need $s_n(p)$ for $-m \leq p \leq N-1$

since $\Omega_n(i,k) = \sum_{p=0}^{N-1} s_n(p-i) * s_n(p-k)$ for $1 \leq i \leq m$ and for $0 \leq k \leq m$

In a matrix formulation, it gives
$$[\Omega_n(i,j)]_{i=1,m\ ;\ j=1,m} \quad [a_i]_{i=1,m} = [\Omega_n(i,0)]_{i=1,m}$$

Let us note that $\Omega_n(i,k) = \Omega_n(k,i)$.

the name of the method is accounted by the fact that the properties of the matrix $[\Omega_n(i,j)]_{i=1,m\ ;\ j=1,m}$ are similar to those of a covariance matrix.

DETERMINATION OF THE OTHER PARAMETERS

Gain

It is proven that $G^2 = E_n$.

Voiced/Unvoiced Decision and Pitch Period

As a by-product of the computation of the linear prediction error sequence, we can determine if the speech part seems voiced or unvoiced by examinig this sequence. If it looks like a random signal, we can guess it is likely to be an unvoiced part of the speech. If it has some periodically spaced peaks, it surely corresponds to a voiced part of the speech and the corresponding pitch period can be extracted (by computing its autocorrelation sequence and scanning for the maximum in the accurate range) since it is the time lap between the peaks.

Spectrum

The LPC provide a short-time spectrum approximation, because the frequency response of a filter whose system function is H(z) can be evaluated by $H(e^{j\Omega})$.

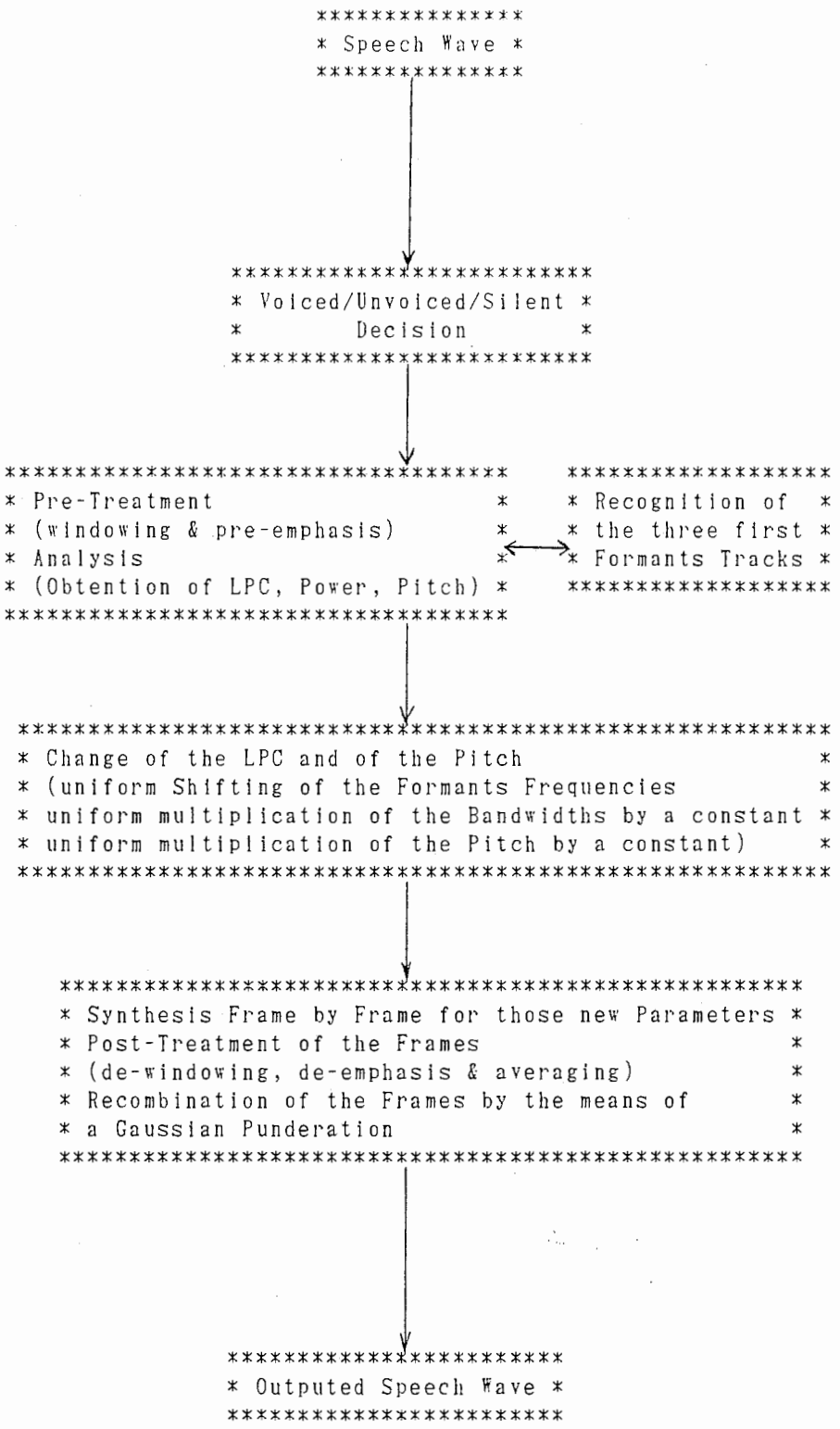
It can be proven that theoretically the poles of the LPC lye inside the unit circle if we use of the autocorrelation method, whereas it is not always the case for the covariance method. This fact is of great computational importance because these poles provide an efficient approach to estimate the formants bandwidths and frequencies since, if the peak due to the pole in the spectrum correponds to a formant, they are related by the formula

pole = exp { ($-\pi$*bandwidth + j*2*π*frequency) * T }

where T is the sampling period for the digitalization of the speech sample.

IV] CONCLUSION

We have quickly dealt with a model for speech production by the output of an all-pole digital filter whose input should be either random noise, either an impulse train related to the pitch, that distinction depending on the voiced or unvoiced aspect of the speech.
The poles of the filter provide easily the formant parameters.
However, it shows poorer results for nasals, stops, and fricatives than for voiced sounds. Also, it may seem that female voices are more difficultly treated than male voices.
Nevertheless, because of its versatility and good results, the LPC method is widely used for speech processing.


Speech Wave
Voiced/Unvoiced/Silent Decision
Pre-Treatment (windowing & pre-emphasis) Analysis (Obtention of LPC, Power, Pitch)
Recognition of the three first Formants Tracks
Change of the LPC and of the Pitch (uniform Shifting of the Formants Frequencies uniform multiplication of the Bandwidths by a constant uniform multiplication of the Pitch by a constant)
Synthesis Frame by Frame for those new Parameters Post-Treatment of the Frames (de-windowing, de-emphasis & averaging) Recombination of the Frames by the means of a Gaussian Punderation
Outputed Speech Wave

****
*   *
* UTILITY SOFTWARE *
*   *
****

```
*****************************************
*                                       *
*           GENERAL USAGE               *
*                                       *
*****************************************
```

For that purpose, several shell routines were developped.
Here they are.

```
echo     " "
echo     "BEFORE ANY TREATMENT WE MUST HAVE 'VOICED' THE FILE"
echo     " "
echo     "IF IT IS SO, ANSWER 'y' AND GO ON"
echo     " "
echo     "ELSE,        ANSWER 'n' AND VOICE IT"
echo     " "
read m
case $m in

y)
          ;;
n)
          cd LPC
          cd TREAT;
          voicing $1;
          cd
          ;;
*)
          echo 'answer correctly next time'
          exit
          ;;
esac

echo     " "
echo     "BEFORE ANY TREATMENT WE MUST HAVE 'FORMANTED' THE FILE"
echo     " "
echo     "IF IT IS SO, ANSWER 'y' AND GO ON"
echo     " "
echo     "ELSE,        ANSWER 'n' AND FORMANT IT"
echo     " "
read m
case $m in

y)
          ;;
n)
          cd LPC
          cd TREAT;
          echo " "
          echo " "
          echo "BEWARE TO SET THE SAME ORDER OF ANALYSIS FOR THE"
          echo " "
          echo "FORMANTING AND THE TREATMENT."
          echo " "
          formant $1;
          cd
          ;;
*)
          echo 'answer correctly next time'
          exit
          ;;
esac


echo     " "
```

```
echo     "BEFORE ANY TREATMENT WE MUST SET THE PARAMETERS"
echo     " "
echo     "ANSWER 'y' TO GO ON"
echo     " "
read m
case $m in

y)
          ;;
*)
          echo 'answer correctly next time'
          exit
          ;;
esac

cd LPC
lavoro $1
cd
```

```
emacs /data/segot/LPC/OLD/a_help.h
emacs /data/segot/LPC/OLD/driving

cc /data/segot/LPC/OLD/foo.c -O
a.out $1 > file_auxilliary
rm a.out

cp /data/segot/LPC/OLD/go go
chmod +x go

cc /data/segot/LPC/OLD/t.h.gen.c -O -o t.h.generator


m=1
grep "/¥* $m ¥*/" file_auxilliary > file_help
if test -s a_storing
then
rm *_storing
fi

while test -s file_help
do
cat file_help /data/segot/LPC/OLD/a_help.h > a.h
cp /data/segot/LPC/OLD/gogo.c zzzzz.c
cc zzzzz.c -O
a.out $1 > gogo
sh gogo
        for toto in `ls $1.*.o`
        do
        mv $toto $toto.$m
        done
(
cd
cd data
cd segot
cd MICRO-VAX

        for toto in `ls $1.*.o.da`
        do
        mv $toto $toto.$m
        done
)
m=`expr $m + 1`
grep "/¥* $m ¥*/" file_auxilliary > file_help
rm *_storing
done

rm a.h file_help file_auxilliary go gogo $1

m=1
cp /data/segot/LPC/OLD/max_m.c zzzzz.c
cc zzzzz.c -O -o max_m
rm zzzzz.c
max_m $m > file_help
while test -s file_help
do
```

```
if test -s $1.$m.o.1
then

        for toto in `ls $1.$m.o.*`
        do
        cat $toto >> $1.$m.o
        rm $toto
        done
(
cd
cd data
cd segot
cd MICRO-VAX
        for toto in `ls $1.$m.o.da.*`
        do
        cat $toto >> $1.$m.o.da
        rm $toto
        done
)
fi
m=`expr $m + 1`
max_m $m > file_help
done

rm max_m file_help t.h.generator
```

```
/*order of linear prediction                    */
#define  M                                        15

/*frequency of sampling in kHz                  */
#define  kHz_SAMPLING_FREQUENCY                   12.0

/*precision of the Bairstow iteration
which provide the roots of A(z)                 */
#define  EPSILON                                  1e-6

/*maximum of data                               */
/*we must add 2, else the program becomes wild, and I cannot figure out why */
#define  MAXIMUM_OF_DATA                           LAST_SAMPLE-FIRST_SAMPLE+2

/*shift of the frame                            */
#define  FRAME_SHIFT                              32

/*length of the frame                           */
#define  FRAME_LENGTH                             256
```

```
#include <stdio.h>

#define STEP_SPEECH_FRAMES              500

main (argc,argv)
int argc ;
char *argv[];
/*this program reads a file and determines its silent zones for
optimizing its slicing for a voice-treatment */
{
char file[81],toto[81];
int max_frame = 0 ;
int i , j , k ;
FILE *file_pointer;
float *voicing ;

strcpy(file,argv[1]);

strcat(file,"_voicing");

strcpy( toto , "/data/segot/MICRO-VAX/" ) ;

strcat( toto , file ) ;

strcpy( file , toto ) ;

if ((file_pointer=fopen(file,"r"))==NULL)
        {
        printf("error¥n");
        exit ( ) ;
        }

while (getc(file_pointer) != EOF)
        max_frame++ ;
max_frame /= sizeof(float) ;
rewind ( file_pointer ) ;
if ((voicing = (float *)malloc(sizeof(float)*max_frame))== NULL )
        {
        printf("error¥n");
        exit ( ) ;
        }
if ( fread(voicing,sizeof(float),max_frame,file_pointer) != max_frame )
                {
                printf("error¥n");
                exit ( ) ;
                }

k = i = 0 ;
j = 1 ;
while ( ( k < max_frame ) && ( *(voicing+k) == 193.00 ) ) k++ ;

while ( k < max_frame-1 )
        {
        printf("#define FIRST_SAMPLE   %d      ",k*32);
        printf("/* %d */¥n",j);
```

```
        i = k + STEP_SPEECH_FRAMES ;
        if ( i >= max_frame - 1 )
                {
                printf("#define LAST_SAMPLE     %d      ",(max_frame-1)*32+256);
                printf("/* %d */¥n",j);
                break;
                }
        else
                {
                while ( *(voicing+i) != 193.00 ) i-- ;
                k = i ;
                }
        if ( i != max_frame - 1 )
                printf("#define LAST_SAMPLE     %d      ",i*32-1);
        else
                printf("#define LAST_SAMPLE     %d      ",(max_frame-1)*32+256);
        printf("/* %d */¥n",j);
        j++ ;
        }
}
```

```
case $1 in
syncovar)
        echo '#include ' '"'"$4"'"'  >> zzzzzzzzzzz.c
        echo ' '        >> zzzzzzzzzzz.c
        echo 'main()'   >> zzzzzzzzzzz.c
        echo  {         >> zzzzzzzzzzz.c
        echo 'printf("main_syncovar %lf %lf %lf %lf %lf",BQ[0],BQ[1],BQ[2],FS3,P_S); ' >>
zzzzzzzzzzz.c
        echo  }         >> zzzzzzzzzzz.c
        echo ' '        >> zzzzzzzzzzz.c
        cc zzzzzzzzzzz.c -lm -O
        a.out > zzzol
        if test -s main_syncovar
        then echo ' '
        else cc /data/segot/LPC/TREAT/_synthesis.c -lm -O -o main_syncovar
        fi
        rm a.out zzzzzzzzzzz.c
        echo "$3 $2" | sh zzzol > $3
        rm zzzol
        ;;
covar)  cat $4 /data/segot/LPC/TREAT/_analysis.c > zzzzzzzzzzz.c
        cc zzzzzzzzzzz.c -lm -O
        a.out $2 $3
        rm zzzzzzzzzzz.c a.out
        ;;
*)      echo "bye-bye"
        ;;
esac
```

```
#include "/data/segot/LPC/OLD/driving"


main(argc,argv)
int argc ;
char *argv[];

{
double id;

if ( BW_STEP == 0 ) id = 1.0 + (double)( BW_RANGE ) ;
else
        {
        id = (double)( - BW_RANGE + atoi( argv[1] ) * BW_STEP);
        if      ( id < 0.0  ) id -= 1.0 ;
        else if ( id >=  0.0 ) id += 1.0 ;

        if ( id <  0.0 ) id = - 1.0 / id ;
        }
if ( id == 0.5 )  id  = 2.0 / 3.0 ;
if ( id == 2.0 )  id  = 1.5 ;
if ( id < 0.5  )  id /= ( 1 - id ) ;
if ( id > 2.0  )  id -= 1.0 ;

printf("double BQ[ 3 ] = { %lf, %lf, %lf};¥n",id,id,id);

if ( FS3_STEP == 0 ) id = FS3_RANGE ;
else id = (double)( -FS3_RANGE+ atoi ( argv[2] ) * FS3_STEP) ;
printf("#define FS3             %lf¥n", id ) ;

if ( PITCH_STEP == 0 ) id = PITCH_RANGE ;
else id = (double)( -PITCH_RANGE+ atoi ( argv[2] ) * PITCH_STEP) ;
printf("#define P_S             %lf¥n", 1.0 + id ) ;

}
```

```
#include "/data/segot/LPC/OLD/driving"


main(argc,argv)
char *argv[];
int   argc;
{
int i , j , k , ff1 , ff2 , ff3 ;
int t1 , t2 , t3 ;

t1 =    BW_STEP ;
t2 =   FS3_STEP ;
t3 =  PITCH_STEP;
ff1 = ( t1 == 0 ) ? 1 : 2* BW_RANGE / ((BW_STEP==0)?1:BW_STEP)+1        ;
ff2 = ( t2 == 0 ) ? 1 : 2* FS3_RANGE / ((FS3_STEP==0)?1:FS3_STEP)+1     ;
ff3 = ( t3 == 0 ) ? 1 : 2* PITCH_RANGE / ((PITCH_STEP==0)?1:PITCH_STEP)+1        ;

printf("go covar %s %s a.h¥n",argv[1],argv[1]);
for ( i = 0 ; i<ff1 ; i++)
        for ( j = 0 ; j<ff2 ; j++)
                for ( k = 0 ; k<ff3; ++k)
                   printf("t.h.generator %d %d %d > t.h.%d¥n",i,j,k,1+ff2*j+ff1*ff2
*i);

for ( i = 0 ; i<ff1 ; i++)
        for ( j = 0 ; j<ff2 ; j++)
                for ( k = 0 ; k<ff3; ++k)
                  printf("go syncovar %s  %s.%d.o t.h.%d¥n",argv[1],argv[1],1+ff2*j
+ff1*ff2*i,1+ff2*j+ff1*ff2*i);

}
```

```
emacs _voicing_tutorial
cd
cd MICRO-VAX
cd NAKA
pdisp $1
cd
cd LPC
cd TREAT
```

```
echo " "
echo "GIVE THE ORDER OF THE ANALYSIS (DEFAULT 21)"
echo " "
read  m
case $m in
[0-9][0-9]) if test $m -le 31
                then echo "OK, I TAKE $m AS THE ANALYSIS ORDER"
                     echo "IN CASE OF DISSATISFACTION HIT THE ^C KEY"
                else echo "I DISSAGREE, I TAKE 21 AS THE ANALYSIS ORDER"
                     echo "IN CASE OF DISSATISFACTION HIT THE ^C KEY"
                     $m = 21
            fi
            echo "#define M $m" >  zzzzzp.c
            echo "#define M $m" > zzzzzzp.c ;;
 *)         echo "I DISSAGREE, I TAKE 21 AS THE ANALYSIS ORDER"
            echo "IN CASE OF DISSATISFACTION HIT THE ^C KEY"
            echo "#define M 21" >  zzzzzp.c
            echo "#define M 21" > zzzzzzp.c ;;
esac
cat /data/segot/LPC/TREAT/preleminary_formant_analysis.c >> zzzzzp.c
cc zzzzzp.c -lm -0
a.out $1 > zzzzzpzzzzzzz
cc /data/segot/LPC/TREAT/first_treatment.c -lm -0
a.out < zzzzzpzzzzzzz
rm zzzzzpzzzzzzz a.out
rm zzzzzp.c
cd
cd MI*
cd NAKA
echo " "
echo "VALIDATE THE RESULTS CONCERNING THE FORMANT TRACKS"
echo " "
echo "AND USE THE pdisp SOFTWARE FACILITY FOR IT"
echo " "
pdisp $1
echo " "
echo "ARE THE RESULTS CONCERNING THE FORMANT TRACKS GOOD ENOUGH? (y/n)"
echo " "
read  m
case $m in
y) cd
   cd LPC
   cd TREAT
   cat second_treatment.c >> zzzzzzp.c
   cc  zzzzzzp.c -lm -0 -o second_treatment.out
   second_treatment.out $1 lissage
   rm /data/segot/MICRO-VAX/$1_formants_validation
   rm second_treatment.out zzzzzzp.c
   exit
   ::
*) cd
```

```
    cd LPC
    cd TREAT
    cat second_treatment.c >> zzzzzzp.c
    cc  zzzzzzp.c -lm -O -o second_treatment.out
```



```
    rm  zzzzzzp.c
    while test -s /data/segot/MICRO-VAX/$1_formants_validation
         do
         cd
         cd LPC
         cd TREAT
         second_treatment.out $1 no_lissage
         cd
         cd MI*
         cd NAKA
         echo " "
         echo "CHECK THE RESULTS CONCERNING THE FORMANT TRACKS"
         echo " "
         echo " "
         echo "AND USE THE pdisp SOFTWARE FACILITY FOR IT"
         pdisp $1
         echo " "
         echo "ARE THE RESULTS CONCERNING THE FORMANT TRACKS GOOD ENOUGH? (y/n)"
         echo " "
         read  m
         case $m in
         y) second_treatment.out $1 lissage
            rm /data/segot/MICRO-VAX/$1_formants_validation
            rm second_treatment.out;;
         *) ;;
         esac
         done
    ;;
esac
```

```
echo ' '
echo 'Give the name of the speech piece to'
echo 'compute the recognition rate of'
echo ' '
read file
if test -s recognition_rate_of_$file
then rm recognition_rate_of_$file
fi
cc /data/segot/LPC/LISTEN/alpha.c -lm -O -o alpha
cc /data/segot/LPC/LISTEN/beta.c -lm -O -o beta
for i in `ls *$file*output*`
do
if test -s recognition_rate_of_$file
        then
        cat recognition_rate_of_$file |  alpha $i > zzzzzz
        mv zzzzzz recognition_rate_of_$file
        else
        beta < $i > recognition_rate_of_$file
fi
echo " $i taken into account"
done
rm alpha beta
cc /data/segot/LPC/LISTEN/chart.c -lm -O -o chart
chart < recognition_rate_of_$file > chart_of_recognition_rate_of_$file
rm chart
```

```
cc /data/segot/LPC/LISTEN/listen_treat.c -O -lm -o listen_treat


echo 'Give your name, just one, please.'
read name


for file in /data/segot/MICRO-VAX/lerat.aiueo.1.o.da /data/segot/MICRO-VAX/jean.aiueo.1.o.d
a

#we may also try a complete test, but it may tire the listener-guinea pig
#for file in `ls /data/segot/MICRO-VAX/*.1.o.da`


do
listen_treat $name $file
echo ' '
echo ' '
echo ' '
echo ' '
echo ' '
sleep 2
done


rm listen_treat
```

```
#include "/data/segot/LPC/OLD/driving"

main(argc,argv)
int argc;
char *argv[];
{
int imax , j ;
int a , b , c ;
int input = 0 ;

for ( j = 0 ; j < strlen ( argv[1] ) ; j++ )
        input = input * 10 + (int) ( *(argv[1]+j) ) - (int) ( '0' ) ;

a = PITCH_STEP ; if ( a == 0 ) a = 1 ;
b = FS3_STEP   ; if ( b == 0 ) b = 1 ;
c = BW_STEP    ; if ( c == 0 ) c = 1 ;

imax = 1+(2*PITCH_RANGE/a+1) + ( 2*FS3_RANGE/b + 1 ) *
        ( 2*FS3_RANGE/b +  ( 2*BW_RANGE/c  + 1 ) * 2*BW_RANGE/c ) ;
if ( input <= imax ) printf ( "%d¥n" , imax-input ) ;
}
```

```
#include <stdio.h>
#include <math.h>
main(argc , argv)
int argc ;
char *argv[];
{
char c;
FILE *TOTO;
int i , j , k ;

TOTO=fopen(argv[1],"r");
while(scanf("clef %d recognition rate %d / %d¥n",&i,&j,&k) == 3 )
        {
        fscanf(TOTO,"%c¥n",&c);
        if (c=='y')
                j++ ;
        printf("clef %d recognition rate %d / %d¥n",i,j,k+1);
        }
fclose ( TOTO ) ;
}
```

```
#include <stdio.h>
#include <math.h>
main()
{
char c;
FILE *TOTO;
int j , k ;
int i;
static char value[121][8] = {
"782","122","485","870","155","815","1244","298","1288","166",
"397","1211","518","848","1277","254","573","1255","595","243",
"529","1299","188","661","1233","826","727","210","1200","804",
"320","1310","287","551","1101","199","925","276","771","606",
"221","694","1167","100","1024","23","1222","45","1145","34",
"903","265","837","1002","177","617","232","1156","584","56",
"1321","474","1","760","639","650","309","936","331","1123",
"672","111","1112","364","1035","408","1266","133","716","375",
"958","67","430","1057","441","1046","683","1134","452","1178",
"749","463","1068","496","1079","507","78","793","89","892",
"628","881","1090","562","1189","12","1013","738","144","705",
"947","353","980","419","859","386","991","342","914","540",
"969"
};

i=0;
while ( scanf("%c¥n",&c) == 1 )
        {
        if (c=='y')
                j = 1 ;
        else
                j = 0 ;
        printf("clef %s recognition rate %d / 1¥n",value[i],j);
        i++ ;
        }
}
```

```
#include <stdio.h>
#include <math.h>
#define ST 11
main()
{
int i,j,k;
float rate [ ST ] [ ST ] ;

for(i=0;i<ST;i++)
        for ( j=0;j<ST;j++)
                rate[i][j]=0.0;

while ( scanf ( "clef %d recognition rate %d / %d ¥n" , &i , &j , &k ) == 3 )
        rate [ ((i-1)%(ST*ST))/ST ] [ (i-1)/(ST*ST) ] = (float)j / (float)k ;

for(i=ST-1;i>=0;i--)
        {
        for ( j=0;j<ST;j++)
                printf( "%.2f   " , rate [i][j] ) ;
        printf("¥n");
        }
}
```

```
#include <stdio.h>
#include <math.h>

main( argc , argv )
int argc ;
char *argv[] ;
{
FILE *results ;
char  v , buffer[100] ;
static char value[121][8] = {
"782","122","485","870","155","815","1244","298","1288","166","397","1211","518","848",
"1277","254","573","1255","595","243","529","1299","188","661","1233","826",
"727","210","1200","804","320","1310","287","551","1101","199","925","276","771",
"606","221","694","1167","100","1024","23","1222","45","1145","34","903","265","837",
"1002","177","617","232","1156","584","56","1321","474","1","760","639","650",
"309","936","331","1123","672","111","1112","364","1035","408","1266","133",
"716","375","958","67","430","1057","441","1046","683","1134","452","1178","749",
"463","1068","496","1079","507","78","793","89","892","628","881","1090",
"562","1189","12","1013","738","144","705","947","353","980","419","859",
"386","991","342","914","540","969"
};

int i;
char begin[200] , roro[210] ;
char responses_buffer[200] ;


strcpy ( buffer, argv[2] ) ;
buffer[strlen(buffer)-7] = '¥0' ;
i=22;
while (buffer[i] != '¥0')
        {
        buffer[i-22] = buffer[i] ;
        i++;
        }
buffer[i-22] = buffer[i] ;


strcpy ( begin , "cd;cd MICRO-VAX;daout -f 12 " );
eheheh : ;
i=0;
printf ("Compare this utterance¥n") ;
strcpy ( roro , begin ) ;
strcat ( roro , buffer ) ;
strcat ( roro , ".661.o.da;cd;cd LPC;sleep 1" ) ;
system ( roro ) ;
printf ("With the following utterance¥n");
strcpy ( roro , begin ) ;
strcat ( roro , buffer ) ;
strcat ( roro , "." ) ;
strcat ( roro , value[i] ) ;
strcat ( roro , ".o.da;cd;cd LPC" ) ;
```

```
system ( roro ) ;
printf ( "Is it the same person? y for yes , n for no , r for repeat  ¥n");
responses_buffer[0] = getchar ( ) ;
v = getchar ( ) ;
```



```
printf("¥n¥n¥n¥n¥n");
switch ( responses_buffer[i] )
        {
        case 'y' :
        case 'n' : break ;
        case 'r' : goto eheheh ;
                   break ;
        default  : printf ("Answer y, n, r, or c, please!¥n¥n¥n¥n");
                   goto eheheh ;
                   break ;
        }
i = 1 ;
beginning : ;
while ( i < 121 )
        {
        printf ("Compare this utterance¥n") ;
        strcpy ( roro , begin ) ;
        strcat ( roro , buffer ) ;
        strcat ( roro , ".661.o.da;cd;cd LPC;sleep 1" ) ;
        system ( roro ) ;
        printf ("With the following utterance¥n");
        strcpy ( roro , begin ) ;
        strcat ( roro , buffer ) ;
        strcat ( roro , "." ) ;
        strcat ( roro , value[i] ) ;
        strcat ( roro , ".o.da;cd;cd LPC" ) ;
        system ( roro ) ;
        printf ( "Is it the same person? (y for yes / n for no / r for repeat ¥n");
        printf ( "                       c for correcting the previous answer )¥n");
        printf ( "                       ( %d / 121 )¥n",i+1);
        responses_buffer[i] = getchar ( ) ;
        v = getchar ( ) ;
        printf("¥n¥n¥n¥n¥n");
        switch ( responses_buffer[i] )
                {
                case 'y' :
                case 'n' : i++ ;
                           break ;
                case 'r' : break ;
                case 'c' : i-- ;
                           break ;
                default  : printf ("Answer y, n, r, or c, please!¥n¥n¥n¥n");
                           break ;
```

```
                }
        }
printf ("It is finished, do you want to correct the last proof?(y/n)¥n");
v = getchar ( ) ;
responses_buffer[160] = getchar ( ) ;
if ( v == 'y' )
        {
        i = 120 ;
        goto beginning ;
        }

strcpy ( begin , argv[1] ) ;
strcat ( begin , "." ) ;
```



```
strcat ( begin , buffer ) ;
strcat ( begin , ".outputs" ) ;
results = fopen ( begin , "w" ) ;
for ( i = 0 ; i < 121 ; i++ )
        fprintf ( results , "%c¥n" , responses_buffer[i] ) ;

printf("Thank you for your help¥n");
}
```

```
#define FS3_RANGE     10
#define FS3_STEP      0


/* Here , the bandwidth are given in additional percent change */
#define BW_RANGE      5
#define BW_STEP       0

/* Here , the pitch is given in additional percent change */
#define PITCH_RANGE      5
#define PITCH_STEP       0
```

```
**********************************************
*                                            *
*       ESTIMATING THE THREE FIRST FORMANTS  *
*                                            *
**********************************************
```

The shell routine 'formant' does it.
It usage is: formant <file.name>
Eventually, for a file name ending in .ad, e.g.
file name = foo.ad, just use 'formant foo'

In the first stage, the frequencies and bandwidths of the
LPC roots are computed for each frame.
In the second stage, a first estimation of the three first
formant tracks is performed, and the following files are created,
for further processing:
Eventually, for a file name ending in .ad, e.g.
file name = foo.ad

foo_poles double point type file
           stores the values of the LPC roots frequencies

foo_bb double point type file
        stores the values of the LPC roots bandwidths

foo_formants double point type file
              stores the values of the three first formant frequencies

foo_np integer type file
        stores the values of the number of the poles corresponding
        to the three first formants in each frame

foo_formants_validation is a temporary integer type file, created at that
time in the purpose of checking the accuracy of the values of foo_np.
    -1 stands for an unvoiced frame
     0 stands for a neutral degree of certainty
     1 stands for a sure value for the corresponding number in foo_np
     2 stands for the beginning of a probably false zone
     3 stands for the end of this zone

Then a human intervention is asked for, through the 'pdisp' command,
followed by a post-processing to take into account the human indications
untill the results proves satisfactory.

```
echo " "
echo "GIVE THE ORDER OF THE ANALYSIS (DEFAULT 21)"
echo " "
read  m
case $m in
[0-9][0-9]) if test $m -le 31
                then echo "OK, I TAKE $m AS THE ANALYSIS ORDER"
                     echo "IN CASE OF DISSATISFACTION HIT THE ^C KEY"
                else echo "I DISSAGREE, I TAKE 21 AS THE ANALYSIS ORDER"
                     echo "IN CASE OF DISSATISFACTION HIT THE ^C KEY"
                     $m = 21
            fi
           echo "#define M $m" >  zzzzzp.c
            echo "#define M $m" > zzzzzzp.c ;;
 *)         echo "I DISSAGREE, I TAKE 21 AS THE ANALYSIS ORDER"
            echo "IN CASE OF DISSATISFACTION HIT THE ^C KEY"
            echo "#define M 21" >  zzzzzp.c
            echo "#define M 21" > zzzzzzp.c ;;
esac
cat /data/segot/LPC/TREAT/preleminary_formant_analysis.c >> zzzzzp.c
cc zzzzzp.c -lm -O
a.out $1 > zzzzzpzzzzzzz
cc /data/segot/LPC/TREAT/first_treatment.c -lm -O
a.out < zzzzzpzzzzzzz
rm zzzzzpzzzzzzz a.out
rm zzzzzp.c
cd
cd M1*
cd NAKA
echo " "
echo "VALIDATE THE RESULTS CONCERNING THE FORMANT TRACKS"
echo " "
echo "AND USE THE pdisp SOFTWARE FACILITY FOR IT"
echo " "
pdisp $1
echo " "
echo "ARE THE RESULTS CONCERNING THE FORMANT TRACKS GOOD ENOUGH? (y/n)"
echo " "
read  m
case $m in
y) cd
   cd LPC
   cd TREAT
   cat second_treatment.c >> zzzzzzp.c
   cc  zzzzzzp.c -lm -O -o second_treatment.out
   second_treatment.out $1 lissage
   rm /data/segot/MICRO-VAX/$1_formants_validation
   rm second_treatment.out zzzzzzp.c
   exit
   ;;
*) cd
   cd LPC
   cd TREAT
   cat second_treatment.c >> zzzzzzp.c
   cc  zzzzzzp.c -lm -O -o second_treatment.out
```

```
   rm  zzzzzzp.c
   while test -s /data/segot/MICRO-VAX/$1_formants_validation
         do
         cd
         cd LPC
         cd TREAT
         second_treatment.out $1 no_lissage
         cd
         cd MI*
         cd NAKA
         echo " "
         echo "CHECK THE RESULTS CONCERNING THE FORMANT TRACKS"
         echo " "
         echo " "
         echo "AND USE THE pdisp SOFTWARE FACILITY FOR IT"
         pdisp $1
         echo " "
         echo "ARE THE RESULTS CONCERNING THE FORMANT TRACKS GOOD ENOUGH? (y/n)"
         echo " "
         read  m
         case $m in
         y) second_treatment.out $1 lissage
            rm /data/segot/MICRO-VAX/$1_formants_validation
            rm second_treatment.out;;
         *) ;;
         esac
         done
   ;;
esac
```

```
**************************************************
*                                                *
*        PROGRAM FOR EXTRACTING THE              *
*            THREE FIRST FORMANTS                *
*                                                *
**************************************************
```

```
#include <stdio.h>
#include  <math.h>
FILE   *STANDARD_INPUT ;
#include "/data/segot/LPC/TREAT/useful.c"
enum harmony { silent , unvoiced , voiced_consonnant , voiced_vowel } ;
#include "/data/segot/LPC/TREAT/find_formants.c"


main()
{
enum harmony *frame_type ;

char    buffer[81],file_formants[81],file_np[81],file_voicing[81];
FILE    *file_formants_pointer;

int     nw,m,frame_shift,max_frame,frame,*np,*werthe;

float   *frame_type_num ;

double  *a_buffer , *fff , *bbb , *inrm , fs  , power_buffer ;

FILE    *filou;

/*
 *
 *   INPUT SEQUENCE
 *
 */

do
if ( scanf ( "%s" , buffer ) == EOF ) iki() ;
while ( strcmp(buffer,"from_file")!=0);
scanf("%s",file_formants) ;
file_formants[strlen(file_formants)-3] = '¥0';
strcat ( file_formants , "_formants" ) ;

/* buffer contains now the name of the original file of raw data that were
   treated, we are at the beginning of the data to input */

if (lecture_stdin(buffer,"sampling_frequency_in_Hz"))
        scanf ("%lf",&fs);
else iki();

if (lecture_stdin(buffer,"order_of_analysis"))
        scanf ("%d",&m);
else iki();

if (lecture_stdin(buffer,"number_of_data_in_it"))
        scanf ("%d",&nw);
else iki();

if (lecture_stdin(buffer,"frame_shift"))
        scanf ("%d",&frame_shift);
else iki();
```

```
if (lecture_stdin(buffer,"total_number_of_frames"))
        scanf ("%d",&max_frame);
else iki();

if (!lecture_stdin(buffer,"the_frame_number"))    iki();

if (((frame_type=(enum harmony *)calloc(max_frame,sizeof(enum harmony)))==NULL)
                                    ||
    ((bbb=(double *)calloc((m-1)*max_frame+1,sizeof(double)))==NULL)
                                    ||
    ((fff=(double *)calloc((m-1)*max_frame+1,sizeof(double)))==NULL)
                                    ||
    ((a_buffer=(double *)calloc(m,sizeof(double)))==NULL))
        {
        printf("there is no room left for the computations, so exit");
        exit () ;
        }

do
{
if ( scanf ( "%d" ,&frame ) == EOF ) iki() ;
frame -= 1 ;
if ((lecture_stdin(buffer,"is_a_voiced_vowel_one"))
                ||
        (strcmp(buffer,"is_a_voiced_one")==0))
        *(frame_type+frame) = voiced_vowel   ;
else if (strcmp(buffer,"is_a_voiced_consonnant_one")==0)
        *(frame_type+frame) = voiced_consonnant   ;
else if (strcmp(buffer,"is_a_silent_one")==0)
        *(frame_type+frame) = silent   ;
else if (strcmp(buffer,"is_an_unvoiced_one")==0)
        *(frame_type+frame) = unvoiced ;
else iki()                              ;

if   (lecture_stdin(buffer,"its_power_is")==0) iki();
else scanf ("%lf",&power_buffer);

switch ( *(frame_type+frame) )
        {
        case silent     :
        case unvoiced   :
                                break ;
        case voiced_vowel:
        case voiced_consonnant:
                        fff += (m-1)*frame ;
                        bbb += (m-1)*frame ;
                        scratch_stdin ( a_buffer , fff , bbb , m ) ;
                        fff -= (m-1)*frame ;
                        bbb -= (m-1)*frame ;
                        break ;
        }
}

while ( lecture_stdin(buffer,"the_frame_number")&&( frame != max_frame-1 ) ) ;

/*
```

```
 ***   Storing of the pole frequencies    ***
*/
strcpy ( buffer , file_formants ) ;
buffer[ strlen(file_formants) - 9 ] = '¥0' ;
strcat ( buffer , "_poles" );
if ((filou = fopen(buffer,"w"))==NULL)
        {
        printf("there is a file opening problem, so exit¥n");
        exit () ;
        }
if (fwrite(fff,sizeof(double),(m-1)*max_frame,filou) != (m-1)*max_frame )
        {
        printf("there is a file writing problem, so exit¥n");
        exit () ;
        }
fclose ( filou ) ;

/*
 ***   Storing of the bandwidths    ***
*/
strcpy ( buffer , file_formants ) ;
buffer[ strlen(file_formants) - 9 ] = '¥0' ;
strcat ( buffer , "_bb" );
if ((filou = fopen(buffer,"w"))==NULL)
        {
        printf("there is a file opening problem, so exit¥n");
        exit () ;
        }
if (fwrite(bbb,sizeof(double),(m-1)*max_frame,filou) != (m-1)*max_frame )
        {
        printf("there is a file writing problem, so exit¥n");
        exit () ;
        }
fclose ( filou ) ;

/*
 ***   Creation of the file of validation    ***
*/
if (( werthe = (int *) malloc ( 3 * max_frame * sizeof(int) ) ) == NULL )
        {
        printf("there is a memory space problem, so exit¥n");
        exit () ;
        }
strcpy ( buffer , file_formants ) ;
strcat ( buffer , "_validation" );
if ((filou = fopen(buffer,"w"))==NULL)
        {
        printf("there is a file opening problem, so exit¥n");
        exit () ;
        }
for ( frame=0;frame<max_frame;++frame)
        {
        if ( *(frame_type+frame) == voiced_vowel )
                {
                *(werthe+3*frame  ) = 0;
                *(werthe+3*frame+1) = 0;
```

```
                                    ||
    (fwrite(inrm,sizeof(double),3*max_frame,file_formants_pointer)!=3*max_frame))
        {
        printf("Problem of access to the file of formants, so exit¥n");
        exit();
        }
fclose ( file_formants_pointer );

strcpy ( file_voicing , file_formants ) ;
file_voicing[ strlen(file_formants) - 9 ] = '¥0' ;
strcat ( file_voicing , "_voicing" );
if ( ( filou = fopen ( file_voicing , "w" ) ) == NULL )
        {
        printf("Problem of access to the file of voicing, so exit¥n");
        exit();
        }
if ((frame_type_num=(float *)calloc(max_frame,sizeof(float)))==NULL)
        {
        printf("Problem of memory allocation, so exit¥n");
        exit();
        }
for ( frame = 0 ; frame < max_frame ; frame++ )
        {
        switch ( *(frame_type+frame) )
                {

                case silent:
                                *(frame_type_num+frame) = 193.0;
                                break;

                case voiced_consonnant:
                case unvoiced:
                                *(frame_type_num+frame) = 120.0;
                                break;

                case voiced_vowel:
                                *(frame_type_num+frame)=55.0+5.0*cos((double)frame*0.2);
                                break;

                }
        }
fwrite( frame_type_num,sizeof(float),max_frame,filou);
fclose ( filou);
free(frame_type_num);

strcpy ( file_np , file_formants ) ;
file_np[ strlen(file_formants) - 9 ] = '¥0' ;
strcat ( file_np , "_np" );
if ((( filou = fopen ( file_np , "w" ) ) == NULL )
        ||
   (fwrite( np , sizeof(int),3*max_frame,filou)!=3*max_frame))
        {
        printf("Problem of access to the file of np, so exit¥n");
        exit();
        }
fclose ( filou);
```

```
                *(werthe+3*frame+2) = 0;
                }
        else
                {
                *(werthe+3*frame  ) = -1;
                *(werthe+3*frame+1) = -1;
                *(werthe+3*frame+2) = -1;
                }
        }

if (fwrite(werthe,sizeof(int),3*max_frame,filou) != 3*max_frame )
        {
        printf("there is a file writing problem, so exit¥n");
        exit () ;
        }
fclose ( filou ) ;
free(werthe);

/*
 *
 *   THE TREATMENT STARTS
 *
 */

if ((( inrm = (double *) calloc ( 3 * max_frame , sizeof(double) ) ) == NULL )
                                    ||
   ( ( np = (int *) calloc ( 3 * max_frame , sizeof( int ) ) ) == NULL ) )
        {
        printf("there is no room left for the computations, so exit¥n");
        exit () ;
        }

formant (fff,bbb,m,np,max_frame,frame_type) ;

/*
 *
 *   THE TREATMENT IS FINISHED
 *
 */

/* now we store the results */
for ( frame=0;frame<max_frame;++frame)
        if ( *(frame_type+frame) == voiced_vowel )
                {
                *(inrm+3*frame  ) = *(fff+(m-1)*frame+*(np+3*frame  ));
                *(inrm+3*frame+1) = *(fff+(m-1)*frame+*(np+3*frame+1));
                *(inrm+3*frame+2) = *(fff+(m-1)*frame+*(np+3*frame+2));
                }
        else
                {
                *(inrm+3*frame  ) = -1.00 ;
                *(inrm+3*frame+1) = -1.00 ;
                *(inrm+3*frame+2) = -1.00 ;
                }

if (((file_formants_pointer = fopen ( file_formants , "w" )) == NULL )
```

```
free(np);
free(frame_type);
free(inrm);
free(bbb);
free(fff);
/* finished */
}
```

HUMAN INTERVENTION IN THE PROCESS

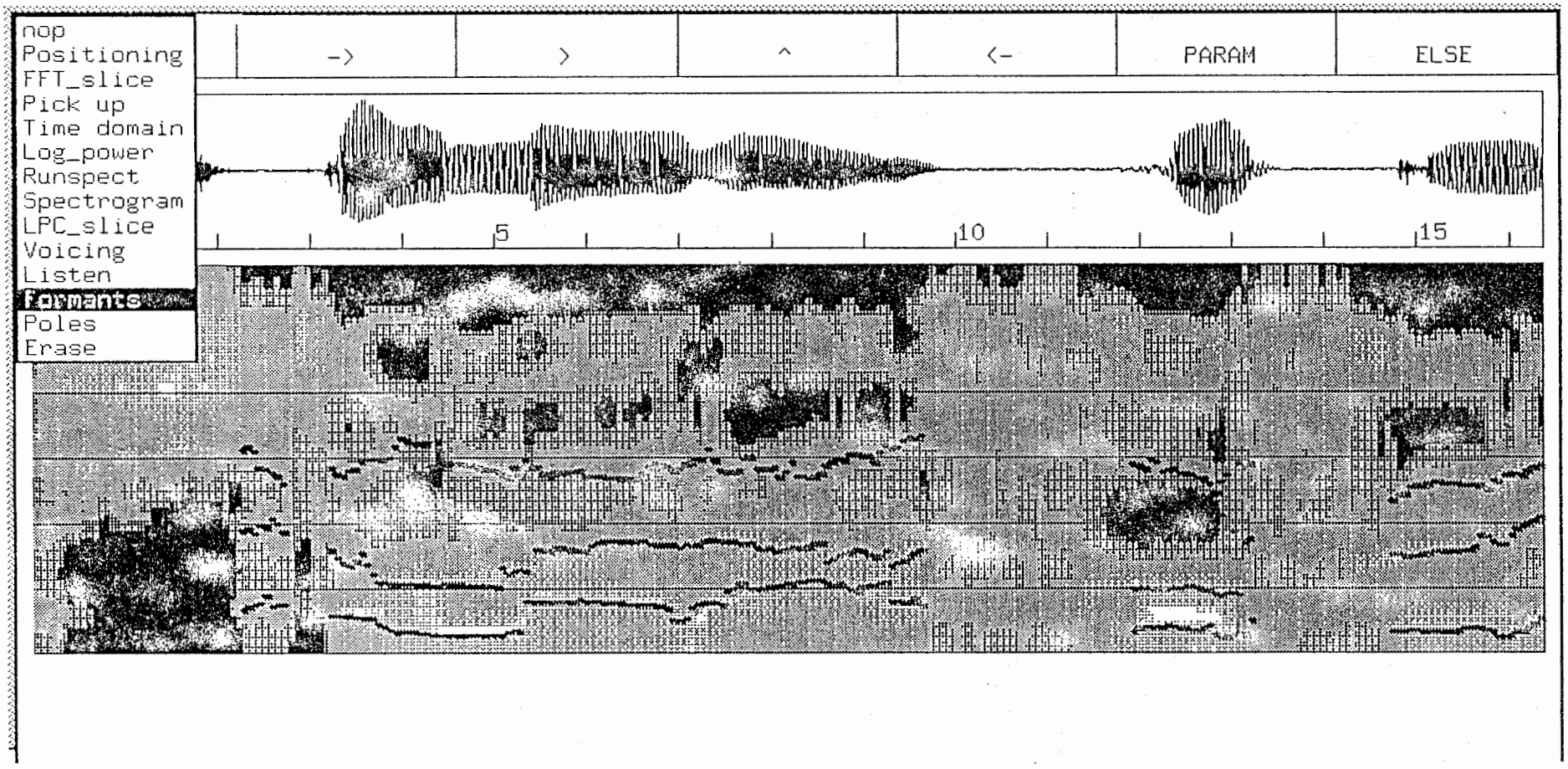
It is possible through the 'pdisp' command.

Formant Displays the current estimations on the instantaneous spectrogram.

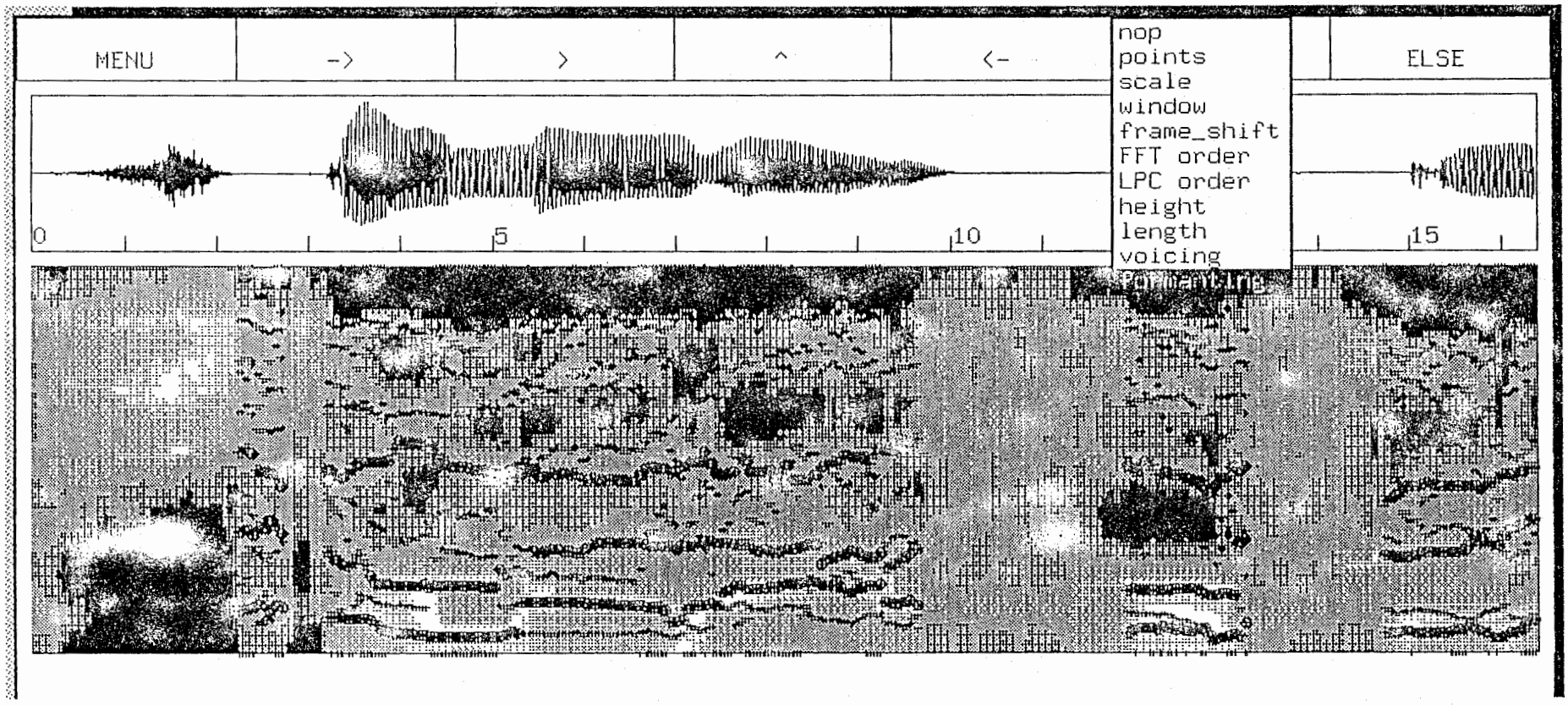
Poles Displays the frequencies of the poles on the instantaneous spectrogram.

formanting Allows the access to a new menu. Its commands are:

formant 1 formant 2 formant 3	for selecting the formant to treat
validate	for confirmating the accuracy of the values between two boundaries defined by consecutive clickings of the mouse.
devalidate	the opposite of validate
link	for demanding the selection of the shortest path between the two selections
assign	for demanding the selection of the closer-to-a-line path between the two selections
treat	for demanding the taking into account of the indications previouly provided and their processing.


nop
Positioning
FFT_slice
Pick up
Time domain
Log_power
Runspect
Spectrogram
LPC_slice
Voicing
Listen
Formants
Poles
Erase
->
>
^
<-
PARAM
ELSE
5
10
15

nop
Positioning
FFT_slice
Pick up
Time domain
Log_power
Runspect
Spectrogram
LPC_slice
Voicing
Listen
Formants
Poles
Erase
->
>
^
<-
PARAM
ELSE
5
10
15


MENU
->
>
^
<-
ELSE
nop
points
scale
window
frame_shift
FFT order
LPC order
height
length
voicing
0
5
10
15

DISPLAYING AND CORRECTING THE PITCH ESTIMATES

The 'pdisp' software (based on 'wdisp') allows it.

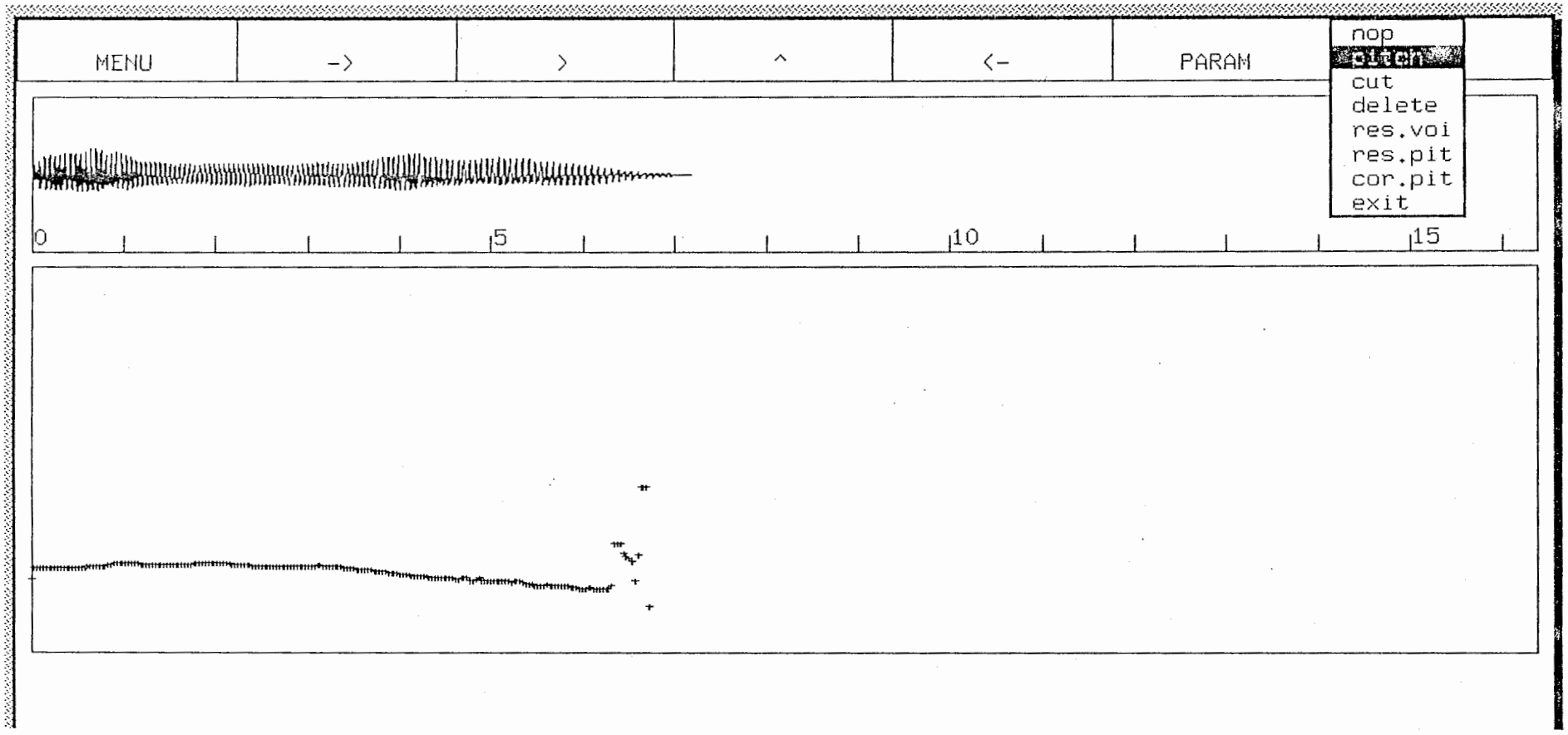
A file containing the pitch estimates is created thanks to:
res.pit Creates a 'pitch' file where all the pitches are set to 0.0 .

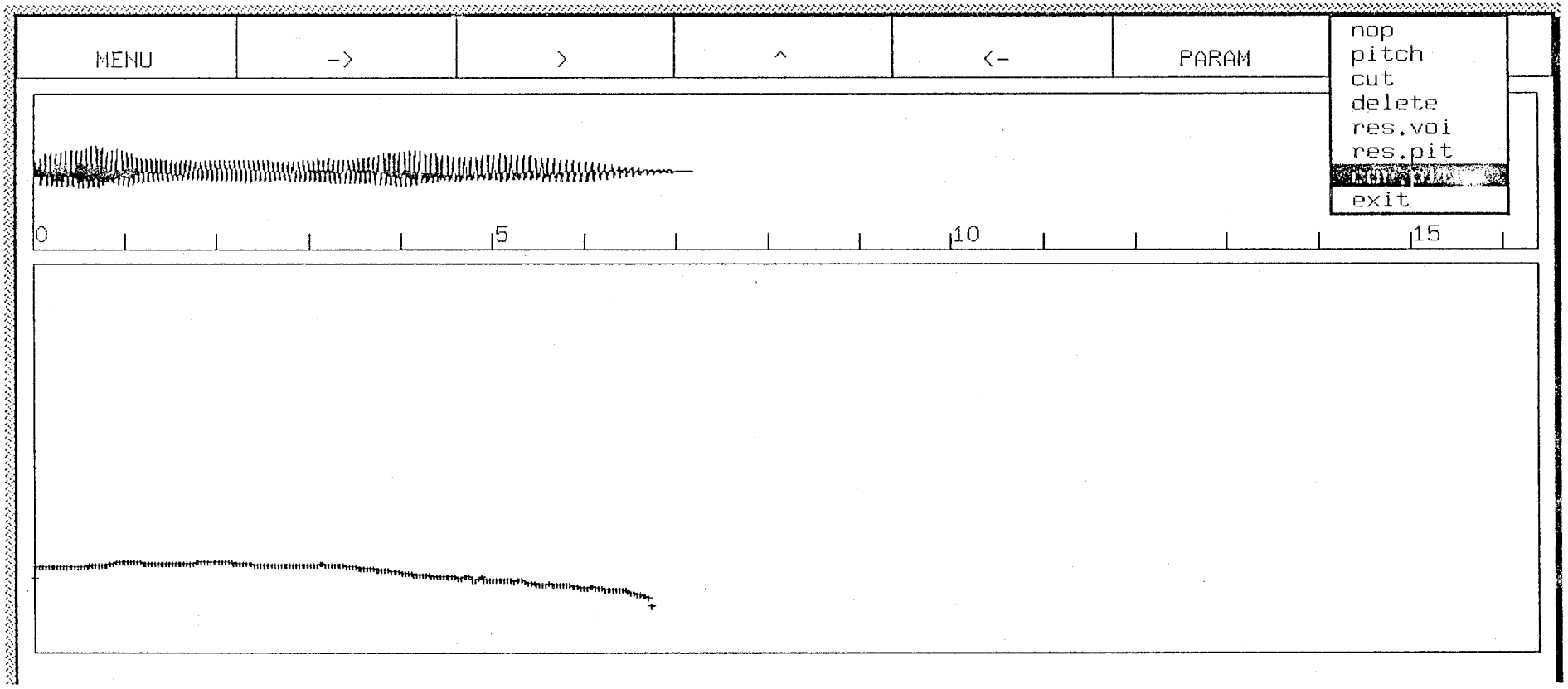
For instance, if the speech file if ./MICRO-VAX/alpha.ad
the pitch file is ./MICRO-VAX/alpha_pitch.
That file is of floating point type, since the pitch values are expressed in Hertz.
The analysis program will store in that file the estimations of those frequencies.

A display of it is available by:
pitch It shows the current pitch values assigned to the frames.
The heigth of the vindow is 1000 Hz.
Each marker corresponds to one frame.
Negative or null values of the pitch, corresponding to non-voiced frames, are not shown.

In case of an error, it is possible to change the pitch estimates:
cor.pit If this command is selected, it will be required that the mouse should be used for clicking on two points, and the intermediary values will be linearly interpolated.

MENU
->
>
<
<-
PARAM
nop
pitch
cut
delete
res.voi
cor.pit
exit
0
5
10
15


MENU
->
>
^
<-
PARAM
nop
pitch
cut
delete
res.voi
res.pit
cor.pit
exit
0
5
10
15


MENU
->
>
^
<-
PARAM
nop
pitch
cut
delete
res.voi
res.pit
exit
0
5
10
15

```
**************************************************
*                                                *
*            MAKING THE DIFFERENCE               *
*                                                *
*       BETWEEN VOICED, UNVOICED AND             *
*       SILENT PARTS IN A SPEECH WAVE            *
*                                                *
**************************************************
```

The 'pdisp' software (based on 'wdisp') allows it.
A certain knack is required for the decision which involves
taking into account the power and the spectrum of the frames.

A file containing those labels is created thanks to:
res.voi Creates a 'voicing' file where all the frames are set to
------- 'unvoiced'.

To help making a decision, the following commands are needed:

Listen — To hear a part of the speech wave by picking it up (remember to turn on the DASBOX).

Runspect — Displays the instantaneous spectrogram of the speech wave with a three-dimensionnal approach.

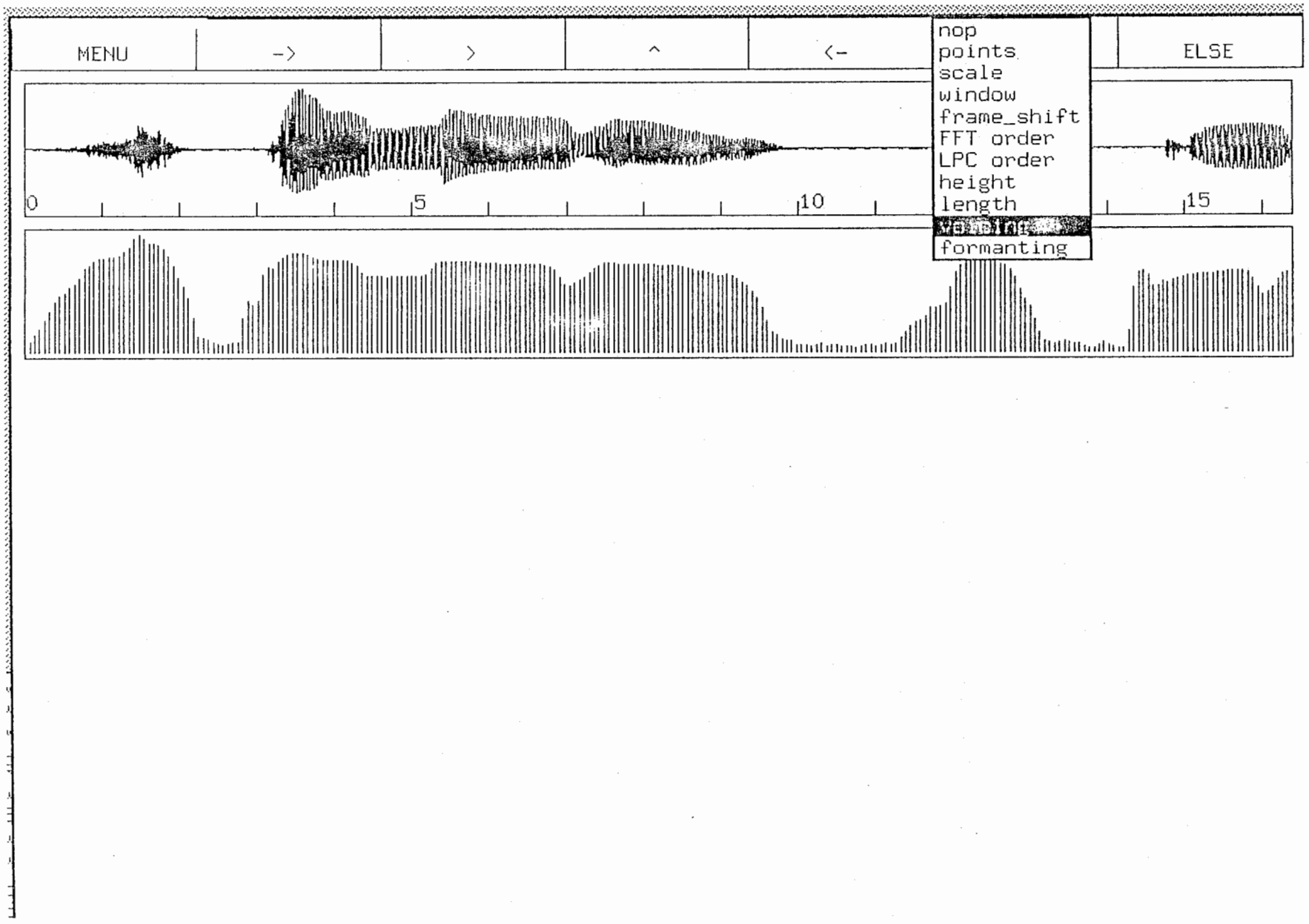
Spectrogram — Displays the instantaneous spectrogram of the speech wave with an intensity approach.

Log_power — Displays the instantaneous power of the speech wave.

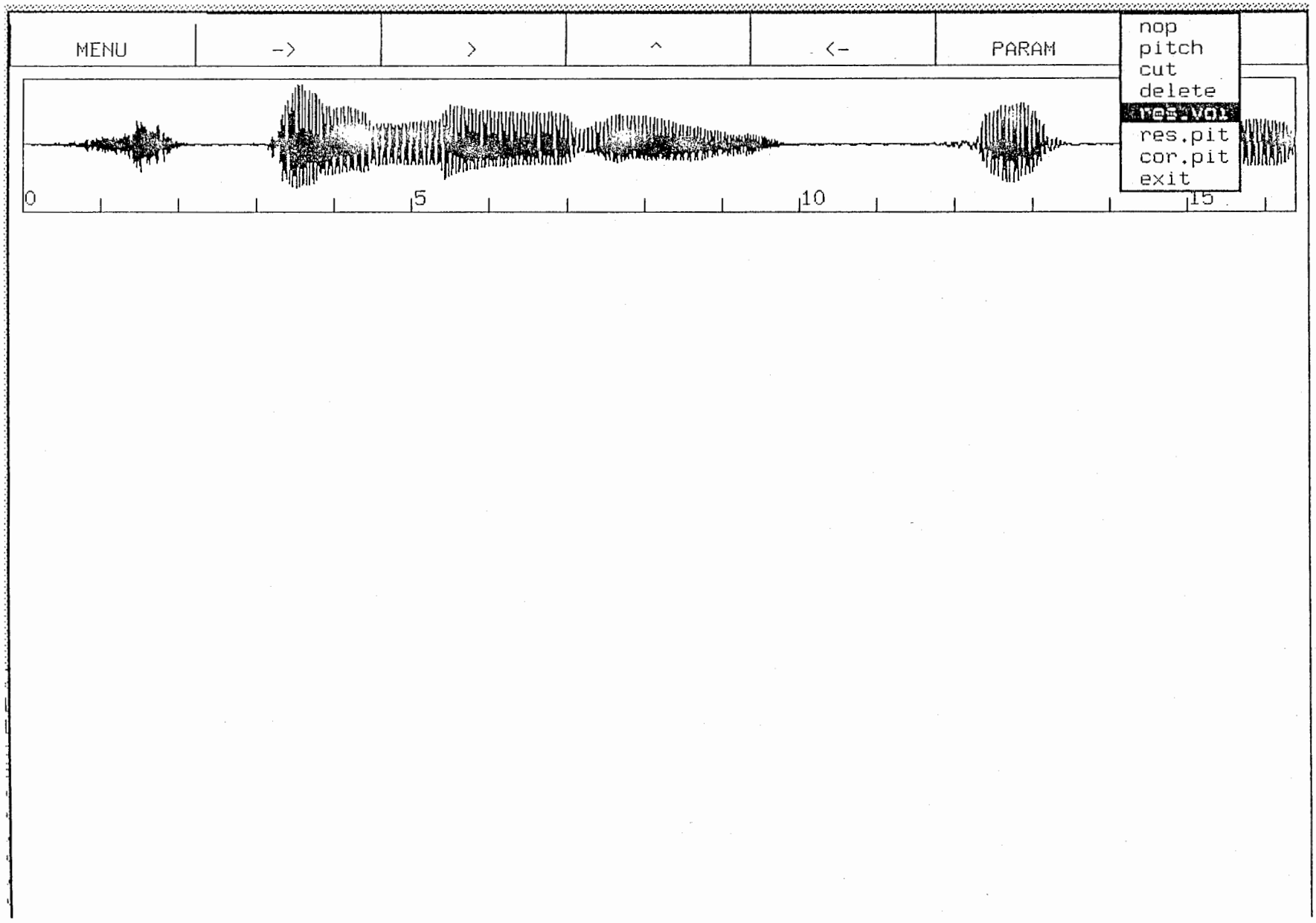
Voicing — Displays the current state of the voicing decision.
The cosine part stands for a voiced one.
The null part stands for a silent one.
The strictly positive constant part stands for an unvoiced one.

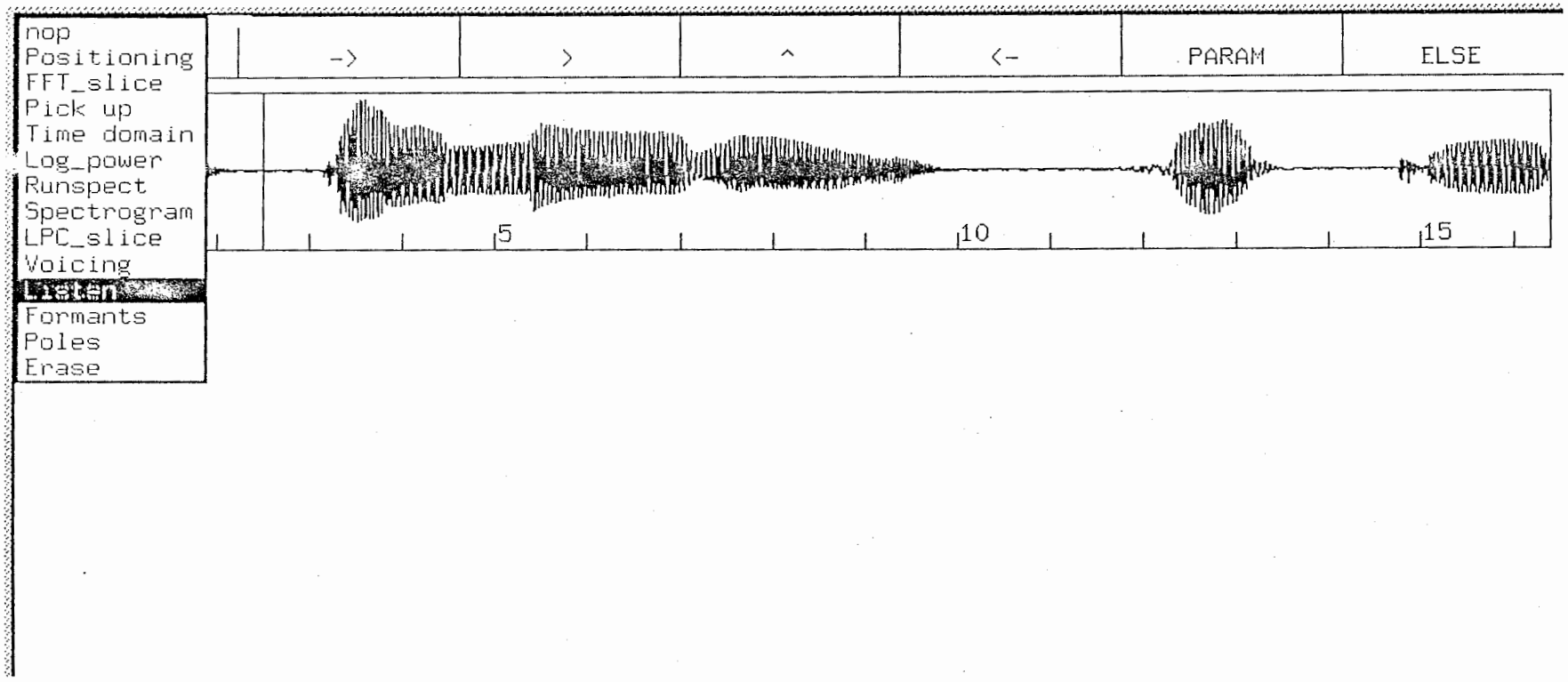
To set the label of a part, that final device proves useful:
voicing once selected, a new menu appears, and
------- through a picking up (courtesy of the mouse) in the power display
or in the spectrogram display, the selection of the voiced, unvoiced
and silent zones is carried out.

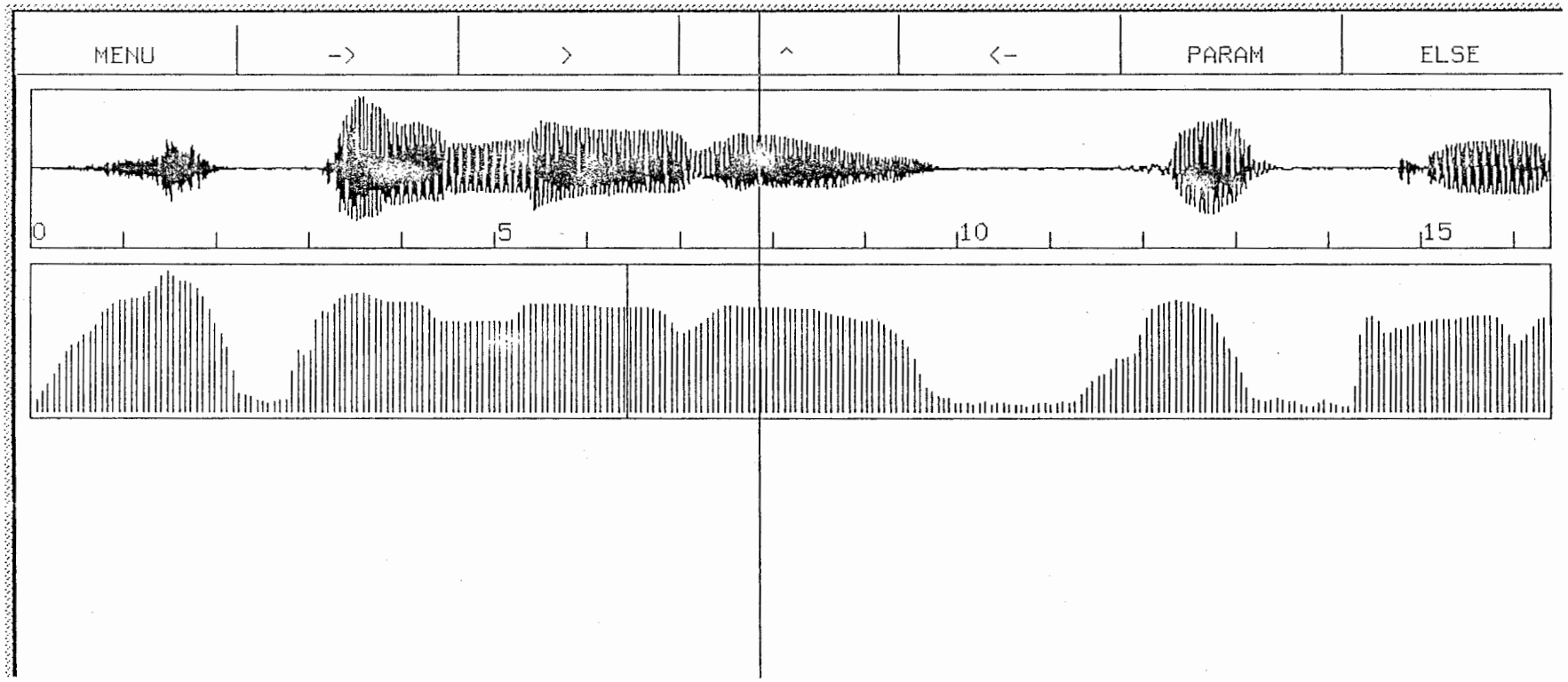
And, for instance, if the speech file if ./MICRO-VAX/alpha.ad
the voicing file is ./MICRO-VAX/alpha_voicing.
That file is of floating point type.
A voiced frame receives the value 55.0 + 5.0*cos(0.2*frame_number).
A silent frame receives the value 190.0.
An unvoiced frame receives the value 120.0.

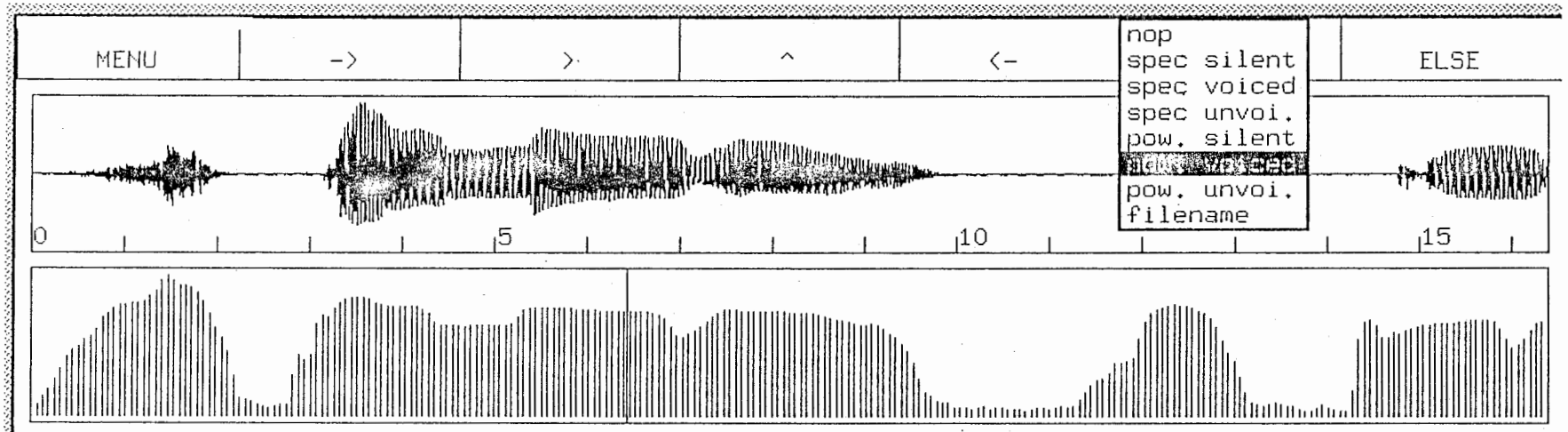

MENU
->
>
^
<-
ELSE
nop
points
scale
window
frame_shift
FFT order
LPC order
height
length
formanting
0
5
10
15

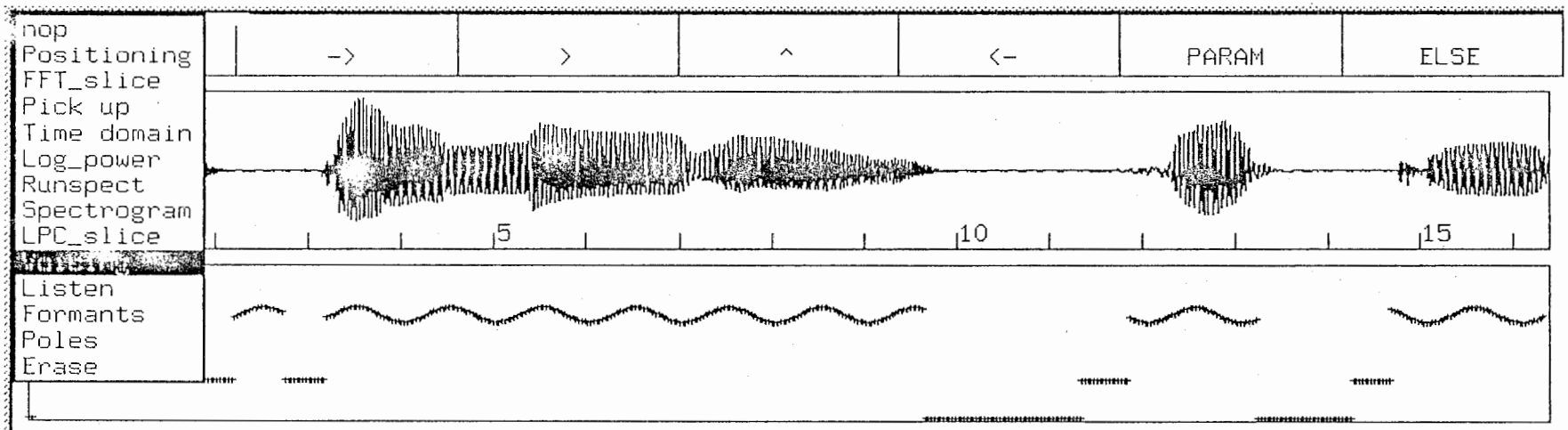
MENU
->
>
^
<-
ELSE
nop
spec silent
spec voiced
spec unvoi,
pow, silent
pow, voiced
pow, unvoi,
filename
0
5
10
15


MENU
->
>
^
<-
PARAM
nop
pitch
cut
delete
res.vol
res.pit
cor.pit
exit
0
5
10
15

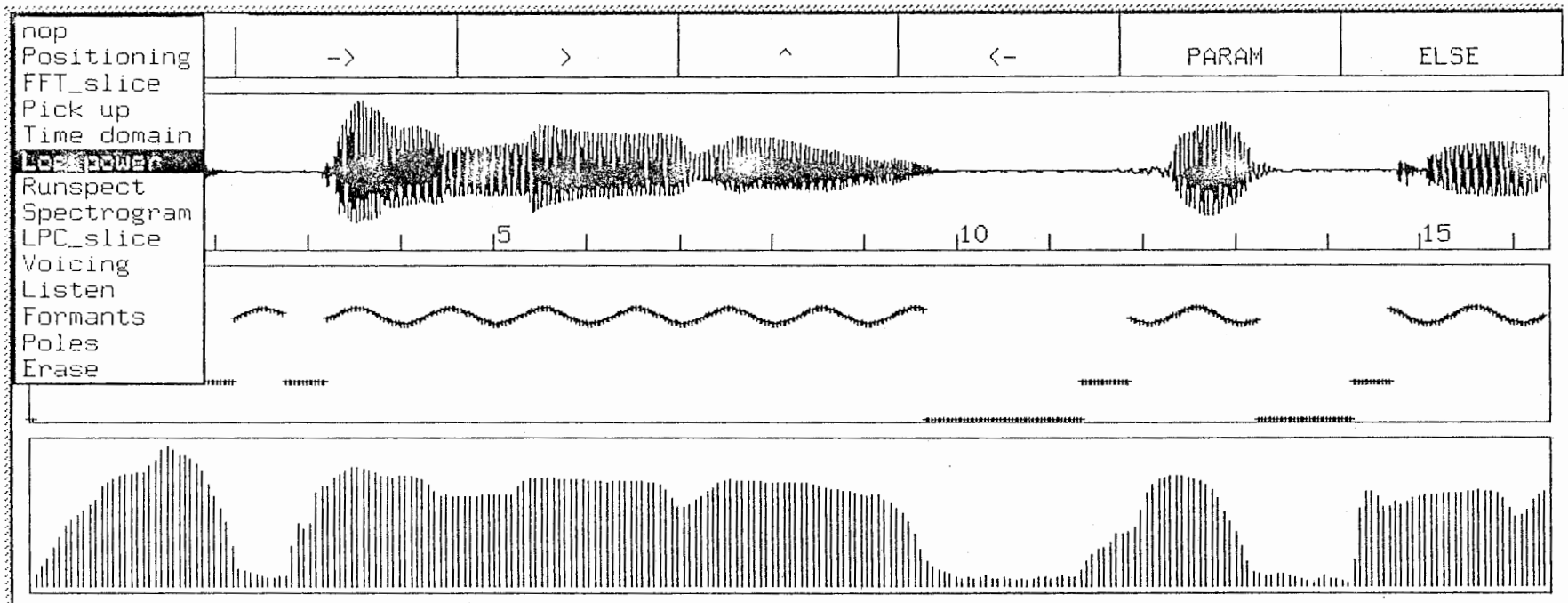

nop
Positioning
FFT_slice
Pick up
Time domain
Log_power
Runspect
Spectrogram
LPC_slice
Voicing
Listen
Formants
Poles
Erase
->
>
^
<-
PARAM
ELSE
5
10
15


MENU
->
>
^
<-
PARAM
ELSE
0
5
10
15


MENU
->
>
^
<-
ELSE
nop
spec silent
spec voiced
spec unvoi.
pow. silent
pow. voiced
pow. unvoi.
filename
0
5
10
15


nop
Positioning
FFT_slice
Pick up
Time domain
Log_power
Runspect
Spectrogram
LPC_slice
Listen
Formants
Poles
Erase
->
>
^
<-
PARAM
ELSE
5
10
15

MENU
->
>
<
<-
PARAM
ELSE
0
5
10
15


nop
Positioning
FFT_slice
Pick up
Time domain
Log power
Runspect
Spectrogram
LPC_slice
Voicing
Listen
Formants
Poles
Erase
->
>
^
<-
PARAM
ELSE
5
10
15

```
*****************************************
*                                       *
*           SYNTHESIS PART              *
*                                       *
*****************************************
```

We modelize the residual signal by a pulse train and, along with the pitch value, create a new residual signal.
This is done in 'pitch_shift'.

We change the bandwidths and the frequencies of the three formants and with our knowledge of the roots, reconstruct the LPC in 'recombination'.

Then we compute the new frames and blend them, using an overlap-add method.

```
#include <stdio.h>
#include  <math.h>

#define   COEFF         4.5
/* the value of COEFF governs the gaussian weight used
   for the recombination of the frames after change of
   the LPC, it must be at least 4.5, because smaller
   values proved not to give an enough important in the
   center weight, thus leading to bad quality speech outputs
 */
#define PI 3.141592653798
FILE    *STANDARD_INPUT;
enum harmony { silent , unvoiced , voiced_consonnant , voiced_vowel } ;
enum boolean { FALSE  , TRUE } ;

#include "/data/segot/LPC/TREAT/useful.c"
#include "/data/segot/LPC/TREAT/_complex.c"
#include "/data/segot/LPC/TREAT/recombination.c"
#include "/data/segot/LPC/TREAT/_main_help.c"
#include "/data/segot/LPC/TREAT/pitch_shift.c"

/*****************************************
 *                                       *
 *      THIS IS THE MAIN PROGRAM         *
 *                                       *
 *****************************************/

main(argc,argv)
int   argc  ;
char *argv[];
{
enum harmony *frame_type ;
enum boolean flag_existence_lpc_residus_power ;

char    buffer[ 90 ] , file_output[ 90 ] , file_inputing[90] ;
char    file_np_char[90],file_inrm_char[90],file_pitch_char[90],taratata[90] ;

short   *s_int ;
int     i , nw , m , *np , ldata , df_counter , skippy , first_sample ;
int     frame_shift , max_frame , frame , j , k ;

float   *pitch_frequency ;

double  *a_buffer , *fff , *bbb , *hamming , *F , *inrm , *x , BQQQ[3] ;
double  df[ 3 ] , dfmax[ 3 ] , dfav[ 3 ] , *s , *punderation , pitch_shift ;
double  *pf , *pb , fs  , power_buffer , ratio , freq_shift_3 , y[256] ;

FILE    *file_output_pointer , *lpc , *residus , *power , *file_fff_pointer ;
FILE    *frame_type_store ;
FILE    *file_bbb_pointer , *file_inrm_pointer , *file_pitch_pointer , *file_np_pointer ;

/******************/
/*                */
/*  PUMP PRIMING  */
/*                */
```

```
/*******************/

if ( argc != 6 )
        {
        printf(" usage _main.c BQ[0] BQ[1] BQ[2] FS_3 PITCH_SHIFT¥n");
        exit ( 1 ) ;
        }

BQQQ[0]      = atof(argv[1]);
BQQQ[1]      = atof(argv[2]);
BQQQ[2]      = atof(argv[3]);
freq_shift_3 = atof(argv[4]);
pitch_shift  = atof(argv[5]);

/*
 *
 * we inquire about the existence of the _storing files
 * so as to decide if we have to create them or not
 *
 * their use is to avoid in case of several computations with
 * the same inputs the reading of the same data with the slow
 * 'fscanf' i/o function, and replace this reading with
 * some file readings which is more efficient
 *
 * a human readable form for the first inputs has been chosen
 * in order to allow direct checking of them
 *
 */

strcpy(taratata,"salut_les_copains");

if ( rename ( "a_storing" , taratata ) == -1 )
        flag_existence_lpc_residus_power = FALSE ;
else    {
        flag_existence_lpc_residus_power = TRUE  ;
        rename ( taratata , "a_storing" ) ;
        }

/*
 *
 *   INPUT SEQUENCE
 *
 */

if(( scanf ( "%s" , file_output ) == EOF )
      ||
   ( scanf ( "%s" , file_inputing ) == EOF ))
        iki() ;

if ( ( STANDARD_INPUT = fopen ( file_inputing , "r" ) ) == NULL )
        {
        printf("exit exit exit exit problem since the beginnig!!!!!exit exit exit¥n");
        exit ( 1 ) ;
        }
```

```
do
if ( fscanf (STANDARD_INPUT, "%s" , buffer ) == EOF ) iki() ;
while ( strcmp(buffer,"from_file")!=0);
fscanf(STANDARD_INPUT,"%s",buffer) ;

buffer[strlen(buffer)-3] = '¥0' ;
strcpy(file_np_char,buffer);
strcat ( file_np_char , "_np" ) ;
if( (file_np_pointer=fopen(file_np_char,"r")) == NULL ) exit();
strcpy(file_inrm_char,buffer);
strcat ( file_inrm_char , "_formants" ) ;
if( (file_inrm_pointer=fopen(file_inrm_char,"r")) == NULL ) exit();
strcpy(file_pitch_char,buffer);
strcat ( file_pitch_char , "_pitch" ) ;
if( (file_pitch_pointer=fopen(file_pitch_char,"r")) == NULL ) exit();

/* buffer contains now the name of the original file of raw data that were
   treated, we are at the beginning of the data to input */

if (lecture(buffer,"sampling_frequency_in_Hz"))
        fscanf (STANDARD_INPUT,"%lf",&fs);
else iki();

if (lecture(buffer,"from_sample"))
        fscanf (STANDARD_INPUT,"%d",&first_sample);
else iki();

if (lecture(buffer,"to_sample"))
        fscanf (STANDARD_INPUT,"%d",&ldata);
else iki();

if (lecture(buffer,"order_of_analysis"))
        fscanf (STANDARD_INPUT,"%d",&m);
else iki();

if (lecture(buffer,"number_of_data_in_it"))
        fscanf (STANDARD_INPUT,"%d",&nw);
else iki();

if (lecture(buffer,"frame_shift"))
        fscanf (STANDARD_INPUT,"%d",&frame_shift);
else iki();
if ( frame_shift == 32  ) skippy = 0 ;
if ( frame_shift == 64  ) skippy = 1 ;
if ( frame_shift == 128 ) skippy = 3 ;
if ( frame_shift == 256 ) skippy = 7 ;

fseek(file_np_pointer,3*first_sample/32*sizeof(int),0);
fseek(file_inrm_pointer,3*first_sample/32*sizeof(double),0);
fseek(file_pitch_pointer,first_sample/32*sizeof(float),0);

if (lecture(buffer,"total_number_of_frames"))
        fscanf (STANDARD_INPUT,"%d",&max_frame);
else iki();

if (!lecture(buffer,"the_frame_number"))    iki();
```

```
if (flag_existence_lpc_residus_power==FALSE)
        {
        if ( (lpc=fopen("a_storing","w")) == NULL )
                {
                printf("cannot open file of lpc, so exit");
                exit();
                }
        if ( (residus=fopen("residues_storing","w")) == NULL )
                {
                printf("cannot open file of residues, so exit");
                exit();
                }
        if ( (power=fopen("power_storing","w")) == NULL )
                {
                printf("cannot open file of powers, so exit");
                exit();
                }
        if ( (frame_type_store=fopen("frame_type_storing","w")) == NULL )
                {
                printf("cannot open file of frame_types, so exit");
                exit();
                }
        }


if (((hamming = (double *) malloc( 2*nw*sizeof(double))) == NULL )
     ||
    ((F = (double *) malloc( 2*nw*sizeof(double) ) ) == NULL ) )
        {
        printf("error of space¥nexit¥n");exit();
        }
for ( i = 0 ; i < 2*nw ; ++i )
        {
        *(hamming+i) = 0.54 + 0.46 * cos(0.01231997119*(2.0*(double)i-(double)(nw-1)));
        *(F+i) = exp(-COEFF*( (double)(2*i) / (double) nw  - 1.0 ) *
                        ( (double)(2*i) / (double) nw  - 1.0 ) ) ;
        }

if ((x=(double *)calloc(nw , sizeof(double))) == NULL )
        {
        printf("there is no room left for the computations, case 1, so exit");
        exit () ;
        }

if (((s=(double *)calloc(ldata-first_sample+1,sizeof(double)))==NULL)
        ||
    ((frame_type=(enum harmony *)calloc(max_frame,sizeof(enum harmony)))==NULL)
        ||
    ((punderation=(double *)calloc(ldata-first_sample+1,sizeof(double)))==NULL)
        ||
    ((s_int=(short *)calloc(ldata-first_sample+1,sizeof(short)))==NULL)
        ||
    ((bbb=(double *)calloc((m-1)*max_frame+1,sizeof(double)))==NULL))
        {
        printf("there is no room left for the computations, case 3, so exit");
```

```
        exit () ;
        }

if (((fff=(double *)calloc((m-1)*max_frame+1,sizeof(double)))==NULL)
        ||
   ((a_buffer=(double *)calloc(m,sizeof(double)))==NULL)
        ||
   ((pf=(double *)calloc(m-1,sizeof(double)))==NULL)
        ||
   ((pb=(double *)calloc(m-1,sizeof(double)))==NULL))
        {
        printf("there is no room left for the computations, case 4, so exit");
        exit () ;
        }
/*  LOOP OF INPUT  */
if (flag_existence_lpc_residus_power==FALSE)
        {
        do
        {
        if ( fscanf (STANDARD_INPUT, "%d" ,&frame ) == EOF ) iki() ;
        frame -= 1 ;
        if (lecture(buffer,"is_a_voiced_vowel_one"))
                *(frame_type+frame) = voiced_vowel    ;
        else if (strcmp(buffer,"is_a_voiced_consonnant_one")==0)
                *(frame_type+frame) = voiced_consonnant    ;
        else if (strcmp(buffer,"is_a_silent_one")==0)
                *(frame_type+frame) = silent    ;
        else if (strcmp(buffer,"is_an_unvoiced_one")==0)
                *(frame_type+frame) = unvoiced ;
        else iki()                                  ;

        if   (lecture(buffer,"its_power_is")==0) iki();
        else fscanf (STANDARD_INPUT,"%lf",&power_buffer);

        if (fwrite(&power_buffer,sizeof(double),1,power) != 1 )
                {
                printf("problem with the storing of powers. Exit");
                exit();
                }

        switch ( *(frame_type+frame) )
        {
        case silent     :
                        for ( i = 0 ; i < nw ; i++ )
                                *(x+i) = 0.0 ;
                        if (fwrite(x,sizeof(double),nw,residus) != nw )
                                {
                                printf("problem with the storing of residues. Exit");
                                exit(1);
                                }
                        break ;

        case unvoiced   :
                        scratch_residual ( x ) ;
                        if (fwrite(x,sizeof(double),nw,residus) != nw )
                                {
```

```
                                printf("problem with the storing of residues. Exit");
                                exit(1);
                                }
                        break ;

        case voiced_vowel:
        case voiced_consonnant: fff += (m-1)*frame ;
                                bbb += (m-1)*frame ;
                                scratch ( a_buffer , fff , bbb , m ) ;
                                if (fwrite(a_buffer,sizeof(double),m,lpc)!=m )
                                        {
                                        printf("problem with the storing of LPC, so exit")
;
                                        exit(1);
                                        }
                                fff -= (m-1)*frame ;
                                bbb -= (m-1)*frame ;
                                scratch_residual ( x ) ;
                                if (fwrite(x,sizeof(double),nw,residus) != nw )
                                        {
                                        printf("problem with the storing of residues. Exit
");
                                        exit(1);
                                        }
                                break ;
        }
        }
        while(lecture(buffer,"the_frame_number")&&( frame!=max_frame-1));
/* END OF THE LOOP OF INPUT */
        if(fwrite(frame_type,sizeof(enum harmony),max_frame,frame_type_store)!=max_frame)
                {
                printf("problem with the storing of powers. Exit");
                exit(1);
                }

        fclose( frame_type_store ) ;
        fclose( lpc ) ;
        fclose( residus ) ;
        fclose( power ) ;
        }

if (flag_existence_lpc_residus_power==TRUE)
        {
        file_inrm_char[strlen(file_inrm_char)-9] = '¥0';
        strcat ( file_inrm_char , "_poles" ) ;
        if( (file_fff_pointer=fopen(file_inrm_char,"r")) == NULL )
                {
                printf("cannot read the poles, so exit¥n");
                exit(1);
                }
        fseek(file_fff_pointer,(m-1)*first_sample/frame_shift*sizeof(double),0);
        fread(fff,sizeof(double),max_frame*(m-1),file_fff_pointer);
        fclose(file_fff_pointer);
        file_inrm_char[strlen(file_inrm_char)-6] = '¥0';
        strcat ( file_inrm_char , "_bb" ) ;
        if( (file_bbb_pointer=fopen(file_inrm_char,"r")) == NULL )
                {
                printf("cannot read the bandwidths, so exit¥n");
```

```
                exit(1);
                }
        fseek(file_bbb_pointer,(m-1)*first_sample/frame_shift*sizeof(double),0);
        fread(bbb,sizeof(double),max_frame*(m-1),file_bbb_pointer);
        fclose(file_bbb_pointer);
        file_inrm_char[strlen(file_inrm_char)-3] = '¥0';
        strcat ( file_inrm_char , "_formants" ) ;
        }


/*
 *
 *      display of the parameters
 *      of the synthesis
 *
 */


printf ( "sampling_frequency_in_hz %f¥n",fs )                                   ;
printf ( "from_sample  %u¥n", first_sample )                                    ;
printf ( "to_sample  %u¥n", ldata )                                     ;
printf ( "order_of_analysis  %u¥n", m )                                         ;
printf ( "number_of_data_in_it %d¥n", nw )                                      ;
printf ( "frame_shift %d¥n" , frame_shift )                                     ;
printf ( "total_number_of_frames %d¥n" , max_frame )                            ;

/*
 *
 *   THE TREATMENT STRARTS
 *
 */

if ((( inrm = (double *) calloc ( 3 * max_frame , sizeof(double) ) ) == NULL )
        ||
    ((pitch_frequency=(float *)calloc(max_frame,sizeof(float)))==NULL)
        ||
   ( ( np = (int *) calloc ( 3 * max_frame , sizeof( int ) ) ) == NULL ))
        {
        printf("there is no room left for the computations, case 10, so exit");
        exit () ;
        }

if ((fread( np , sizeof(int) , 3*max_frame , file_np_pointer ) != 3*max_frame )
        ||
    (fread(pitch_frequency,sizeof(float),max_frame,file_pitch_pointer)!=max_frame)
        ||
    (fread( inrm , sizeof(double) , 3*max_frame , file_inrm_pointer ) != 3*max_frame ))
        {
        printf("problem of acces to the files concerning the formants, so exit.¥n");
        exit(1);
        }

fclose ( file_np_pointer ) ;
fclose ( file_pitch_pointer ) ;
fclose ( file_inrm_pointer ) ;
```

```
ldata = 0 ;
df_counter   = 0 ;
for ( i = 0 ; i < 3 ; ++i )
        {
        dfmax[ i ] = 0 ;
        dfav [ i ] = 0 ;
        }


if ( (lpc=fopen("a_storing","r")) == NULL )
        {
        printf("cannot open file of lpc coefficients, so exit¥n");
        exit(1);
        }

if ( (frame_type_store=fopen("frame_type_storing","r")) == NULL )
        {
        printf("cannot open file of frame_type, so exit¥n");
        exit(1);
        }

if ( (residus=fopen("residues_storing","r")) == NULL )
        {
        printf("cannot open file of residues, so exit¥n");
        exit(1);
        }

if ( (power=fopen("power_storing","r")) == NULL )
        {
        printf("cannot open file of powers, so exit");
        exit(1);
        }

if ( fread(frame_type,sizeof(enum harmony),max_frame,frame_type_store) != max_frame )
                {
                printf( "cannot read the frame_type, so exit¥n" );
                exit(1);
                }

for ( frame = 0 ; frame < max_frame ; ++frame )
{
ratio = 1 ;
if ( fread(&power_buffer,sizeof(double),1,power) != 1 )
                {
                printf( "cannot read the power, so exit¥n" );
                exit(1);
                }

np += 3*frame*skippy ;
inrm += 3*frame*skippy ;
pitch_frequency += frame*skippy ;

switch ( *frame_type )
        {
        case silent     :       if ( fread(x,sizeof(double),nw,residus) != nw )
                                        {
```

```
                                printf( "cannot read residues, so exit¥n" );
                                exit(1);
                                }
                        break ;

case unvoiced   :       if ( fread(x,sizeof(double),nw,residus) != nw )
                                {
                                printf( "cannot read residues, so exit¥n" );
                                exit(1);
                                }
                        break ;

case voiced_consonnant: if ( fread(a_buffer,sizeof(double),m,lpc) != m )
                                {
                                printf( "cannot read lpc coefficients, so exit¥n"

                                exit(1);
                                }
                        if ( fread(x,sizeof(double),nw,residus) != nw )
                                {
                                printf( "cannot read residues, so exit¥n" );
                                exit(1);
                                }
                        from_resid_to_wave (x,a_buffer,nw,m) ;
                        break ;

case voiced_vowel :

                        fff += frame*(m-1) ;
                        bbb += frame*(m-1) ;

                        ++df_counter ;

                        *(inrm+3*frame  ) *= (1+freq_shift_3/100.0);
                        *(inrm+3*frame+1) *= (1+freq_shift_3/100.0);
                        *(inrm+3*frame+2) *= (1+freq_shift_3/100.0);
                        recombination(a_buffer,fff,pf,bbb,pb,m,frame,np,inrm,BQQQ,
;



                        for ( i = 0 ; i < 3 ; ++i )
                        {
                        if (*(fff+*(np+3*frame+i))!= 0.0)
                        df[i]=(*(pf+ *(np+3*frame+i))-*(fff+*(np+3*frame+i)))/
                                *(fff+*(np+3*frame+i))*100;
                        else df[i] = 0.0 ;
                        if ( fabs(df[i])>=fabs(dfmax[i])) dfmax[i]=df[i]  ;
                        dfav[i] += fabs ( df [ i ] ) ;
                        }

                        if ( fread(x,sizeof(double),nw,residus) != nw )
                                {
                                printf( "cannot read residues, so exit¥n" );
                                exit(1);
                                }
                        /* and now we start the synthesis */
                        if ( pitch_shift != 1.0 )
                                pitch_shifting(hamming,a_buffer,x,pitch_frequency,
```

```
        frame,frame_shift,pitch_shift,nw,m,fs);
                                from_resid_to_wave (x,a_buffer,nw,m) ;

                                ratio = 0.0 ;
                                for ( i = 0 ; i < nw ; ++i )
                                        ratio += *(x+i) * *(x+i) ;
                                if (ratio!=0.0)
                                        ratio = sqrt (power_buffer/ratio) ;

                                for ( i = 0 ; i < nw ; ++i )
                                        *(x+i) *= ratio ;
                                /* de-Hamminging */
                                for ( i = 0 ; i < nw ; ++i )
                                        *(x+i) /= *(hamming+i) ;
                                /* de-emphasis */
                                for ( i = 0 ; i < nw ; ++i )
                                        y[i] = *(x+i) ;
                                for ( i = 1 ; i < nw ; ++i )
                                        *(x+i) = y[i] + *(x+i-1) ;
                                /* now, we set the average value of that frame to zero,
                                   since it is so usually in real speeches */
                                ratio = 0.0 ;
                                for ( i = 0 ; i < nw ; ++i )
                                        ratio += *(x+i) ;
                                ratio /= (double) nw ;
                                for ( i = 0 ; i < nw ; ++i )
                                        *(x+i) -= ratio ;
                                fseek(lpc,sizeof(double)*m,1) ;
                                bbb -= frame*(m-1) ;
                                fff -= frame*(m-1) ;
                        break;
        }


np -= 3*frame*skippy ;
inrm -= 3*frame*skippy ;
pitch_frequency -= frame*skippy ;

/* the method here is called overlap-add */

if ( frame_shift == nw )
        {
        for ( i = 0 ; i < nw ; ++i )
                *(s+ldata+i) = *(x+i) ;
        ldata += nw ;
        }
else
        {
        if ( frame == 0 )
                for ( i = 0 ; i < frame_shift ; ++i )
                        *(s+i) = *(x+i) ;
        else
                for ( i = 0 ; i < frame_shift ; ++i )
                        {
                        *(s+ldata+i) += *(x+i) * *(F+i) ;
                        *(punderation+ldata+i) += *(F+i) ;
```

```
                        }

        for ( i = frame_shift ; i < nw - frame_shift ; ++i )
                {
                *(s+ldata+i) += *(x+i) * *(F+i) ;
                *(punderation+ldata+i) += *(F+i) ;
                }

        if ( frame == max_frame-1 )
                {
                for ( i = nw - frame_shift ; i < nw ; ++i )
                        *(s+ldata+i) = *(x+i) ;
                ldata += nw ;
                }
        else
                {
                for ( i = nw - frame_shift ; i < nw ; ++i )
                        {
                        *(s+ldata+i) += *(x+i)**(F+i) ;
                        *(punderation+ldata+i) += *(F+i) ;
                        }
                ldata += frame_shift ;
                }
        }
frame_type++;
}
free(np);
free(frame_type);
free(inrm);
free(bbb);
free(fff);


/* we have finished the looping treating the frames one by one */
if (frame_shift != nw )
        {
        for ( i = frame_shift ; i < ldata-frame_shift ; ++i )
                *(s+i) /= *(punderation+i) ;
        }

free(punderation) ;

fclose( lpc ) ;
fclose( residus ) ;
fclose( power ) ;
fclose( frame_type_store ) ;

if ( df_counter != 0 )
        {
        for ( i = 0 ; i < 3 ; ++i )
                {
                dfav[ i ] /= (double) df_counter ;
                printf ( "formant# %d¥n" , i+1 ) ;
                printf ( "max_ratio_of_change %f¥n" , dfmax[ i ] ) ;
                printf ( "shift_average %f     in %%¥n" , dfav[ i ] ) ;
                }
```

```
        }

/* and now, we save the synthesized signal */
for ( i = 0 ; i < nw  ; ++i )
        *(s+i) *= 0.54 - 0.46 * cos((double)i/(double)nw * PI ) ;


for ( i = 0 ; i < ldata ; ++i )
        {
        if ( fabs(*(s+i)) > 31800.0 )
                {
                if ( *(s+i) > 0.0 )     *(s+i) = 31800.0 ;
                else     *(s+i) = - 31800.0 ;
                }
        *(s_int+i) = round ( *(s+i) ) ;
        }

strcpy(buffer,"/data/segot/MICRO-VAX/");
strcat(buffer,file_output);
strcat(buffer,".da");
strcpy(file_output,buffer);
if ((file_output_pointer = fopen  ( file_output , "w" )) == NULL)
        {
        printf( "Problem of file opening¥n");
        printf( "Cannot open %s¥n",file_output);
        exit(1);
        }
if (fwrite ( s_int , sizeof ( short int ) , ldata , file_output_pointer ) != ldata )
        {
        printf( "Problem of file writing¥n");
        printf( "Cannot write %s¥n",file_output);
        exit(1);
        }
fclose ( file_output_pointer ) ;
}
```

```
#include <stdio.h>
#include <math.h>

pitch_shifting ( hamming , a_buffer , x , pitch_frequency ,
frame , frame_shift , pitch_shift , nw , m , Hz_sampling_frequency )
double *hamming , *x , *a_buffer , pitch_shift , Hz_sampling_frequency ;
float  *pitch_frequency ;
int     frame , frame_shift , nw , m ;
{
int alpha , i , k , pitch , location ;
double *pitch_pulse , *pulse_train ;
static int preceeding_location;

if ((( pitch_pulse = (double *)malloc(sizeof(double)*nw) ) == NULL )
        ||
    ((  pulse_train = (double *)malloc(sizeof(double)*nw*2) ) == NULL ))
        {
        printf ("no room left for the computation, so, exit¥n");
        exit(1);
        }
if (( frame == 0 )||( *(pitch_frequency+frame-1) < 40.0 ))
        location = nw / 2 ;
else
        {
        location = preceeding_location - frame_shift ;
        while ( location <= 0 )
                location += (int) (Hz_sampling_frequency
        * pitch_shift /(double) *(pitch_frequency+frame-1) );
        }
preceeding_location = location ;

pitch = (int) ( Hz_sampling_frequency * pitch_shift / (double) *(pitch_frequency+frame) )

/* location and pitch found */
*pitch_pulse =  -1.0  ;

/* but, if we try a synthesis by residual signal
for ( i = 0 ; i < (int) ((double)pitch/pitch_shift) ; ++i )
        *(pitch_pulse+i) = *(x+location+i) / *(hamming+location+i) ; */


alpha = (nw-frame_shift*frame<0) ? 0 : nw-frame_shift*frame ;

for ( i = alpha ; i < 2*nw ; ++i )
        {
        k = i-location-nw;
        while (k<0)
                k += pitch ;
        while (k>=pitch)
                k -= pitch;
        *(pulse_train+i) = *(pitch_pulse+k) ;
        }

for ( i = 0 ; i < nw ; ++i )
        *(x+i) = *(pulse_train+i+nw) * *(hamming+i) ;
```

```
/* the residual signal has been modelized possibly by a pulse train */

free(pulse_train);
free(pitch_pulse) ;
}
```

```
/*
 *
 *  from the poles and the formants,
 *  we compute a new set of linear predictor coefficients by
 *  shifting or modifying the original frequencies and bandwidths
 *
 *  inputs  original_frequencies                ff
 *          original_bandwidths                 bb
 *          frequencies_change      shift ratio for the formant frequencies
 *          bandwidths_change       change ratio for the formant bandwidths  bq
 *          fs                      sampling frequency in Hz
 *          m                       order of the computation
 *          frame
 *          np                      number of poles corresponding to formants
 *          inrm                    normalized formant frequencies
 *
 *  outputs final_frequencies       pf
 *          final_bandwidths        pb
 *          a                       linear prediction coefficients after change
 *
 *  last change by Segot, june 23, 1987
 */

recombination ( a , ff , pf , bb , pb , m , frame , np , inrm , bq , fs )

int    *np ;
double *a , *bq ;
double *inrm ;
double *pb , *pf , *bb , *ff ;
double fs ;
int    m , frame ;
{
register i , j ;
struct complex r[30] , z[30] ;

/* we put the original values in the final arrays and we will change only the
   necessary ones */
for ( i = 0 ; i < m-1 ; ++i )
        {
        *(pf+i) = *(ff+i) ;
        *(pb+i) = *(bb+i) ;
        }

/* we presume the array ff to be sorted in an increasing order */
/* and now we scan for the values to change and do it */
/* the scanning is performed thanks to *np, the formants numbers */
for ( i = 0 ; i < 3 ; ++i )
        {
        j = *(np +3*frame + i ) ;
        *(pf+j)  = *(inrm+3*frame+i) ;
        *(pb+j) *= *(bq+i) ;
        if ((j != m-1 )&&(*(ff+j+1) == *(ff+j)))
                {
                *(pf+j+1) = *(pf+j) ;
                *(pb+j+1) = *(pb+j) ;
```

```
                }
        if ( *(ff+j-1) == *(ff+j) )
                {
                *(pf+j-1) = *(pf+j) ;
                *(pb+j-1) = *(pb+j) ;
                }
        }


/* now, we compute the roots corresponding to the new frequencies and bandwidths */
i = 0 ;
while ( i < m-1 )
        {
        z[i] = polcartes(exp(- *(pb+i) *PI/fs),*(pf+i) *2*PI/fs) ;
        if (( *(pf+i+1) == *(pf+i) )&&( *(pf+i) != 0 )&&( *(pf+i) != fs/2 ))
                {
                z[i+1] = conjugate( z[i] );
                i += 2 ;
                }
        else
                i++ ;
        }

/* now we compute the coefficients corresponding to the new roots */
/* those coefficients are put in the complex array r[] */
r[0] = negation(z[0]);
r[1].real = 1;
r[1].imaginary = 0;
for(i=1;i<m-1;++i)
        {
        r[i+1].real=1;
        r[i+1].imaginary=0;
        for(j=1;j<i+1;++j)
        r[i-j+1] = soustraction(r[i-j],multiplication(r[i-j+1],z[i]));
        r[0] = negation(multiplication(r[0],z[i]));
        }
/* we suppress now the imaginary parts (must be 0.0) of the array r to get the new lpc */
for(i=0;i<m;++i)
        *(a+i) = r[m-1-i].real;

}
```

```
#include <stdio.h>

/*
 *
 *  this function performs the comparison of two character strings
 *  it returns in case of equality 1 , otherwise 0 .
 *
 */

int equal_strings ( s1 , s2 )
  char s1 [ ] , s2 [ ] ;
  {
        if (strcmp ( s1 , s2 ) == 0 )
                return 1;
        else    return 0;
  }

/*
 *
 * this function exits of the computation should an error occur
 *
 */

void iki ( )
{
printf ( "The file to treat is not correctly formatted!¥n");
printf ( "So, we abandon its treatment.")                    ;
exit   (     )                                               ;
}

/*
 *
 * this funtion reads on a file a character string
 * and compares it to the string "constant" given in input
 * returning 1 in case of an equality
 *           0 otherwise
 *
 */

int lecture ( buffer , constant )
char buffer [ ] , constant [ ] ;
{
fscanf ( STANDARD_INPUT ,"%s" , buffer ):
return (strcmp(buffer , constant )==0):
}


/*
 *
 * this funtion reads on the standard I/O a character string
 * and compares it to the string "constant" given in input
 * returning 1 in case of an equality
 *           0 otherwise
 *
 */
```

```
int lecture_stdin ( buffer , constant )
char buffer [ ] , constant [ ] ;
{
scanf ( "%s" , buffer );
return (equal_strings ( buffer , constant )) ;
}

/*
 *
 * This function reads from a file
 *  a    linear correlation coefficients
 *  freq frequencies of the formants
 *  bw   bandwidths  of the formants
 *
 *  m is an input, and represents the order of analysis
 *     ( a[m], freq[m-1],bw[m-1] )
 *
 */

void scratch ( a , freq , bw , m )
int     m                       ;
double  *a , *freq , *bw ;
{
char z [ 62 ] ;
int i         ;

if(!lecture(z,"*************************************************************"))
        iki() ;
fscanf(STANDARD_INPUT, "%61c" , z );
if(!lecture(z,"**************************************************************"))
        iki() ;
fscanf (STANDARD_INPUT, "%26c", z );
z[26] = '¥0'    ;
a [0] = 1.0     ;
for ( i = 1 ; i < m ; ++i )
{
if (!lecture(z,"*")) iki();
if (!fscanf(STANDARD_INPUT,"%*d %1c",z)) iki();
z[1]='¥0';
if (!equal_strings(z,"*")) iki();
if ( fscanf(STANDARD_INPUT,"%lf * %lf * %lf",a+i,freq+i-1,bw+i-1)==0)   iki();
}
if(!lecture(z,"*************************************************************"))
        iki();
}



/*
 *
 * This function reads from the standard I/O
 *  a    linear correlation coefficients
 *  freq frequencies of the formants
 *  bw   bandwidths  of the formants
 *
```

```
 *  m is an input, and represents the order of analysis
 *     ( a[m], freq[m-1],bw[m-1] )
 *
 */

void scratch_stdin ( a , freq , bw , m )
int     m                       ;
double  *a , *freq , *bw ;
{
char z [ 62 ] ;
int i         ;

if(!lecture_stdin(z,"*************************************************************"))
        iki() ;
scanf( "%61c" , z );
if(!lecture_stdin(z,"*************************************************************"))
        iki() ;
scanf ( "%26c", z );
z[26] = '¥0'    ;
a [0] = 1.0     ;
for ( i = 1 ; i < m ; ++i )
{
if (!lecture_stdin(z,"*")) iki();
if (!scanf("%*d %1c",z)) iki();
z[1]='¥0';
if (!equal_strings(z,"*")) iki();
if ( scanf("%lf * %lf * %lf",a+i,freq+i-1,bw+i-1)==0)   iki();
}
if(!lecture_stdin(z,"*************************************************************"))
        iki();
}

/*
 *
 * This function reads from the file to treat the residual signal
 *
 */

void scratch_residual ( y )
double *y ;
{
char buffer[81] ;
int  i          ;

if (!lecture(buffer,"*********************************"))    iki() ;
fscanf(STANDARD_INPUT,"%35c",buffer);
if (!lecture(buffer,"*********************************"))    iki() ;
i = 0 ;
while ( !lecture(buffer,"*********************************") )
        {
        if ( fscanf(STANDARD_INPUT,"%*d * %lf", y+i ) == 0 )          iki() ;
        i++ ;
        }
}

short round(y)
```

```
double y ;
{
short z ;
z = (short) y ;
if ( y-z > .5 ) z++ ;
else if ( z-y > .5 ) z-- ;
return ( z ) ;
}
```

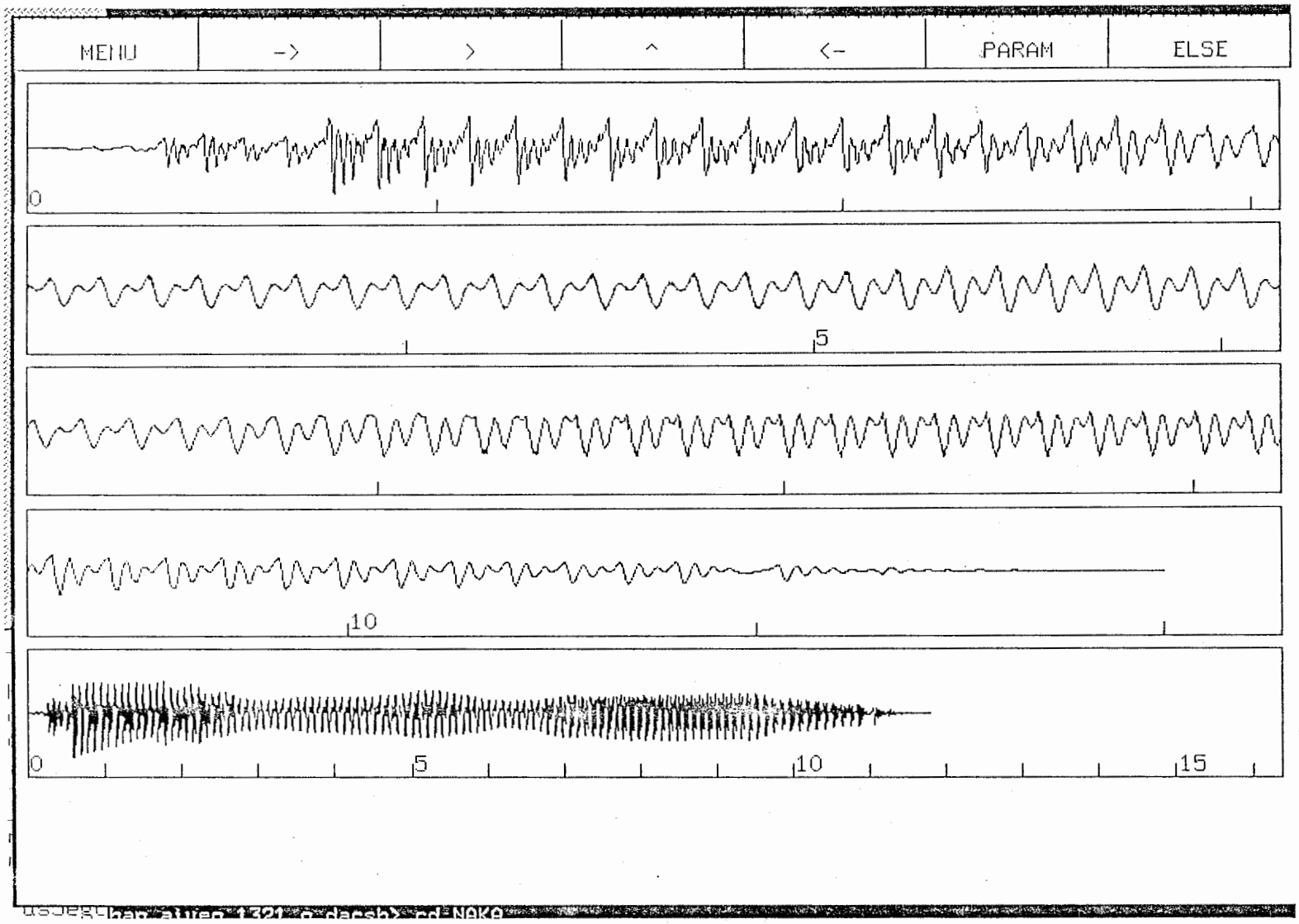
```
#include <stdio.h>
#include <math.h>
/* a few standard arithmetic operations on
 * complex numbers, implemented for convenience
 */
struct complex
{
double real ;
double imaginary ;
};

struct complex addition ( a , b )        /* returns a + b */
struct complex a , b ;
{
a.real += b.real ;
a.imaginary += b.imaginary ;
return ( a )    ;
}

struct complex multiplication ( a , b )  /* returns a * b */
struct complex a , b ;
{
struct complex c     ;
c.real = a.real*b.real-a.imaginary*b.imaginary;
c.imaginary = a.imaginary*b.real+a.real*b.imaginary;
return ( c )         ;
}

struct complex polcartes ( a , b )   /* converts from polar coordinates
                                        into a complex number         */
double a , b ;
{
struct complex c ;
c.real = a * cos ( b ) ;
c.imaginary = a * sin ( b ) ;
return ( c )        ;
}

struct complex conjugate ( z )       /* returns the conjugate of z */
struct complex z ;
{
z.imaginary =  -z.imaginary     ;
return ( z )                    ;
}

double module ( z )                  /* returns the module of z */
struct complex z ;
{
double y ;
y = sqrt ( z.real * z.real + z.imaginary * z.imaginary );
return ( y )                                            ;
}

struct complex negation ( z )        /* returns the inverse of z */
struct complex z ;
```

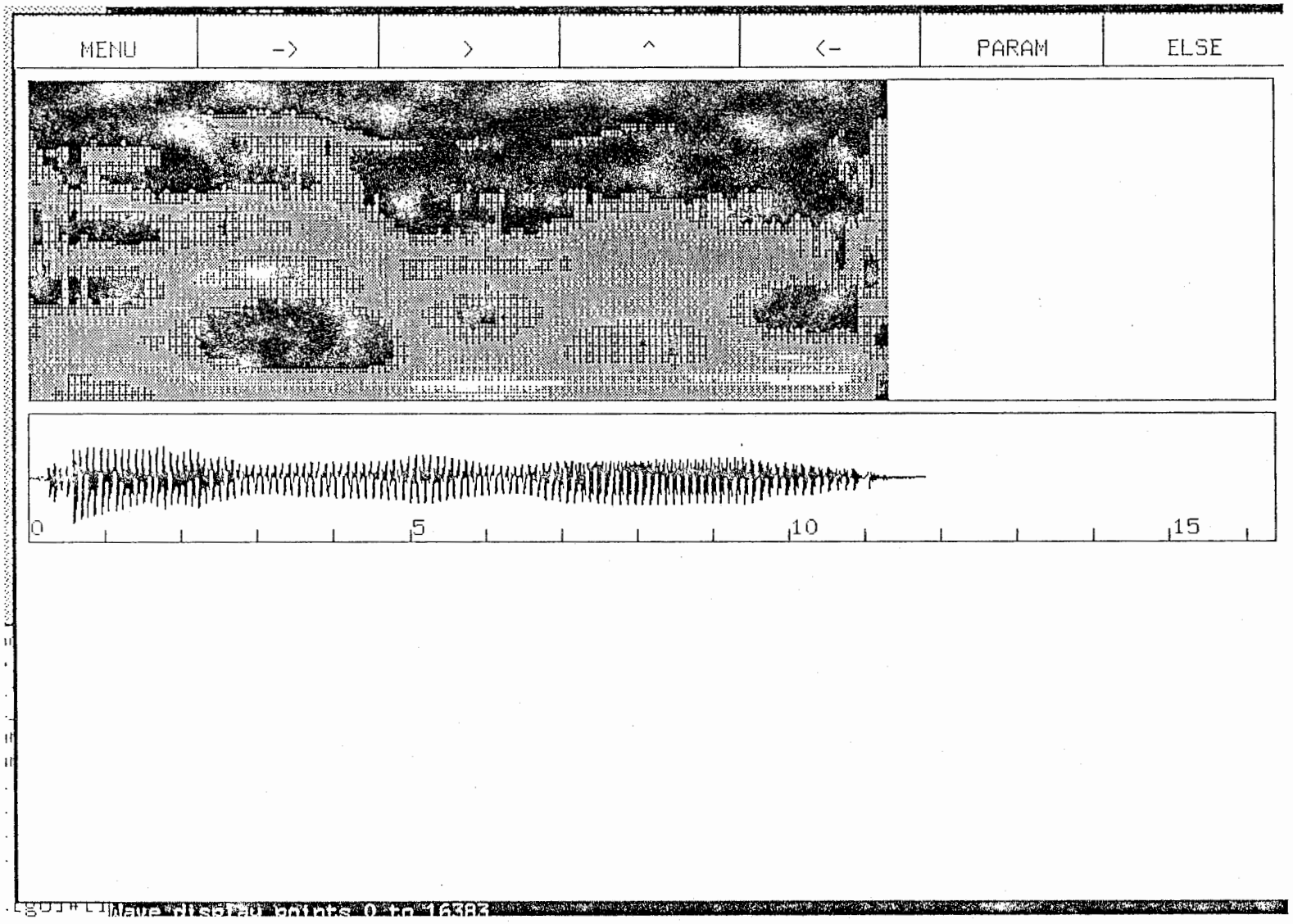
```
{
z.real  =       -z.real          ;
z.imaginary =   -z.imaginary     ;
return ( z )                     ;
}

struct complex soustraction ( z , y )  /* returns z - y */
struct complex y , z ;
{
z.real -= y.real ;
z.imaginary -= y.imaginary ;
return ( z )    ;
}

struct complex division ( z , y )      /* returns z / y */
struct complex z , y ;
{
struct complex q ;
double a ;
if ( y.imaginary == 0.0 )
        {
        z.real /= y.real ;
        z.imaginary /= y.real ;
        return ( z ) ;
        }
else if ( y.real == 0.0 )
        {
        q.real = z.imaginary/y.imaginary ;
        q.imaginary = - z.real/y.imaginary ;
        return ( q ) ;
        }
else
        {
        a = y.real/y.imaginary + y.imaginary/y.real ;
        q.real = ( z.real/y.imaginary - z.imaginary/y.real ) / a ;
        q.imaginary = ( z.real/y.real + z.imaginary/y.imaginary ) / a ;
        return ( q ) ;
        }
}
```

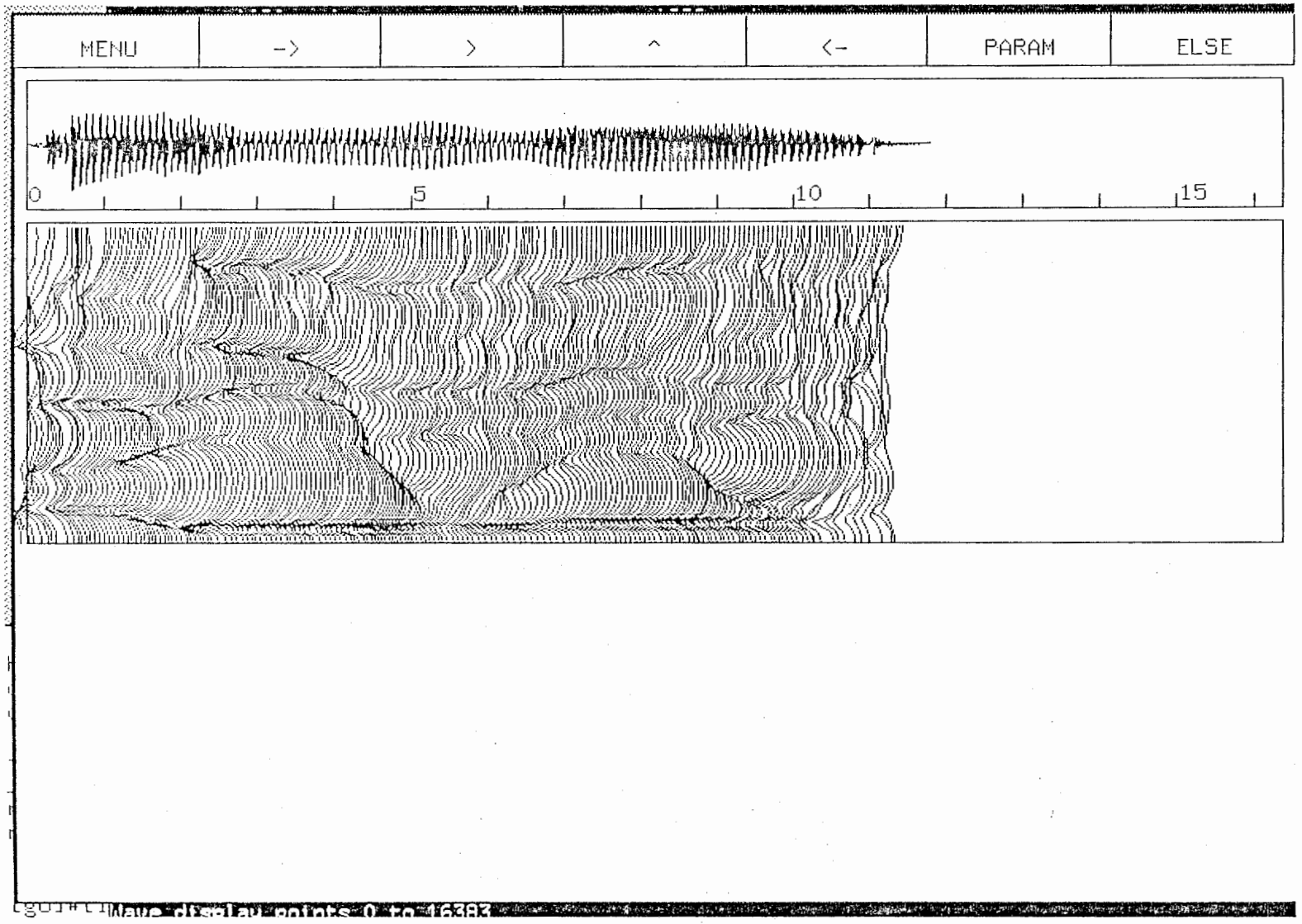
```
/* compute the residual signal */

from_resid_to_wave( x , aa , nw , m )
double *aa ;
double    *x ;
int nw , m ;
{
int i , j ;
double sum;

for ( i = 1 ; i < nw ; ++i )
        {
        sum = 0.0 ;
        for ( j = 1 ;(( j < m )&&( j < i )); ++j )
                sum -= *(aa+j) * *(x+i-j);
        *(x+i) += sum ;
        }
}

/*
 *
 * This function performs the display of the original and final parameters
 *
 */
void display ( m , a , aa , frequencies , ff , fb )
double *aa , *a , *frequencies , *ff , *fb ;
int    m ;
{
int    i , j ;

  printf("*************************************************************¥n");
  printf("*  index  *   original  *    original      *    final      *¥n");
  printf("*         *     LPC     *   frequencies    *     LPC       *¥n");
  printf("*************************************************************¥n");
  *aa = 1.000000      ;
  printf ( "*    0    *  %+.6f  *",   *aa ) ;
  printf ( "                    * %+.6f   ¥n",*aa ) ;
        for    ( i = 1 ; i < m ; ++i )
                        {
  printf ( "*   %2d    *  %+.6f  *   ", i, *(a+i));
  for ( j = 0 ; j < (5-(int)(log10(*(frequencies+i-1)))) ; ++j )
        printf ( " " ) ;
  printf ("%-5.6f  ",*(frequencies+i-1));
        printf ( "* ");
  printf ( "%+5.6f¥n",*(aa+i));
                        }
  printf("*************************************************************¥n");
  printf("*********************************************¥n");
  printf("*  index  *   normalized     *   normalized  *¥n");
  printf("*         *   frequencies    *   bandwidth   *¥n");
  printf("*********************************************¥n");
        for    ( i = 1 ; i < m ; ++i )
                        {
  printf ( "*   %2d    *" , i ) ;
```

```
  for ( j = 0 ; j < (5-(int)(log10(*(ff+i-1)))) ; ++j )
        printf ( " " ) ;
  printf ("%-5.6f   ",*(ff+i-1));
        printf ( "   * ");
  printf ( "%-5.6f¥n",*(fb+i-1));
                        }
  printf ( "*********************************************¥n");
  }
```

****
*   *
* ORIGINAL DATA *
*   *
****

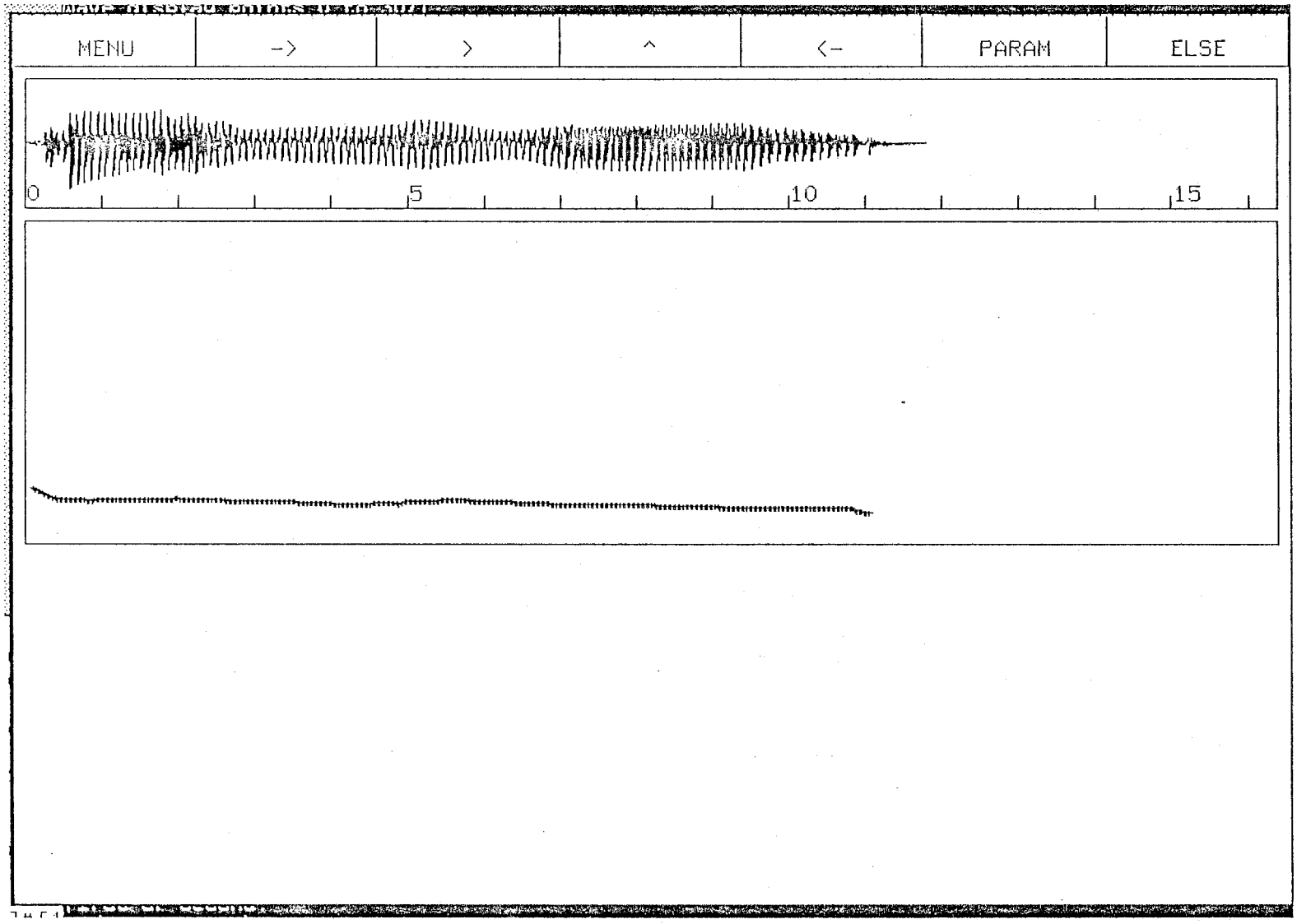

MENU
->
>
^
<-
PARAM
ELSE
0
5
10
0
5
10
15


MENU
->
>
^
<-
PARAM
ELSE
0
5
10
15

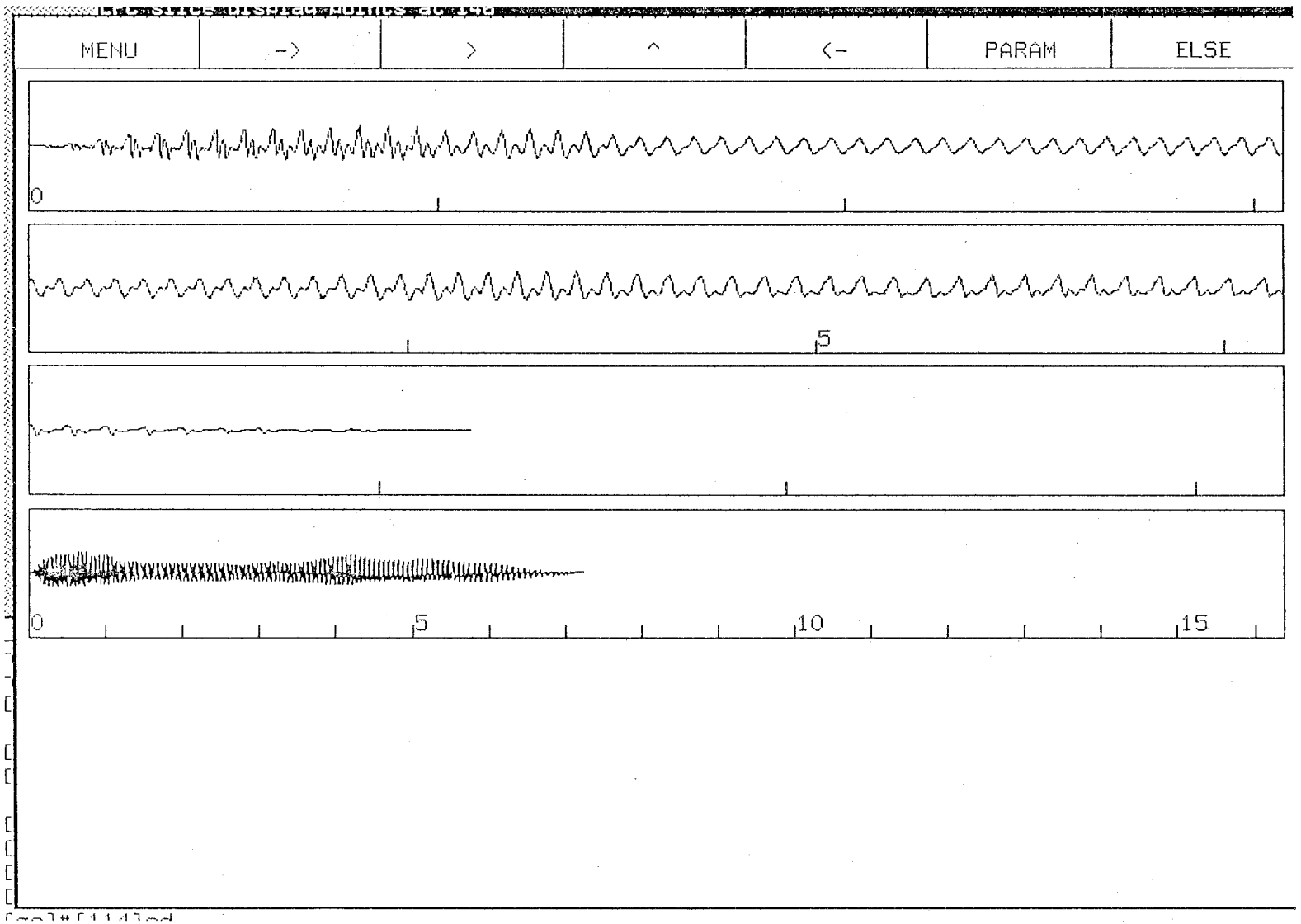

MENU
->
>
^
<-
PARAM
ELSE
0
5
10
15

MENU
->
>
>
<-
PARAM
ELSE

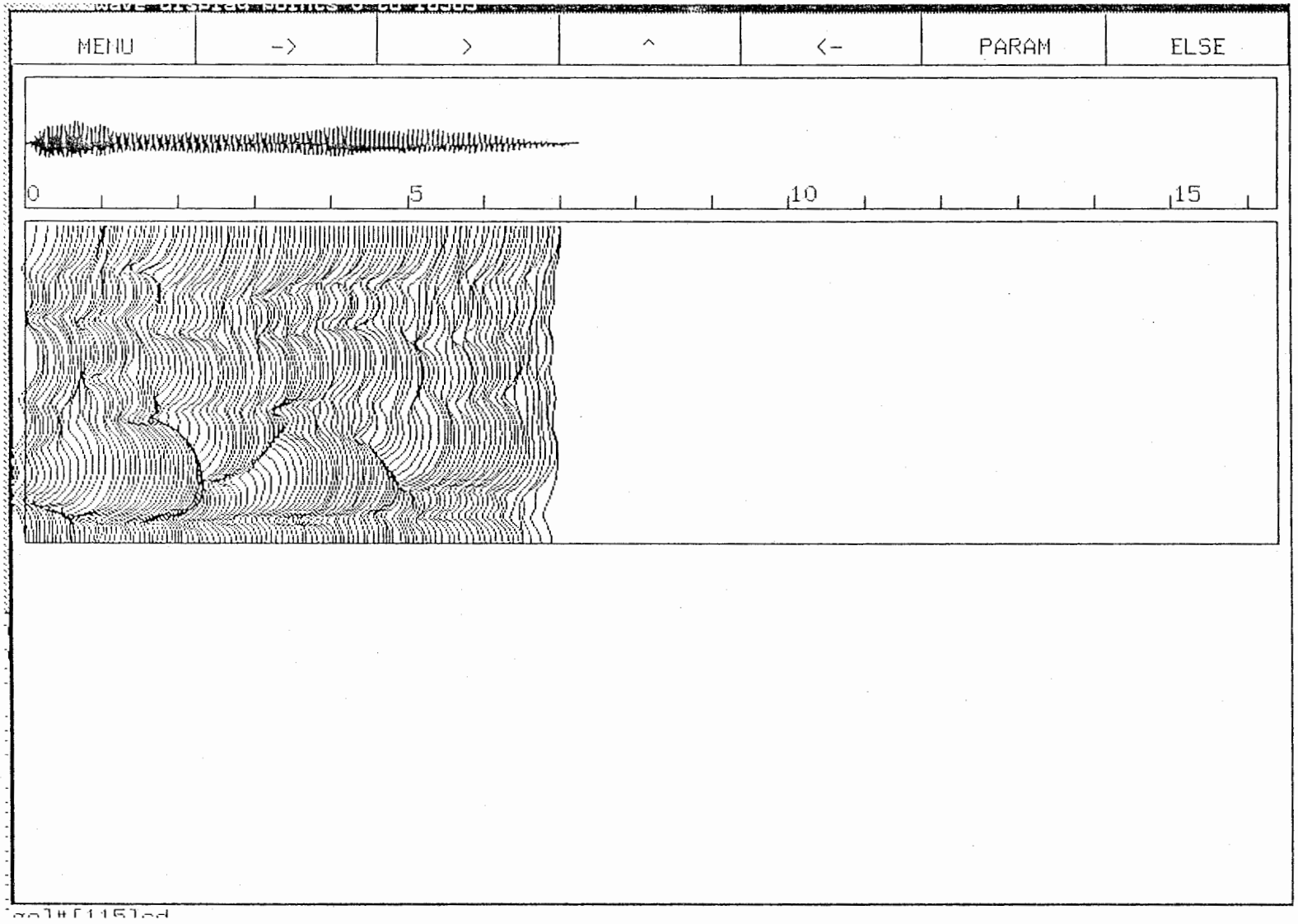
MENU
->
>
<
<-
PARAM
ELSE
0
5
10
15


MENU
->
>
^
<-
PARAM
ELSE
0
5
10
15

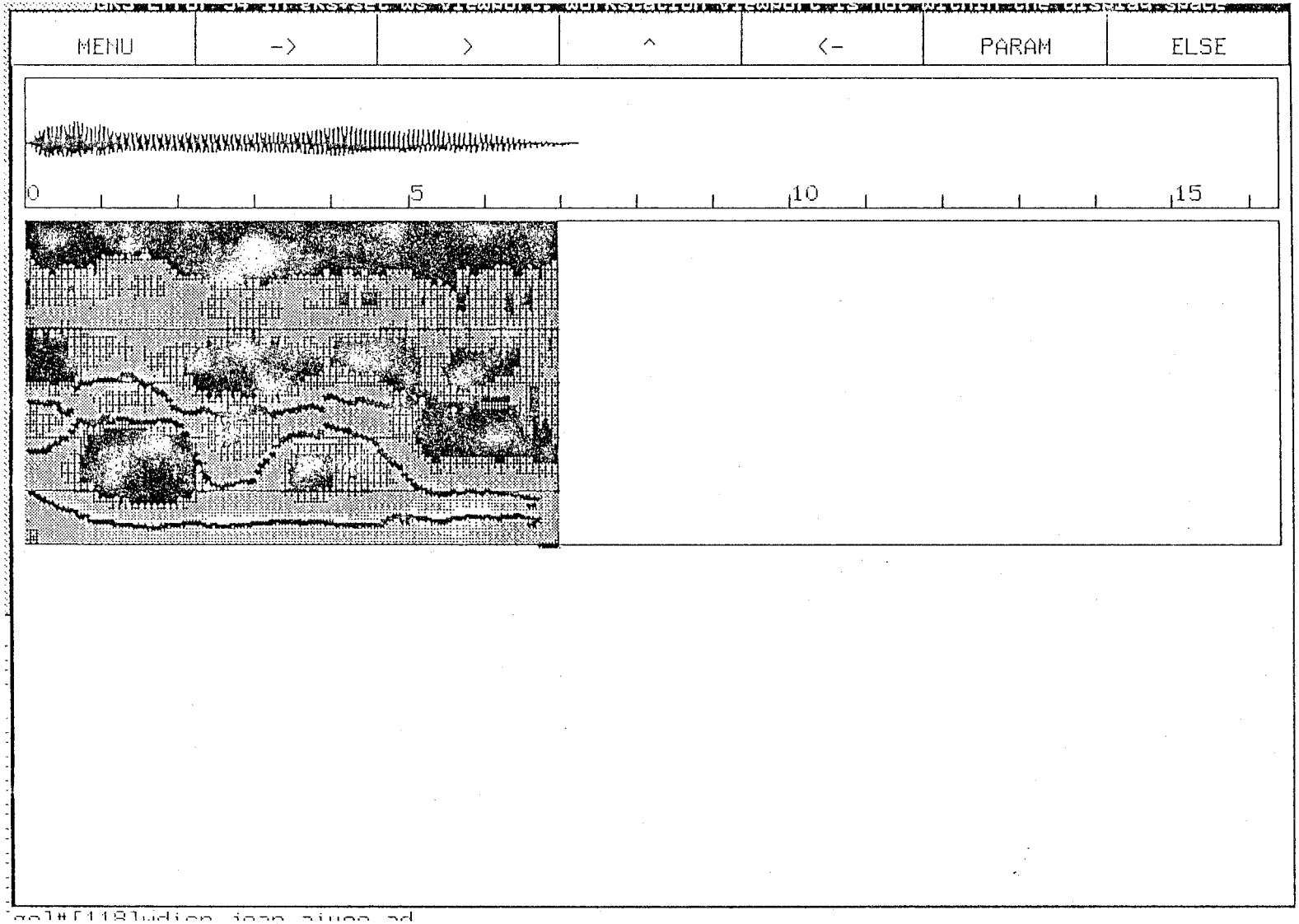
****
*   *
* ORIGINAL DATA *
*   *
****

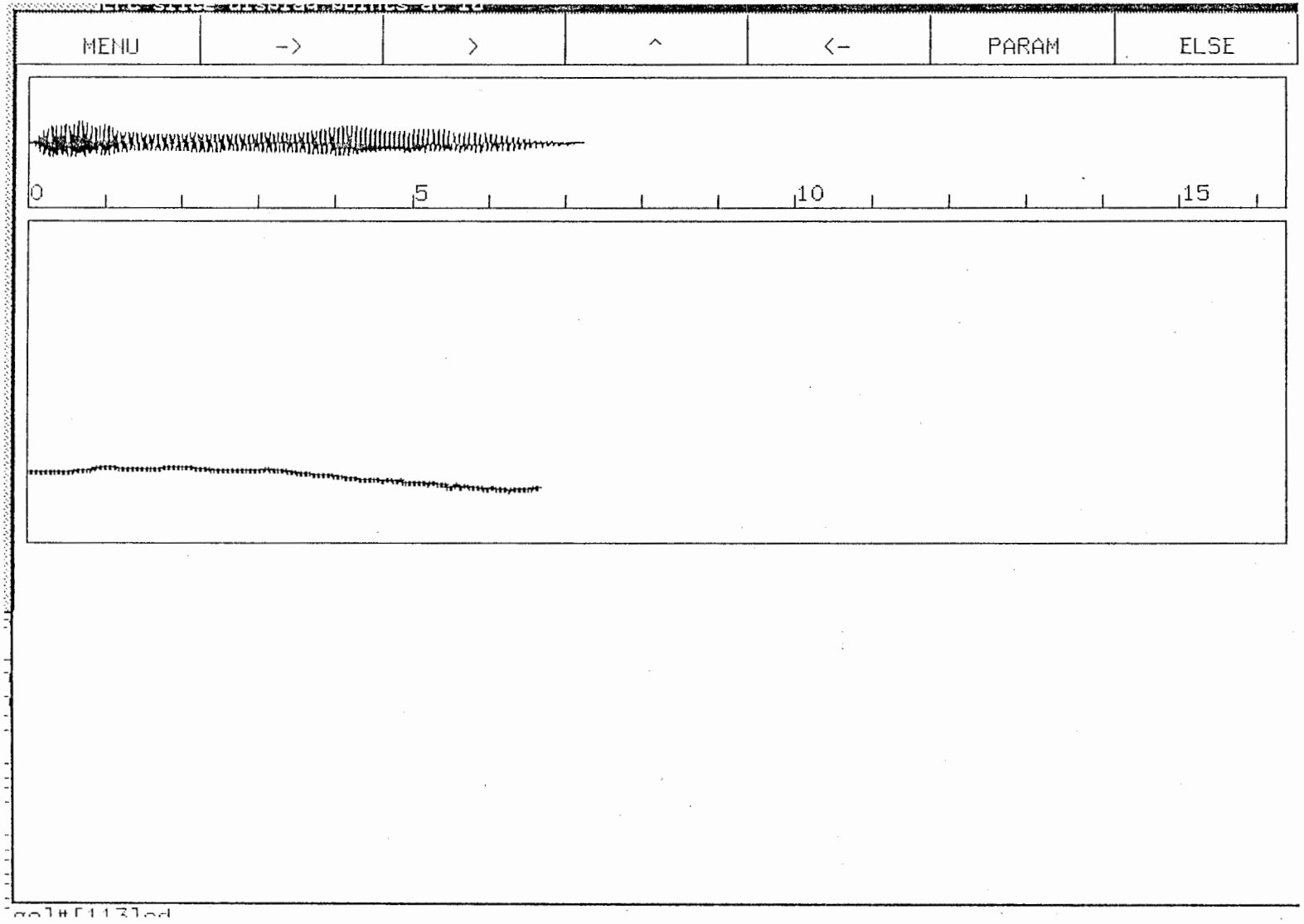

MENU
->
>
^
<-
PARAM
ELSE
0
5
0
5
10
15

MENU
->
>
^
<-
PARAM
ELSE
0
5
10
15


MENU
->
>
^
<-
PARAM
ELSE
0
5
10
15

MENU
->
>
<-
PARAM
ELSE


MENU
->
>
^
<-
PARAM
ELSE
0
5
10
15

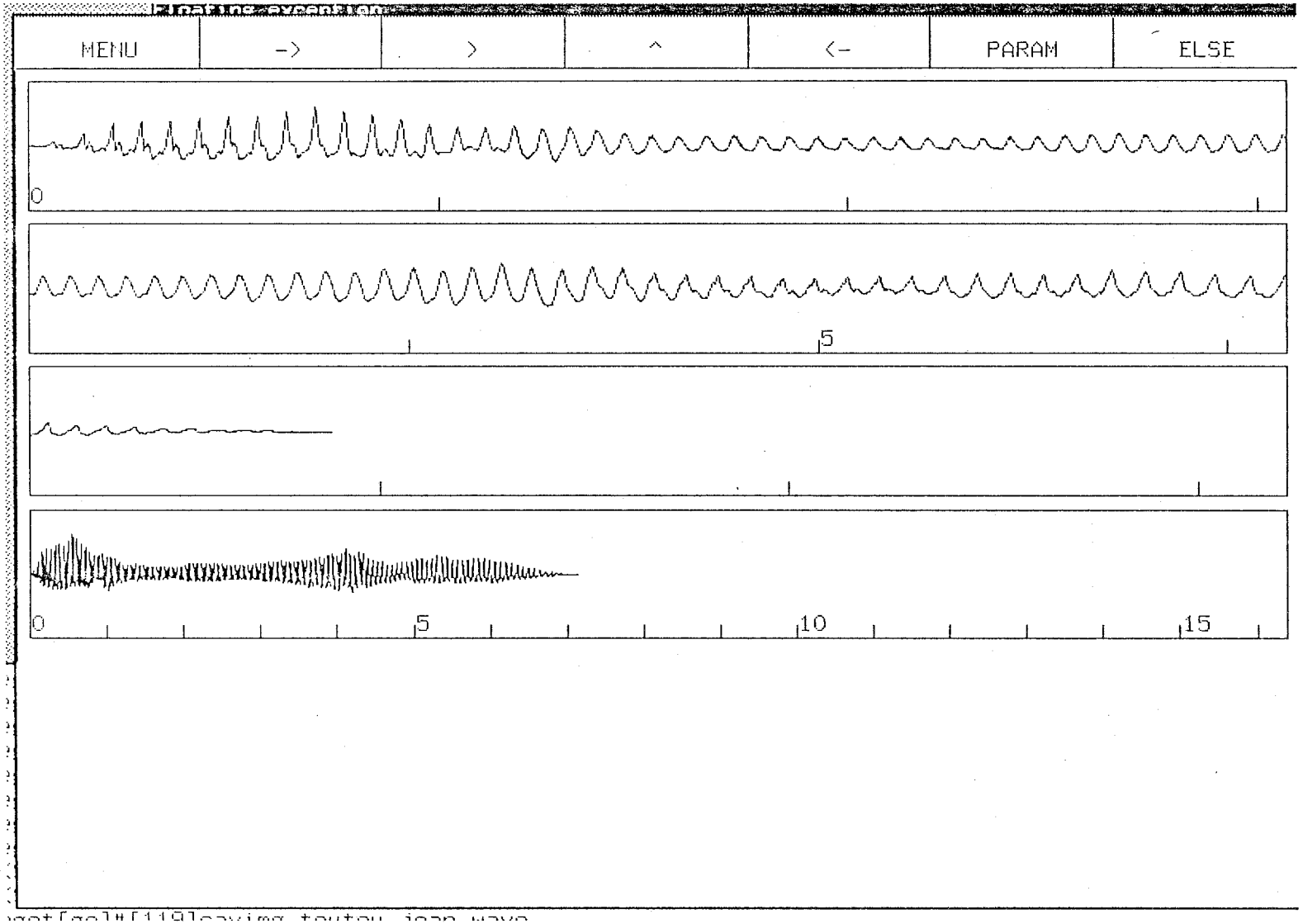

MENU
->
>
^
<-
PARAM
ELSE
0
5
10
15

****
*   *
* MODIFIED DATA *
*   *
****

Bandwidths multiplied by 5.0

Frequencies shifted down by 10%

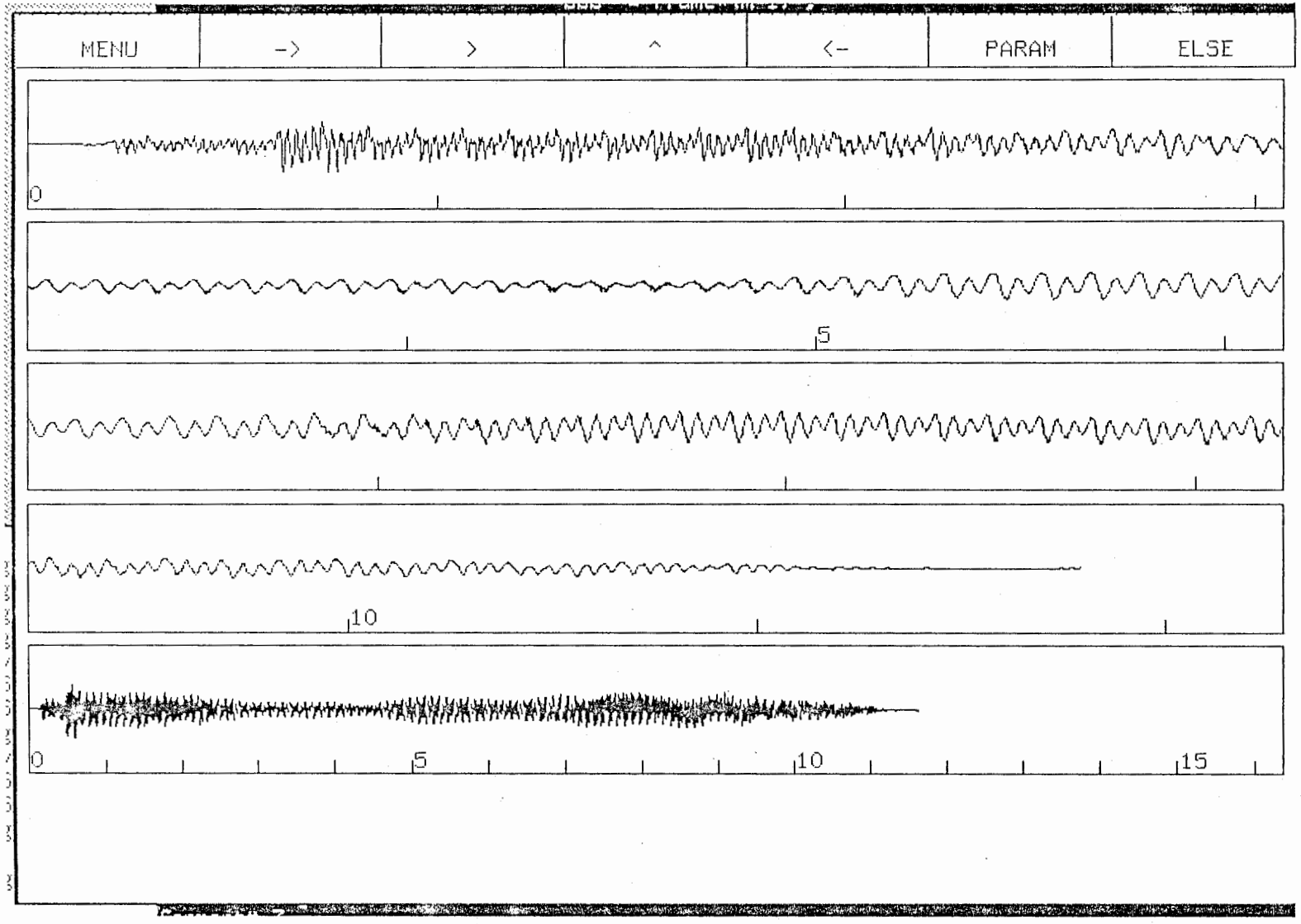
MENU
->
>
<
<-
PARAM
ELSE
antfont#1221save\ma tmutmu.ipan.fnrmants


MENU
->
>
^
<-
PARAM
ELSE
0
5
0
5
10
15

```
*****************************
*                           *
*       MODIFIED DATA       *
*                           *
*****************************
```

Bandwidths divised by 5.0

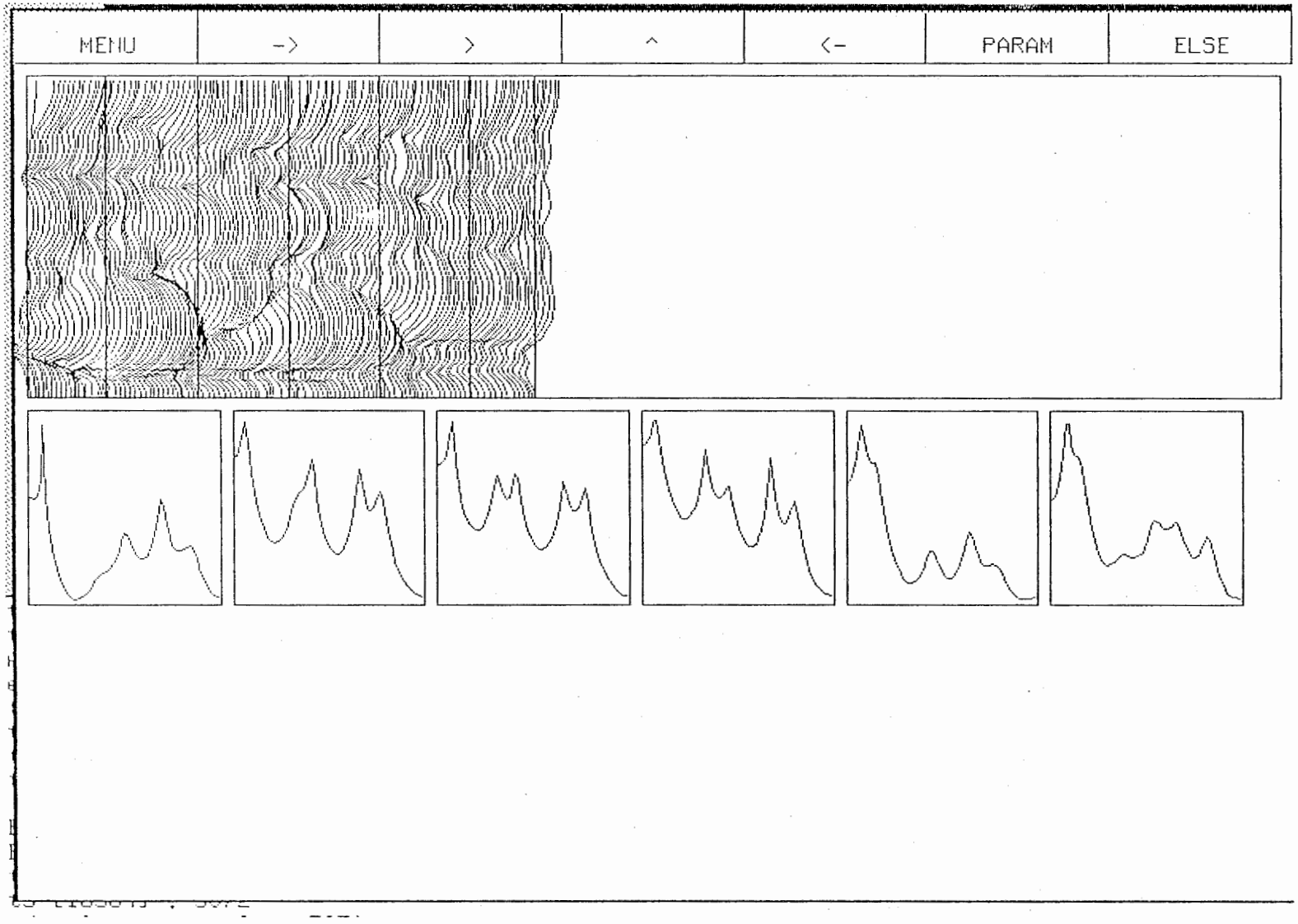
MENU
->
>
>
<-
PARAM
ELSE


MENU
->
>
^
<-
PARAM
ELSE
0
5
10
0
5
10
15

```
*****************************
*                           *
*       MODIFIED DATA       *
*                           *
*****************************
```

Pitch divised by 2 (in Hz).0

MENU
->
>
<
<-
PARAM
ELSE
0
5
10
15

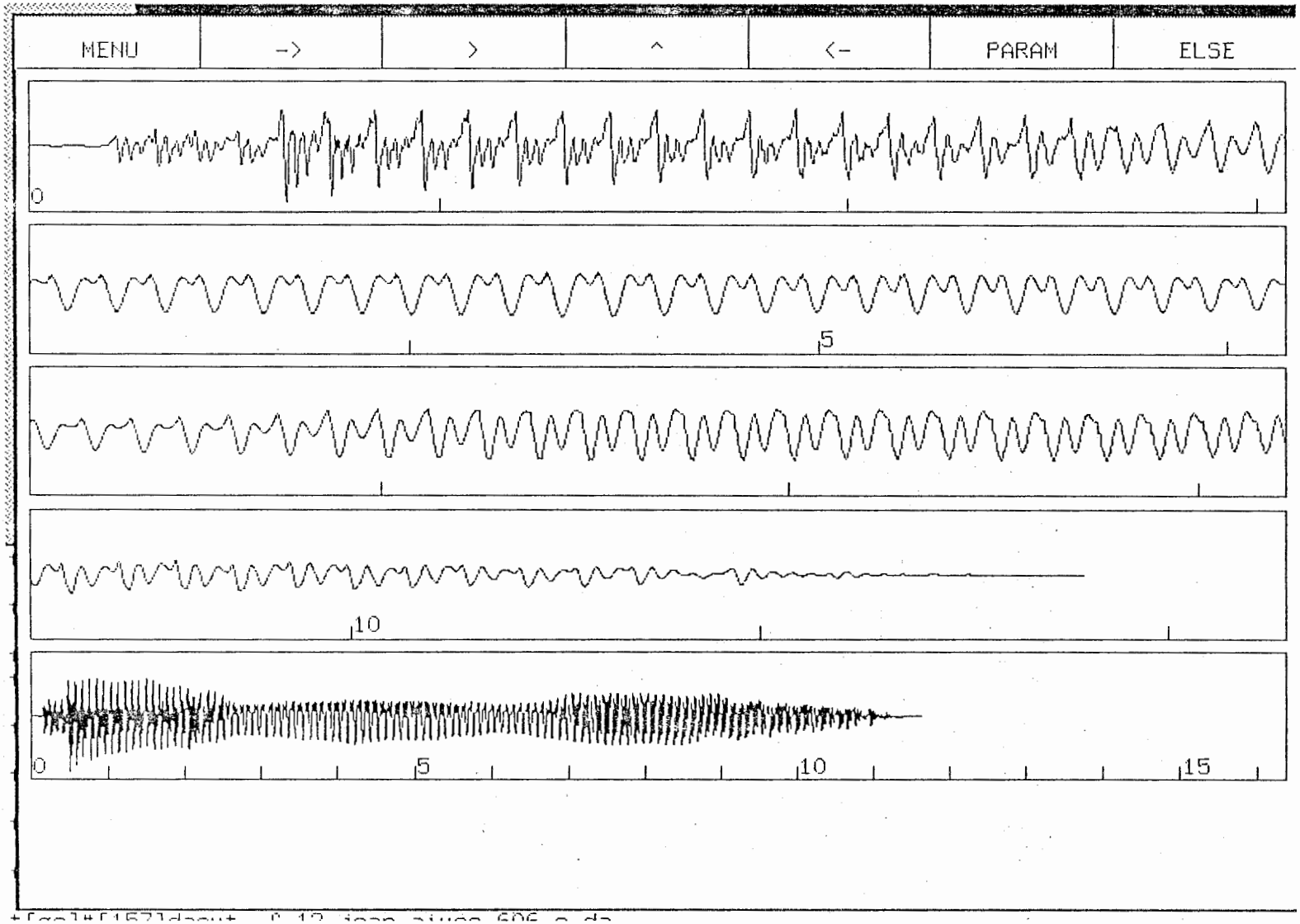

MENU
->
>
^
<-
PARAM
ELSE

MENU
->
>
<
<-
PARAM
ELSE
0
5
10
15

```
*****************************
*                           *
*       MODIFIED DATA       *
*                           *
*****************************
```

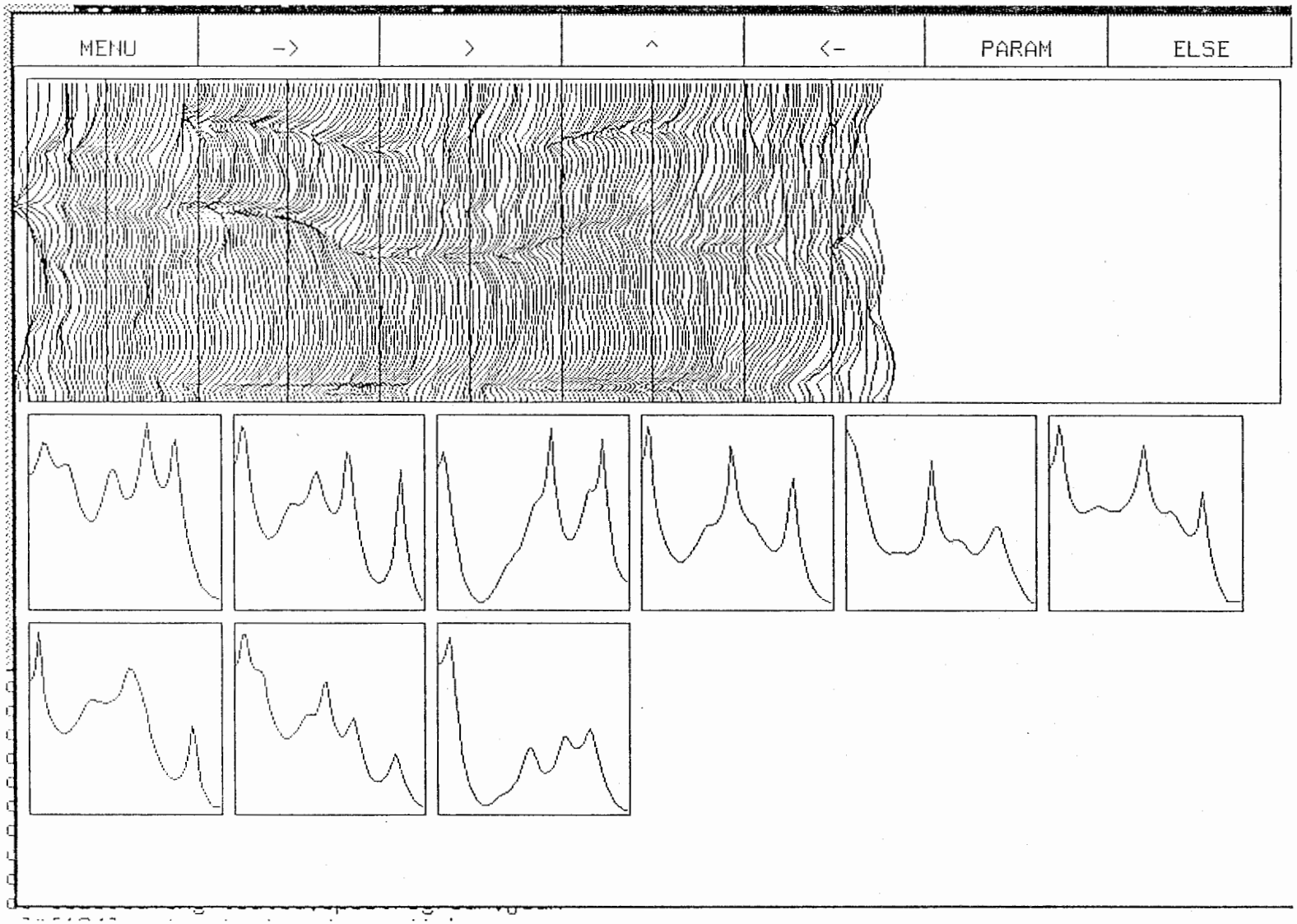
Frequencies shifted down by 10%

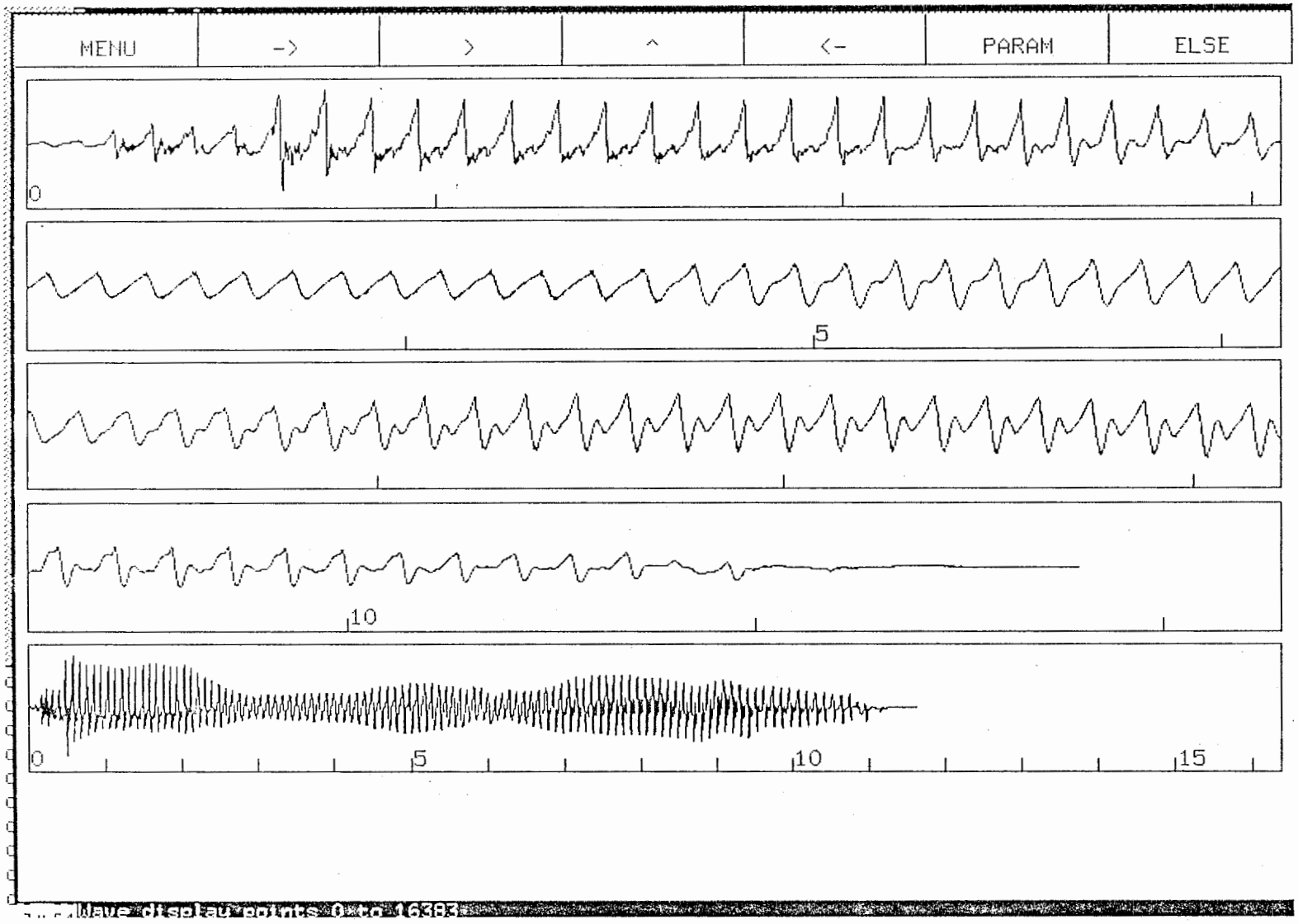
MENU
->
>
>
<-
PARAM
ELSE


MENU
->
>
^
<-
PARAM
ELSE
0
5
10
0
5
10
15

```
*****************************
*                           *
*        MODIFIED DATA      *
*                           *
*****************************
```

Bandwidths multiplied by 5.0


MENU
->
>
^
<-
PARAM
ELSE


MENU
->
>
^
<-
PARAM
ELSE
0
5
10
0
5
10
15

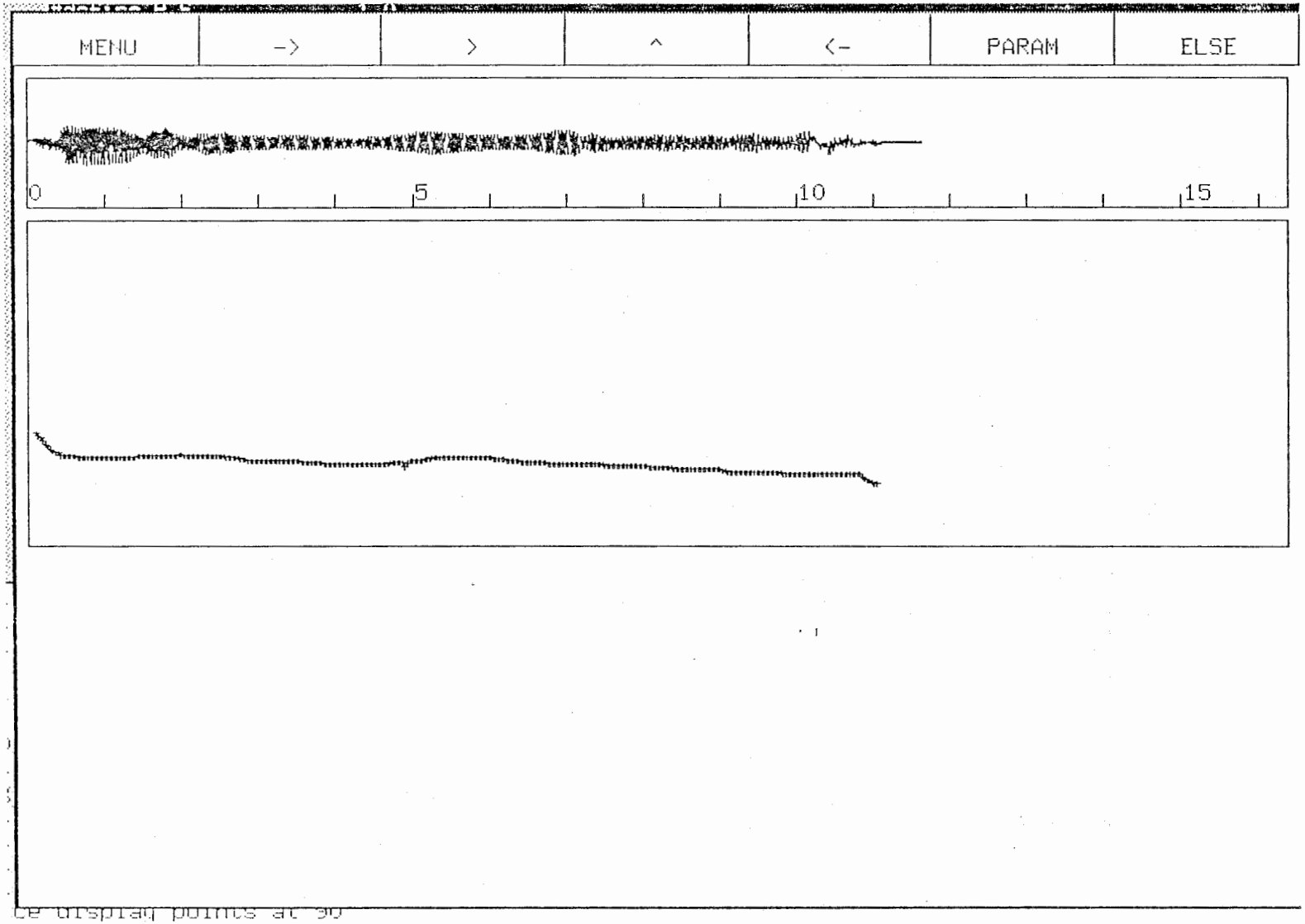
```
*****************************
*                           *
*       MODIFIED DATA       *
*                           *
*****************************
```

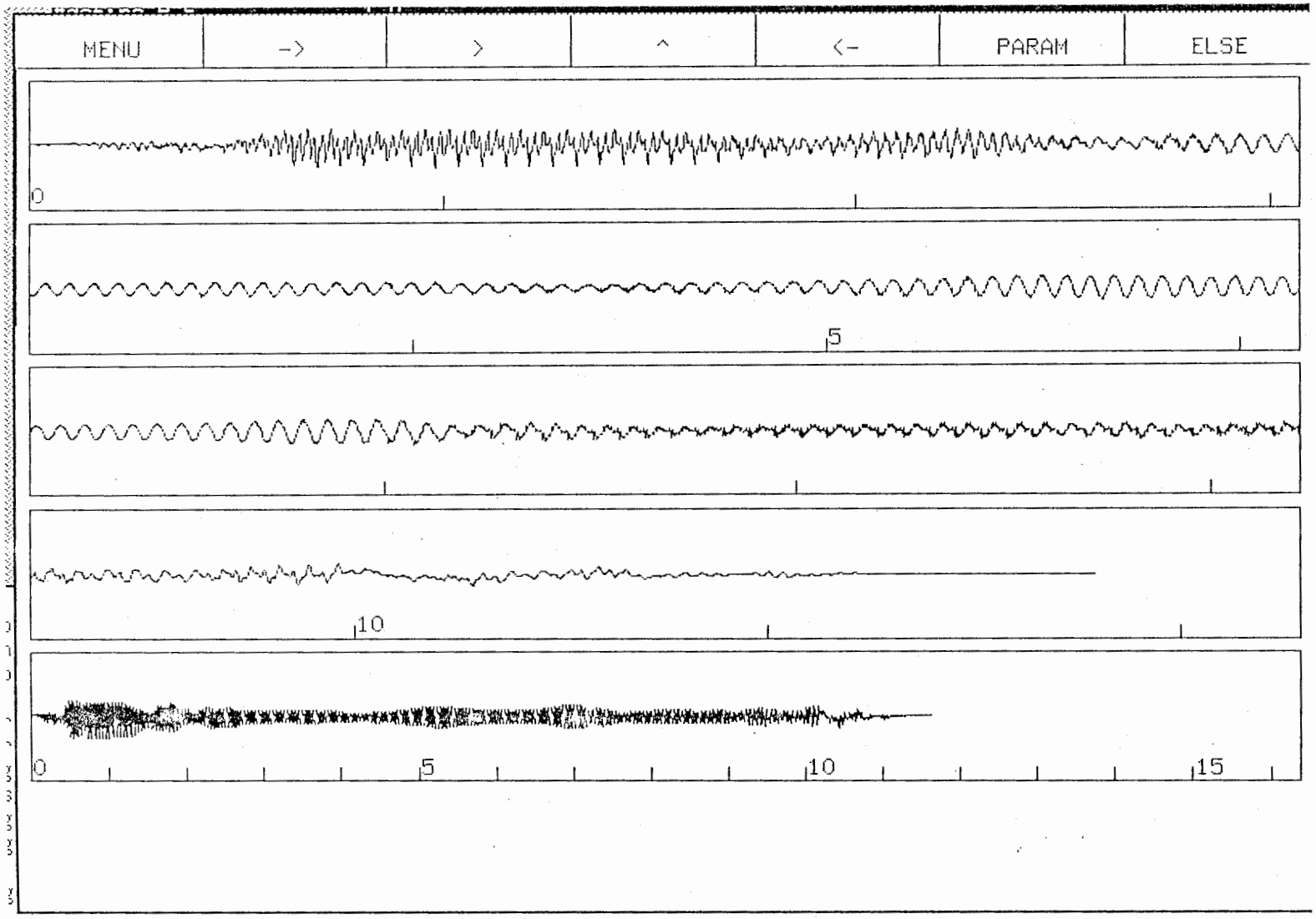
Bandwidths divised by 5.0

Frequencies shifted up by 10%

Pitch multiplied by 2.0 (in Hz)

MENU
->
>
<-
PARAM
ELSE


MENU
->
>
^
<-
PARAM
ELSE
0
5
10
15


MENU
->
>
^
<-
PARAM
ELSE
0
5
10
0
5
10
15

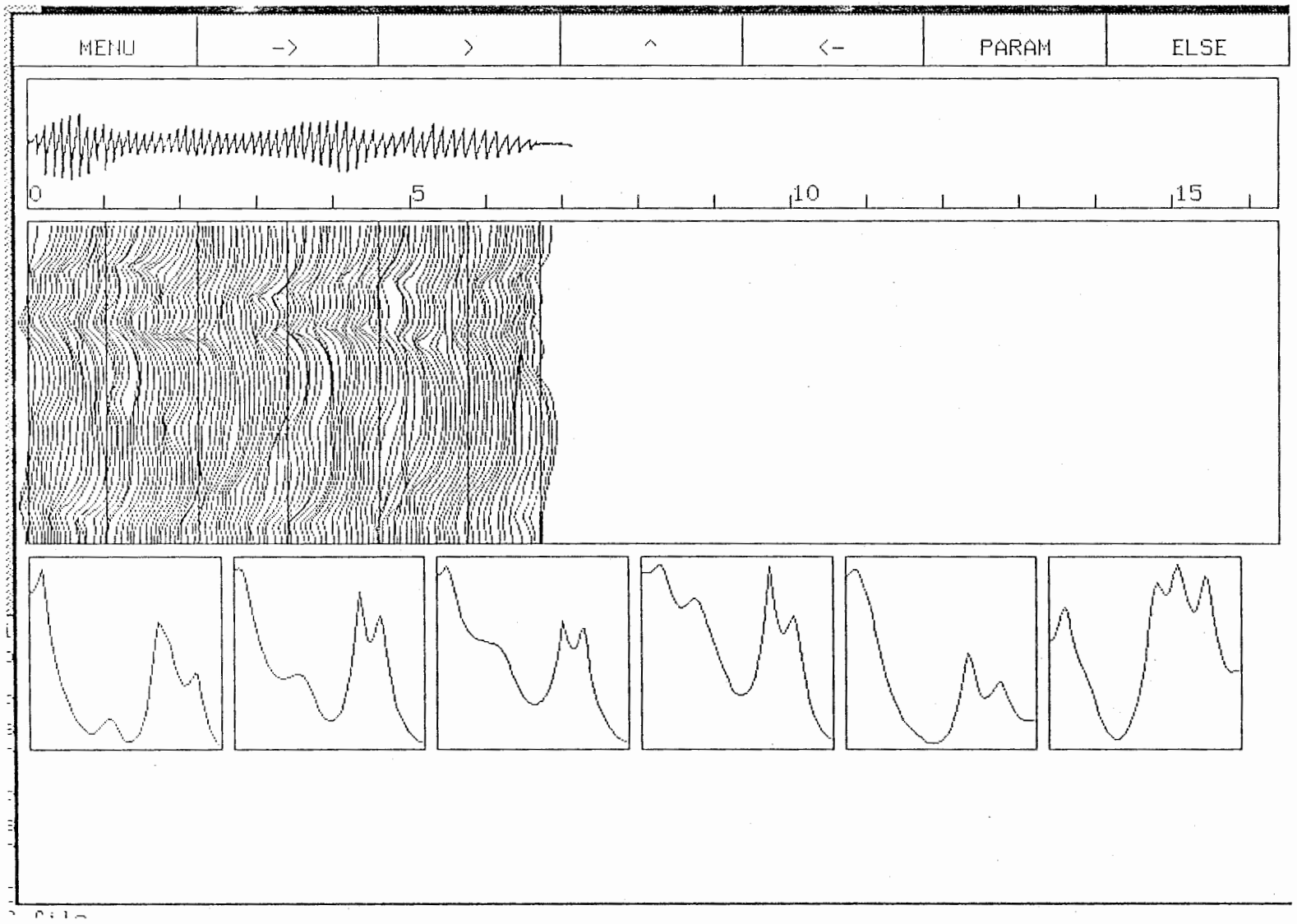
****
*   *
* MODIFIED DATA *
*   *
****

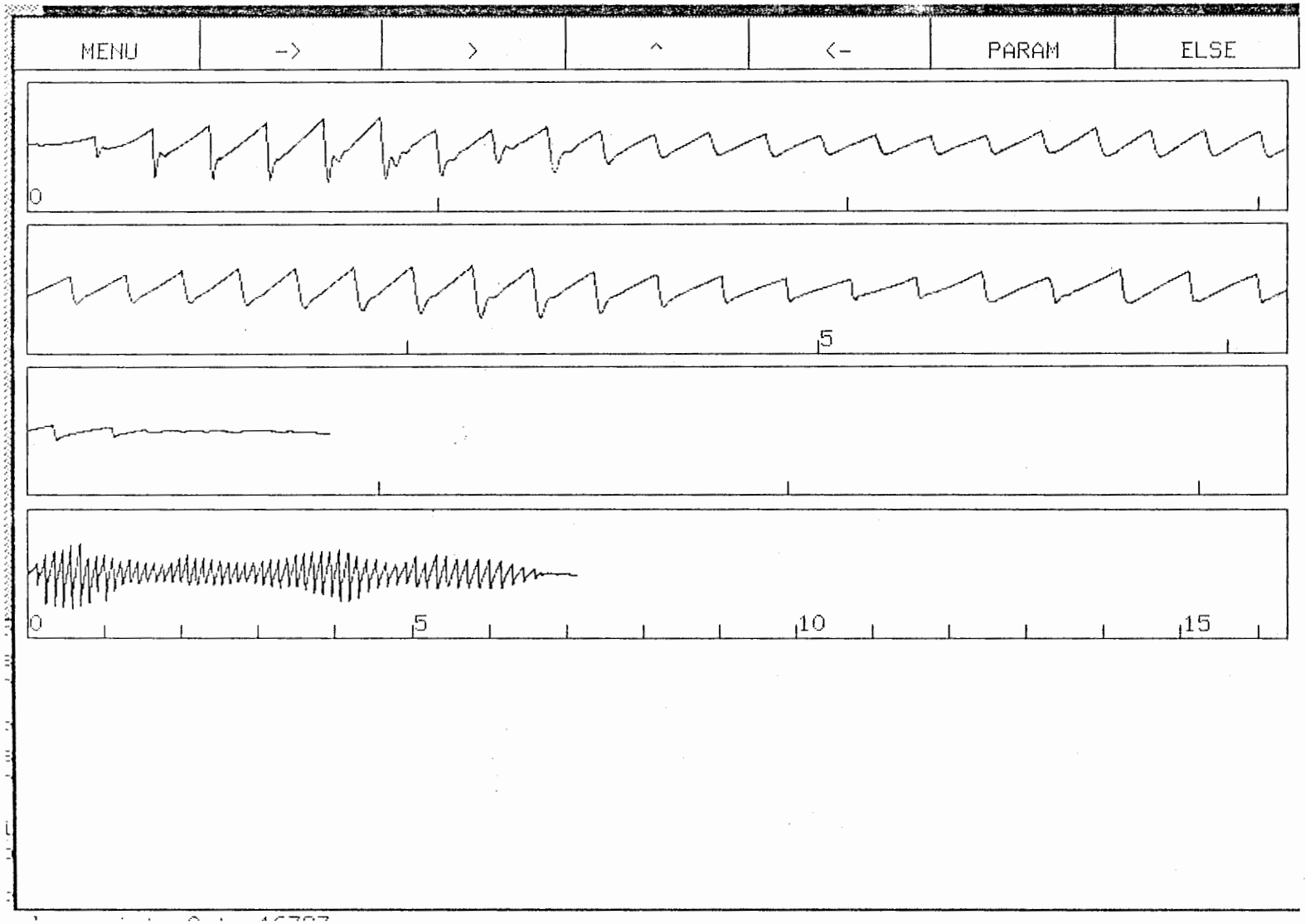
Bandwidths multiplied by 5.0

Frequencies shifted down by 10%

Pitch divised by 2.0 (in Hz)

MENU
->
>
^
<-
PARAM
ELSE
0
5
10
15

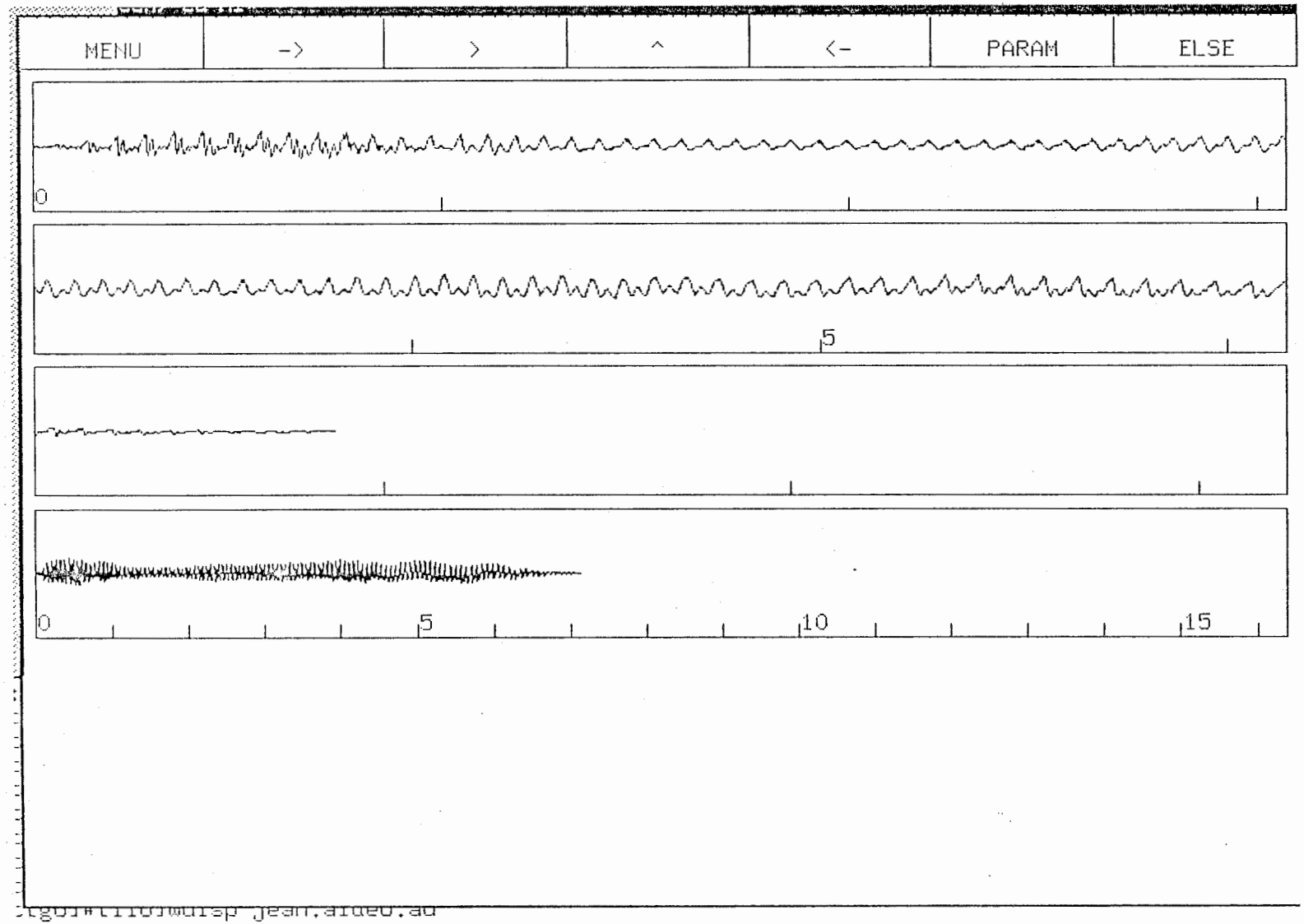

MENU
->
>
^
<-
PARAM
ELSE
0
5
10
15


MENU
->
>
^
<-
PARAM
ELSE
0
5
0
5
10
15

```
*****************************
*                           *
*       MODIFIED DATA       *
*                           *
*****************************
```

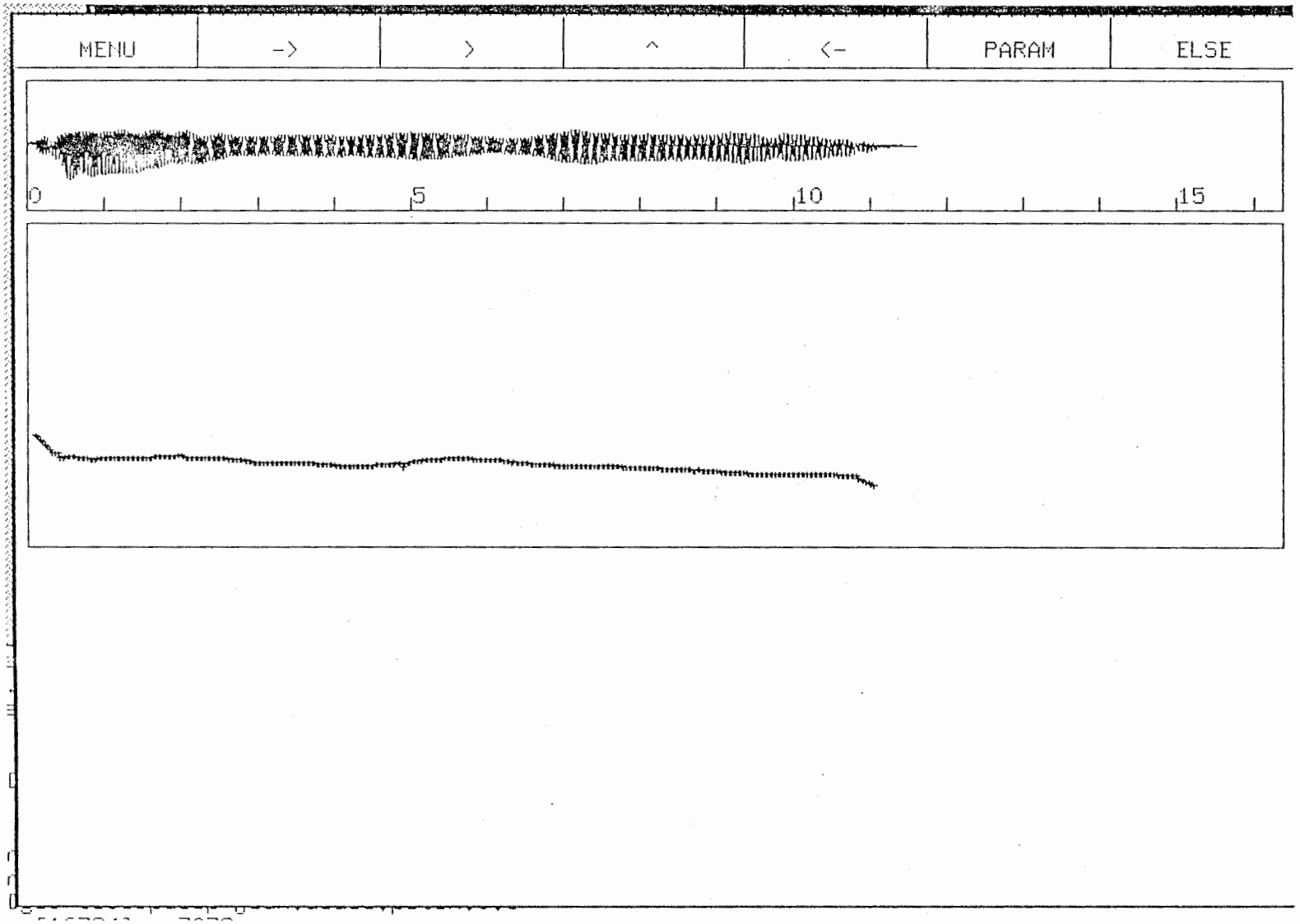
Frequencies shifted up by 10%

MENU
->
>
<
<-
PARAM
ELSE

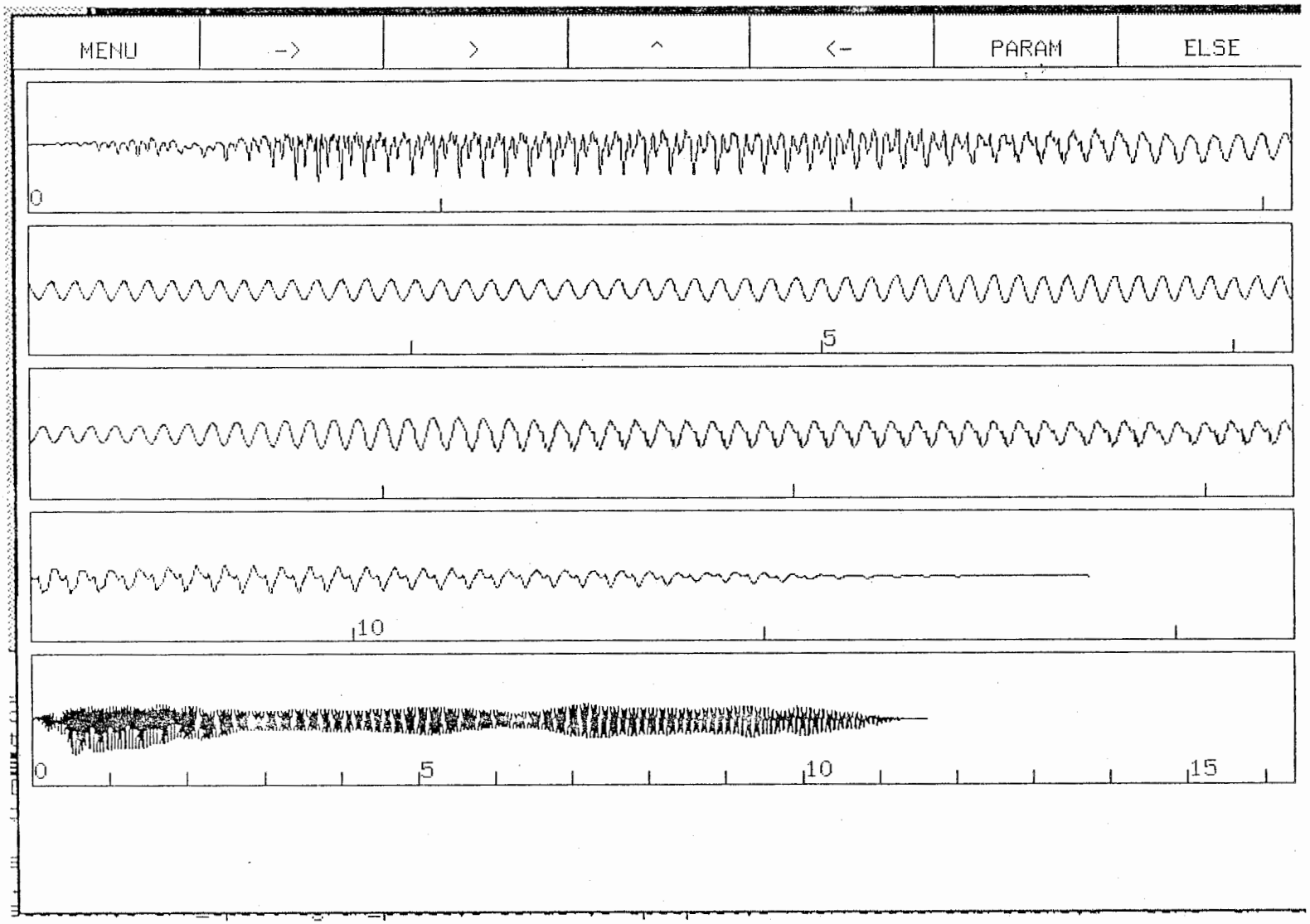

MENU
->
>
^
<-
PARAM
ELSE
0
5
0
5
10
15

```
*****************************
*                           *
*        MODIFIED DATA      *
*                           *
*****************************
```

Pitch multiplied by 2 (in Hz).0


MENU
->
>
^
<-
PARAM
ELSE
0
5
10
15

MENU
->
>
^
<-
PARAM
ELSE


MENU
->
>
^
<-
PARAM
ELSE
0
5
10
0
5
10
15

```
*****************************
*                           *
*       MODIFIED DATA       *
*                           *
*****************************
```

Bandwidths divised by 5.0

Frequencies shifted up by 10%

MENU
->
>
>
<-
PARAM
ELSE