Chant Manual

P-F. Baisnee
and the Chant group.
Ircam
29/4/85

TABLE OF CONTENTS

INTRODUCTION 1

CHAPTER I - INTRODUCING CHANT 9
1.1 SIMPLE LEARNING SESSIONS WITH CHANT (VAX VERSION) 10
1.1.1 Session 1: using default values 10
1.1.1.1 What you can do 10
1.1.1.2 How you can do it 10
1.1.1.3 How it works 11
1.1.2 Session 2: interactive modification of the parameters 12
1.1.2.1 What you can do 12
1.1.2.2 How you can do it 12
1.1.2.3 How it works 13
1.1.3 Session 3: using parameter files 14
1.1.3.1 What you can do 14
1.1.3.2 How you can do it 14
1.1.3.3 How it works 15
1.1.4 Session 4: functions and transitions 16
1.1.4.1 What and what for ? 16
1.1.4.2 Immediate functions and function files 16
1.1.4.3 Time scaling 17
1.1.4.4 Scaling and offset for the parameter values 18
1.1.4.5 Transitions 19
1.2 FOFs AND FILTERS: TECHNIQUES IN CHANT 21
1.2.1 The FOF synthesizer 21
1.2.1.1 The production model 21
1.2.1.2 The FOF synthesizer and its specificity 21
1.2.2 Filters in CHANT 25
1.2.3 Main differences between the VAX and the FPS versions 25
1.2.3.1 Filters 25
1.2.3.2 Noise in the FOFs 25
1.2.4 Parallel synthesis on the array processor 29
1.3 SOME AUTOMATIC CALCULATIONS AND RULES 31
1.3.1 Automatic calculation of the formants' bandwidths 31
1.3.2 Automatic calculation of the amplitudes of the formants 34
1.3.3 Automatic calculation of the complement 34
1.3.4 Adjusting the amplitudes of the formants 34
1.3.5 Adjustment of first and second formant frequencies (bending) 36
1.3.6 Vibrato on the fundamental frequency 36
1.3.7 Correction of the spectrum 40
1.4 BRIEF SUMMARY OF THE PARAMETERS ACCORDING TO THEIR CONCERN 41
1.4.1 Fundamental frequency 41
1.4.1.1 Random variations of the fundamental 41
1.4.1.2 vibrato (fundamental frequency vibrato) 41
1.4.1.3 random variations of the vibrato 42

1.4.2 spectrum 42
1.4.2.1 slope of the spectrum 42
1.4.2.2 automatic calculation of the spectrum 43
1.4.3 loudness of the sound 44
1.4.4 local formant envelope 44
1.4.5 control over the synthesis 44
1.4.6 Control over filters 45
1.4.6.1 In the FPS version: 45
1.4.6.2 In the VAX version 45
1.4.7 Controlling the array processor 45
1.5 RUNNING CHANT ON THE FPS ARRAY PROCESSOR 47

CHAPTER II - HOW TO BUILD YOUR OWN RULES IN CHANT: USER'S SUBROUTINES..... 49
2.1 CHANT ALGORITHMS AND USER'S SUBROUTINES 51
2.1.1 Short description of the basic CHANT algorithms and automatic calculations in relation to user's subroutines. 51
2.1.1.1 Evaluation of some of the internal CHANT variables: 51
2.1.1.2 Evaluation of the functions for parameters: 51
2.1.1.3 Vibrato: 51
2.1.1.4 Cslope: 52
2.1.1.5 Bending: 52
2.1.1.6 Automatic corrections of formant amplitudes and bandwidths; Automatic calculation of the complementary formant: 52
2.1.1.7 Correction: 52
2.1.2 Position in time of CHANT algorithms in relation to the three users 53
2.2 CHANT INTERNAL AND EXTERNAL VARIABLES..... 54
2.2.1 Variables corresponding to the usual CHANT parameters 54
2.2.2 Internal variables 55
2.3 IMPLEMENTING USER'S SUBROUTINES..... 59
2.3.1 FORTRAN 59
2.3.1.1 User model 59
2.3.1.2 Variables 61
2.3.1.3 Coding 62
2.3.1.4 Compiling 63
2.3.2 C language 63
2.4 RUNNING YOUR USERS ON THE VAX..... 64
2.5 RUNNING YOUR USERS ON THE FPS ARRAY PROCESSOR 65
2.6 SOME SIMPLE ALGORITHMS AND USER'S SUBROUTINES..... 66
2.6.1 A user for generating clicktracks 66
2.6.2 Scalers and random variations of CHANT parameters 68
2.6.3 Iteration and general scalers: scaling filters' amplitudes with bramp 69
2.6.4 Correlation between the filters' amplitude and coefamp 69
2.6.5 Low frequency modulation on coefamp 70
2.6.6 Attenuation algorithm 72
2.7 INTERNAL SUBROUTINE ENV.F 75
2.8 USING PHONEME DICTIONARIES: USER MODEL PHON.F..... 77

CHAPTER III - THE USER'S LIBRARY 78
3.1 HOW TO USE THE LIBRARY 79
3.2 USER'S SUBROUTINE MODEL: USER.F 80
3.3 USER MODEL FOR USING PHONEME DICTIONARIES: PHON.F 81
3.4 USER CYM.F 84
3.5 USER ADDIT.F 87
3.6 OTHER USERS 89
3.6.1 user qon.f 89
3.6.2 user cda.f 89

CONCLUSION: CHANT AND FORMES - CHANT AND THE 4X 90

SELECTED BIBLIOGRAPHY 91

appendix 1: CHANT PARAMETERS LISTED IN ALPHABETIC ORDER 92

appendix 2: DEFAULT VALUES OF STANDARD CHANT PARAMETERS (FILE SCR.PAR) 104

Appendix 3: PHON.F, A BASIC USER FOR USING PHONEME DICTIONARIES 107

appendix 4: FREQUENCY TABLES 114

appendix 5: PHONEMES 116

LISTE DES FIGURES

Figure 1. Local envelope 5

Figure 2. Repeated excitation with a global envelope. 6

Figure 3. Formant: spectrum envelope. 7

Figure 4. Envelope of a spectrum with several formants. 8

Figure 5. Structure of a FOF synthesizer 22

Figure 6. Log magnitude FOF spectra 24

Figure 7. Effect of the noise algorithm on a FOF signal: -1- normal FOF -2- noisy FOF. 27

Figure 8. Effect of the noise algorithm on a spectrum: -1- one formant without noise -2- same formant with noise. 29

Figure 9. FOF and Filter synthesis on the FPS array processor 30

Figure 10. Automatic bandwidth as a function of formant center frequency 33

Figure 11. Vibrato: the effect of vibamp and vibfreq 38

Figure 12. Vibrato: random variations of the vibrato amplitude 39

Figure 13. Amplitude scaler for attenuation, as a function of formant frequency 73

Figure 14. Frequency table 114

Figure 15. Frequencies and their musical notation- frequency ranges of some instruments 115

Figure 16. Formant frequencies, amplitudes and bandwidths for different phonemes 116

Figure 17. Formant frequencies in relation to fundamental frequency.(from: Johan Sundberg, Synthesis of Singing, Swedish Journal of Musicology, 1978, 60.) 117

INTRODUCTION

This is an introduction to the CHANT program, developed by Xavier Rodet and Yves Potard. It is a program originally devoted to speech and song synthesis. It is able to synthesize human singing voices, and in its present form can produce single or repeated notes or continuous phrases on vowel sounds. It is also able to produce a variety of non-vocal sounds with resonant spectra, as well as transformations between vocal and non-vocal sounds. At Ircam, CHANT has been running on the VAX 780. A much faster version, implemented on the FPS array processor by Yves Potard and Jan Vandenheede, is also available.
The CHANT synthesizer can be driven in two different ways at present; first, parameters can be supplied in parameter files, and additional rules can be implemented in Fortran or C programming languages; second, one can use the FORMES object oriented language to control the synthesizer. This manual is concerned with the first possibility.

It is important to know that a great attention has been given to the ease of use even for the beginner,to the point where **YOU CAN SYNTHESIZE A SUNG VOWEL ONLY BY TYPING A SERIES OF "CARRIAGE-RETURN'S"** after having started the program! Try it, it costs nothing, and that's the easiest synthesis of your life! Then you can begin modifying the parameters one by one for a certain effect... or in order to feel what they really do!

The development of the program has been continuously aided by users and composers. Generally the adopted solutions are the result of some tenebrous compromise between different necessities. It is hoped that as many users as possible be confronted with "CHANT" and that they propose improvements.

In the physical world, a great variety of objects can vibrate after they have been excited, producing more or less pleasant sounds. Except for certain rare circumstances (near absolute 0), this vibration is damped sooner or later.
Let us consider some examples:
After being hit, a bell goes on resonating for a long while, but a wood block sound is attenuated very fast. The same is true if you hit a violin string, it gives a short sound. That is probably why some people like to use a bow: it allows them to give many excitations in a very short time.
The sound so produced is characterized by:

- the resonant properties of the object (e.g. violin in air)
- the rate of repetition of the excitation, and thus of the resonance, which causes the produced sound to be periodic at that rate, what we call, *in that case*, the fundamental frequency or pitch.

To give another example, think of an intrument in which the air plays a greater role in the resonant object: trumpet, reed instrument, vocal tract, etc... You know that if you tap your cheek with your finger, you hear the sound of the resonant mouth cavity (also called vocal tract), and that the sound changes if you change the shape of the mouth's resonant cavity, by moving the tongue for example. As the use of a finger for exciting one's vocal tract is not so efficient, again, most people like to use their glottis to provide repeated

excitations to their vocal tract, and according to their success they call the result speech or singing or some other names... Again the sound so produced is caracterized by the resonant properties of the object (mouth shape) and the rate of repetition of the excitation (glottis). It is periodic (and generally harmonic). *In that case*, the fundamental frequency of the sound is the rate of repetition and gives the pitch of the tone. For the second time, "in that case" is underlined.The other "case" was the one of the bell cited at the beginning: there is only one excitation and obviously the pitch, if there is a pitch, comes from elsewhere. First let us say what a resonance is: it is a perodic movement or a sum of them. The simplest movement is a pure sinusoid (with damping of course). Some, like the bell vibration, are a summation of damped sinusoids called partials. In this case there is no necessity that the sinusoids be harmonically related (which means that the total sound is not necessarily periodic), and the pitch can come from a complex combination of the frequencies of the different inharmonic partials. All these heavy explanations are to help you remember our definition of the "**FUNDAMENTAL FREQUENCY**" in the CHANT program: it is "**THE RATE OF REPETITION OF THE EXCITATION**" and its inverse is the fundamental period (see Fig. 2).

For a usual vocal sound (or wind instrument for example), there is no mystery: the fundamental frequency gives the pitch (440 Hz. for A4). But if you want to produce the sound of a bell or tam-tam struck once every ten seconds, it means that the excitation shall happen every ten seconds, and thus that the parameter, in the program, giving the fundamental frequency shall have the value 1/10 =.1 Hertz.

The total resonance of the object is described by:

- The characteristics of each of the partial resonant components
- The rapidity of the excitation of each of them

In terms of spectrum, a resonance is a "peak" of the amplitude spectrum of the sound, called "**FORMANT**" (see Fig. 3 and 4). Its characteristics are:

— central FREQUENCY
— maximum AMPLITUDE (related to loudness)
— "sharpness" of the summit, called BANDWIDTH and measured at -6 dB from the top
— extension of the low-amplitude parts of its spectrum on each side of the central frequency,also called width of the skirts, and referenced as "EXCITATION TIME" ("TEMPS D'EXCITATION" in French), for a reason to be explained below.

In terms of signal, a resonance (formant) is a damped sinusoid, with a more or less short ascending portion (excitation portion). We call **LOCAL ENVELOPE** the envelope defined by this ascending portion,the damped portion and a final attenuation that will be explained with the parameter **ATTEN** (see Fig. 1). It is LOCAL because for usual vocal or instrumental sounds its duration is of the order of the fundamental period (10 milliseconds for 100Hz) and very much smaller than the overall (global) envelope of the sound (see Fig. 2). The resulting signal so defined is called **FOF.**[1] Its characteristics are:

1. "FOF" stands for "Fonction d'Onde Formantique", that is "formant wave function".

- the frequency of the sinusoid which is the central frequency of the formant (resonance)
- the amplitude of the damped sinusoid which is the amplitude of the formant.
- the speed of the damping which gives the bandwidth of the formant : Note that a slow damping means a small bandwidth (sharp peak), a fast damping means a large bandwidth (broad peak).
- the duration of the ascending portion of the envelope of the damped sinusoid which yields the width of the skirts of the formant. Note that a fast attack means large skirts, while a long attack means reduced width of skirts. As this notion is not very familiar, we give here the names of the corresponding parameters in the program: **tex1, tex2, tex3,...** etc... "Tex", which stands for "temps d'excitation" (duration of excitation), is followed by an integer refering to the formant's number.

Input to the program is specified by collections of parameters which often have clear physical correlates: formant center frequency, vibrato frequency, vocal effort, amount of random variation of the fundamental frequency, etc... These parameters can have **FIXED VALUES** for the duration of the sound, or can be controlled by **FUNCTIONS** (of time) allowing for changing values.

The output of the program is a soundfile which contains a "phrase" consisting of one or more notes or events. In addition, the program outputs a file containing the values of nearly all parameters used to specify the sound, which allows one to resynthesize the sound easily if necessary. This file is called **scr.par** and is written in your current directory.

The method of computation for these sounds is also related to physical correlates. For each resonance area in the sound there is a sine wave whose frequency is the center frequency of the resonance area and whose amplitude is related to the relative amplitude of the resonance. This sine wave is damped according to an envelope whose duration is in the order of one or two periods of the fundamental frequency (in the order of 2 milliseconds for a fundamental of 500 Hz.), and the envelope is triggered at the beginning of each fundamental period. For a sound with five formants (resonance areas) there will be five sine waves whose envelopes are triggered simultaneously at the beginning of each fundamental period. The resulting sound has a harmonic series with a fundamental at the frequency of triggering of the sine wave envelopes, and with the relative amplitudes of the harmonics controlled by the frequencies, amplitudes, and envelopes of the sine waves. The actual form of the envelope for each sine wave controls the bandwidth of the formant area.

Advantages of this system are:

1. Clear correlation between parameters and physical characteristics of the sound;

2. In most cases, information used for the computation of the sound need only be updated once every pitch period or **quantum**. The CHANT parameter for controlling the quantum is the frequency (and not the period!) for updating the parameters; in the VAX version of CHANT, it is automatically set to the fundamental frequency, but on the FPS array processor, it can be set by the

user.

In addition to this so-called **FOF synthesis technique,** CHANT now enables the use of **FILTERS.** These two techniques will be explained a little further. For a more extensive description of FOF and FILTER techniques and their application to the modeling of singing voices and instruments, it is strongly suggested that the new user of the program CHANT address him or herself to the articles mentioned in the bibliography.

NOTE: The command **man chant** should display some information about the CHANT program. In particular, it should answer the following questions:
- Where are the sources of the CHANT program and the CHANT library located on the system;
- Where is the present manual kept (for corrections).

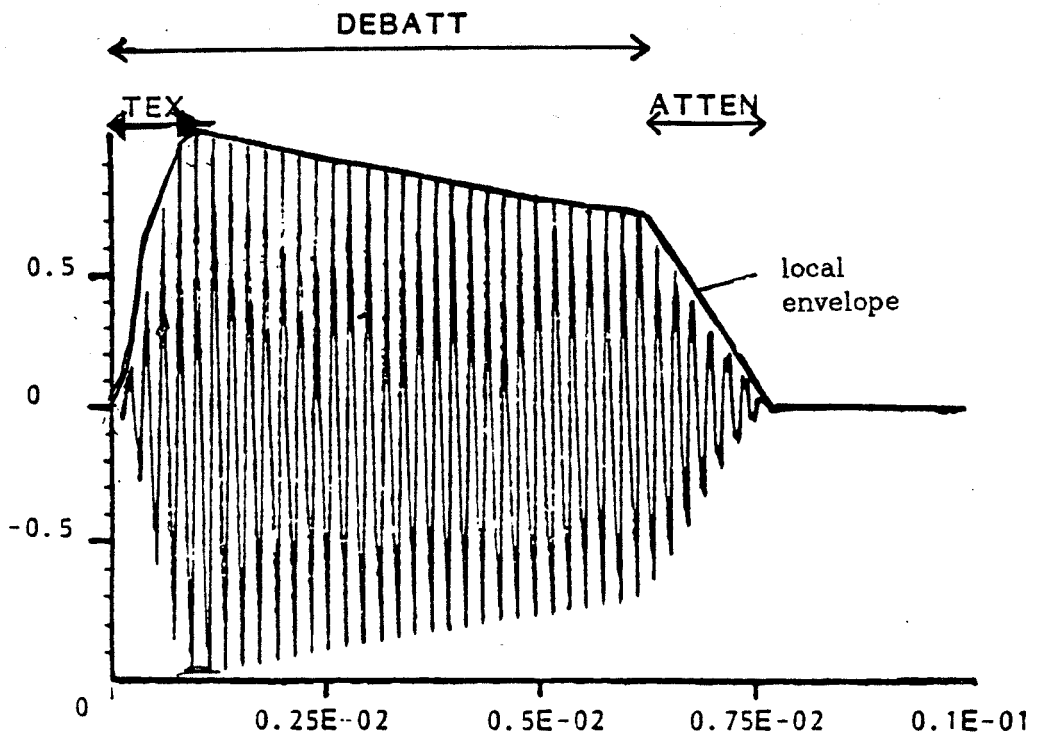


Figure 1. Local envelope

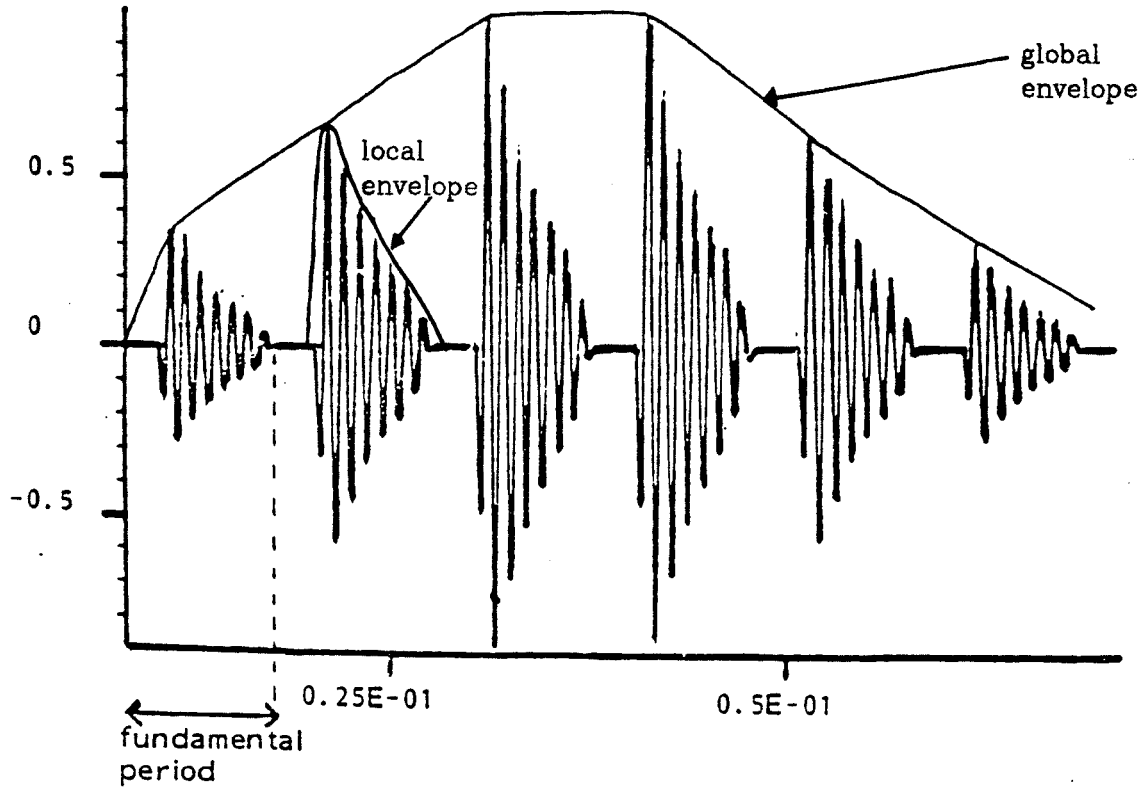


Figure 2. Repeated excitation with a global envelope.

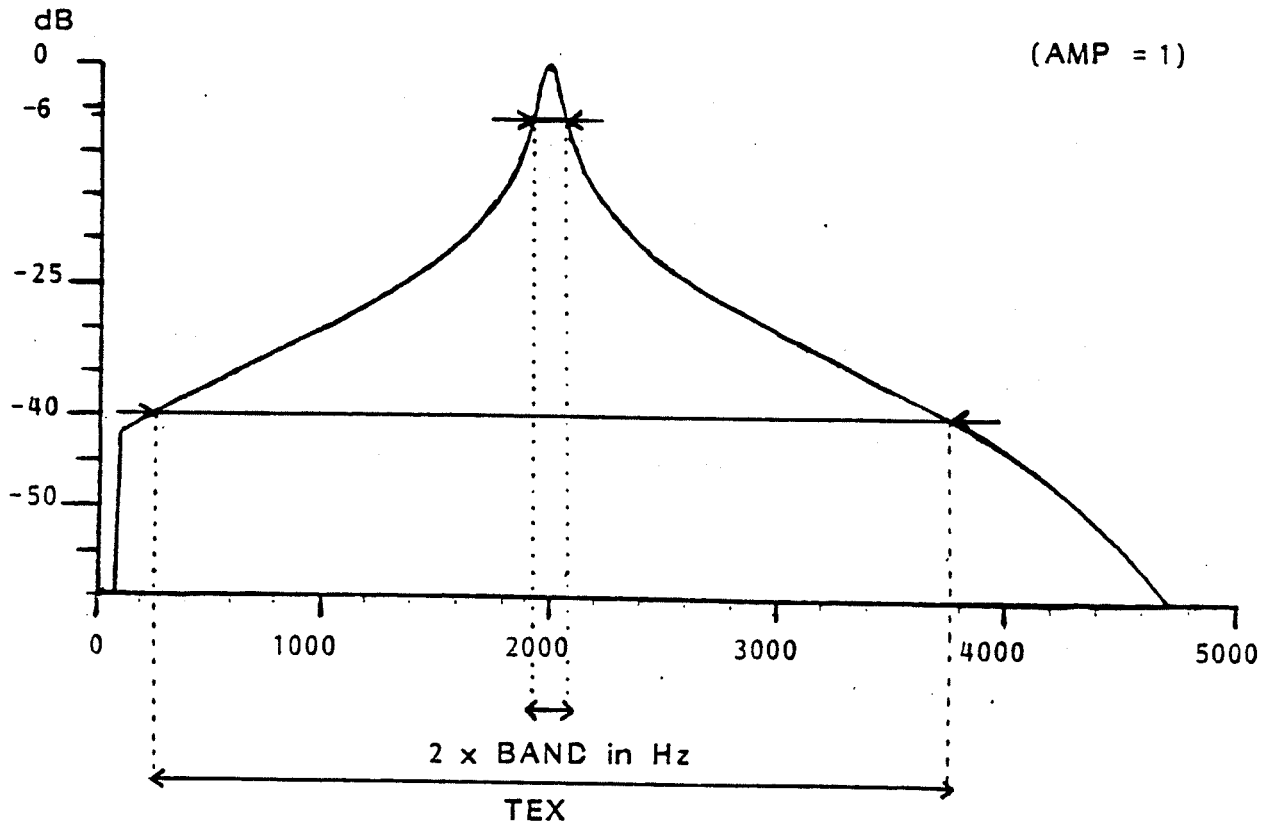


Figure 3. Formant: spectrum envelope.

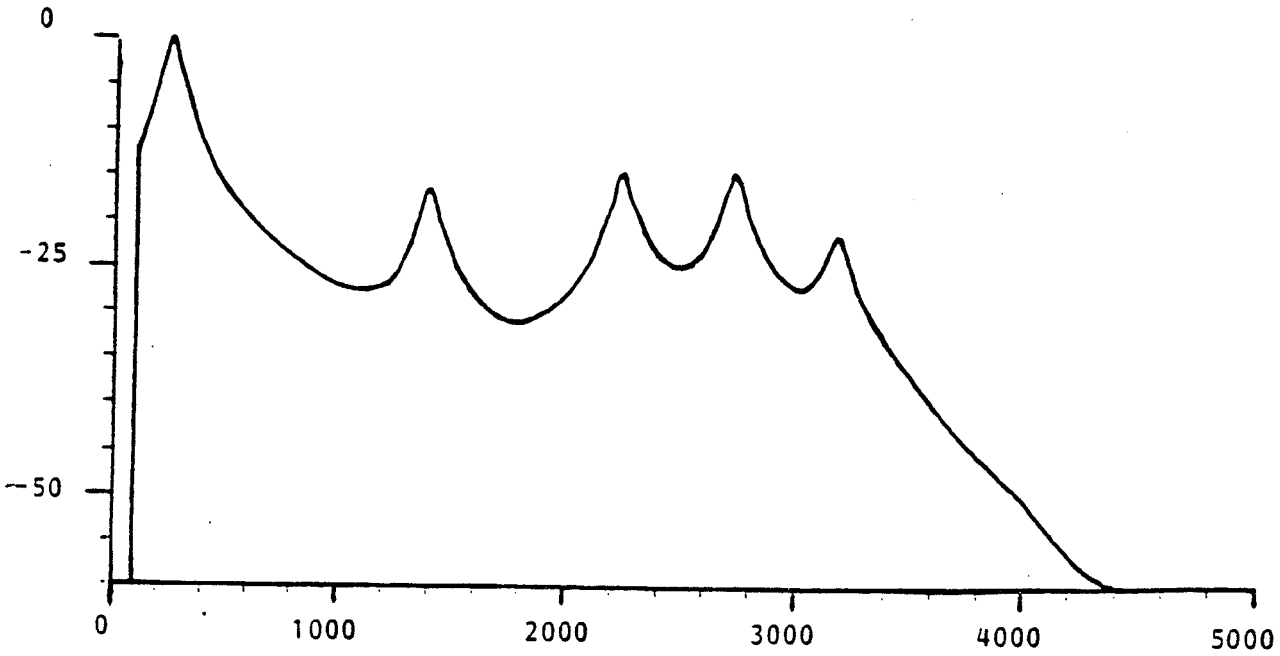


Figure 4. Envelope of a spectrum with several formants.

CHAPTER I - INTRODUCING CHANT

Before explaining CHANT synthesis techniques, parameters and rules, we first of all give a guideline for some simple CHANT sessions. This chapter should illustrate two important features, that make the program easy to use:

- every parameter has a default value, which allows the user to specify only those parameters he wishes to play with.
- almost every parameter has a perceptual relevance, which, with a little training, enables the user to predict the result of a particular synthesis.

IMPORTANT NOTE:
The VAX and the FPS versions of CHANT differ:
-1- The FPS version, which is faster, offers more possibilities than the VAX version.
-2- Some CHANT parameters have different or additional meanings in the FPS version.

These differences will be described further. The first sessions presented here are intended for the VAX version; but the FPS version is strongly recommended for further experiments with CHANT.

1.1 SIMPLE LEARNING SESSIONS WITH CHANT (VAX VERSION)

1.1.1 Session 1: using default values

1.1.1.1 *What you can do*

By running CHANT and then typing some "carriage returns", you can use the default values of every parameter to synthesize a default CHANT sound. Default values define a sung vowel "A", 1.3 seconds long, with a fundamental frequency of 100 Hz.

1.1.1.2 *How you can do it*

Under shell, you just have to type :

chant

Here is the complete session, as it is displayed on the screen. Words between "[" and "]" indicate what you are to type. For instance, [RETURN] indicates that you have to strike the "carriage return" key on your terminal.

```
$ [chant] [RETURN]

        CHANT-Vax I.R.C.A.M. Version 3.d
     synthese par regles de X.Rodet

     parameters file     :          [RETURN]
     parameter           :          [RETURN]

     computing           :    press <return>
     notions             :    press <?> <return>
     other file          :    press <f> <return>
     parameter           :          [RETURN]

     time set            : 1.300 seconds
     parameters file output scr.par

     I'm computing from  .000  up to  1.300 seconds
          file scr.flt about : 164 blocks


 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2
          sound file 'carl' output ...
                                ... wait
$ [PLAY chant] [RETURN]
$
```

1.1.1.3 *How it works*

The "**chant**" command runs the executable code of the CHANT program, on the VAX 780.

parameter file :	The user can supply the program with values, prepared in advance, and which would erase the default values for the corresponding parameters. CHANT is asking for the name of a file containing a list of parameters with assigned values. [RETURN] simply means that you do not have such a file.
parameter :	Here, you could interactively change any default value, just by typing the name of the parameter to be changed followed by a space, then the new value and a carriage return. Just typing the name (followed by a carriage return) would display the current value of the parameter. [RETURN] simply means: I don't want to change any value.
computing : press <return> notions : press <?> <return> other file : press <f> <return>	
parameter :	[RETURN] simply means: I don't want anything but to compute the sound. You could also load new files by typing "f" followed by [return], or modify parameters again. Typing ? [return] would lead you into a kind of help loop, where you can obtain a brief summary of the parameters of your choice. Try it.

CHANT next displays the duration of the sound to be calculated, then it warns you that a file called **scr.par** has been produced on your current directory. We already mentioned that this file contains almost all CHANT parameters with their values for the current synthesis -- in the present case, all of them are default values. Each run of CHANT will erase the preceding scr.par file. Before going further, you can have a look at the default values in that file. (either use an editor or the "cat" and "more" commands).

The sound is then calculated (you can follow the calculation with the display of the current time of the synthesis), and a file named **chant** is produced on your current soundfile directory.

You can hear the result with the command **play.**

NOTE:
Every new synthesis will erase any pre-existing soundfile "chant". Rename every soundfile you intend to keep with the appropriate command ("mvsf" for CARL soundfiles, or "mv" for UNIX soundfiles).

1.1.2 **Session 2: interactive modification of the parameters**

1.1.2.1 *What you can do*

You can synthesize more than one note or event -- up to 999 --, using CHANT default values for the spectrum, but assigning new values to the following parameters:

the number of notes or events to be synthesized: **nnote**
the corresponding durations: **dr1 to dr999**
the corresponding fundamental frequencies: **f1 to f999**

NOTE:
It is not enough to define frequencies and durations for each of the "notes" you want: *you have to specify the number of notes or events* assigning a value to the parameter **nnote.**

1.1.2.2 *How you can do it*

From now on, carriage returns ([RETURN]), that end every line of the session, will not be figured. What you are to type is enclosed in brackets.

```
$ chant
      parameters file    :
      parameter          :      [nnote]
      nnote          =            1.00000      ? [4]
      parameter          :      [nnote]
      nnote          =            4.00000      ?
      parameter          :      [f1 = 50]
      parameter          :      [f2 = 100]
      parameter          :      [f3 = 300]
      parameter          :      [f4 = 150]
      parameter          :      [dr1 = 1]
      parameter          :      [dr2 = 1.5]
      parameter          :      [dr3 = .7]
      parameter          :      [dr4 = 1.3]
      parameter          :

      computing          :        press <return>
      notions        :        press <?> <return>
      other file     :        press <f> <return>
      parameter          :      [dr4 = .8]
      parameter          :

      computing          :        press <return>
      notions        :        press <?> <return>
      other file     :        press <f> <return>
      parameter          :

      time set :   4.000 seconds
       parameters file output scr.par

       I'm computing from   .000   up to    4.000 seconds

ETC .....
$
```

1.1.2.3 *How it works*

Parameters' default values are replaced with the values you type online. You can ask for a parameter value just by typing the name, then change the value or leave it unchanged (this is shown with the parameter nnote). Or you can assign values in a simpler way, just by typing the name of the parameter and the new value; the equal sign may be omitted.
You can check in the **scr.par** file that the parameters have effectively received their new values.

NOTE:
Try the **ls** command and notice that in addition to scr.par, CHANT produced two other files: **recitant** and **out.**
File **recitant** contains the following:

```
/usr/local/bin/chante
/usr/local/bin/calcul
chmod +x out
out
rm carl
```

chante (and not chant) reads the parameter values and builds the **scr.par** file (**chante** corresponds to the "first pass" of the CHANT program); **calcul** (the "second pass" of the CHANT program) is in charge of the synthesis, and **out** -- check by yourself the content of the executable file "out" --, just writes out the soundfile **chant** using a temporary file "carl".

1.1.3 Session 3: using parameter files

1.1.3.1 *What you can do*

CHANT can synthesize notes or events with a user defined spectrum. As the parameters involved may be numerous, the interactive modification of default values parameter after parameter becomes a heavy work. Instead, you can prepare in advance a **parameter file** containing every desired modification, and give it as an input to "chant".

1.1.3.2 *How you can do it*

Create a file named "**test.par**", for instance, which contains the parameters used in the previous example, as follows:

```
nnote = 4
f1 = 50
f2 = 100
f3 = 300
f4 = 150
dr1 = 1
dr2 = 1.5
dr3 = .7
dr4 = .8
```

When running **chant**, you just have to type the name of your parameter file, *omitting the ".par" extension:*

```
$ chant
      parameters file   :      test
      parameters file   :
      parameter         :

      computing         :      press <return>
      notions           :      press <?> <return>
      other file        :      press <f> <return>
      parameter         :

      time set :  4.000 seconds
       parameters file output scr.par
ETC .....
```

1.1.3.3 *How it works*

"**Chante**" (which is the first part of "**chant**") can read ordinary ASCII files *that end with the ".par" extension* and that follow a simple syntax. From a formal point of view, a parameter file contains a list of ASSIGNMENTS, COMMENTS, SEPARATORS.

ASSIGNMENTS The assignments are composed of
a PARAMETER NAME, followed by
the ASSIGNMENT CHARACTER "=" or by a BLANK, followed by
the PARAMETER VALUE.
There are three ways to assign values to parameters:

- direct assignment: "f1=100" or "f1 100" for instance.
- direct assignment as an "immediate function" (see next section)
- indirect assignment, by calling a function file (see next section).

SEPARATORS

- <RETURN> carriage return
- " " blank
- "," comma, which is used to separate various assignments on the same line.
- ";" semi-colon, which is used to end immediate functions (see next section).

COMMENTS

- "<" starts a comment
- ">" this is not recognized as the end of a comment, but it can be used for readability.
  A comment always ends with a carriage return.

PARAMETERS MAY BE LISTED IN ANY ORDER, and may be separated by as many blanks or blank lines as desired.
You can supply "chante" with ANY NUMBER OF PARAMETER FILES, and thus divide your

parameters and organize them at your convenience into different files.
Any **scr.par** file is a good example of a parameter file.

1.1.4 Session 4: functions and transitions

We do not give a session guideline here; try the different features by yourself.

1.1.4.1 *What and what for ?*

One often wishes to have parameters evolving in time. This can be achieved through the use of FUNCTIONS of time. In CHANT, functions are not algebraic expressions; they are *breakpoint functions*, each point being defined by a value and time pair. The values of the function between two points are obtained at each quantum by linear interpolation.

1.1.4.2 *Immediate functions and function files*

Almost every CHANT parameter can be assigned a function following two different methods.

First, you can use the so-called "**IMMEDIATE FUNCTIONS**" (immediate because the function is described directly, either interactively, or in a parameter file).
The format for immediate function assignment is as follows:

```
PARAMETER = /i                    <starts the function assignment>
value  time                       <break-points>
value  time
 .     .
 .     .
;                                 <ends function assignment>
```

The equal sign can be omitted.
Example: glissando on fundamental frequency of the first note or event

```
nnote 1
dr1          2
f1 = /i
80           0
100          1.2
200          2
;
```

In this example, f1 has the value 80 at time 0. (seconds), the value 100 at time 1.2 (sec.), etc...

Each of the "times" can be given in true values as in the above example, or as an increment from the preceding "time", this being indicated by the prefix **p** (p stands for plus). The above example can be written equally (considering only the breakpoints lines):

80	**0**	**or**	**80**	**0**	**or**	**80**	**0**
100	**1.2**		**100**	**p1.2**		**100**	**p1.2**
200	**p.8**		**200**	**2**		**200**	**p.8**

Obviously, the first "time" must be non-incremental .

(2) Second, you can use a normal ascii **FUNCTION FILE** containing the breakpoint lines of the function to be assigned. The default exension for a function file is **".fun"**, which is strongly recommended for clearness.
The format of the assignment, which can be done interactively as well as from a parameter file, is as follows (/f stands for file):

parameter =/f filename.fun

Note that the file must only contain the (value,time) pairs. The ending semi-colon is not required, as the end of the file marks the end of function assignment.

NOTE:
Here are some remarks to guide the choice between functions in or out of the parameter-file (immediate functions vs. function files). If the text of the function is in a *separate function file* the system will have to search for the function file, which may take some precious seconds especially if the system is loaded. Having a lot of files for functions constituted only by a bunch of points will encumber your directory. Furthermore, the function-files do not mention in which parameter-file they are used, so their management is not always very simple, except through tricky prefixed and suffixed names: It might be easier to see the functions in the parameter-file and not risk deleting them prematurely. But if the text of the function is long, it will make your parameter file a little less clear, and it can be very cumbersome if the function is repeated many times in the same parameter-file or in many different parameter-files.

1.1.4.3 *Time scaling*

The times specified in the breakpoint functions are in fact scaled by "chante" so that the begin time of the function is 0., and the end time is equal to the sum of the notes or event durations.
When using "absolute" "true" values, as in the previous examples, it makes no difference. But you can use proportions of the total duration of the synthesis instead of absolute time values. In the present version of CHANT, ***you must start every function with a zero time, to obtain a correct scaling***. (This may change in the future, check by yourself).

For instance if we synthesize a phrase *4 seconds long* and give the following function :

13 0
17 1.5
15 2

the program will use the "scaled" equivalent:

13 0
17 3
15 4

The **scr.par** file which is created automatically will contain the scaled time values.

It is possible to avoid time scaling by using a special function assignment:
parameter =/il **instead of** **parameter =/i** for immediate functions.
parameter =/fl **instead of** **parameter =/f** for function files.

1.1.4.4 *Scaling and offset for the parameter values*

The *values* of the function are multiplied or divided by a factor, and they are increased or diminished by an offset. Default values for the offset and the factor are 0 and 1, which induces no change. But you can reset the factor or the offset and their use, using some special signs, followed by the new values:

_* sets a scaling factor (multiplication)
_/ sets a scaling factor (division)
_+ sets a positive offset (addition)
_– sets a negative offset (substraction)

For instance:

1	0	is equivalent to	1	0
2	.1		2	.1
_*4		is equivalent to		
1	.2		4	.2
_/2		is equivalent to		
1	.4		.5	.4
_+35		is equivalent to		
1	.6		36	.6
_+0_*3		is equivalent to		
2	.9		6	.9

More than one reset can be placed on the same line as in the last example. Note that any reset will affect every value until a new reset is done.

You can also modify function files. However, in this case, all the values are affected by the arithmetic operators. The function file must be followed by the sign "_" (underline) and the operators and factors or offsets:

parameter =/f file.fun_*4

1.1.4.5 *Transitions*

A transition is *a subset of a function* preceded by a line composed of the letter "**t**" and the "**true**" (or absolute) duration of the transition, and followed by a line composed with the letter "**f**".

Example :

```
t       4.
                222     0
                444     1.5
                333     1.7
                222     2
f
```

The subset will be "scaled" to the indicated duration, thus leading to a subset of the function, as follows:

```
222     0
444     3
333     3.4
222     4
```

A transition should always be followed by a (value,time) point of the function which constitutes its *anchor point on the time axis*, in the sense that the last point of the subset will coincide with the following anchor point (you can think of it as a "transition toward" this anchor point). An example of a correct use of a transition could be:

```
100     0
                t               4.
                                        102     0
                                        104     1.5
                                        103     1.7
                                        107     2
                f
107     7
```

which is equivalent to :

```
100     0
                                        102     3
                                        104     6
                                        103     6.4
107     7
```

The transition will be 4 seconds long and will end at time 7. The indicated duration is "**TRUE**" (or absolute) in the sense that it is **NOT** scaled by the duration of the phrase. Note that the last value of the transition is the same as that of the anchor point. If not, the program would send an error message ("erreur de continuite a la sortie d'une transition") and would keep the value of the anchor point .

Two main reasons for using transitions :

- By varying the duration of a transition, one can play with the rapidity of a gesture.
- By embedding a transition in another, one can build the description of a gesture in which some segments vary and some others do not.

The second feature (embedded transitions) is available in some versions of CHANT, *but not in those available at Ircam when writing this manual. (Check by yourself if the version you are using offers this feature).* Anyway, here is an example of embedded transitions:

```
0  0
     t  6
          0  0
               t  4.
                    11  0
                    14  1.5
               f
          14  6
     f
14  6
               ^
               | embedded (inner) transition
     ^
     | embedding (outer) transition
```

The outer transitions is 6 sec. long; the last 4 s. are constituted by the inner transition. This means that:
Whatever the duration of the outer transition, its last 4 seconds will be a gesture from 11 to 14.
The number of levels of nesting is only limited by the capacity of the computer in those version of CHANT that allow embedded transitions.

NOTE:
The duration of a transition can NOT be longer than the time between the point before the transition and the anchor point, otherwise the beginning of the transition is skipped.

1.2 FOFs AND FILTERS: TECHNIQUES IN CHANT

For a detailed explanation of the CHANT synthesizer, you should refer to the articles mentioned in the bibliography. The goal here is just to clear some points, and describe the possibilities of the techniques used.

1.2.1 The FOF synthesizer

1.2.1.1 *The production model*

The CHANT synthesizer results from the following analysis of the singing voice, and production model:

The sonic wave of the voice is produced by a stream of air expired from the lungs into the vocal tract, through the larynx to the lips and nostrils. At various points, *three sonic sources* may disturb this wave:

- The vocal cords can modulate the stream of air breathed out through the larynx, producing quasi-periodic or *voiced* sounds.
- Narrowing of the buccal (mouth) cavity at certain points (lips, tongue palate, and glottis for whispered sounds) produces aperiodic sounds, which are known as *fricatives*.
- Interrupting and then releasing the air stream in the buccal cavity produces *plosive* sounds.

The resulting sound itself is modified by the resonance properties of the vocal tract, which acts like a filter. These filtering modifications may be described by the *transfer function* of the vocal tract, which varies continuously in speech or singing production.

The voice production is often represented by a model composed of:
-1- a source of excitation P (a periodic one for voiced sounds, and a random one for fricative and plosive sounds);
-2- a linear filter F1, with a given transfer function.

The source itself can be represented by a simple source P2 with a flat spectrum, and a filter F2 which represents the spectral envelope of the source. A single filter F can be used instead of F1 and F2, and the spectrum of the sound is obtained as the product of the flat source by the gain of the filter F.

1.2.1.2 *The FOF synthesizer and its specificity*

Instead of distinguishing the source and the filter F, the FOF technique associates the excitation and the resonance:
-1- The spectrum is described in terms of formants (peaks in the transfer function of the filter F);
-2- for each excitation, CHANT calculates the output signal corresponding to a given

formant: a **LOCAL ENVELOPE** is applied to a sinusoid, resulting in a **FOF** ("Fonction d'Onde Formantique", which means "formant wave function" in French). (Refer to the figures in the introduction.)

This could be compared to the use of parallel filters excited by impulses at the rate of the fundamental frequency, each filter corresponding to one formant. An automatic calculation (see section 1.3) is in charge of the amplitude balance between formants (this balance is not obtained directly because the model does not use filters in series). The final output is the sum of individual FOF signals.

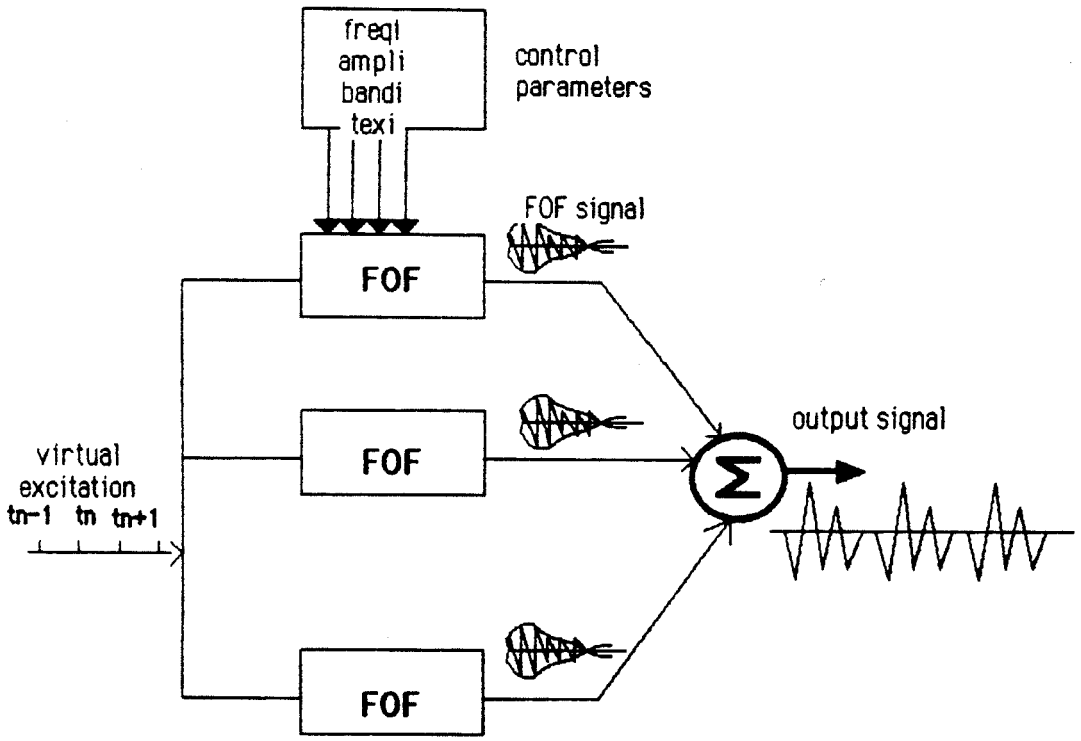


Figure 5. Structure of a FOF synthesizer

Let us remind you of the basic parameters that control a formant:

- **freqi,** the frequency of the sinusoid which is the center frequency of the i^{th} formant (resonance)
- **ampli,** the amplitude of the damped sinusoid which is the amplitude of the i^{th} formant.
- **bandi,** bandwidth of the i^{th} formant, obtained through the speed of damping: Note that a slow damping means a small bandwidth (sharp peak), a fast damping means a large bandwidth (broad peak).
- **texi,** the duration of the ascending portion of the envelope of the damped sinusoid which yields the width of the skirts of the i^{th} formant. Note that a fast attack means large skirts, a long attack means reduced width of skirts.

Once a FOF is triggered, there is no way to change its parameters; the user can play with this specificity of the FOF synthesis, deviating from vocal sounds. For instance, provided that each FOF has long excitation and damping (parameters **band, debatt, atten, tex**), *the user can superpose many FOFs for a given formant.*

The FOF synthesizer allows a kind of additive synthesis: formants with zero bandwidths correspond in fact to pure sinusoids; adjusting the formant parameters **tex, debatt, atten** and **phase** allows one to produce a sinusoid which is continuous from one FOF to the other. This will be described further (see chapter III).

The user can predict the effects of the basic parameter variations with the help of two different representations:
-1- a temporal one, considering a single FOF and its temporal properties;
-2- and a "spectral" one, considering each formant as a more or less resonating (narrow) filter (but remembering that parameters do not change continuously but in a discrete way from one FOF to the other, and that FOF superpositions may occur).

Figure 6 represents different FOF spectra and shows the effect of the parameter **tex.**

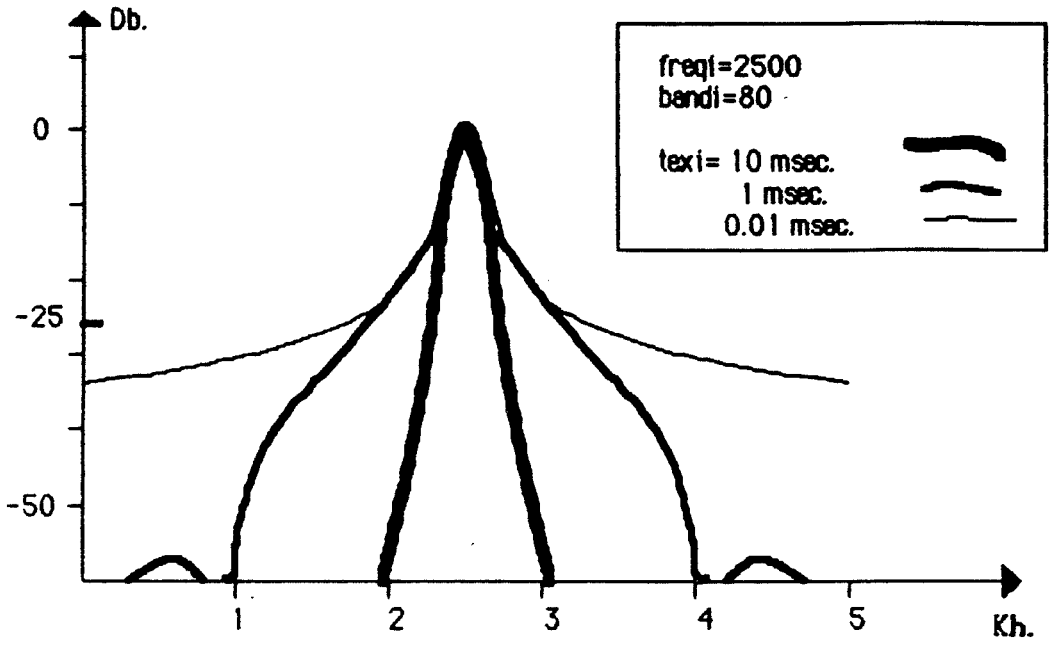


Figure 6. Log magnitude FOF spectra

1.2.2 **Filters in CHANT**

Second order parallel filters are now available in CHANT, allowing a source to be filtered. The control over filters is close to that of the FOFs; the user can specify *the amplitude, the center frequency and the bandwidth* of a given filter.

For historical reasons, the VAX and the FPS array processor versions of CHANT differ, in particular regarding the control over the filters.

1.2.3 **Main differences between the VAX and the FPS versions**

The user may be confused at first by the differences described below. He or she may learn to use the array processor version only, as it offers more possibilities. The VAX version was only used in this manual for the first learning sessions, but from now on, the array processor version is strongly recommended. However, here are the differences between the two versions:

1.2.3.1 *Filters*

In the VAX version, you can use only ten filters; the input to the filters can only be white noise. The filters' center frequencies are given by those of the formants (that is freq1, freq2 ... freq10). The amplitudes are specified by the parameters **vuser61 to vuser70,** and the bandwidths by **vuser81 to vuser90.**

In the FPS version, you can use up to 200 filters; you can specify their amplitudes **(amplf1 to amplf200),** their bandwidths **(bandf1 to bandf200),** and their center frequencies **(freqf1 to freqf200).** Filters can be used in parallel with FOFs, and with various input sources (see section 1.2.4).

1.2.3.2 *Noise in the FOFs*

In the FPS version , you can add noise (random variations) in the first ten formant wave functions (FOFs). The parameters **vuser61 to vuser70,** (which in the VAX version control the filter amplitudes), correspond to the noise amplitude, and **vuser81 to vuser90** (filter bandwidths in the VAX version), to the noise bandwidth.
These features are not available in the VAX version.

Here are some explainations about the noise algorithm:

The noise is produced by random variations of each FOF's LOCAL ENVELOPE.
The "**noise bandwidth**" corresponds to the frequency of new random value selections. The random values (between −0.5 and +0.5) specify the LOCAL amplitude variations, expressed as proportions of the central LOCAL amplitude value, and a linear interpolation is performed between two random value selections.
The "**noise amplitude**" is a scaler on the random values, and consequently, on the random variations; the greater the noise amplitude, the greater the random variations of the LOCAL ENVELOPE.
This can be summarized in the following formula:

local_amplitude = local_amplitude * (1 + random_value * noise_amplitude)

From a spectral point of view, the frequency of selection of new random values corresponds to the bandwidth of the main resulting formant (see the following spectra).

Figures 7 and 8 show the effect of the noise algorithm on the FOF signals and on the resulting spectra.

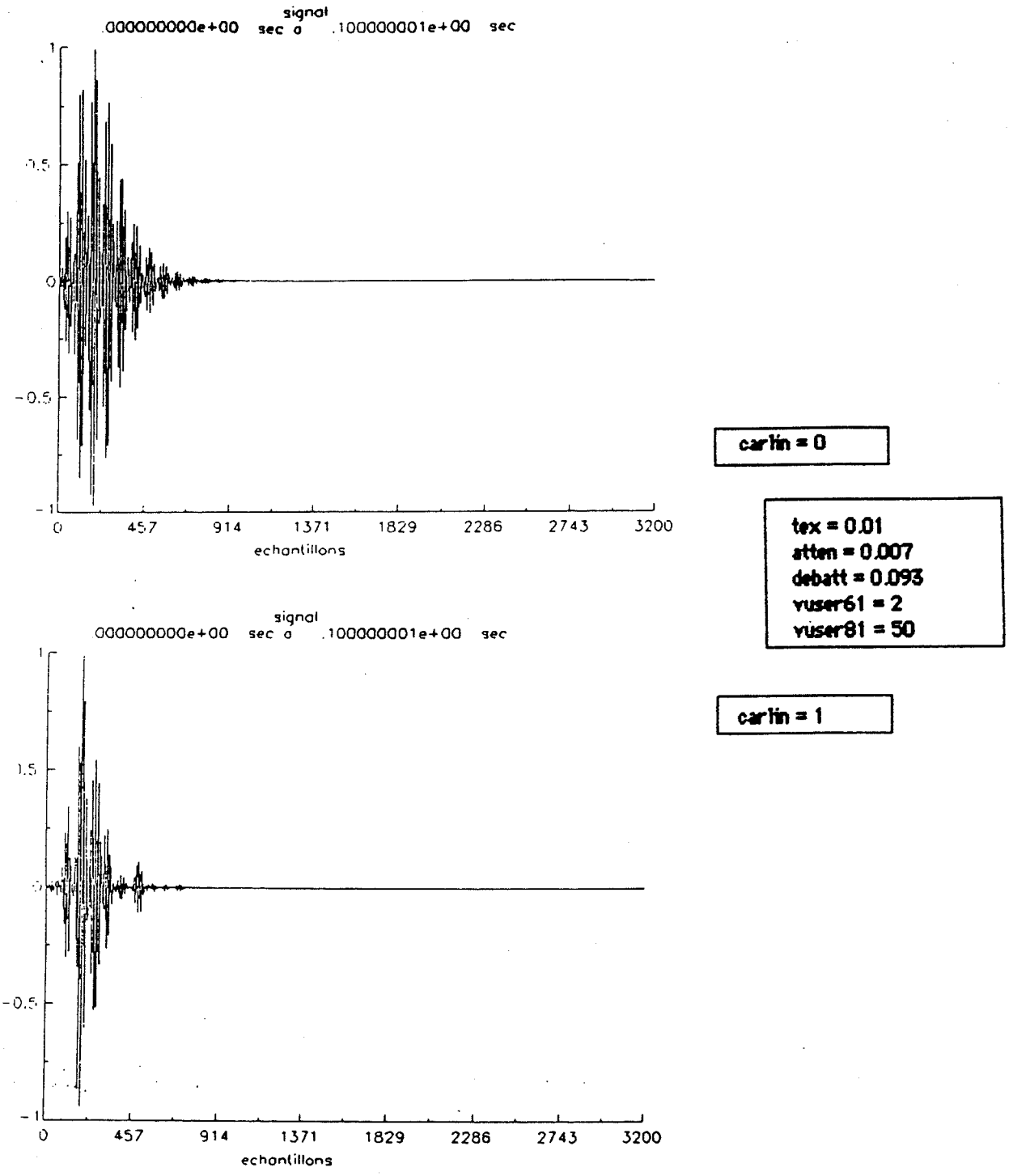


Figure 7. Effect of the noise algorithm on a FOF signal: -1- normal FOF -2- noisy FOF.

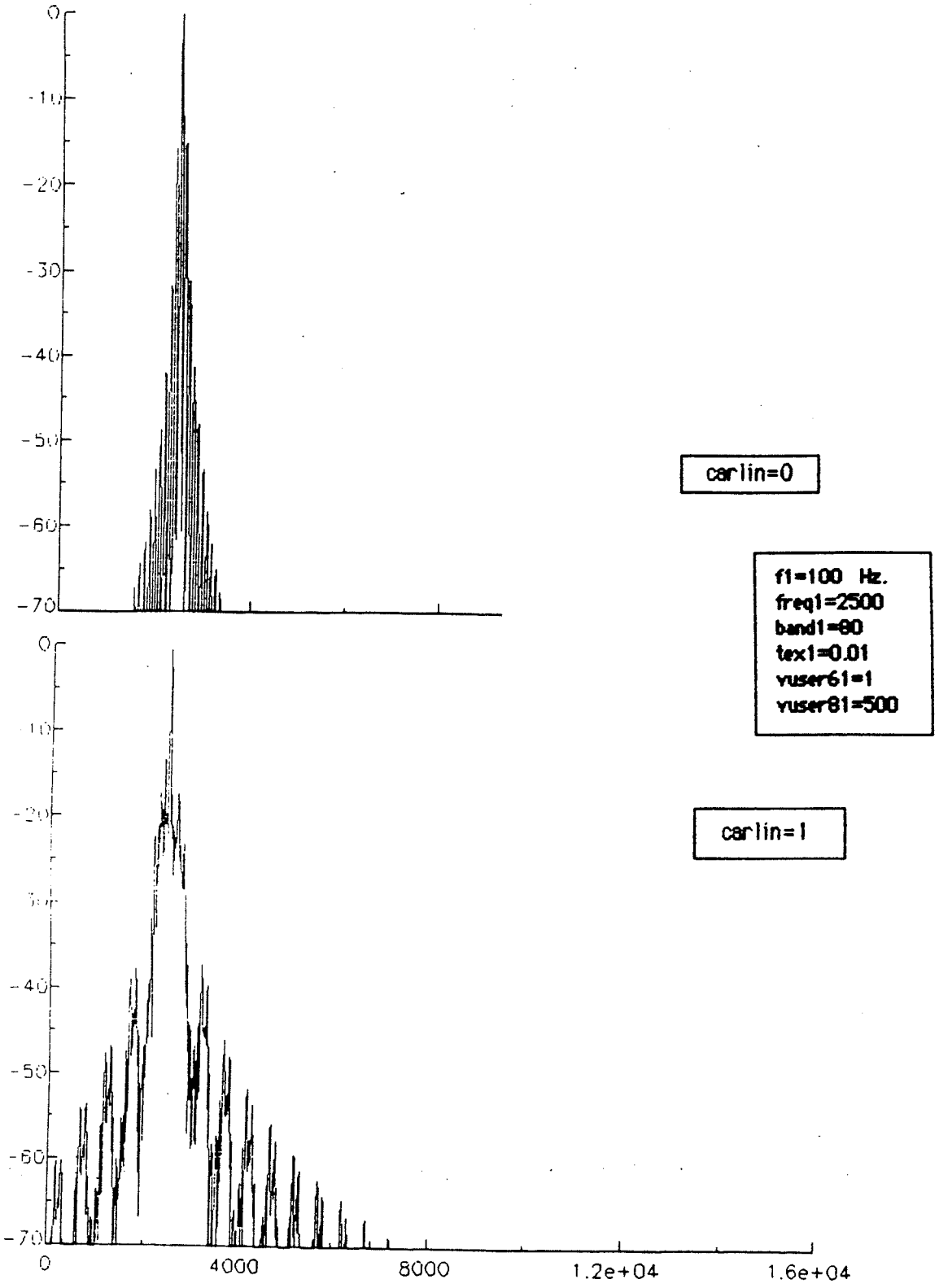


Figure 8. Effect of the noise algorithm on a spectrum: -1- one formant without noise -2- same formant with noise.

1.2.4 Parallel synthesis on the array processor

Input to the filters may be one of the following in the FPS version of CHANT:

White noise: The parameter **play** is used to send white noise as the input to the filters, when positive. It is a flag, and a scaler on the noise amplitude.

External source: An already existing soundfile can be used as the input to the filters, when the parameter **apf** is positive; **apf** is a scaler on the input amplitude, in this case.

Noise and an external source can be used at the same time, as inputs for the filters, and filters can be used in parallel with FOFs. In a future version, the output of the FOFs will also be allowed as inputs to the filters.

Figure 9 represents the implementation of the CHANT program on the array processor. Parameters are evaluated (for each quantum) on the VAX, and then passed to the FPS where the synthesis takes place.

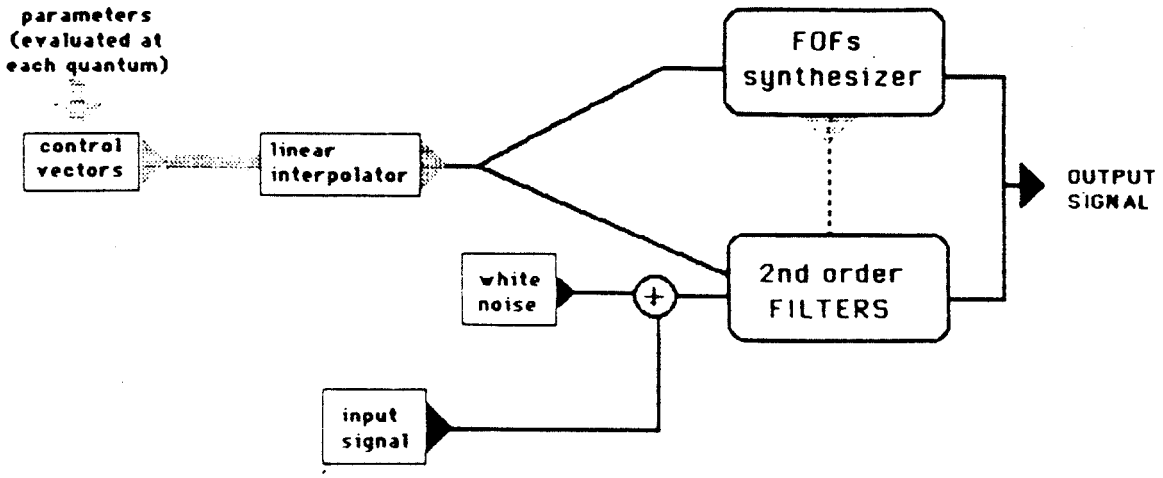


Figure 9. FOF and Filter synthesis on the FPS array processor

NOTE:
Some parameters -- mainly those which control the filters --, have a different meaning in the VAX and the FPS versions of CHANT; refer to the alphabetic list of parameters (appendix 1).

1.3 SOME AUTOMATIC CALCULATIONS AND RULES

CHANT is a *synthesis by rule* system originally designed for modeling the singing voice. In other words, the knowledge learned through analysis of the singing voice has been "stored" in algorithms (or rules, or "knowledge models"). These algorithms may be executed or not when running CHANT, depending on the value of some standard parameters or flags. Note that these rules can be skipped at runtime, and replaced with your own rules (see next chapter on user's subroutines).

For several rules -- jitter, transitions between two notes or events, tremolo, hoarseness, etc... --, the parameters involved are self-explanatory. Some other algorithms require some explanations, for a better use of CHANT. We must introduce here a somewhat artificial distinction between the so-called "**automatic calculations**", and the other **algorithms:** "**Automatic calculations**" are algorithms derived from analysis of the singing voice, and which enable the user *not to specify some standard CHANT parameters* (mainly formant amplitudes and bandwidths), although obtaining a satisfactory result. There are three automatic calculations in CHANT, for the formants' amplitudes and bandwidths, and for a complementary formant -- the so-called "complement".

We shall describe the following automatic calculations and algorithms:[2]

- Automatic calculation of the formant bandwidths.
- Automatic calculation of formant amplitude.
- Automatic calculation of the complement.
- Adjustments of formant amplitudes
- Adjustment of first and second formant frequencies.
- Vibrato on the fundamental frequency.
- Correction of the spectrum

1.3.1 Automatic calculation of the formants' bandwidths

This automatic calulation takes place if the flag **atb** is set to 1; it is skipped otherwise.

When the algorithm is used, the bandwidths specified in the parameter files and default bandwidth values are ignored. The formant bandwidths are computed according to a function of the formant center frequencies. The user can shape the variation of the bandwidth with the center frequency, assigning values to the following CHANT parameters:

2. the order chosen here does not correspond to the sequencing of the algorithms in the CHANT program.

fref1
fref2
fref3
bref1
bref2
bref3

Fref1,2,3 are *reference frequencies;* a *reference bandwitdth* **(bref1,2,3)** corresponds to each *reference frequency.*
CHANT calculates a parabolic curve determined by the three (reference frequency, reference bandwidth) points, and assigns an automatic bandwidth to every formant in relation to its center frequency according to the function so defined.
Figure 10 should make this point clear; the reference values of the figure are better than the default ones for modeling a singing voice.

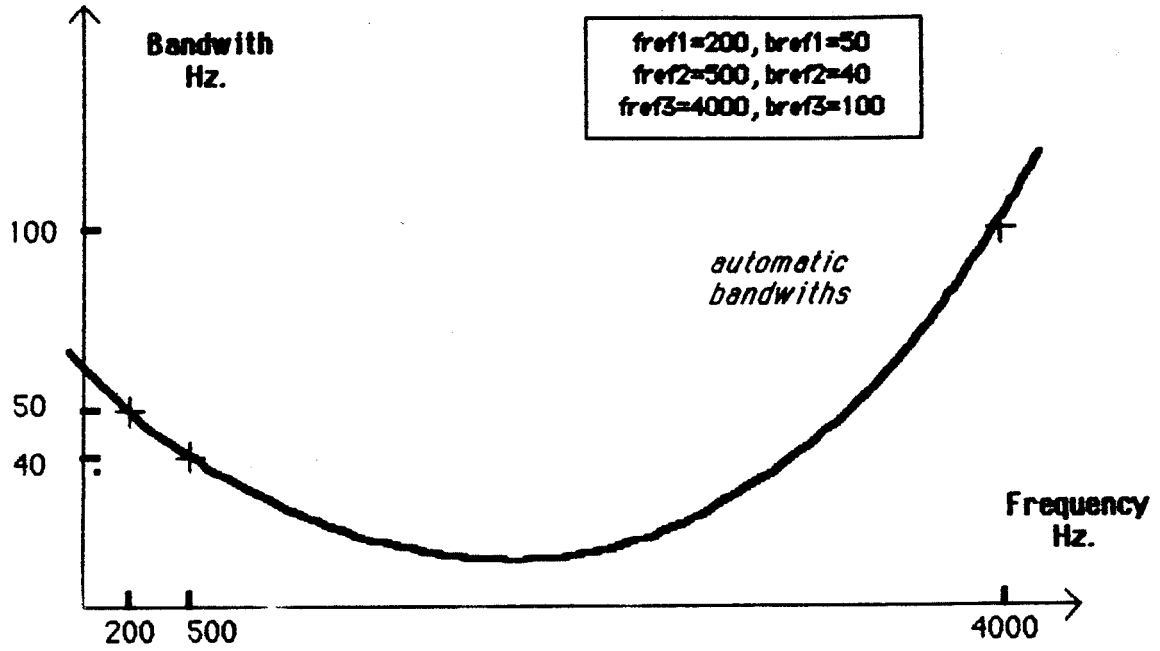


Figure 10. Automatic bandwidth as a function of formant central frequency

NOTE:
-1- *Reference frequencies and bandwidth must be constant:* the polynomial which gives the automatic bandwidth values in relation to the central frequencies of the formants is calculated only once, at the beginning of the sound.
-2- Beware that the polynomial you use always has positive values; negative bandwidths are not allowed !

1.3.2 Automatic calculation of the amplitudes of the formants

The automatic calculation of the amplitude of the formants simulates a filter series and, according to the frequency of each filter, fixes the formant amplitudes. The goal of this calculation is *to take into account the interactions between formants: when two or more formants approach one another, the filter simulation enables the reinforcement of their amplitudes.*
This automatic calculation is skipped when the flag **ata** is zero. It is executed if it is greater than zero; the given values for ampl1,ampl2, etc... are then modified to obtain the "automatic" amplitude values.

The automatic calculation has another function: *it scales every formant amplitude with a function that varies as 1/f.* Thus, if you specify an amplitude of 1 for every formant, and use the automatic amplitude algorithm, the resulting spectral envelope will not be "flat", but decreasing with formant frequency. The 1/f function has proved correct in approximating natural voice spectra.

1.3.3 Automatic calculation of the complement

A supplementary formant, known as the "complement", is automatically assigned when the parameter **atc** has a value greater than 0 in the FPS version, and a value of 1 in the VAX version. In the FPS version, in the case where automatic calculation of formant amplitudes is executed and the complement is calculated, **atc** is also a scaler on the complement's amplitude.
Adding a complement reinforces the first formant, the fundamental partial and the low register.
Its frequency is specified by the parameter **fcomp,** which is 80 Hz. by default. When using n formants, the complement aotomatically receives the number n+1.

1.3.4 Adjusting the amplitudes of the formants

When a singer sings loudly, the spectrum of the voice is different from when he or she sings softly. In a synthesized voice sound, increasing the amplitude without changing the spectrum produces what has been called the "volume-control effect". The impression is that of a recorded voice being turned up on an amplifier, not that of a singer exerting more effort to produce a louder sound.

It was found that for a constant distance between source and listener (the distance from loudspeaker to listener if there is no added reverberation for the synthesized sound) there are combinations of loudness and spectrum which are generally judged as humanly possible, and others which are judged as distinctly impossible. For a given loudness, the threshold was observed to be quite sharp between spectral weightings that seemed possible and ones that seemed impossible, and the range of possible spectra changed when varying the fundamental frequency.

In CHANT, several parameters allow one to take into account these observations. One can control the spectrum in relation to the dynamic amplitude of the sound, the type of the voice which is being used, and the fundamental frequency. The parameters involved

are:

- **cslope (flag, and slope of the spectrum),**
- **sex (type of the voice being used),**
- **f0moyen (the middle frequency of the range of the voice),**
- **the fundamental frequency,**
- **and coefamp (the dynamic amplitude, which modifies the spectrum).**

The way the algorithm works is as follows:
A region of possible ratios between *amplitude and spectral slope* was found for five fundamentals spanning the male vocal range and for five from the female range. A formula was found for each voice range which maintains humanly possible ratios between amplitude and spectrum. The parameter **sex** determines which of the two formulas is employed. They are of the general form:

adjusted formant amplitude =
given formant amplitude * coefamp * fundamental-frequency-linked scaler.

The scaler varies linearly with the fundamental frequency and has a negative slope (it decreases as the fundamental increases).
For the male voice (**sex = 1**) the scaler is given by:

3 + 1.1*(400-fundamental frequency)/300

For the female and castrati voices (**sex=0 and sex=2**):

0.8 + 1.05*(1000-fundamental frequency)/1250

The parameter **cslope** controls whether the previous formulas are used or not.
-1- **If cslope = -1,** they are used, thus providing an automatic correction of the spectrum in relation to coefamp, the fundamental frequency, and the type of the voice. Only formants which have a higher center frequency than that of the first formant are adjusted.
-2- **If cslope is positive,** the automatic adjustment is skipped. In this last case, **cslope** acts like a scaler on the amplitude of every formant which has a higher center frequency than formant 1, in relation to formant 1 and and the "fundamental" (the latest in the VAX version only) amplitude.
In fact, the formula used is somewhat more complicated; there is an additional factor, and the final scaler on formant amplitudes is:
scaler = cslope * exp (ajus1 * atan (ajus2 * alog(fundamental/f0moyen)))
Ajus1 and **ajus2** are standard CHANT parameters. Note that **ajus1** has a default value of 1, which yields a scaler equal to cslope.

In both cases -- whether cslope is positive or not --, there is another adjustment, which depends on the dynamic amplitude (coefamp) and the fundamental frequency: the louder and higher the sound, the richer the spectrum. This is achieved through the use of a scaler, which is:
scaler = coefamp * (fundamental_frequency/f0moyen) ** ajus3

Note that **ajus3** has a default value of zero; there is no adjustment in relation to the fundamental frequency in this case.

The parameter **envelo**, which scales every formant amplitude, does not modify the spectrum.

NOTE:
These adjustments are made in the main program for the second pass of CHANT,[3] the sources of which are in "capap.f" for the array processor version; you can check in this file how the adjustments work. (Try "man chant" to find the location of "capap.f" on the system).

1.3.5 Adjustment of first and second formant frequencies (bending)

Johan Sundberg has measured the changes in first and second formant frequencies with changes in fundamental, which are summarized in the appendices. As the fundamental moves to higher values, the first formant remains fixed until the fundamental reaches its frequency, and then it tracks the fundamental at the same frequency. The second formant moves downward in pitch as the fundamental moves higher than 200 hz, but when the interval between fundamental and second formant has decreased to an octave plus about 30 hz, that interval is maintained between the second formant and the fundamental, and the second formant moves in parallel with the fundamental as it rises further.

The CHANT algorithm does not differentiate usual female and male voices (sex=0 and sex=1); the previous rules are in both cases applied to the first and the second formants. For a castrato voice (sex=2), only the first formant is adjusted.
NOTE:
The CHANT parameter **cor**, *beside its main function (see the "correction" algorithm, section 1.3.7 below)*, allows a modification of the effect of the "bending" algorithm: if 0, the two first formants are left unchanged; if set between 1 (which is the default value) and 0, **cor** acts like a scaler on the frequency corrections to be applied to the first and second formants.

1.3.6 Vibrato on the fundamental frequency

In the singing voice, the vibrato is an oscillation --which has a given repetition rate -- of the fundamental frequency around a center frequency which is perceived as the pitch of the note. The vibrato frequency varies from 5 to 7 Hz. -- and variations are quite noticeable between these values --, but it is rather constant for a given voice. For high fundamentals, the vibrato is faster than for lower ones. It is not strictly periodic as there

3. The first pass of the CHANT program corresponds to the **chante** command; it reads the parameter files, prepares a data file for the second pass, and writes out the file **scr.par** on your current directory. The second pass, which corresponds to different programs in the VAX and FPS versions, evaluates the parameters for the synthesis according to the data, to the standard CHANT rules, and eventually to user defined rules.

are some random variations in its frequency (or repetition rate), and in its amplitude. Vibrato is important in the definition and the recognition of a given timbre, as it sweeps across vocal tract resonances corresponding to the formants, and, consequently, reveals them to the ear.

CHANT parameters which enable the user to simulate a vibrato are the following:

vibamp:	amplitude of the vibrato expressed as a proportion of the fundamental frequency.
vala1 and vala2:	amplitudes of the random amplitude variations of the vibrato, expressed as ratios of vibamp.
tvala1 and tvala2:	times between selection of new random values for vala1 and vala2.
vibfreq:	frequency of the vibrato (in Hertz).
valf1 and valf2:	amplitude of the random variations of the vibrato frequency, expressed as a ratio of vibfreq.
tvalf1 and tvalf2:	times between selections of new random values for valf1 and valf2.

These parameters are explained in detail in the alphabetic list of parameters (see appendix 1); here, we shall underline some points, and give a simple example.

A variation around a central value is often expressed in CHANT as a ratio of the parameter it is applied to, whether the variation is a random or a periodic one.
- The corresponding parameter can specify *the maximum possible excursion above and below the central value – that is, half the "total excursion" (or half the range of variation)* –; this is the case for the parameter **vibamp.**
- Or it can correspond to the *range of the variation (the "total excursion")*, as is the case for the parameters **vala1, vala2, valf1, valf2.**[4]

Now, suppose that the random parameters are null, and that the other parameters are given the following values:

- **fundamental frequency = 100 Hz,**
- **vibamp = 0.1 (that is 10% of the fundamental frequency),**
- **vibfreq = 2 Hz. (one period every 0.5 seconds).**

The CHANT vibrato algorithm will result in a fundamental frequency which describes a sinusoid, starting from 100 Hz, evolving between 90 and 110 Hz., with a period of 0.5 seconds (Figure 11).

4. In the FORMES version, and possibly in a future array processor version of CHANT, only one of the two possibilities mentioned will be chosen, and used as a standard.

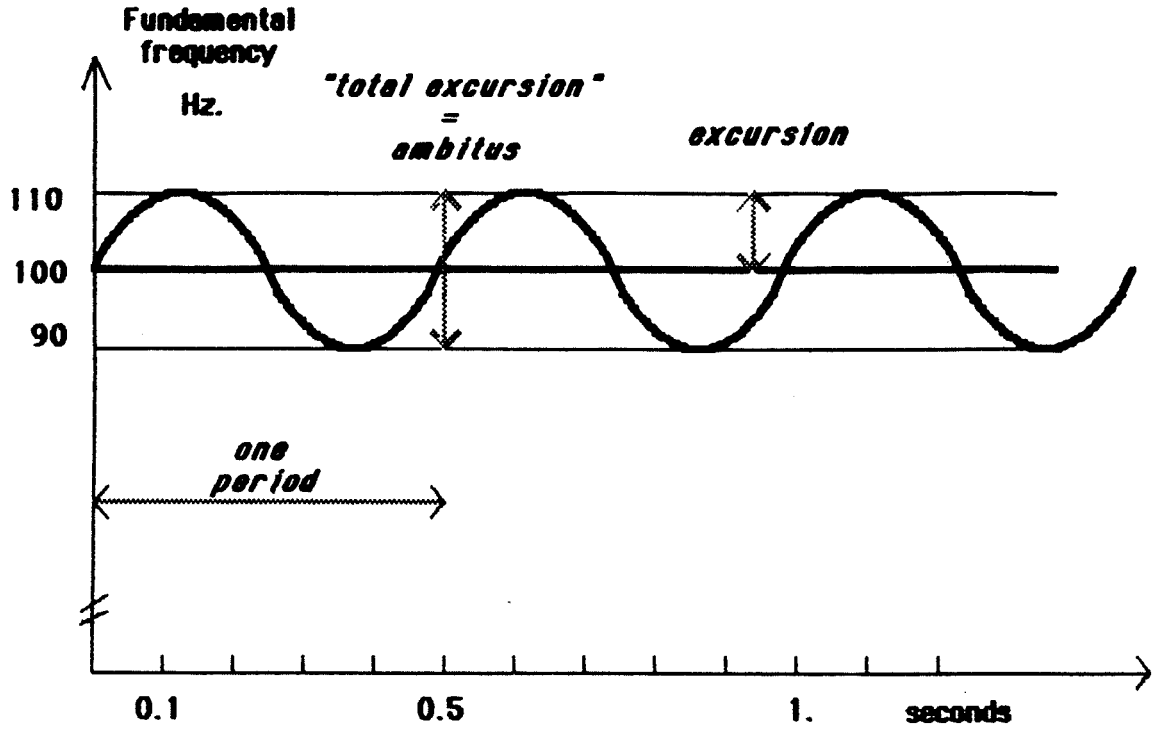


Figure 11. Vibrato: the effect of vibamp and vibfreq

Suppose we now add random variation to the amplitude of the vibrato:

- **vala1** = 1.
- **tvala1** = 2 sec.
- **vala2** = 0.

These values correspond to a variation of 50% above and below the central value of the vibrato amplitude, at most. (The range or "total excursion" is 100% of the central value, at most).
New random values will be given to the vibrato amplitude every 2 seconds.
The second random parameter, vala2, is null (no random variations), in order to simplify the example.
These values result in a variation of the parameter vibamp between:
0.15 (that is: 0.1 + (0.1 * 0.5)) and 0.05 (that is: 0.1 - (0.1 * 0.5))
The resulting fundamental frequency evolves between:
115 (that is: 100 + (100 * 0.15)) and 95Hz. (that is: 100 - (100 * 0.15))

Figure 12 shows the maximum, minimum, and average "total excursions". The crosses correspond to random value selections for vibamp.

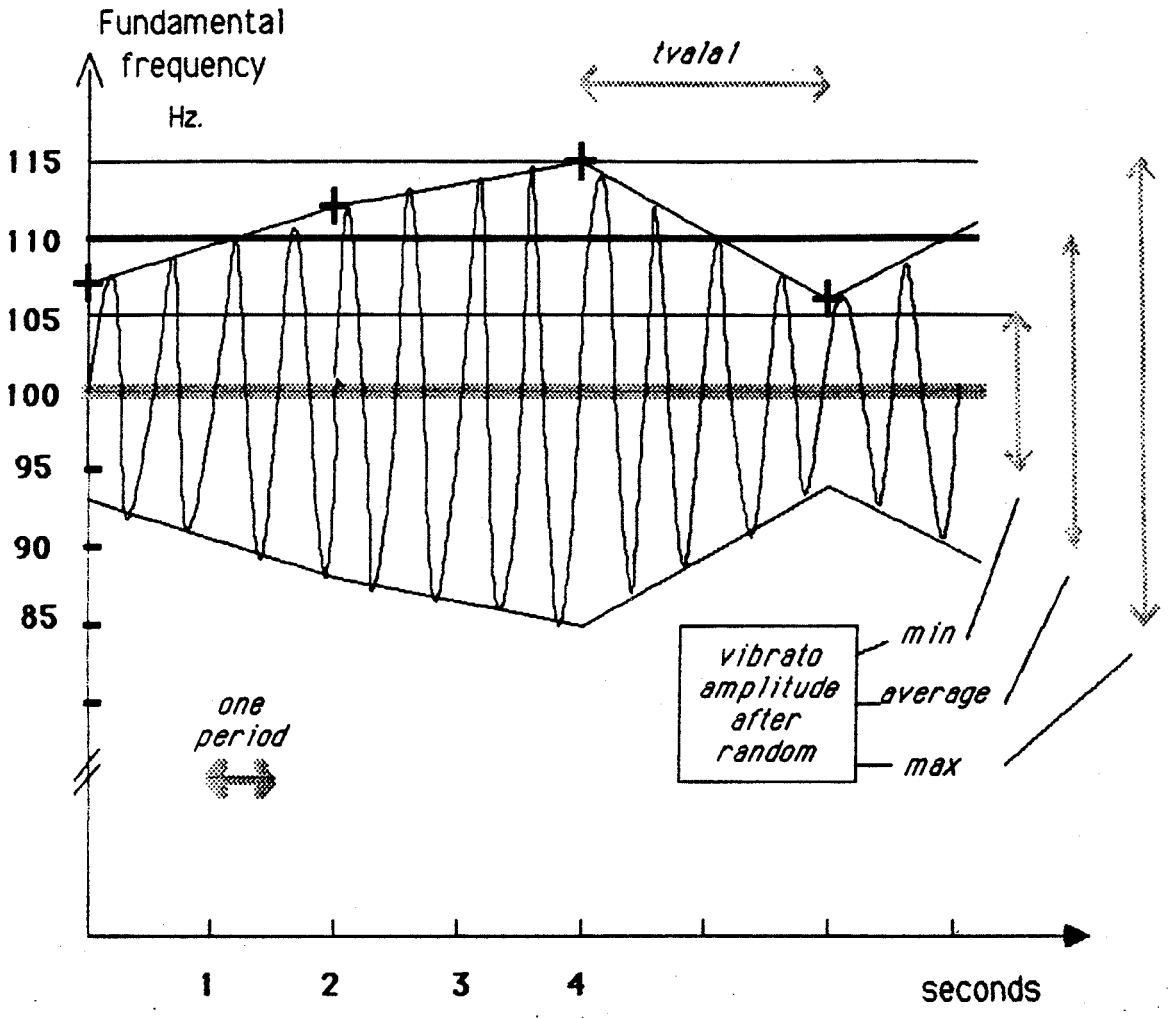


Figure 12. Vibrato: random variations of the vibrato amplitude

Random variation on the vibrato frequency can be added as well:

- **valf1 = 1.**
- **tvalf1 = 1.**
- **valf2 = 0**

Every second, the vibrato frequency will be given a new value, between 1Hz. (2 Hz. - (0.5 * 2 Hz.), and 3 Hz. The vibrato will slow down, or accelerate between two random selections, according to the random values.

1.3.7 Correction of the spectrum

The formant amplitudes, as specified by the user or as calculated by the automatic amplitude calculation algorithm, are not absolute but relative amplitudes: they are intended for controlling the amplitude balance between formants. This balance would be respected in the output signal, without further calculations, if every formant had the same bandwidth and the same skirt-width, which is usually not the case. In fact, the bandwidth and the width of the skirts of a formant are related to the energy corresponding to that formant in the output signal. The narrower the formant bandwidth and the skirts, the more the FOF superpositions that occur for a given formant, and the greater the resulting energy.
The "correction" algorithm modifies each formant amplitude according to the parameters **tex** (duration of the attack of a FOF, which corresponds to the skirtwidth of the corresponding formant), and **band** (bandwidth) of the given formant, in order to respect the "balance" as specified by the user.

It is the parameter **cor** which controls whether the algorithm is executed (if cor is different than 0) or not (cor=0).
NOTE:
-1- **Cor** is also used for controlling the "bending" algorithm (2.3.5 above).
-2- A similar amplitude correction is automatically applied to the filters; the algorithm cannot be skipped in the present CHANT version, but it will be optional in a future version.

1.4 BRIEF SUMMARY OF THE PARAMETERS ACCORDING TO THEIR CONCERN

In this section we describe the parameters for synthesis, trying to respect some structural or hierarchical order while placing the more important of them first.
These parameters concern:

— fundamental frequency control
— spectral shape description
— loudness of the sound
— local envelope of the sinusoids (formant waveforms)
— general control over the synthesis

As they can influence several of these characteristics, some parameters occur in more than one section. You can refer to the alphabetic list (see appendix 1), which gives a more detailed explanation of the role of each parameter.
NOTE:
Parameter names should always be lowercased when using CHANT.

1.4.1 Fundamental frequency

f1,...999 The **Fundamental Frequencies** for the notes or events of the phrase.
dr1,...999 The corresponding **DuRations** for each note or event (in seconds).
ttr One-half the **TRansition Time** between successive notes or events.
nnote The **Number of NOTEs or events** in the phrase being synthesized (up to 999).

1.4.1.1 *Random variations of the fundamental*

jitt1,2,3 **Random variation** of type 1,2,3 expressed as a **ratio of the fundamental frequency (jitter).** In fact, the ratio expresses the *maximum "total excursion" (the ambitus)*, that is twice the maximum excursion above and under the fundamental frequency. Types 1,2,3 differ by the time between selections of new random values, and by the ratio value.
tjitt1,2,3. **Times between selections of new random values** for jitt1,2,3.
seed If negative,its absolute value initializes the random series. Default is zero.

1.4.1.2 *vibrato (fundamental frequency vibrato)*

vibamp **AMPlitude of the VIBrato.** (the maximum excursion above and under the center frequency -- in other words half the ambitus --, expressed as a ratio of the center frequency).
vibfreq The **FREQuency of the VIBrato** in hertz.

1.4.1.3 *random variations of the vibrato*

vala1
vala2 Ratios of the **random variations of the Vibrato Amplitude (the maximum "total excursion" or ambitus).**

valf1
valf2 Ratios of the **random variations of the Vibrato Frequency (the maximum "total excursion" or ambitus).**

tvala1
tvala2 **Times between selections** of new random values for variations of the **Vibrato Amplitude**

tvalf1
tvalf2 **Times between selections** of new random values for variations of the **Vibrato Frequency.**

seed If negative,its absolute value initializes the random series. Default is zero.

1.4.2 spectrum

freq1,...200 The **center FREQuencies (in Hz.) for the formant regions.**

ampl1,...200 The **AMPLitudes of each formant region.** Only the ratios between formants' amplitudes are significant.

band1,...200 The **BANDwidth in hertz at –6db for each formant region.**

tex1,...200 **EXcitation Time (duration in sec.)** for the formant, i.e. **skirt-width.**

nof The **Number Of Formants.** The possible range is 1 to 200 in the FPS version.

fcomp **Center FREQuency of the Complement** (the complement is a "complementary formant" that may be added below the first formant to reinforce its amplitude, and -- in the VAX version only --,that of the fundamental).

apf Stands for "**AmPlitude of the Fundamental**". **In the VAX version of CHANT only,** a sine wave with a frequency equal to the fundamental frequency can be added to the sound in order to boost the fundamental partial. In the array processor version, this feature has been abandonned and **apf** has a different meaning (see below).

hn HoarseNess: Random variation of formant amplitudes. **Hn** represents the maximum range of the variation (twice the excursion above and below the formant amplitude). Selections of new random values occur at every quantum.

1.4.2.1 *slope of the spectrum*

hollow Controls the **amplitude of the "fundamental" (in the VAX version only), that of the first formant, and that of every formant which has a lower center frequency than formant1, relative to the other formants' amplitudes.** It is a scaler, with a default value of 1.

cslope Controls the amplitude of formants 2 and upwards (in fact, of every formant which has a higher center frequency than formant 1), in

relation to that of formant 1 and complement.
If positive, it is a **scaler** on high formant amplitudes, which can be modified through the CHANT parameters **ajus1** and **ajus2.**
If negative, it acts like a **flag to implement automatic calculation** of high formants' amplitudes in relation to that of the first formant, and according to the type of the "voice **(sex, f0moyen),** to the fundamental frequency **(f0),** and to the dynamic amplitude **(coefamp).** (see section 1.3 on automatic calculations).

coefamp **COEFficient of AMPlitudes of formants.** For (de)crescendi; induces dynamic changes in the spectrum.

dur1,...200 **Attack DURation,** for each of the formants, at the very beginning of the synthesis.

durf In the VAX version only; durf is the equivalent of dur1-200, for the fundamental (see **apf**).

dvr1,...200 **Decay DURation** during which amplitudes are modified, for each of the formants, at the very end of the synthesis.

dvrf In the VAX version only; dvrf is the equivalent of dvr1-200, for the fundamental (see apf).

dga,dgf **General Duration of the Attack and decay (Fin)** at the very beginning and at the very end of the whole "phrase" to be synthesized; for each formant, the parameters duri and dvri are respectively scaled by dga and dgf.

exa and exf Permit **EXponential curves** for the **Attacks and decays (Fin),** (at the very beginning and at the very end of the sound). Default values are 1, which leads to linear attacks and decays.

sex type of the voice being used: 0 is female, 1 is male, 2 is castrato. Modifies the spectrum through the cslope algorithm.

ajus1
ajus2
ajus3 **AdJUStments in amplitude** of "fundamental" (on the VAX only), and formants in relation to fundamental frequency (see section 1.3 on automatic calculations).

f0moyen **Mean (MOYEN) frequency** for amplitude changes in relation to fundamental frequency. F0moyen is the middle frequency of a given voice frequency range.

1.4.2.2 *automatic calculation of the spectrum*

ata If set to 1, **formant Amplitudes are calculated AuTomatically.**

atb If set to 1, **formant Bandwidths are calculated AuTomatically .**

atc If set to 1, **"Complement" is calculated AuTomatically. Also a scaler on the complement amplitude when positive, and in the FPS version.**

bref1,2,3 **REFerence Bandwidths** used for bandwidths automatic calculation.

fref1,2,3 **REFerence Frequencies** used for bandwidths automatic calculation.

hn **HoarseNess:** Random variation of formant amplitudes.

ajus1,2,3 **AdJUStments in amplitude** of "fundamental" (in the VAX version only), and formants.

f0moyen **Mean (MOYEN) fundamental** of a singer's voice range.

1.4.3 loudness of the sound

amp Specifies the **absolute AMPlitude** of the whole note or phrase. (The output signal is scaled at the very end of the synthesis so that its maximum amplitude is equal to **amp).**

coefamp For (de)-crescendi. Modifies amplitude and spectrum.

envelo **ENVELOpe:** the classical "potentiometer" effect: modifies the amplitude without altering the spectrum.

tremolo Causes a **periodic change in the amplitude of each formant.**

dur1,...200 **Attack DURation** for each formant, and for the very beginning of the sound. Relative amplitudes (ampl1,2,...) are modified.

dvr1,...200 **Decay DURation** for each formant, for the very end of the sound. Relative amplitudes are modified.

dga,dgf **General Duration of the Attack and decay (Fin)** for the beginning and the end of the "phrase" to be synthesized. **for each formant, the parameters dur and dvr are respectively scaled by dga and dgf.**

exa and exf Permit **EXponential curves for the Attacks and decays (Fin).**

1.4.4 local formant envelope

tex1,...200 **EXcitation Time** for each local formant envelope (ie. pitch synchronous excitation).

band1,...200 **BANDwidth** for each formant. Bandwidths correspond to the rate of decay of the local envelope (small bandwidths: long decay; large bandwidths: short decay).

debatt **Beginning time of the final ATTenuation** of local envelopes .

atten **ATTENuation duration** for local formant envelopes .

cor When 0 avoids an automatic **CORrection of the amplitude of each formant** in relation to the parameters tex and band. (Also used as a scaler on the "bending" correction).

1.4.5 control over the synthesis

dsil **Duration in seconds of silences** to be calculated before and after the sound.

dsk **In the FPS version, the frequency (in Hz.)** to be used for refreshing parameter values for the synthesis. It is the inverse of the quantum (ie. the duration between successive parameter evaluations, expressed in seconds). Out of use in the VAX version, in which the quantum is set to the fundamental period.

e **Sampling rate** for the output sound file. The lower the sampling rate and the quantum are, the faster the synthesis.

phase1,...200 Coefficient of **continuity of phase** for formant-sinusoids (FOFs).

tfin **End Time of the synthesis** regardless of **nnote** and durations.

tdeb **Beginning Time of the synthesis** (0 by default).

user It is a flag. If 1, the CHANT program **calls a user's subroutine (user defined rules).**

vuser1,...500 **Variables at the USER's disposal,** from vuser1 to vuser500. Some

numbers are already used by CHANT, namely:
In the FPS version:
Vuser61 to 70 = the amplitude of the noise to be generated around the first ten formants.
Vuser81 to 90 = the bandwidth of the noise (in Hz.) to be generated around the first ten formants.
In the VAX version:
Vuser61 to 70 = the amplitude of the filters (frequencies are given by the formants' ones). **Vuser81 to 90** = the bandwidth of the filters (in Hz.)

carlin In the FPS version, carlin control the noise to be added to the FOFs (see below).
In the VAX one's, it controls whether or not a soundfile is to be produced after the synthesis is done (carlin different than zero, and carlin=0, respectively).

1.4.6 Control over filters

1.4.6.1 *In the FPS version:*

freqf1,...200 **Center frequencies** of the filters.
bandf1,...200 **Bandwidths** (in Hertz) of the filters.
amplf1,...200 **Amplitude** of the filters. Only the proportions between filters' amplitudes are significant.

1.4.6.2 *In the VAX version*

freq1,...10 The center frequencies of the filters are given by those of the first ten formants.
vuser61,...70 Amplitudes of the filters.
vuser81,...90 Bandwidths of the filters.

1.4.7 Controlling the array processor

In the FPS version of CHANT (relative to the VAX one), some parameters either have some additional functions, or are used in a different way.

dsk **Frequency (the inverse of the quantum)** for evaluating the parameters.

debatt When running CHANT on the array processor, debatt has a special supplementary function: it is used for controlling the filters.
if negative, **only filters** are used for the synthesis, excluding FOFs.

apf **If -1, filters are not to be used with an external sound source.**
If positive or null, apf is a **scaler on the input amplitude** of the external source sent into the filters.

play Equivalent to **apf**, for using **white noise** instead of an external source as the input to the **filters.** Play is a scaler on the input amplitude of the white noise.

carlin **If zero, no noise in the FOFs. If positive, acts like a scaler** on the amplitude of the noise which is superimposed to the "normal" FOFs signals.

1.5 RUNNING CHANT ON THE FPS ARRAY PROCESSOR

Running CHANT on the FPS array processor presents several advantages. Not least is the fact that the computation is about 20 times faster at least, on average. In fact, the speed of calculation is also dependent on the VAX, which drives the array processor.

The array processor can only manage one process at a time. Jobs submitted to the FPS wait in a queue for their time to run. The "**apq**" command displays the current state of the queue.

The command that runs CHANT on the array processor is **apch.** It is documented online (try "man apch"). Remember that the **chant** command which runs CHANT on the VAX is equivalent to the following series of commands:

chante **<prepares the parameter data for the synthesis>**
<and outputs an scr.par file.>
calcul **<runs the synthesis>**
out **<outputs a soundfile>**

The equivalent sequence for the array processor becomes:

chante
capap **<computes the parameters for the synthesis>**
apch **<runs the synthesis and outputs a soundfile>**

The **apch** command accepts several options, such as the sampling rate and the duration of the soundfile to be output. A Unix "pipe" can be used between **capap** and **apch,** to avoid the production of intermediate files of data, which slows down the synthesis and may consume a lot of space on the disks.
At this stage, *it is strongly recommended to use an executable file where all the commands are grouped together.* You can copy the following model into a file named **apgo,** for instance, and make it an executable file with the command:
chmod +x apgo
To run CHANT on the array processor you have only to type **apgo** under shell. Here is the model:

```
chante <<%
ap
MY_PARAM



%
capap | apch -R16000 -T7.68 MY_SND_FILE
```

- **ap.par,** that should be placed on your working directory, is a file that contains the parameters controlling the array processor:

```
<FILE AP.PAR>

< quantum >
dsk=50
< carlin: noise in the FOFs; 0: no noise; >=0: scaler on noise amplitude >
carlin=0
< play: noise in the filters; -1: no noise; >=0: scaler on noise amplitude >
play -1.
< apf: external source in the filters >
<     -1: no external source >
<     >=0: scaler on external source amplitude >
apf= -1
< debatt: if -1, no FOFs but an only filter synthesis >
<debatt -1
```

This file must always be used, with a set of parameter values of your choice.

- **MY_PARAM.par** is the parameter file used for the current synthesis (replace it with your own file). You could add more files, one per line.

- **The "<<%" and "%" signs** delimit the input to the **chante** command. This is to avoid typing the parameter file names and parameter values each time you run chante. Between these two signs, the first two lines indicate parameter files. The **three blank lines** are in fact "carriage returns" (remember that chante expects you to type three carriage returns before producing the scr.par file).

- **MY_SND_FILE** should be replaced by the name of the output soundfile.

NOTE:
-1- You can replace "**MY_PARAM**" by "**$1**", and "**MY_SND_FILE**" by "**$2**". The Unix system would replace **$1** with the first argument of the command **apgo,** and **$2** with the second argument, thus making **apgo** a more general program.
-2- The display produced by capap and apch slightly differs from that produced by chant.

CHANT provides a facility for writing code in Fortran subroutines or in C functions. In such subroutines, (the so-called user's subroutines), one can load new data, modify the value of any usual CHANT parameter or internal variable, or even create one's own parameters (the so-called vusers) and establish one's own rules to drive the synthesis. One may not only implement completely new rules: it is possible to interact with the already existing ones in the standard CHANT program, and to override or replace them. Of course, a good knowledge of standard CHANT possibilities is required before implementing user's subroutines.
At run-time, in the FPS array processor version, user's subroutines are called at each quantum. (In the VAX version , the quantum is synchronous to the pitch, thus the user's subroutines are called at each fundamental period). Each time a subroutine is called, parameters may receive algorithmically calculated new values. Between two quanta, CHANT performs a linear interpolation for each parameter's values, and for each sample.

Former users of CHANT have created a library of documented User's subroutines, intended for the modelling of some instruments or for more general purpose, and available for everyone to use. Looking at some of those subroutines may be a good way of learning. We shall describe the library further, and concentrate here on the construction of user's programs.

After having written code using a chosen programming language, the user must create an equivalence table which matches the *mnemonic names* of the new parameters, with their rather *non-mnemonic internal names* (the latter are in fact elements of a big array called vuser, which is itself part of a bigger array named valcha). Mnemonic names should be used when writing the user's subroutines, to make the code easier to read; non-mnemonic ones are to be used in the parameter files where the user's new parameters can be given a value, as any of the standard CHANT parameters.
The next step is to give any new parameter a default value. This must be done in a usual ".par" suffixed file. We can summarize this entire process:

User's subroutine =
code (rules)
+ equivalence table (for the new parameters)
+ default environment (default values for the new parameters).

There are three kinds of user's subroutines:

- pre-user,
- user,
- post-user.

Pre-user subroutine (or **user1**) is called before anything else, particularly, before parameters receive their values from the functions specified in the parameters files, and before the transitions between two notes are calculated. Thus, there is very little that you can do there, except changing temporarily the current time in order to explore functions with a different timing, for instance circularly as in a Music-V oscillator (but all of them will follow the same timing). You could also read files containing data the very first time user1 is called (that is at time 0 of the synthesis), or generate scores by means of some compositional algorithms.

User (or **user2**), is called after parameters are given their new values for the current quantum through the invocation of functions, and after transitions between two notes (or events) are calculated.

Post-user (or **user3**) is called after almost every CHANT algorithm (rule) has been applied to parameters and before signal calculation.

What we shall call here a "**user**" (or sometimes, a user's subroutine), is in fact the reunion of user1, user2 and user3 -- and eventually some additional subroutines or functions --, in a single program. User2 and user3 are the best places in which to write most algorithms.
One can already guess that some knowledge of CHANT standard algorithms and internal variables[5] is required before implementing a user's subroutine. We shall examine these two points first, and then describe how to implement and use user's subroutines.

5. Internal variables are those variables that do not correspond to any usual CHANT parameter; nonetheless, they are used internally by the CHANT program, and one may use them in a user's subroutine. They are opposed to the usual CHANT parameters or "external variables".

2.1 CHANT ALGORITHMS AND USER'S SUBROUTINES

The advanced user may wish to modify some standard CHANT rules, or interact with them. For instance, one may like the vibrato frequency and amplitude to evolve automatically during the span of a note or event. To achieve this, one must know not only which are the variables involved and the name by which they are accessible in a subroutine, but also where the calculation of the standard vibrato takes place relative to the three user's subroutines (user1 user2 and user3). We will next shortly describe the basic CHANT algorithms, in relation to user's subroutines. Their general scheduling relative to user1 user2 and user3 will be summarized afterwards in a figure.

2.1.1 Short description of the basic CHANT algorithms and automatic calculations in relation to user's subroutines.

2.1.1.1 *Evaluation of some of the internal CHANT variables:*

In the present version of CHANT, some CHANT internal variables *are evaluated before execution of user1 subroutine*. Let us just mention the variable **tapres** -- "apres" means *after* in French -- which is the end time of the current note or event (see: CHANT internal variables in section 2.2).
Tapres is calculated using the duration of the current note, which is a standard CHANT parameter. Thus, if you wish to modify the duration of the current note algorithmically in user1, you should also change the value of **tapres.**

2.1.1.2 *Evaluation of the functions for parameters:*

The so-called "immediate functions" by means of which you can specify values for any CHANT standard parameter, as well as for any of the vusers you have created, *are evaluated between user1 and user2 subroutines*. Thus, in user1, those variables or vusers defined as immediate functions have not yet received their values for the current quantum (they still have those from the preceding quantum). Let us repeat that this is the reason why most of your algorithms should be implemented in user2 or in user3.

2.1.1.3 *Vibrato:*

Vibrato is applied to the fundamental frequency of the current note or event *between user2 and user3*, and according to the standard CHANT parameters' vibfreq and vibamp values (as well as their respective random parameters).
It should now be clear that any interaction with the vibrato algorithm should take place in user2. There, you can modify vibamp or vibfreq before the fundamental frequency is affected by the CHANT algorithm.

2.1.1.4 *Cslope:*

The spectrum adjustments linked to cslope algorithms take place *between user2 and user3* subroutines.

2.1.1.5 *Bending:*

The bending algorithm (adjustment of the center frequencies of the two first formants) is executed *between user2 and user3* subroutines.

2.1.1.6 *Automatic corrections of formant amplitudes and bandwidths; Automatic calculation of the complementary formant:*

These automatic adjustments of the spectrum are executed -- only if CHANT parameters **ata atb** and **atc** respectively have a value of one--, *between user2 and user3 subroutines*. When using these automatic calculations, amplitude or bandwidth modifications placed in user2 subroutine would not be effective. In this case, the right place is the user3 subroutine.

2.1.1.7 *Correction:*

The final correction of formant amplitudes takes place *after user3* subroutine.

2.1.2 Position in time of CHANT algorithms in relation to the three users

At each quantum, CHANT executes the following sequence of algorithms:

Sequencing of CHANT rules relative to the three users

```
|    + some internal CHANT parameters are given a value
-------
USER 1
-------
|    + function evaluation: parameters receive their value for the current
|      quantum
|    + transition calculations
|
|
|
|
\/
-------
USER 2
-------
|    + jitter
|    + vibrato of the fundamental frequency
|    + bending
|    + automatic calculations (atb, then ata and atc)
|    + cslope
|    + hollow
|    + tremolo
|    + hoarseness
|
\/
-------
USER 3
-------
|    + correction
|
|
\/
------
SYNTHESIS (on the Vax or on the array processor)
------
```

NOTE:
In the array processor version, the main program for the second pass of CHANT (FPS version) is "**capap.f**" ("man chant" will give you the location of the CHANT sources in the system). A little familiarity with Fortran will enable you to check in this file the exact sequencing of rules and calls of user's subroutines, as well as the exact way a standard rule is implemented.

2.2 CHANT INTERNAL AND EXTERNAL VARIABLES

In order to be able to access CHANT variables, the user only has to declare them as common variables. This means that every fortran subroutine or C function will "know" the address of those variables in the computer memory, and thus will either read or affect their value when required. This point will be clarified later; at present, we just need to know the name and the function of every variable.

We can draw a more precise distinction between the already mentioned categories of variables --external and internal-- available to the user in CHANT.

2.2.1 Variables corresponding to the usual CHANT parameters

The first class of variables includes every usual CHANT parameter (f1, nnote, freq1, freq2, etc...) accessible through the ".par" suffixed parameter files. The reader can refer to the alphabetic list of parameters (see appendix 1).

Notice that there is not a strict correspondence between variable names in parameter files, and variable names in user's subroutines: two simple rules must be respected:

- -1- Every indexed name in a parameter file --indexed name refers to parameter names that end with integers, for example dr1, dr2, etc...--, corresponds to an element of an array in the user's subroutine. The integer is then enclosed in parentheses. For instance, dr1 (the duration of the first "note") must be refered to by dr(1). The following table gives some more examples.

Correspondence between internal and external names of CHANT usual parameters

PARAMETER FILE (external name)	USER'S SUBROUTINE (internal name)
ttr	ttr
freq1	freq(1)
jitt3	jitt(3)
bandf100	bandf(100)

- -2- The other rule is as follows:
  In Fortran, the lengh of variable names cannot exceed 6 characters. For those CHANT parameters whose names are longer, retain only the first six letters in the user's subroutines. Coefamp, for instance, must be abbreviated as coefam.

We next give the complete list of CHANT standard parameters, as they should be accessed from a user's subroutine; for "indexed" variables, the dimensions of the corresponding arrays are figured between parentheses.

list of usual chant parameters available in user's subroutines

ajus(3)	e
amp	envelo
ampl(200)	exa
amplf(200)	exf
apf	f(999)
asif(10)	f0moye
ata	fcomp
atb	fref(3)
atc	freq(200)
atten	freqf(200)
band(200)	hn
bandf(200)	hollow
bref(3)	jitt(3)
bsif(10)	nnote
carlin	nof
coefam	phase(200)
cor	play
cslope	seed
debatt	sex
dga	tdeb
dgf	tfin
tex(200)	tremol
dr(999)	ttr
tjitt(3)	tvala(2)
dsil	tvalf(2)
dsk	user
dur(200)	vala(2)
durf	valf(2)
dvr(200)	vibamp
dvrf	vibfre

NOTE:
Vuser1 to vuser500 are the non-mnemonic names for the user defined variables.
The parameters asif and bsif (1 to 10) are the mnemonic names corresponding to vuser61-70 and vuser81-70, used in the VAX version as the amplitude and the bandwidth of the filters.

2.2.2 Internal variables

Under the second category we find some variables the inexperienced user of CHANT is not concerned with, because they are not accessible from a parameter file, but only from user's subroutines. Some of them correspond to the result of a CHANT algorithm, and are available only in one or two of the three user's subroutines.

Here is a list of all the internal variables available:

init : boolean: 1 before first call of user's subroutines, not 1 after. Useful in user1,user2 and user3 for code which is to be executed only once at the very beginning of the sound.

newnot : boolean: 1 when current quantum corresponds to the beginning of a new note or event (current quantum is the first one after the theoretical -- without quantization error -- begin time of the new note.

inote : Number of the current note or event.

t : (absolute) Current time .

tavant : (absolute) begin time of the current note or event.

tapres : (absolute) end time of the current note or event.

eto : Duration of the current fundamental period expressed in number of samples (ie. in 1/e sec. units, where e stands for the current sampling rate).

rst : Time between the beginning of the current period and the time of the first sample of this current period expressed in number of samples (1/e s. unit).

tmvib : Current Pulsation (omega/2 * PI) for the vibrato frequency expressed in radians/2*PI .[in user3 only].

pp : e/f000 where f000 is the fundamental frequency after application of transition calculation. (e is the sampling rate).[in user2 only].

f000 : fundamental frequency after transition calculations.[in user2 only].

wfreq0 : fundamental frequency after vibrato.[in user3 only].

inof : Number of formants as specified in the parameter file (=nof).

inofp1 : Number of formants (inofp1=nof+1 if atc=1; inofp1=nof if not).[in user3 only].

wfreq(i) : frequency of the i^{th} formant after automatic calculations.[in user3 only].

wampl(i) : amplitude of the i^{th} formant after automatic calculations.[in user3 only].

NOTE:
If you are to read the CHANT program code at some point, you may find some OUT OF USE OR UNIMPORTANT internal variables.
Here is the list of these variables:

chp(i)
deto
dmax
f0
f0moy
fase
fmax (see dvr, in the alphabetic list of parameters)
foncs
ieto
iii
lim
pfreq(i)
saveto
sortir
tti
tto
xmots

The next table distinguishes the most important variables -- the ones you will most probably play with -- according to their availability in the three user's subroutines. Internal names for standard CHANT parameters which have some correspondent internal variables, are enclosed in brackets. For instance the standard parameter **nof** has a corresponding internal variable **inof.**

list of important internal variables according to their availability in the user's subroutines

USERS 1 2 3	USERS 2 3	USER 3
init		
inote		
newnot		
t		
tavant		
tapres		
[nof] inof		
		inofp1
[f(inote)]	f000	wfreq0
[ampl(i)]		wampl(i)
[freq(i)]		wfreq(i)

NOTE:
You must notice that if parameters that are given a value via an immediate function are evaluated at each quantum, constant parameters are not. Thus, if parameter f1 (the fundamental frequency of the first note or event) is given a constant value C in a parameter file, and if you want to add a random number frand to f1 (= C) in your user's subroutine via the statement: f1 = f1 + frand, you will not get what you want: at successive quanta, f1, not being reinitialized, will take the values C+frand(1), C+frand(1)+frand(2), etc... One way to get rid of that would be to use an immediate function in the parameter file:

```
f1 = /i
C  0
C  1 ;
```

At each quantum, between user1 and user2, f1 would take back its constant value, before the execution of your statement (which, by the way, should be placed in user2).

2.3 IMPLEMENTING USER'S SUBROUTINES

Most of the user's subroutines in the library are written in Fortran, for some historical reasons mainly. It is thus worthwhile to read the next section even if you plan to use the C language.

2.3.1 FORTRAN

The present version of CHANT is running Fortran 77 which offers some features resembling the facilities of the structured programming languages (closing endif for an "if then" statement for instance), and uses the f77 compiler on the Vax.

In order to create your own subroutines, you should first copy the model user.f, which you can find on the system (in the users' library) to your working directory under the name of your choice. Your "user" should have the ".f" extension for fortran programs:
rcp igor:/usr/local/doc/chant/model/user.f erik:/MY_DIRECTORY/MY_USER.f
(Upper-cased words indicate what is left to your choice. The "rcp" command copies a file from the VAX-750 (igor) to the VAX-780 (erik).)

2.3.1.1 *User model*

Here follows a commented "user" model (remember that in Fortran, comments are introduced with a "c" at the beginning of a line):

```
c*************************************************************************
      subroutine user1
c     executed before parameters evaluation
c     Place in this user auxiliary files reading,
c     CHANT score generation, and initializations.
c     When using additive synthesis, phases must be non zero.
c*************************************************************************
c     include for common blocks, declarations, and equivalence tables
      include '/usr/local/include/chant/comsyn.f'
      include '/usr/local/include/chant/equiva.f'
c     include '/MY_HOME_DIRECTORY/MY_DIRECTORY/MY_COMUSE.f'
c     include '/MY_HOME_DIRECTORY/MY_DIRECTORY/MY-EQUSE.f'

      if (init .eq. 1.) then
      write(0, *)' user MY_USER '
c     INSERT HERE YOUR OWN CODE FOR INITIALIZATIONS
      end if

c     INSERT HERE THE CODE TO BE EXECUTED AT EACH QUANTUM

      return
      end
c*************************************************************************
      subroutine user2
c     executed after parameter evaluation and transition calculations,
c     before vibrato and automatic calculations.
c*************************************************************************
      include '/usr/local/include/chant/comsyn.f'
      include '/usr/local/include/chant/equiva.f'
c     include '/MY_HOME_DIRECTORY/MY_DIRECTORY/MY_COMUSE.f'
c     include '/MY_HOME_DIRECTORY/MY_DIRECTORY/MY-EQUSE.f'

c     INSERT YOUR CODE HERE

      return
      end
c*************************************************************************
      subroutine user3
c     executed after automatic calculations ata atb atc, before correction
c     and synthesis
c*************************************************************************
      include '/usr/local/include/chant/comsyn.f'
      include '/usr/local/include/chant/equiva.f'
c     include '/MY_HOME_DIRECTORY/MY_DIRECTORY/MY_COMUSE.f'
c     include '/MY_HOME_DIRECTORY/MY_DIRECTORY/MY-EQUSE.f'

c                pi  = 3.14159   this can be useful if using trigonometric
c                pis2 = 1.57079          functions in user3!
c     INSERT YOUR CODE HERE

      return
      end
c*************************************************************************
```

2.3.1.2 *Variables*

Notice the "include" statements at the beginning of each of the three subroutines; their function is to "load" the mentioned files.

/usr/local/include/chant/comsyn.f
The first file (comsyn.f) is for declaring the CHANT variables which should be common to the user's subroutines and to the chant synthesizer. (You can have a look at the file, and notice the Fortran "common" statements). Of course, this file *must be "included"* in your program.

/usr/local/include/chant/equiva.f
In equiva.f, you will find an equivalence table between the mnemonic names of usual CHANT parameters, and their internal representation as a big array named "valcha". Although not obligatory, including this file is strongly recommended, unless you feel like using valcha(4369) instead of bandf(1) for the first formant bandwidth in your own code...

The next two files are optional too, but they will make your work easier. They are the equivalent, for the user's own variables (vusers), of the comsyn.f and equiva.f files for standard CHANT parameters.

MY_COMUSE.f
Here you can declare variables that you want to be common to the three users. Sharing the same variables makes them able to retrieve values calculated in the previous user(s). As an example, you can have a look at **/usr/local/include/chant/comuse.f,** which is used in some users of the library.
You could include the common statements directly in your code, instead of using an "include" file.

MY-EQUSE.f
Following the same syntax (see equiva.f on the system), one can give the already mentioned equivalence table between mnemonic names and the internal ones of every variable one wishes to create.
One can also group declarations and dimensioning for every variable in this file, if necessary.
As an example, an extract of an already existing file follows. The "mnemonic" name of every variable is separated from the internal one by a comma, and both are enclosed in a pair of parentheses. You can use up to 500 vusers.

An extract of equivalence table for user cda

```
c     DECLARATIONS, DIMENSIONING AND EQUIVALENCES FOR USER CDA
c     ********************************************************
      dimension ffr(20),bwr(20),texr(20),ffst(20),fast(20),far(20)
      real npat1,npat2,ngr1,ngr2,npart,limit,inharq
      dimension pat1(10),pat2(10)
      dimension sopra(10)
      equivalence       (ffr(1)                    , vuser(1)),
     .                  (coupur                    , vuser(21)),
     .                  (reject                    , vuser(22)),
     .                  (corect                    , vuser(23)),
     .                  (corr                      , vuser(24)),
     .                  (ampr                      , vuser(25)),
     .                  (per                       , vuser(26)),
     .                  (coc                       , vuser(27)),
     .                  (cocf                      , vuser(28))
      equivalence       (ffst(1)                   , vuser(41)),
     .                  (devfl                     , vuser(101)),
     .                  (alfa                      , vuser(102)),
     .                  (beta                      , vuser(103)),
     .                  (valmax                    , vuser(104)),
     .                  (valmin                    , vuser(105)),
     .                  (daatt                     , vuser(106)),
     .                  (daext                     , vuser(107))
c     etc...
```

Files MY_USER.f, MY_COMUSE.f and MY_EQUSE.f may be placed in any directory. You should add a default environment file in the same directory. It should have the usual ".par" extension for parameter files, and it is a good practice to name it as your user, MY_USER.par. Each of your vusers should be given a default value, via their internal names.

example:

```
 vuser1 = 1
 vuser101 =/i
 1   0
 1   1 ;
```

2.3.1.3 *Coding*

It is not the purpose here to give a Fortran manual. Let us just remind you that variables that are not declared in the previously mentioned files will be local to one subroutine; variables whose names begin with a letter in the range [i-n] will implicitly be integers; they will otherwise be floats. Arrays must be declared.
Vusers should be accessed via their mnemonic names (ffr instead of vuser1, and devfl instead of vuser101 with the previous equivalence table for instance).

You can of course extract some algorithms from other users, and insert them in your

own, instead of starting from scratch.

2.3.1.4 *Compiling*

The next step is to compile your user and link it with the CHANT environment. If you are not familiar with the commands involved, procede as follows:

-1- Type the following commands into a file named makeuser:

```
f77 -Ns700 -Nq300 -Nn500 -c $1.f
echo ld /usr/local/lib/chant/module $1.o /usr/local/lib/chant/newcha.a -lF77 -lI77
-lm -lc -o /MY_DIRECTORY/${2-$1}
ld /usr/local/lib/chant/module $1.o /usr/local/lib/chant/newcha.a -lF77 -lI77 -lm -lc
-o /MY_DIRECTORY/${2-$1}
```

-2- The shell command <**chmod +x makeuser**> makes makeuser an executable file.

-3- To produce the executable code for your user, just type :

<makeuser MY_USER>

under shell.
This will produce -- if your subroutines are correct -- an executable file named MY_USER on the chosen directory. You could give makeuser a second argument, which would become the executable file name.
NOTE:
-1- The users' library (see next chapter) contains the previous model **makeuser**.

2.3.2 C language

If you plan to use the C language for your users, just adapt the different steps described for Fortran to C language programming and compiling. C users not having been used extensively, you may encounter problems; see a local expert in this case.
NOTE:
one of the reasons why C has not been used extensively is that for computing efficiency, all the variables had to be stored in a contiguous zone of memory. In Fortran, all the variables are grouped in a big array called valcha (in fact, the internal names available in the subroutines are just an intermediate level: see the equiva.f file...) In C, variables are elements of a big structure. If C functions are more easily reusable, an elegant way of invoking such elements of the structure was not found. (This could possibly be done with pointers or via define statements (preprocessing)).

2.4 RUNNING YOUR USERS ON THE VAX

Running users' subroutines on the VAX presents little interest. Nonetheless, there is a model in the users' library (see next chapter), for compiling and linking your subroutines to the CHANT environment on the VAX, if necessary. This model is called **make_vaxuser.**

2.5 RUNNING YOUR USERS ON THE FPS ARRAY PROCESSOR

To run a user's subroutine, the procedure to follow differs slightly from the one you already know (see section 1.5, "Running chant on the FPS array processor").
First, the CHANT variable **user** must be set to one in a parameter file (in the default environment file for instance); this will warn the CHANT program that you are intending to use user's subroutines interleaved with CHANT standard algorithms.
Second, the **capap** command must be replaced by your executable user name.

Here is a shellscript model for running your user.

```
chante <<%
MY_USER
ap
MY_PARAM



%
/MY_DIRECTORY/MY_USER | apch MY_SND_FILE
```

MY_USER.par is the default environment parameter file for the user's subroutine.
ap.par contains the usual parameters controlling the array processor (dsk,carlin, play, apf and debatt).
MY_PARAM.par contains the parameters values for the current synthesis.
MY_USER is the executable code for the user's subroutine.
MY_SND_FILE is the output sound-file name.

See "man apch" on the system for the various options of the apch command.

2.6 SOME SIMPLE ALGORITHMS AND USER'S SUBROUTINES

2.6.1 A user for generating clicktracks

Here is simple user's subroutine. User1 just prints out on the terminal the name of the user the very first time it is called (that is when the boolean internal variable **init** has a value of one).
User2 is an empty subroutine
User3 is intended for generating clicks in a simple way, for multitracking synchronization for instance.

```
c          CLICKTRACK USER
c ***********************************************************
c user1 (pre-user)
      subroutine user1
c ***********************************************************
      include "/usr/local/include/chant/comsyn.f"

      if(init.eq.1) write(0,*)'**** CLIC user *****'

      return
      end

c ***********************************************************
c user2 (user)
      subroutine user2
c ***********************************************************
      include "/usr/local/include/chant/comsyn.f"

c empty subroutine

      return
      end

c ***********************************************************
c user3 (postuser)
      subroutine user3
c ***********************************************************
      include '/usr/local/include/chant/comsyn.f'
      include '/usr/local/include/chant/equiva.f'

c CLICK TIMING
      real click(4)
      data click /0.,10.,20.,1000./
      if (init.eq.1) i=1
      wfreq0=1/(click(i+1)-click(i))
      dsk=wfreq0
      eto=e/wfreq0
      i=i+1

      return
      end
c ***********************************************************
```

The way subroutine user3 works is as follows:
Variable click is declared as an array of real numbers, containing four elements for the present case. In the data statement, ocurrence times for the clicks are specified (in seconds). Clicks will be produced at time 0., 10., 20. . Calculation will be ended before time 1000. through the use of parameter dr1 that will be set in the appropriate parameter file to a value slightly superior to the ocurrence time of the last click to be calculated (at time 20).
The fundamental frequency for the synthesizer, wfreq0, is made equal to the interval of time between current and next clicks. Note that any previous value of wfreq0 is erased.

Dsk is made equal to the fundamental frequency, to make the computation faster and to avoid timing errors due to the quantum value.
The internal variable eto must be corrected too, as the new period expressed in number of samples.
At last, a counter indicating the current click number, initially set to one, is incremented.

Here are all the parameters you need to use in the parameter file.

```
<quantum>
dsk 50
<bruit dans les fofs (randi)a 0 pas de bruit>
carlin=0
<si pas de souffle play =-1>
play -1.
<si pas de source exterieure  apf = -1>
apf= -1
<si debatt=-1 pas de fof>
<debatt -1
user 1
atc 0
dga 0
dgf 0
nof 5
nnote 1
dr1 22
tex1 .0001
tex2 .0001
tex3 .0001
tex4 .0001
tex5 .0001
```

This will produce three clicks, with very short attack times. The corresponding spectrum has five formants.

2.6.2 Scalers and random variations of CHANT parameters

Scalers on CHANT variables enable the user to have as many levels of control as desired. One could for instance add in user2 a control over coefamp with a scaler named coes:

coefamp = coefamp * coes

Random variations are often used in the CHANT program and in user's subroutines. The idea is to add or substract a random proportion of the chosen variable to the central value of this variable (as given in a parameter file for instance). A random scaler provides a control over the maximum excursion; this scaler can express -- as a ratio of the central value --, either the maximum excursion ABOVE and UNDER the central value -- that is, half the maximum "total excursion" or ambitus --, or twice this maximum excursion -- that is the ambitus of the variation.
The formulas :

variable = variable * (1 + 2*(ran(l)-.5) * random_scaler)
variable = variable * (1 + (ran(l)-.5) * random_scaler)
are quite general, and correspond respectively to the two possibilities we have just mentioned. Here are some comments:
the function **ran** produces random numbers between 0 and 1, in Fortran;
ran(l)−0.5 produces a number between −.5 and +.5 ;
the excursion above or under the central value is:

variable * random_scaler * (ran(l)−.5)

for the second formula;
the "random-scaler" should be a variable declared in the equivalence table, with a mnemonic name of your choice and an internal name "vuseri", where i is an integer (not already used in the equivalence table, and in the range from 1 to 500).

2.6.3 Iteration and general scalers: scaling filters' amplitudes with bramp

Iteration on formants or filters is often useful, particularly in the case where you want to apply the same "general" scaler to every formant or filter parameter.
In the next example, amplitudes of each of the "nof" filters are scaled with the scaler bramp.

```
c-------------------------------------
c filter amplitudes
    do 3 i=1,nof
    amplf(i) = wampl(i) * bramp
3    continue
c-------------------------------------
```

Note that this algorithm must be placed in user3. In fact, it uses the CHANT internal variable wampl(i), which is the amplitude of the i^{th} formant after automatic amplitude calculation (that takes place between user2 and user3). In user2, the correct version should use ampl(i) instead of wampl(i).

2.6.4 Correlation between the filters' amplitude and coefamp

Suppose you have created the previous general scaler called bramp, and you want the filters' amplitude to be related that of the FOFs, that is, to coefamp.
A single vuser brq and the following code (placed in user2) allows that.

```
c ***********************************************************
c bramp correlation with coefam
    if (brq .gt. 0.) then
    bramp=bramp*((1-brq)+(brq*coefam))
    end if
c ***********************************************************
```

Vuser brq is a flag and a scaler at the same time:
if zero, there is no correlation and the algorithm is skipped.

if greater than zero, (and implicitely lower than one), brq is taken somewhat like a correlation coefficient. The greater it is, the greater the influence of coefamp over bramp will be.

2.6.5 Low frequency modulation on coefamp

Low frequency modulation on CHANT standard parameters has been used extensively in user's subroutines. In the present case, the principle consists in modulating the value of coefamp with a sinusoid. Amplitude and frequency of the modulation should be vusers so they could be changed at will from a parameter file. In addition, random variations could possibly be applied to the previous two parameters.
This examplifies some typical ways of using vusers, namely:

- use of flags,
- use of random scalers,
- modification of the quantum.

The variables involved are described in the next table.

Variables for low frequency modulation on coefamp			
Mnemonic name	Internal name	default value	Function
coemfl	vuser111	-1.	flag for low frequency modulation on coefamp
coema	vuser112	.2	amplitude of low frequency modulation on coefamp
coemf	vuser113	5.	period of low frequency modulation on coefamp
coemr	vuser354	0.	flag for random on modulation
coemar	vuser352	0.	random on amplitude of low frequency modulation on coefamp
coemfr	vuser353	0.	random on frequency of modulation on coefamp

The flag variable **coemfl** may take several values, with different effects:
-1: no modulation, the algorithm is skipped;
0 : sinus modulation;
1 : cosinus modulation;

Random parameters **coemar** and **coemfr** are scalers on the central values **coema** and **coemf,** and they express ,as a proportion, twice the maximum possible excursion under and over the central values -- that is the maximum "total excursion", or ambitus.

The algorithm uses some internal local variables (which need not be declared as vusers):
temf: current period of modulation,

tema: current amplitude of modulation,
tzero: begin time of current period,
tlocal: local time ie. time elapsed from tzero. This is to avoid discontinuities in phase values.
coemod: final value of scaler to be applied to coefamp current value.
pi: Pi constant; it must be initialized somewhere else in the program:
(pi = 3.14159 .)

The algorithm follows:

```
c ***********************************************************
c coefam low-frequency modulation
      if ( coemfl .gt. -1) then

c  if random on modulation parameters is used:
      if (coemr.gt.0) then

      if (init .eq. 1) then
      temf=0
      tzero=0
      tlocal=0
      endif

      if (tlocal .ge. temf) then
      tzero=tzero+temf
      temf=coemf * ( 1 + (ran(l)-.5)*coemfr )
      tema=coema * ( 1 + (ran(l)-.5)*coemar )
      if (coemr.eq.1) dsk=5/temf
      endif
      tlocal=t-tzero

c  no random on modulation parameters
      else
      temf=coemf
      tema=coema
      tlocal=t
      endif

c  modulation algorithm
      if ( coemfl .eq. 0 ) then
      coemod=1+tema*sin(2*pi/temf*tlocal)
      coefam= coefam*coemod

      else if ( coemfl .eq. 1 ) then
      coemod=1+tema*cos(2*pi/temf*tlocal)
      coefam= coefam*coemod
      end if

      endif

c ***********************************************************
```

If coefamp is set to a constant value of one, the default values of the vusers will not alter the output amplitude. If coemfl is set to zero, amplitude will describe a sinusoid every five seconds, starting from 0.8 (ie. 1 - 0.2), reaching a maximum value of one 2.5 seconds later.

The use of random raises two problems:

- to ensure phase continuity, a local time must be used at each period;
- the quantum must be a sub-multiple of the current period.

If **coemr** is set to zero, random will not be used. Giving **coemr** a value of one will divide every (random) period into five "quanta"; this is to be used with "high" frequencies of "low" modulation. A value of two leaves the quantum unchanged, which is acceptable if the quantum is very small with respect to the period.

2.6.6 Attenuation algorithm

The attenuation algorithm presented here has a function which is similar to that of the **cslope algorithm** of the standard CHANT environment: it enables one to simulate a vocal or instrumental effort by controlling the amplitude of higher formants. The idea is that of a low-pass control over the spectrum, with a variable spectral slope. The control is more powerful than that of the cslope algorithm.

The basic algorithm works as follows:

The user can specify two frequencies:
coupur (low frequency for the attenuation algorithm),
and **reject** (high frequency for the attenuation algorithm).

The amplitude of formants whose center frequencies are lower than **coupur** are not modified by the algorithm.
Those whose center frequencies are higher than **reject** are affected in a simple way; their amplitudes are scaled in the user3 subroutine by the value of **corect.**
The amplitudes of those formants whose center frequencies are comprised between **coupur** and **reject** are scaled by a factor depending on the center frequency, and on **coupur, reject** and **corect.** The next figure shows the amplitude scaling factor for attenuation, as a function of formant frequency, for given values of **coupur, reject,** and **corect.** Roughly, the scaling factor describes half a sinusoid between **coupur** and **reject.**

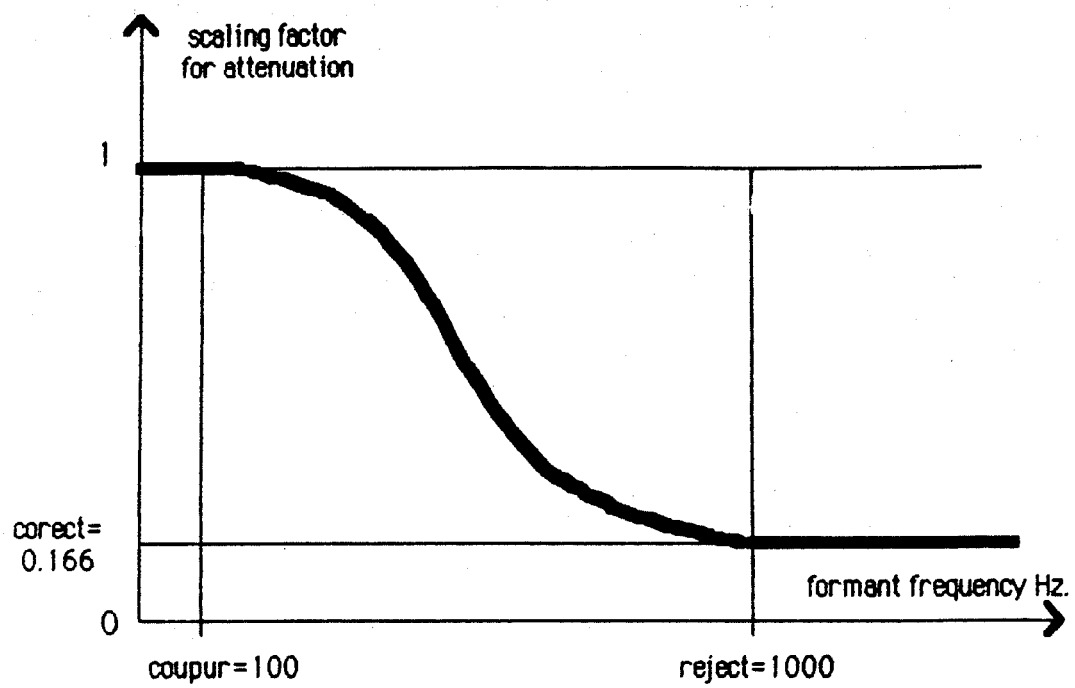


Figure 13. Amplitude scaler for attenuation, as a function of formant frequency.

A lot of rules can be added to this basic algorithm: random variation and low frequency modulation on **corect** value, **corect** correlation with coefamp, pitch dependent weighting of the attenuation, etc...
The code reproduced below is the simplest version of the algorithm; additional features can be found in the "user" **cda.f** in the library (see next chapter).

Variables for low-pass control over the spectrum			
Mnemonic name	Internal name	default value	Function
coupur	vuser21	50.	low frequency of attenuation
reject	vuser22	1000.	high frequency of attenuation
corect	vuser23	1.	formant amplitude scaler for attenuation of higher formants

```
c ***********************************************************
c ATTENUATION ALGORITHM (in user3 subroutine)
c ***********************************************************
c protection against negative values for corect
      rr = max(.001,corect)

c attenuation
      do 50 i=1,inofpl
      if ( wfreq(i) .le. coupur) then
      arg=1.
      else if (wfreq(i) .ge. reject) then
      arg=rr
      else
      arg = log(rr) *
     .      (1. + sin(3.1416 *(wfreq(i)-(coupur+reject)/2.)/
     .                        (reject-coupur))) / 2.
      arg=exp(arg)
      end if
      wampl(i) = wampl(i) * arg
50      continue
c ***********************************************************
```

NOTE:
the previous algorithms were extracted from user cda.f, which can be found in the user's library online, along with some useful commented algorithms ("Portamento", "Vibrato evolving during the span of a note or event", for instance).

2.7 INTERNAL SUBROUTINE ENV.F

There is a useful internal CHANT subroutine for generating envelopes over any vuser or standard CHANT parameter. You can find it in the user's library.

To be used, env.f must be compiled, then linked with your user's subroutine. This is done automatically when compiling a user's subroutine. In your own subroutine, to generate an envelope over a parameter X, just call env in the following way:

X = env (dstac, dext, datt, rappor, ampini, ampfin, offset)

Arguments dstac, dext, etc..., have the following functions:

dstac : Ratio between durations of the envelope and the whole note.
If 1, the envelope duration is equal to the note duration. If .5, the envelope lasts half the duration of the note, and the parameter keeps a constant value for the other half. This enables "staccati".

datt : Attack duration as a ratio of the total event duration.

dext : Decay duration as a ratio of the total event duration.

rappor : Ratio of the target value to be reached at the end of the attack.

If one, the maximum value is reached at the end of the attack;
if .7, for instance, .3 of the total motion is left to be fullfilled.
The reason for this is that attack, steady state and decay are realized with portions of sinusoids, so that the "steady" state is not really steady, which appeared to be convenient because being nearer to a natural model.

offset : Target value around which the motion is happening.

ampini : Distance from the starting point of the deviation towards the offset.

ampfin : Distance from offset towards the ending point of the deviation.

For example,
Suppose you want an envelope starting from 4, going down to 2, and then staying at 2:
offset = 2
ampini = −2 (the minus sign because the amplitude decreases during the attack)
ampfin = 0

Another example:
offset = .5
ampini = .5

ampfin = .5
rappor = .7
dstac = 1
If applied to the parameter coefamp, these values will lead to an envelope:
- starting from 0 (offset–ampini = .5 – .5), reaching .7 of the maximum amplitude (as given by parameters amp, coefamp and envelo) at the end of the attack;
- increasing and then decreasing between this point and the maximum value during the "steady" state (which is in fact a portion of a sinusoid);
- and decreasing from .7 of the maximum value to 0 during the decay.

2.8 USING PHONEME DICTIONARIES: USER MODEL PHON.F

CHANT allows you to associate a name to some given formant frequencies, thus defining *"phonemes"*. Such phonemes can be grouped into *"phoneme dictionaries"* which are ordinary ascii files with a "**.pho**" extension.
Instead of specifying formant frequency parameters with immediate functions, you can create a "phoneme file" containing a list of user defined "phonemes" which refer to the dictionary. An automatic spectral interpolation between phonemes is then performed at run-time.
For doing this, you should insert some code in your own user. (Refer to section 3.3).

NOTE:
When using the standard CHANT program, you can call --by setting the variable **user** to 1 -- *a default user's subroutine*, which enables you to use phonemes.

CHAPTER III - THE USER'S LIBRARY

User's subroutine models and user's subroutines have been stored on the system in a "library". Every user of CHANT can benefit from the knowledge acquired at Ircam on the modeling of different types of voices or instruments, and can take advantage of some useful algorithms allowing global control over the dynamics of the spectrum.
This chapter is not exhaustive, as new "users" can be added to the library, and as old ones will be added when updated. You will find some explanations on the use of the library in the file **README**, and a commented index of the library contents in the file **INDEX**.

3.1 HOW TO USE THE LIBRARY

You can find the library under the following directory:

/usr/local/doc/chant/model

which can be accessed from the VAX-750

Users' subroutines may be copied into the directory of your choice on the VAX-780; you can extract some algorithms to build new "users".
You can also run a user's subroutine as it is, compiling and linking it to the CHANT program on the VAX-780 (see chapter II).
Remember that almost every "user" has a corresponding parameter file which contains default values.

You can add new "users" to the library. They should be clearly commented. If you find some bugs in an already existing "user", please report them to the CHANT and FORMES group.

3.2 USER'S SUBROUTINE MODEL: USER.F

"**User.f**" is a commented model for Fortran user's subroutines (see chapter II).

3.3 USER MODEL FOR USING PHONEME DICTIONARIES: PHON.F

This model contains the code for running your user's subroutine with "**phoneme dictionaries**".

To use this feature, you need:

- a **phon.f** like user's subroutine and its default parameter file (a **phon.par** like file),
- a **phoneme dictionary,**
- and a "**spectral interpolation phrase file**".

When running CHANT, you can call a default "user's subroutine" which is intended for using phoneme dictionaries. The model "**phon.f**" is to be used if you intend to use phonemes along with your own rules; it can be included in your own "user", or you can start adding rules in a copy of "phon.f". Some comments on the code (which can be found in the appendices):
/usr/local/include/chant/params.f must be included in user1. This file contains some declarations.
The code is divided between user1 and user2 subroutines, and needs some functions and subroutines which can be found after user3, or in "/usr/local/lib/chant".
Note that the parameters **coefamp** and **envelo** must be given their value *using an immediate function:* these parameters are modified in the "user", and they must retrieve the user defined value at each quantum through function evaluation. The **phon.par** file sets **coefamp** and **envelo** to 1 (using immediate functions); in addition, it sets the parameter **user** to 1, so that the "user" is called at each quantum.

Phoneme dictionaries contain lists of "phoneme" definitions. A phoneme definition is as follows:

PHONEME-NAME
ENVELO-VALUE **COEFAMP-VALUE**
FREQ1 **FREQ2** **FREQ3 ..... FREQ9**
<BLANK-LINE>

Here is an example:

< The first phoneme, QUON in the present case, is a default one >

```
QON
1          1
700        1500    3050    3950    4900    6150    7400    8750

A
1          1
600        1000    2450    2700    3240

A2
1          1
450        950     2200    2800    3500

E
1          1
400        1700    2300    2900    3400

SILENCE
0          0
500        1000    2000    3000    4000
```

You can associate amplitude values of your choice to a given phoneme, through **envelo** and **coefamp** values in the phonem definition.
The *first phoneme* in a dictionary always defines a "default phoneme", for which you can specify *up to ten default formant frequencies*. For the previous example, for instance, one could set the parameter **nof** to 8, although the phonemes in the dictionary only have five formants. Formants 6, 7 and 8 would automatically take the default frequency values 6150, 7400, and 8750 respectively. This enables the user to define phonemes with different numbers of formants, and use them in the same synthesis.
Dictionaries should have the "**.pho**" **extension. The default phoneme dictionary is dico.pho** which can be found in the library, along with other phoneme dictionaries.

In the "**phrase file**" for spectral interpolation, which should also end with the "**.pho**" **extension,** each line must respect the following syntax:

DURATION-of-INTERPOLATION PHONEME-NAME

For instance, the following phoneme file:

```
0.1     A
0.9     E
2.      E
0.1     A2
1.      E
```

describes the following spectral interpolations:

from time zero to 0.1 second:	phoneme A
from 0.1 to 1:	a 0.9 sec. long transition from A to E.
from 1. to 3:	a 2 second long "steady state" on phoneme E.
from 3 to 3.1:	a quick transition from E to A2.
etc ...	

Note that CHANT does not scale the durations given in a phoneme file, and that the total duration of the specified interpolations should be ***at least as long*** as the corresponding parameter file's duration.

When running your "user", CHANT will ask you for the phoneme dictionary and the phoneme file(s) to be used. If using a command file instead of the interactive mode, use the following model:

```
chante <<%
MY_USER
ap
MY_PARAM



%
cat > foo <<%
MY-DICTIONARY
MY-PHONEMFILE
%
/MY_DIRECTORY/MY_USER < foo | apch MY_SND_FILE
rm foo
```

MY_USER.par is the default parameter file associated to your "user". It should contain the default parameter file for phon.f, which is **/usr/local/doc/chant/model/phon.par.** (on the VAX-750).

3.4 USER CYM.F

This user is oriented towards cymbal simulation. Nonetheless, the rules it contains allow derivations from the original model, towards non-intrumental like sounds, and the algorithm can be added to some other "users".

In short, the parameter **envelo** is modified in user2 subroutine, as follows:
For each note, envelo increases during **cymneq** seconds, from zero to its value as specified by the user in the parameter file.
It then decreases exponentially until it reaches the value **cymlev.**
Then, envelo keeps cymlev value until the end of the current note is reached.

In addition, random variations are applied to coefamp and to the fundamental frequency. The algorithm is executed only if the current time of the synthesis is greater than **cymb** (beginning time for the cymbal algorithm), and lower than **cyme** (end time for the cymbal algorithm).

In user3 subroutine, one can modify the formant bandwidths with the general scaler **bws;** one can also randomly change every formant frequency (random scaler **ffrs**), which is an important feature in the "cymbal effect". At last, one can "stretch" the formants, multiplying every formant center frequency with a same scaler **(sts).**

Here is the code of the "user" cym.f, as it can be found in the library:

```
c******* mnemonic names and meaning of the vusers *******
c cf cym.par for default values, and cym_comuse.f for
c the equivalence table
c*****************************************************
cymb  beginning time for entering in the cymbal algorithm
cyme  end time (for exiting the cymbal algorithm)
cymneq  quantum for reinitializing envelo value
cymlev  value for envelo for the end of the note or event
c sts   stretching general scaler (on formant frequencies)
c ffrs  formant frequency random (general) scaler
c bws   (formant) bandwidth (general) scaler
c*****************************************************************
      subroutine user1
c*****************************************************************
      include '/usr/local/include/chant/equiva.f'
      include '/usr/local/include/chant/comsyn.f'

c THE EQUIVALENCE TABLE FOR THE CYMBAL VARIABLES:
      include '/usr/local/include/chant/cym_equse.f'
c (IF NOT PRESENT IN /usr/local/include/chant, MAKE A COPY FROM
c /usr/local/doc/chant/model INTO YOUR OWN DIRECTORY);
c NOTE THAT THE EQUIVALENCE TABLE CAN BE INSERTED IN YOUR
c OWN CODE.

      if ( init .eq. 1. ) then
      print *,'               subroutine for cymbal  '
      end if

      return
      end
c*****************************************************************
      subroutine user2
c*****************************************************************
      include '/usr/local/include/chant/equiva.f'
      include '/usr/local/include/chant/comsyn.f'
c     The equivalence table for the cymbal variables:
      include '/usr/local/include/chant/cym_equse.f'

c *** cymbal algorithm ***
      if ( (t .gt. cymb) .and. ( t .lt. cyme) ) then

      if (al2 .eq. 0) then
      al2=1
      al4=1
      tal2=0.
      end if

      if ( t .ge. tal2) then
      al1=al2
      al3=al4
      temp=(ran(l)**2)
      al2=temp*2+1
      al4=temp+1
```

```
      tal1=t
      tal2=t+.05
      endif

      f000=f000*(al1+(al2-al1)*(t-tal1)/.05)
      coefam=coefam*(al3+(al4-al3)*(t-tal1)/.05)

      if ( (t-tavant) .lt. cymneq) then
      envelo = envelo * (t-tavant)/cymneq
      else
      arg = (t-(tavant+cymneq))* 7.4
      if (arg .lt. 50) then
      envelo = envelo * ( exp (-arg)+ cymlev)
      else
      envelo = cymlev
      end if
      end if

      end if
c *** end of cymbal algorithm ***

      return
      end
c*********************************************************************
      subroutine user3
c*********************************************************************
      include '/usr/local/include/chant/equiva.f'
      include '/usr/local/include/chant/comsyn.f'
c     The equivalence table for the cymbal variables:
      include '/usr/local/include/chant/cym_equse.f'

c** random on formant frequencies, stretching, and bandwidth scaler **
      do 100 i=1,inofp1
      wfreq(i) = wfreq(i)*sts*(1+(ran(1)-.5)*ffrs)
      band(i) = band(i)*bws
100    continue

      return
      end
c*********************************************************************
```

NOTE:
This version cannot be used with phoneme dictionaries; if you want to do this, just insert the include statements and the code in a copy of **phon.f** .

3.5 USER ADDIT.F

This is a basic user which allows a kind of additive synthesis in CHANT:
Formant bandwidths are set to 0;
tex debatt and **atten** are made equal to the fundamental period;
phase (1 to inofp1) are set to 1, to ensure phase continuity.
With such values, to each formant corresponds a sinusoid with a frequency equal to that of the formant.

Automatic calculations, which in fact are intended for simulating a singing voice, are skipped (ata=atb=atc=0). Some CHANT algorithms (vibrato, cslope, etc...) may have an inapropriate effect too, when using additive synthesis, and the corresponding parameters are set in the default parameter file so that the algorithms are ineffective (vibamp=0, cslope=1...).

One can derive from this model, and play with the parameters phase1-200 or with FOF superpositions for instance.

The default parameter values can be found in the library (file addit.par); the code is reproduced below (note that this user only uses standard CHANT variables).

```
c*********************************************************************
      subroutine user1
c*********************************************************************
      include '/usr/local/include/chant/equiva.f'
      include '/usr/local/include/chant/comsyn.f'

      if ( init .eq. 1) then
      print *,'    subroutine for additive synthesis '
      end if

      return
      end
c*********************************************************************
      subroutine user2
c*********************************************************************
      include '/usr/local/include/chant/equiva.f'
      include '/usr/local/include/chant/comsyn.f'

c to avoid a pulsation at the rate of fundamental
      if ( init .eq. 1) then
      debatt=1./f000
      atten=1./f000

      do 100 i=1,inof,1
      phase(i)=1.
      band(i)=0.
      tex(i)=atten
100       continue

      ata=0
      atb=0
      atc=0
      vibamp=.000000
      dga=0.
      dgf=0.

      end if

      return
      end
c*********************************************************************
      subroutine user3
c*********************************************************************
      include '/usr/local/include/chant/equiva.f'
      include '/usr/local/include/chant/comsyn.f'

      return
      end
c*********************************************************************
```

3.6 OTHER USERS

In addition to the simple models, the library includes some complex users that contain specialized rules. As an example, let us mention the following programs:

3.6.1 user qon.f

Qon.f is a user written by Yves Potard, and intended for simulating classical soprano voices. "Qon.f" stands for "Queen Of the Night", because it was used to synthesize the famous Mozart aria, as one of the first tests of the CHANT program.

3.6.2 user cda.f

Cda.f is a "user" which contains many algorithms, either extracted from other subroutines, or original ones. It is oriented towards the use of "synthetic instruments" derived from abstract models, and allows various global controls over the spectrum (stretching of the formant frequencies for instance). In addition, one can control the CHANT filters in a very precise way (harmonic or inharmonic series, random variation of the filter frequencies, modulation of the filter amplitudes, bandpass control, phoneme spectral envelopes on the filters, etc...).
This "user" was used to produce the tape for Gerard Grisey's piece "Les Chants De l'Amour", hence its name.

CONCLUSION: CHANT AND FORMES - CHANT AND THE 4X

"FORMES is an interactive system designed for Music Composition and Synthesis. It includes an interactive object-oriented (or, better, "process-oriented") programming language, and libraries of "ready-to-use" examples, algorithms and programs. Its most remarquable orientation is **Composition and Scheduling of Processes.** *FORMES can be used for the control of a sound synthesizer or for score elaboration, or other applications."* ["FORMES Beginner's Guide" - Xavier Rodet - Ircam].

For our concern, FORMES is intended as a general interactive environment for driving the CHANT synthesizer. As such, it makes the use of "users' subroutines" an obsolete practice. Users' subroutines should be replaced with FORMES functions, which allow a much more flexible environment and a more powerful control over the synthesis. However, this manual should be helpful for understanding the CHANT synthesizer, its parameters and standard algorithms. In addition, it should enable you to translate any user's subroutine into FORMES functions.

A version of CHANT should soon be implemented on the 4X, the appropriate patch of the machine allowing a real-time control over several CHANT parameters. The excitation should be simulated with wave tables, and filters should be implemented.

SELECTED BIBLIOGRAPHY

[1] J.B. Barrière, *Chreode-I: chemin vers une nouvelle musique avec l'ordinateur*, (august 1984).

[2] G. Bennett, *Singing Synthesis in Electronic Music*, in "Research Aspects of Singing", ed. J. Sundberg, Publication 33. Stockholm: Royal Swedish Academy of Music, pp. 34-50.

[3] P. Cointe, *Manuel FORMES*, Ircam (december 1982).

[4] P. Cointe, *FORMES par l'Exemple*, Ircam (1984).

[5] X. Rodet, Y. Potard, J.B. Barrière, *The CHANT Project: From the Synthesis of the Singing Voice to Synthesis in General*, Computer Music Journal 8 (3) MIT Press, (1984).

[6] X.Rodet, *Time-domain Formant Wave Function Synthesis*, Proc. ASI-NATO, Bonas, France, (July 1979).
(also in Computer Music Journal 8 (3)).

[7] X. Rodet, *FORMES Beginner's Guide*, Ircam (September 1984).

[8] Y. Potard, *Manuel d'Utilisation de l'Array processeur CHANT avec FORMES*, (not yet printed, a first version by X. Rodet is available).

[9] X. Rodet, G. Bennett, *Synthèse de la Voix Chantée par Ordinateur*, Conférences des Journées d'Etudes, Festival International du Son, Paris (1980).

[10] X. Rodet, P. Cointe, *Formes: Composition and Scheduling of Processes*, Computer Music Journal 8 (3) MIT Press, (1984).

[11] J.Sundberg, *The Acoustics of the Singing Voice*, Scientific American (March 1977).

Dicography:

J.B. Barrière, ed. 1983. *Ircam: un portrait-recherche et création.*

appendix 1: CHANT PARAMETERS LISTED IN ALPHABETIC ORDER

ata Stands for "AuTomatic Amplitudes". It is a flag.
If set to 1, formant amplitudes are calculated automatically and then scaled by the respective values of **ampl1,2,3** etc... , which allows you to take into account the particular shape of an individual glottal-source spectrum. If you have no special idea about a certain individual voice , set **ata** and **ampl1,2,3...** to 1.
If set to 0, formant amplitudes are given by the parameters **ampl1,2,3,...** (see below). Default for **ata** is 1 (Automatic).

atb Stands for "AuTomatic Bandwidths". It is a flag. If set to 1, formant bandwidths are calculated automatically. If set to 0, formant bandwidths are specified by **band1,2,3,...** (see below). Default is 1 (Automatic).

atc Stands for "AuTomatic Complement". It is a flag. If set to 1, the amplitude of the "complement", a resonance area centered at the frequency given by **fcomp** and used to boost the region of the spectrum below the first formant, is calculated automatically. Default is 1 (Automatic). In the FPS version, it is also used as a scaler on the complement amplitude, when positive.

ajus1 Amount of change in amplitude of "fundamental" (in the VAX version only, see **apf**) and first formant (FPS and VAX versions) in relation to fundamental frequency. 0 means no change; .25 is a good value. Effective only when cslope is greater than 0. (see section 1.3 on automatic calculations and algorithms).

ajus2 When clsope is positive, controls the shoulders of the curve effecting change in high formant amplitudes in relation to fundamental frequency. 5.7 is a good value. (see section 1.3 on automatic calculations and algorithms).

ajus3 Amount of change in amplitude of formants in relation to fundamental frequency and dynamic amplitude. 0 means no change; .5 is a good value. (see section 1.3 on automatic calculations and algorithms).

amp Specifies the absolute amplitude of the note or phrase. (The output signal is scaled at the very end of the synthesis so that its maximum amplitude is equal to amp.) Values range between 0 and 1 (the default value is 1). **Amp** can only be a constant value and NEVER a function overwise you'll get into troubles. If you want an envelope over the sound, well, we have a nice parameter called **envelo** to do that (exactly in the classical meaning of the potentiometer). And **amp** is there only to specify the overall

loudness of the sound. The loudness of a sound can also be altered dynamically during a note or phrase by a function applied to **coefamp** (but see the exact effect of that parameter which modifies also the spectrum). The following values for **amp**, with an automatic calculation of the spectrum, gave the impression of equally spaced dynamic gradations. They are listed with their dB and musical notation equivalents.

amp value	**db**	**musical notation**
1	0	ff
0.708	-3	f
0.501	-6	mf
0.355	-9	mp
0.2	-14	p
0.12	-19	molto p
0.063	-24	pp
0.035	-29	ppp

The range 0 to -29 db (**amp** values 1 to .035) was found to be a realistic dynamic range for a human voice at constant distance from the listener.

ampl1,2,3...200 The amplitudes of each formant area. These values are the real amplitudes of the formant areas if **ata** is set to 0 (in fact, only the proportions between the amplitudes are significant: if you multiply all the **ampl1, 2 etc...** by the same factor, it will make no change). If **ata** is flagged to 1 the amplitudes are first calculated automatically, and then scaled by the respective values of **ampl1,2,3,etc...**, which allows you to take into account the particular shape of an individual glottal-source spectrum. If **ata=1**, and if you have no special idea about a certain individual voice, set **ampl1,2,3,etc...** to 1 (which is the default value).

amplf1,2,...200 In the FPS version:
The amplitudes of CHANT filters. Note that the filter output amplitude also depends on **apf** or **play** values (that is the amplitude of the filter's input), and on **amp** and **coefamp** if FOFS are used in parallel with the filters.
For the VAX version, see **vuser.**

apf In the VAX version:
stands for "AmPlitude of Fundamental". It is the amplitude of a sine wave of frequency equal to the fundamental frequency of the note, added to the sound to provide a boost for the fundamental partial (useful for certain rare cases like special female voices or voices with much chest resonance). The attack and decay times of this sine wave are given by **dur0** and **dvr0** respectively, which, like **dur1,2,3,...** and **dvr1,2,3...** are scaled by **dga** and **dgf.**

In the FPS version:
controls the external source input to the filters.
If -1, filters are not to be used with an external sound source (that is, an already existing soundfile).
If positive or null, **apf** is a scaler on the input amplitude of the external source sent into the filters.

atten A rarely changed parameter, it specifies the attenuation time of the pitch-synchronous impulse used to generate the sound. In other words, **atten** controls the duration of the final damping of the LOCAL ENVELOPE which is applied to each of the sinusoidal impulses. Because an exponential decay never reaches zero, and you wish that the computer ends its calculation before eternity, the program uses an additional damping that begins at time **debatt** seconds after the beginning of the excitation and is **atten** seconds long, which means that the envelope necessarily reaches zero after time **debatt+atten.** This has nearly no influence on the spectrum provided that **debatt+atten** is large enough. Otherwise, it mainly causes a broadening of formant bandwidths. The default value is .007 (7 milliseconds). When computing very high notes (above C5) **atten** can be reduced to as low as .002, with a significantly faster compute time resulting. Reductions in **atten** tend to blur the vowel definition, but this is not a problem at high fundamental frequencies.

band1,2,3...200 The bandwidth in Hertz at -6db for each formant region. The values are ignored if **atb** is flagged, in which case the bandwidths are calculated automatically, and the parameter output file will contain the values calculated for the last fundamental frequency of the note or phrase.

bandf1,...200 The bandwidth in Hertz at -6dB of the filters; for the FPS version only. (For the VAX version, see **vuser**).

bref1,2,3 are the reference bandwidths used to model the resonance characteristics of the human vocal tract. They give bandwidths for formant areas at the center frequencies defined by **fref1,2,and 3,** and are employed for the automatic calculation of formant bandwidths when the parameter **atb** is flagged (see "automatic calculations", in section 1.3). Default values are **bref1=38, bref2=30, bref3=75.** This means that in automatic calculation, formants at frequencies **fref1,2 and 3** are respectively attributed bandwidths of 38, 30 and 75 Hertz, and others the value given by a cubic interpolation between those points.

carlin In the array processor version:
this is a flag and a scaler for the amplitude of noise in the FOFs.
If 0: no noise

If greater than zero: noise with an amplitude scaled with **carlin** value.
In the VAX version:
Controls whether or not (**carlin** greater than 0 and **carlin=0,** respectively) a sound file will be produced after the synthesis. If not, intermediary data files are not removed.

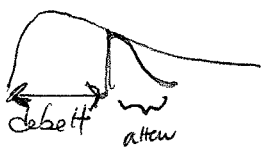
coefamp It is used to make dynamic changes in the course of a note or event or phrase relative to a maximum amplitude set by **amp.** Values in **amp** are scalers for **coefamp** -- a value of 1 results in the full value of amplitudes being employed, .5 results in amplitudes being cut to half, etc... A dynamic function can be defined as a relative change, for example a crescendo of 12db, and then be applied to a variety of target amplitudes. A 12 db crescendo applied to an **amp** of 1 will give a crescendo roughly from mf to ff, applied to an **amp** of .25 it will give a crescendo from p to mf. The default value is 1, meaning no change in **amp. Coefamp** modifies the spectrum through the "cslope" algorithm (see "automatic calculations" in section 1.3).

cor If set to 0, it avoids a CORrection of each formant amplitude according to **tex** and **band.** Leave it to 1 unless you know what you are doing.
Cor is also used as a scaler on the "bending" correction of the two first formants.

cslope is a scaler controlling the amplitude of the formants which have a higher center frequency than that of formant 1, in relation to the amplitude of the first formant and that of the "fundamental" (the latest in the VAX version only; see **apf**). A value of 1 produces relative amplitudes as defined either by the automatic adjustment of formant amplitudes, or the specified values of **ampl1,2,3...** (see ampl). A value of 2 doubles the high formant amplitude relative to the first formant (and "fundamental" in the VAX version), a value of .5 halves them. A negative value for **cslope** acts as a flag to implement the automatic adjustment of the high formant amplitudes according to the loudness of the note (**coefamp**) and its fundamental frequency. This automatic link permits the modeling of vocal effort through the parameter **coefamp.** In that case the exact effect of **amp** and **coefamp** and fundamental frequency is described in the manual (see section 1.3, automatic calculations). The default value is -1 (flagged for automatic adjustment).

debatt The time between the beginning of the LOCAL ENVELOPE (which is applied to each impulse) and the beginning of its attenuation (which is **atten** seconds long). Default value is .01 (10 milliseconds). This parameter is rarely changed (see atten). Like

épate le sommet d'un formant



atten, it can be reduced to as low as .002 when computing sounds with fundamentals higher than C5. Reductions in **debatt** also tend to blur vowel definition, which is again not a problem when the fundamental frequency is higher than C5.
When using large values for **debatt,** FOF superpositions may occur; small bandwidth values along with large tex values will then produce resonating sounds. Note that the number of allowed superpositions is limited, and that the corresponding message "stack is full ..." may occur; reducing **debatt** value is the appropriate solution, in this case.
NOTE:
When running CHANT on the array processor, **debatt** also has another function: if negative, *only filters* are used for the synthesis, excluding FOFs.

dga

General duration of the attack (in seconds) for the *first* note or event of the synthesis. This is a scaler for the durations given for the attack of each formant area by the parameters **dur1-10.** (dur1-10 specify ratios of dga). The default value is .35 (seconds). The usable range for modeling vocal sounds is from .1 to 1; less than .1 becomes more like an instrumental attack, more than 1 produces a noticeable timbre transformation. **Dga** is not used for dynamic changes over the course of a long note or phrase -- its function is to model the onset characteristics of a voice. **Dga** is automatically checked against the duration of the first note of the specified phrase, and is corrected so as never to be longer than half that note's duration.

dgf

Analogous to **dga,** it is the scaler which multiplies the individual decay times of the formant areas defined by **dvr1-10.** Default value is .85 (seconds). **Dgf** is not used for dynamic changes over the course of a long note or phrase -- its function is to model the decay quality of a voice *at the very end* of a vocal sound. The useable range is from .1 to 1.5. **Dgf** is automatically checked against the last note of the specified phrase, and is corrected so as never to be longer than half that note's duration.

dr1,2,3...999

The durations for each note or event, in seconds. A zero duration will produce bizarre results; the only limit in total duration is the capacity of the sound disk onto which the soundfile is being written. Default value is 1.3 (second) for **dr1,** 0 for the rest.
Note: the floating point intermediary sound file used during computation takes twice the space of the final integer sound file. Therefore, a total of two times the duration of the final sound must be available on the disk you are writing on.

droite Not in use at present.

dsil The duration in seconds of the silence between the beginning of the sound file and the beginning of the sound itself, also the time between the end of the sound and the end of the file. Default value is 0 (seconds). The silence protects against pops at the onset and end of the sound. It is a duration added to the total duration of the given notes; sound-files are thus **2*dsil** longer than the sum of their component durations. In very precisely controlled functions it is advisable to take **dsil** into account when specifying the times for break-points (see description of functions, section 1.1.4).

dsk In the array processor version, **dsk** is the inverse of the time quantum used for the calculation, ie. the frequency of parameter evaluation through function invocation and CHANT calculations.
It is not in use in the VAX version, in which the quantum is equal to the fundamental period.

dur1,2,3... Attack times for each of the formant areas, at the very beginning of the synthesis. These values seldom need to be changed; their default values establish their relationship, and they can be scaled by **dga** (they are ratios of dga). Default values are **dur1=1.00, dur2=1.15, dur3=1.30, dur4=1.45, dur5=1.60** (seconds). The default values are set such that **dga** gives the actual length of the overall attack in seconds.

durf In the VAX version only; **durf** is the equivalent of **dur1-200,** for the fundamental (see **apf**).

dvr1,2,3... Decay times for the individual formant areas, for the very end of the synthesis. Their values, are set such that the value for **dgf** is the actual length in seconds of the overall decay. All formants begin their decays at the duration **fmax*dgf** before the end of the sound; **fmax** is an internal CHANT variable which is used to store the longest formant decay (ie. the greatest **dvri).** The default values are **dvr1=1, dvr2=.965, dvr3=.930, dvr4=.910, dvr5=.890.** Higher formants decay more rapidly than lower ones.

dvrf In the VAX version only; **dvrf** is the equivalent of dvr1-200, for the "fundamental" (see **apf**): it specifies the decay duration for the "fundamental".

e The sampling rate. The default is 16000, which is adequate for most vocal sounds since they have little or no content higher thant 8000 hz. Note: this parameter is not written into the output parameter file.

exa and exf Permit exponential curves for the attacks and decays specified by **dur1...200** and **dvr1...200.** Default values are 1, which results in linear attack and decay. Values greater than 1 cause the curves to arch, those less than 1 cause the curves to sag. Little noticeable difference has been found with changes in these parameters in the range of durations used for vocal onset and decay.

fcomp **fcomp** is the center frequency for a resonance area added below the first formant to reinforce it and the fundamental partial. This resonance area is called the "complement". the default value for **fcomp** is 80 Hz. **Fcomp** is used only when **atc** is flagged (see atc), causing the amplitude of the complement to be calculated automatically.

fref1,2,3 stands for "Fréquence de REFérence". These values are the reference frequencies used to model the resonance characteristics of the human vocal tract. They are the points of reference for the automatic calculation of formant bandwidths according to their center frequencies (see atb and bref). Default values are **fref1=200, fref2=500, fref3=4000** (Hz). These parameters are part of the package of values that define a particular voice type and vowel sound. For details on these packages see the appropriate section in the manual (automatic calculations, section 1.3).

freq1,...200 The center frequencies (in Hz) for the formant regions. Up to 200 can be specified; five is the number normally used for vocal sounds. If all three automatic adjustments are flagged, five frequencies for these parameters, plus five values for their corresponding **texs,** are sufficient to define a vowel.

freqf1,...200 In the FPS version, the center frequencies (in Hz.) for the filters. (In the VAX version, filter frequencies are given by **freq1,...freq10**).

f0moyen Pivot frequency for amplitude changes in relation to fundamental frequency (see automatic calculations, section 1.3). The default value (200 Hz.) is good for a baritone for instance.

f1,...999 The fundamental frequencies for the notes or events of the phrase (in Hertz). Zero values will produce bizarre results. Default value is 100 (Hz.) for **f1,** 0 for the rest.

gauche Not in use.

hn HoarseNess: Random variation of formant amplitudes. **Hn** represents the maximum range of the variation (twice the excursion above and below the formant amplitude). Selections of

new random values occur at every quantum.

hollow A scaler which multiplies the amplitudes of the "fundamental" (in the VAX version only), and that of every formant which has a lower center frequency than that of formant 1 (including the complement), in relation to higher formants. It is the inverse function to **cslope**, but it does not have the possibility for automatic adjustment that **cslope** has. Default value is 1, resulting in no change of the given values.

jitt1,2,3 Scalers controlling the random variation of the fundamental frequency. They are linked respectively to: **tjitter1**, **tjitter2**, **tjitter3** (the times between selections of new random values). **jitter1+jitter2+jitter3** = twice the total desired excursion ABOVE (or below) the mean value of the fundamental frequency, i.e. the TOTAL desired "excursion" (range). If **jitter1+jitter2+jitter3 = .03** the range of the random variations will be .03 times the fundamental frequency (from +1.5% to -1.5% of the fundamental). Each of them should be approximately 1/3 of the total desired excursion If **jitter1+jitter2+jitter3 = .02** and the fundamental frequency is 134 Hertz, the total excursion in Hertz will be .02 times the fundamental frequency: .02*134 = 2.68 Hertz, that is from 1.34 Hz. above (134+1.34=135.34) to 1.34 Hz below (134-1.34=133.66).

nnote is the number of notes or events in the phrase being synthesized: as few as 1, as many as 999. **Nnote** may be smaller than the number of notes defined, resulting in a truncated phrase. Default value is 1.

nof The number of formants. The possible range is 1 to 200. The majority of vocal sounds are made with 5 formants. Fewer formants can be specified by nof than the number defined, in which case only the first nof formants are calculated. Default value is 5. (If **atc** is flagged, formant **nof+1** is used to store the values for the "complement", thus leaving only 199 formants available.)

phase1,...200 Coeficient of continuity of phase of formant-sinusoids. At each fundamental period (number i) beginning, the phase of the sinusoid for the formant number j is set to angle(i-1,j)*phase(j), where angle(i-1,j) is the phase angle of the sinusoid of the formant number j at the end of the preceeding period of number i-1. The default value being 0, the phase is usually reset to 0. If the value is 1, the phase is CONTINUOUS from period to period, and in between, the phase is weighted between those extreme values (that is why it is called coefficient of phhase continuity). This trick allows you to go continuously from a speech or instrumental

sound to an additive-synthesis sound: you only have to write a user that sets all the **tex(i)** equal to **atten,** sets **debatt** equal to the fundamental period and sets the bandwidths to 0 (all this for the continuity of the local amplitude envelopes). This is to allow additive synthesis of inharmonic sounds and progressive transitions between those sounds and usual harmonic sounds of Chant. When phase parameters are null, which is the default and the average case, they are not copied in the **scr.par** file.

play *In the array processor version,* this parameter -- together with **apf** --, controls the input sent to the filters. **Play** is to be used when sending white noise, and **apf** when sending an already existing soundfile. The use of one of those parameters is not exclusive of the other. If both are zero, filters are not used. If **play** is greater than zero, white noise is sent to the filters. **Play** can be used as a scaler on noise input amplitude in this case. If zero, there is no noise in the filters.

seed Seed for initializing the random algorithm. Default is 0.

sex Type of the voice being used: 0 is female, 1 is male, 2 is castrato. These flags effect the automatic relationship for **cslope** to **coefamp** and fundamental frequency, and the adjustment of formant frequencies (bending), which are discussed in section 1.3 (CHANT automatic calculations and rules). For a male voice, use a male vowel package and **sex=1,** for a female voice use a female vowel package and **sex=0.** Default is 1.

stereo Not in use.

tdeb Stands for "Temps de Debut" --which means "begin time" in French-- (see **tfin**). For synthesizing less than the total duration of the defined phrase. The calculated sound will start abruptly at **tdeb.** Important: **tdeb** is never recorded in the output parameter file. Its use is purely temporary and it is not stored with the other accumulated values. Default value for **tdeb** is 0 (seconds). If you set **tfin to a value less or equal to tdeb** only one fundamental excitation and resonance will be calculated, which is a nice way to do that, and sometimes you wish it, to make for example a nice smooth and exact envelope of the spectrum of the sound that you are calculating. For experiments, one often wishes to synthesize only a short section of a long sequence,and not to have to wait for the long, long run of the poor lonesome computer, and this without altering the beautiful set of parameters courageously written in the ".par" file: That's why one can use the pair of paramaters called **tdeb** and **tfin** to indicate to the program that you want only the sound of this critical section . The computed sound will be of a duration of **tfin–tdeb** and identical to the same

section extracted from the totally computed sound. To avoid errors at production time the values of those parameters are printed at run-time. The default value for **tdeb** is (naturally) 0 and for **tfin** is 1000000 so that you can compute without trouble sounds up to 1000000 seconds long... Furthermore, like **e** these parameters ARE NOT written into the output parameter file (**scr.par**) in order to allow you to use this file for production after the "fine tuning" of the parameters without having to modify (or worse ,forgetting to modify...) their values (which thus will always be the default values unless you explicitely modify them).

tjitt1,2,3

The times between selections of new random values between +.5 and -.5, which are scaled by **jitter1, jitter2,** and **jitter3,** and used in turn as continuously changing random micro-tonal scalers for the fundamental frequency (f). A package of values for these parameters has been found that approximates 1/f random variation with a slight weighting toward variation at a period of approximately 1.1 seconds. These values were found to produce a good balance between lifelike irregularity in the synthesized voice along with freedom from jolting or awkward fluctuations in pitch. These are the default values:

jitt1 = .01 **tjitt1** = .05 **(seconds)**
jitt2 = .01 **tjitt2** = .1111
jitt3 = .01 **tjitt3** = 1.2188

tex1,...200

The excitation time for the pitch-synchronous impulses which produce the sound. The excitation time controls the width of the "skirts" of the formant areas, i.e. the bandwidth at -40 db. Values usually range between .001 and .005., but you can use very large ones for smoothing the attack to the point where it is no longer an attack but an increase in loudness.

tfin

Stands for "Temps de Fin" (see **tdeb**). It is a convenience for synthesizing less than the total duration of the defined phrase. If **tfin** is less than the cumulative duration of the specified notes, the resulting sound will stop abruptly when **tfin** is reached. Important: **tfin** is never recorded in the output parameter file. Its use is purely temporary and it is not stored with the other accumulated values.

tremolo

Causes a periodic change in the amplitude of each formant area which is synchronous with the vibrato and slightly dephased for each formant area.
Default value is 0 (no change);
A non zero value is a scaler on the amplitude of the variation.

ttr One-half the transition time between successive notes in a phrase. The pitch transition begins **ttr** seconds before the end of a note and continues ttr seconds past the beginning of the next. It is automatically checked and adjusted so as never to be longer than one- tenth the duration of the shortest of the two notes it functions between. The default value is .07 (seconds). Values lower than .02 produce an instrumental rather than a vocal transition; values as high as .18 can be used if the skips between successive pitches are larger than a fifth. The transition curve is a sine-wave section from 90 degrees to 270 degrees.

tvala1, tvala2 Times between selections of new random values for variations of the vibrato amplitude. The random values, which are in the range (+.5, -.5), are scaled by **vala1** and **vala2** and used in turn to change continuously the vibrato amplitude.

tvalf1, tvalf2 Times between selections of new random VALues for variations of the vibrato Frequency.

user Flag for no-call (0), or call (1) of the user's subroutine (a default one is available, which offers the possibility of using phoneme dictionaries). At each quantum, the user defined rules are interleaved with the CHANT standard algorithms, when the parameter **user** is flagged.

vala1, vala2 Stands for "Vibrato-ALeatoire-Amplitude". Scalers controling the random variation of the vibrato amplitude value. They are linked respectively to **tvala1** and **tvala2** (the times between selections of new random values). **vala1+vala2** = twice the total desired excursion of the vibrato amplitude value ABOVE (or below) its mean value (**vibamp**), i.e. the TOTAL desired "excursion" (range). If **vala1+vala2 =.03** the total range of the value (of the vibrato amplitude) will be .03 times the mean vibrato amplitude (from +1.5% to -1.5%) Each of them should be approximately 1/2 of the total desired excursion. If **vala1+vala2 =.02** and the vibrato amplitude (**vibamp**) = .04 (4%) ,the "total excursion" will be .02 times the mean vibrato amplitude : .02*.04 = .008 ,that is from .004 . above (.04+.008=.048) to .004 below (.04-.008=.032).

valf1, valf2 Stand for "Vibrato-ALeatoire-Frequence". Scalers controling the random variation of the vibrato frequency value. They are linked respectively to **tvalf1** and **tvalf2** and function like **vala1** and **vala2.**

vibamp The amplitude of the vibrato. It is expressed as the ratio between 1/2 the vibrato "total excursion" (range) and the center frequency around which the vibrato is operating. For example, a value of .03 for **vibamp** results in a vibrato which moves from 1.03 times the center frequency to 0.97 times the center frequency, which

results in a "total excursion" of 0.06. The default value is 0.02.

vibfreq The frequency of the vibrato in Hertz. For singers, this value varies between roughly 4 and 7, and small changes within this range are quite noticeable. The default value is 5.1.

vuser1,...500 Variables at the USER's disposal, from **vuser1** to **vuser500.** Some numbers are already used by CHANT:
In the FPS version:
Vuser61 to 70 = the amplitude of the noise to be generated around the first ten formants.
Vuser81 to 90 = the bandwidth of the noise (in Hz.) to be generated around the first ten formants.
In the VAX version:
Vuser61 to 70 = the amplitude of the filters (frequencies are given by those of the formants).
Vuser81 to 90 = the bandwidth of the filters (in Hz.)

appendix 2: DEFAULT VALUES OF STANDARD CHANT PARAMETERS (FILE SCR.PAR)

< notes parameters >

dr1 = 1.3000 , f1 = 100.0000

< formantics parameters >

freq1 = 609.0000 , ampl1 = .0278
band1 = 77.6438 , tex1 = .0030
dur1 = 1.0000 , dvr1 = 1.0000

freq2 = 1000.0000 , ampl2 = .0137
band2 = 88.4311 , tex2 = .0030
dur2 = 1.1500 , dvr2 = .9650

freq3 = 2450.0000 , ampl3 = .0070
band3 = 122.9401 , tex3 = .0030
dur3 = 1.3000 , dvr3 = .9300

freq4 = 2700.0000 , ampl4 = .0078
band4 = 127.8438 , tex4 = .0030
dur4 = 1.4500 , dvr4 = .9100

freq5 = 3240.0000 , ampl5 = .0018
band5 = 137.6589 , tex5 = .0030
dur5 = 1.6000 , dvr5 = .8900

< filters parameters >

freqf1 = .0000
amplf1 = .0000
bandf1 = .0000

freqf2 = .0000
amplf2 = .0000
bandf2 = .0000

freqf3 = .0000
amplf3 = .0000
bandf3 = .0000

freqf4 = .0000
amplf4 = .0000
bandf4 = .0000

freqf5 = .0000
amplf5 = .0000
bandf5 = .0000

< other parameters >

fref1 = 200.0000
fref2 = 500.0000
fref3 = 4000.0000
bref1 = 75.0000
bref2 = 75.0000
bref3 = 150.0000
ajus1 = .0000
ajus2 = 5.7000
ajus3 = .0000
f0moyen = 200.0000
fcomp = 80.0000
nnote = 1.0000
nof = 5.0000
debatt = .0100
atten = .0070
dsil = .0000
tremolo = .0000
dga = .3500
dgf = .8500
exa = 1.0000
exf = 1.0000
vibfreq = 5.1000
vibamp = .0200
jitt1 = .0100
jitt2 = .0100
jitt3 = .0100
vala1 = .0000
vala2 = .0000
valf1 = .0000
valf2 = .0000
hn = .0000
cslope = -1.0000
hollow = 1.0000
coefamp = 1.0000
envelo = 1.0000
amp = 1.0000
ata = 1.0000
atb = 1.0000
atc = 1.0000
ttr = .0700
apf = .0000
durf = .8500
dvrf = 1.0000
tjitt1 = .0500
tjitt2 = .1111
tjitt3 = 1.2188
tvala1 = 5.0000
tvala2 = 5.0000
tvalf1 = 1.0000
tvalf2 = 1.0000
sex = 1.0000
dsk = .0000
cor = 1.0000

```
seed     =       .0000
vuser61   =     -1.0000
vuser62   =     -1.0000
vuser63   =     -1.0000
vuser64   =     -1.0000
vuser65   =     -1.0000
vuser66   =     -1.0000
vuser67   =     -1.0000
vuser68   =     -1.0000
vuser69   =     -1.0000
vuser70   =     -1.0000
user      =       .0000
play      =       .0000
carlin    =      1.0000
stereo    =       .0000
gauche     = -1000000.0000
droite     = -1000000.0000
```

Appendix 3: PHON.F, A BASIC USER FOR USING PHONEME DICTIONARIES

```
c*************************************************************************
      subroutine user1
c*************************************************************************

c variables for phon.f:
      include '/usr/local/include/chant/params.f'
c common variables for phon.f (common statements may appear directly here):
      include '/usr/local/include/chant/phon_comuse.f'
c equivalence table for standard CHANT variables:
      include '/usr/local/include/chant/equiva.f'
c common standard CHANT variables:
      include '/usr/local/include/chant/comsyn.f'
c your own common variables and equivalence table:
c     include '/MY_DIRECTORY/MY_COMUSE.f'
c     include '/MY_DIRECTORY/MY-EQUSE.f'

      integer gnol,debs,phocan,phofil(30)
      real dph(1000)
      external ouvre,delnom,blanc,trans

      common vide
      logical vide
      data phofil /md,mi,mc,mo,lpt,mp,mh,mo,
     .  lesp,lesp,lesp,lesp,lesp,lesp,lesp,lesp,lesp,lesp,lesp,lesp,
     .  lesp,lesp,lesp,lesp,lesp,lesp,lesp,lesp,lesp,lesp/

      if (init .eq. 1.) then
      write(0, *)'      USER FOR PHONEME DICTIONARIES'
      end if

      if ( init  .eq. 1) then
      phocan = 14
      vide = .true.
      len = 0
      lstr = 80
      debs = 2
      total = 0.
code tronque a lcode caracteres
code truncated at lcode characters
      lcode = 10
chargement du dictionnaire des phonemes
c loading the phoneme dictionary
      write(0,*)'     default phoneme dictionary : dico.pho'
      do 20 i=1,lstr
20    str(i) = lesp
      write(0,*)'                    other phoneme dictionary ? : '
      read (tti,30) str
30    format (80a1)
      do 35 i=1,lstr
      call asc(str(i))
35    continue
```

```
c la subroutine "blanc"                 test si la chaine est vide
c the "blanc" subroutine tests empty strings
            call delnom
            call blanc (str,lstr)
            if (vide) then
                  do 31 i=1,30
31                nom(i) = phofil(i)
                  len = 13
            else
                  call trans (str,1,lstr,nom,len)
                  do 40 i=1,len
40                if (nom(i) .eq. lpt) go to 45
                  nom(len+1) = lpt
                  nom(len+2) = mp
                  nom(len+3) = mh
                  nom(len+4) = mo
                  len = len+4
            end if
45          canal  = phocan
            age    = 'old'
            mode   = 'sequential'
            forme  = 'formatted'
            call ouvre
            lectur = 1
            kas = 0
50          do 55 i = 1,lstr
55          str(i) = lesp
60          read (canal,65,end=140) (str(i),i=1,lstr)
65          format (80a1)
            do 19000 i=1,lstr
            call asc(str(i))
19000       continue
            if (str(1) .eq. linf) go to 60
            call blanc (str,lstr)
            if (vide) go to 60
            go to (95,180) lectur
c purge des espaces et tabs de part et d'autre de la chaine str(lstr)
c "purging" tabs and blanks before and after the string
95          call cadre (str,lstr,gnol)
            go to (100,110,120) kas+1
code-lecture
100         iii = iii+1
            code (iii,1) = gnol
            call maj (str,gnol)
            do 105 i=1,gnol
            code (iii,i+1) = str(i)
c le k pour print seulement
c k is for printing only
            k = i+1
105         continue
            go to 130
c tenv- et tcoe-lecture
c lecture de la 1ere valeur tenv dans la chaine str(gnol)
```

```
c reading the first value tenv in the string str(gnol)
110         call val1 (str,gnol,val)
            tenv (iii) = val
            long = gnol
            call cadre (str,long,gnol)
c lecture de la 2eme valeur tcoe dans la chaine str(gnol)
c reading the second value tcoe in the string str(gnol)
            call val1 (str,gnol,val)
            tcoe(iii) = val
            go to 130
c tfreq(iii,i)-lecture
120         j1 = 0
125         call val1 (str,gnol,val)
            j1 = j1+1
            tfreq (iii,j1) = val
            call blanc (str,gnol)
            if (vide) go to 130
            long = gnol
            call cadre (str,long,gnol)
            go to 125
c difference noruse-hybrid
c les formants hauts sont charges avec les valeurs par defaut
c high formants are "loaded" with default values
130         do 135 lkj=j1+1,nof
135         tfreq(iii,lkj) = tfreq(1,lkj)
            kas = kas+1
            if (kas .eq. 3) kas=0
            go to 50
140         close (canal,status='keep')
            if (lectur .eq. 2) go to 195
            imax = iii
c lecture du fichier-phonemes de l'utilisateur
c reading the user's phoneme file
150         jjj = 0
153         do 155 i=1,lstr
155         str(i) = lesp
            write(0,*)'                    phoneme file    : '
            read (tti,165) str
165         format (80a1)
            do 166 i=1,lstr
            call asc(str(i))
166         continue
            call delnom
            call blanc (str,lstr)
            if (vide) go to 153
            call trans (str,1,lstr,nom,len)
            do 170 i=1,len
170         if (nom(i) .eq. lpt) go to 175
            nom(len+1) = lpt
            nom(len+2) = mp
            nom(len+3) = mh
            nom(len+4) = mo
            len = len+4
```

```
175         call ouvre
            lectur = 2
            go to 50
c lecture de la 1ere valeur tenv dans la chaine str(gnol)
c reading the first value tenv in the string str(gnol)
180         jjj = jjj+1
            call cadre (str,lstr,gnol)
            call val1 (str,gnol,val)
            dees(jjj) = val
            long = gnol
            call cadre (str,long,gnol)
c lecture de la 2eme valeur (ascii) phon(jj) dans la chaine str(gnol)
c reading the second value (ascii) phon(jj) in the string str(gnol)
            call maj (str,gnol)
            phon (1,1) = gnol
            do 185 i3=1,gnol
            phon (1,i3+1) = str(i3)
c le k pour print seulement - k is for printing only
            k = i3+1
185         continue
comparaison des codes-utilisateurs et des codes-dictionnaires
comparing the code of the phoneme file and that of the dictionary
190         continue
            nbcar = phon(1,1)
c nbcar1 pour print seulement-nbcar for printing only
            nbcar1 = nbcar+1
            do 194 i=1,imax
            do 192 k=debs,nbcar1
            if (phon(1,k) .ne. code(i,k)) go to 194
192         continue
            vals(jjj) = i
            go to 50

194         continue
c           echoue - error
            write (0,1931) (phon(1,lkj),lkj=debs,nbcar1)
1931        format (' le phoneme-user ',9a1,' n''est pas dans le dictionnairé)
            write (0,1932)
1932        format ('                    GAFFE, GAFFE, .....')
            go to 50
c           la phrase est entree - the "phrase" is loaded
195         continue
            jmax = jjj
            do 196 j=1,jmax
196         total = total + dees(j)
            close (canal,status='keep')


            total = tfin - tdeb
            write(0,*)' beginning time = ',tdeb,' end time = ',tfin
            write(0,*)'       computing time = ',total
c initialisations pour user2 - initializations for user2
            iseg = 0
```

```
            t2  = 0.
            val2 = vals(1)


            end if

            return
            end


c*************************************************************************
            subroutine user2
c*************************************************************************

            include '/usr/local/include/chant/phon_comuse.f'
            include '/usr/local/include/chant/comsyn.f'
            include '/usr/local/include/chant/equiva.f'
c       include '/MY_DIRECTORY/MY_COMUSE.f'
c       include '/MY_DIRECTORY/MY-EQUSE.f'
            real incrp

c ***********************************************************
c phoneme file initialisation for formants-coefamp-envelo
101         if (t .lt. t2) go to 10
c parameters for new segment
            iseg = iseg+1
            if (iseg .gt. jmax) go to 10
            dee = dees(iseg)
            val1 = val2
            val2 = vals(iseg)
            t1  = t2
            t2  = t2 + dee
            goto 101
10          do 20 nfor=1,inof
            if (tfreq(val2,nfor) .ne. 0.) freq(nfor) =
     .      (tfreq(val1,nfor) * (t2-t) + tfreq(val2,nfor) * (t-t1))
     .      / dee
            coefa1 = coefam * (tcoe(val1) * (t2-t) + tcoe(val2) * (t-t1))
            if (dee .ne. 0.) coefam = coefa1 / dee
            envel1 = envelo  * (tenv(val1) * (t2-t) + tenv(val2) * (t-t1))
            if (dee .ne. 0.) envelo = envel1  / dee
20          continue

            return
            end

c*************************************************************************
            subroutine user3
c*************************************************************************

            include '/usr/local/include/chant/phon_comuse.f'
            include '/usr/local/include/chant/comsyn.f'
            include '/usr/local/include/chant/equiva.f'
```

```
c     include '/MY_DIRECTORY/chant/MY_COMUSE.f'
c     include '/MY_DIRECTORY/MY-EQUSE.f'

      return
      end

c ************************************************************
      function irasar(n)
c ************************************************************
      irasar=min( (ran(l)*n + 1),n)
      return
      end
c*****************************************************************************
      subroutine maj (itab,l)
c*****************************************************************************
c     lower-case to upper case conversion

      parameter (la = 065)
      parameter (lz = 090)
      parameter (ma = 097)
      dimension itab(80)
      do 10 i=1,l
c on systems like pdp-10
c     if (itab(i) .gt. 'z'  .or. itab(i) .lt. 'a') go to 10
c        z majuscule          a majuscule
c on vax-vms
c     if (itab(i) .le. 'z'  .and. itab(i) .ge. 'a') go to 10
c     itab(i) = itab(i) - 'a' + 'A'
c                 lower a ^    ^ upper A
c on unix-system upper-case to lower-case conversion
      if (itab(i) .ge. la  .and.  itab(i) .le. lz)
     .  itab(i) = itab(i) - la + ma
c             a majuscule ^    ^ a minuscule
c next time do you convert the characters of the NON-A ?
c ok, here we have a lower-case
10    continue
      return
      end

c*****************************************************************************
      subroutine val1 (itab,kit,conc)
c*****************************************************************************
c     ascii-characters input
c     return first real numerical value

      parameter (lmoin = 045)
      parameter (lesp  = 032)
      parameter (ltab  = 009)
      parameter (lpt   = 046)
      parameter (l0    = 048)
      parameter (l9    = 057)
      dimension itab(80)
c
```

```
            conc = 0.0
            deci = 0.0
            i=1
            isign=1
            if (itab(1).ne.lmoin) go to 1
            i=2
            isign=-1
1           do 10 ib = i,kit
            if (itab(ib).eq.lesp .or. itab(ib) .eq. ltab) go to 15
            if (itab(ib).eq.lpt) go to 20
            if (itab(ib).lt.l0 .or. itab(ib).gt.l9) go to 10
            conc = (conc*10.)+(itab(ib)-l0)
10          continue
15          ifin=ib-1
            go to 40
20          deci=0.
            do 25 j=ib+1,kit
            if (itab(j).eq.lesp .or. itab(j) .eq. ltab) go to 30
c                   space ^             tab ^
25          continue
30          ifin = j-1
35          j = j-1
            if (j .eq. ib) go to 40
            deci = deci/10.+ (itab(j)-l0)
            go to 35
40          conc=isign * (conc+(deci/10.))
c string-cleaning
            do 50 i=1,ifin
50          itab(i) = lesp
c
            return
            end
c*******************************************************************************
```

appendix 4: FREQUENCY TABLES

oct oct'	0 1 c/8	1 2 c/4	2 3 c/2	3 4 c	4 5 c*2	5 6 c*4	6 7 c*8	7 8 c*16	8 9 c*32
UT	32.7025	65.405	130.81	261.62	523.24	1046.48	2092.95	4185.92	8371.84
UT#	34.6475	69.295	138.59	277.18	554.36	1108.72	2217.44	4434.88	8869.76
RE	36.7075	73.415	146.83	293.66	587.32	1174.64	2349.28	4698.56	9397.12
RE#	38.89125	77.7825	155.565	311.13	622.26	1244.52	2489.04	4978.08	9956.16
MI	41.20375	82.4075	164.815	329.63	659.26	1318.52	2637.04	5274.08	10548.16
FA	43.65375	87.3075	174.615	349.23	698.46	1396.92	2793.84	5587.68	11175.36
FA#	46.24875	92.4975	184.995	369.99	739.98	1479.96	2959.92	5919.84	11839.68
SOL	48.99875	97.9975	195.995	391.99	783.98	1567.96	3135.92	6271.84	12543.68
SOL#	51.91375	103.8275	207.655	415.31	830.62	1661.24	3322.48	6644.96	13289.92
LA	55	110	220	440	880	1760	3520	7040	14080
LA#	58.27	116.54	233.08	466.16	932.32	1864.64	3729.28	7458.56	14917.12
SI	61.73625	123.4725	246.945	493.89	987.78	1975.56	3951.12	7902.24	15804.48

Figure 14. Frequency table

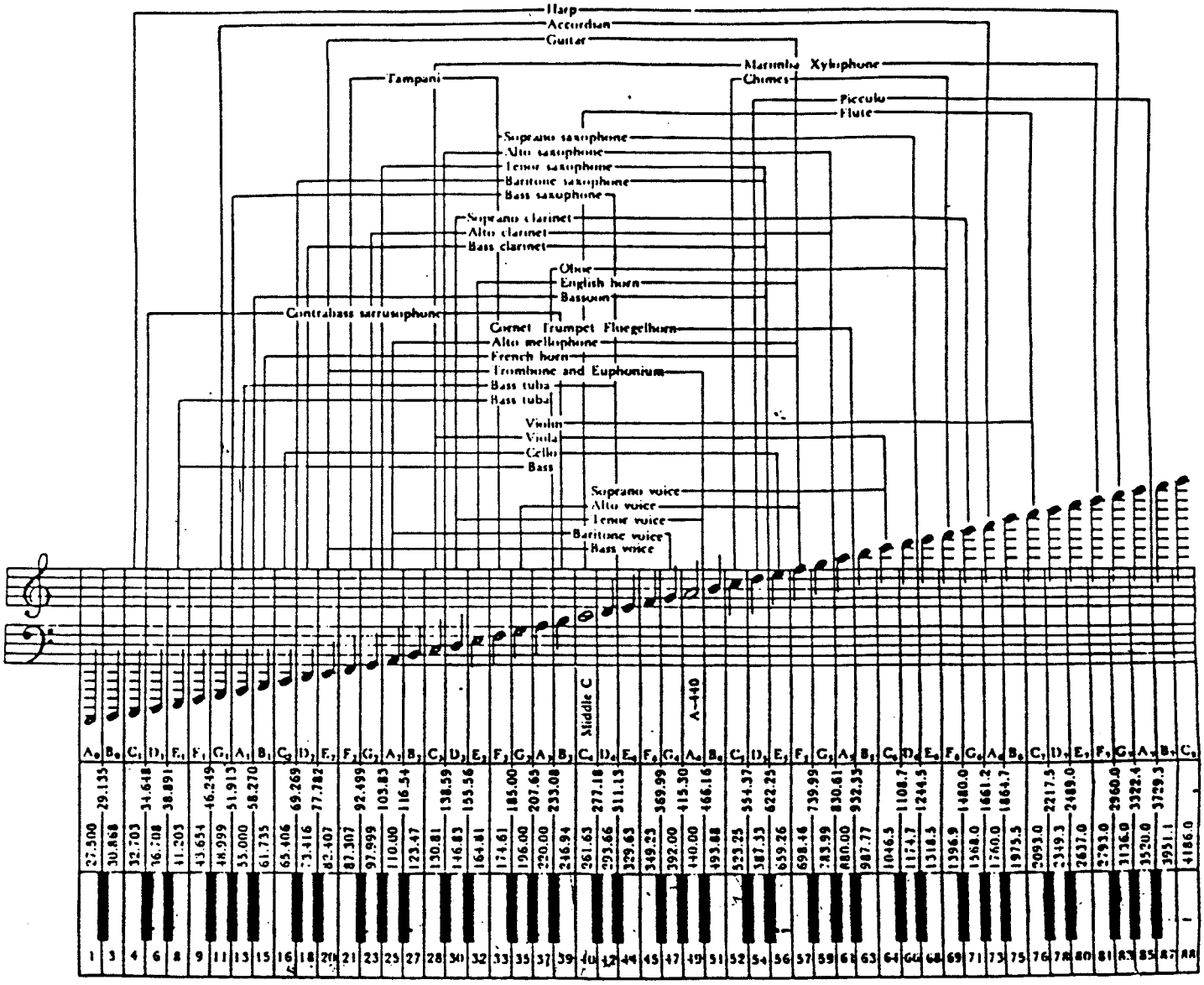


Figure 15. Frequencies and their musical notation- frequency ranges of some instruments

appendix 5: PHONEMES

"Male Phonemes"															
phoneme	F1			F2			F3			F4			F5		
	fr Hz.	amp dB.	band Hz.	fr Hz.	amp dB.	band Hz.	fr Hz.	amp dB.	band Hz.	fr Hz.	amp dB.	band Hz.	fr Hz.	amp dB.	band Hz.
a	609	0	78	1000	-6.1	88	2450	-12	123	2700	-11	128	3240	-23.8	138
e	400	0	64	1700	-9	81	2300	-8	101	2900	-11	119	3400	-19	134
i	238	0	73	1741	-19.6	108	2450	-16.5	123	2900	-19.6	132	4000	-31.7	150
o	325	0	73	700	-11.8	80	2550	-26	125	2850	-21.9	131	3100	-27.9	135
ou	360	0	51	750	-11.8	61	2400	-29.4	168	2675	-26.4	184	2950	-35.4	198
	415	0	45	1400	-12.3	64	2200	-15.6	93	2800	-18.4	114	3300	-27	129
	300	0	66	1600	-13.6	93	2150	-12.3	108	2700	-14.9	122	3100	-23.4	131
	400	0	73	1050	-12.3	90	2200	-19.1	118	2650	-19.6	127	3100	-28.7	135

"Female Phonemes"															
phoneme	F1			F2			F3			F4			F5		
	fr Hz.	amp dB.	band Hz.	fr Hz.	amp dB.	band Hz.	fr Hz.	amp dB.	band Hz.	fr Hz.	amp dB.	band Hz.	fr Hz.	amp dB.	band Hz.
a	650	0	69	1100	-8.3	95	2860	-12.8	95	3300	-11.7	102	4500	-19.2	120
e	500	0	75	1750	-8.9	104	2450	-10.4	123	3350	-14.5	140	5000	-22.6	165
i	330	0	89	2000	-14.3	114	2800	-10.9	132	3650	-9.8	145	5000	-18.8	162
o	400	0	86	840	-12.3	109	2800	-25.7	130	3250	-24	138	4500	-30.8	157
ou	280	0	70	650	-18.2	132	2200	-47.7	156	3450	-50.2	224	4500	-51.7	272

Figure 16. Formant frequencies, amplitudes and bandwidths for different phonemes

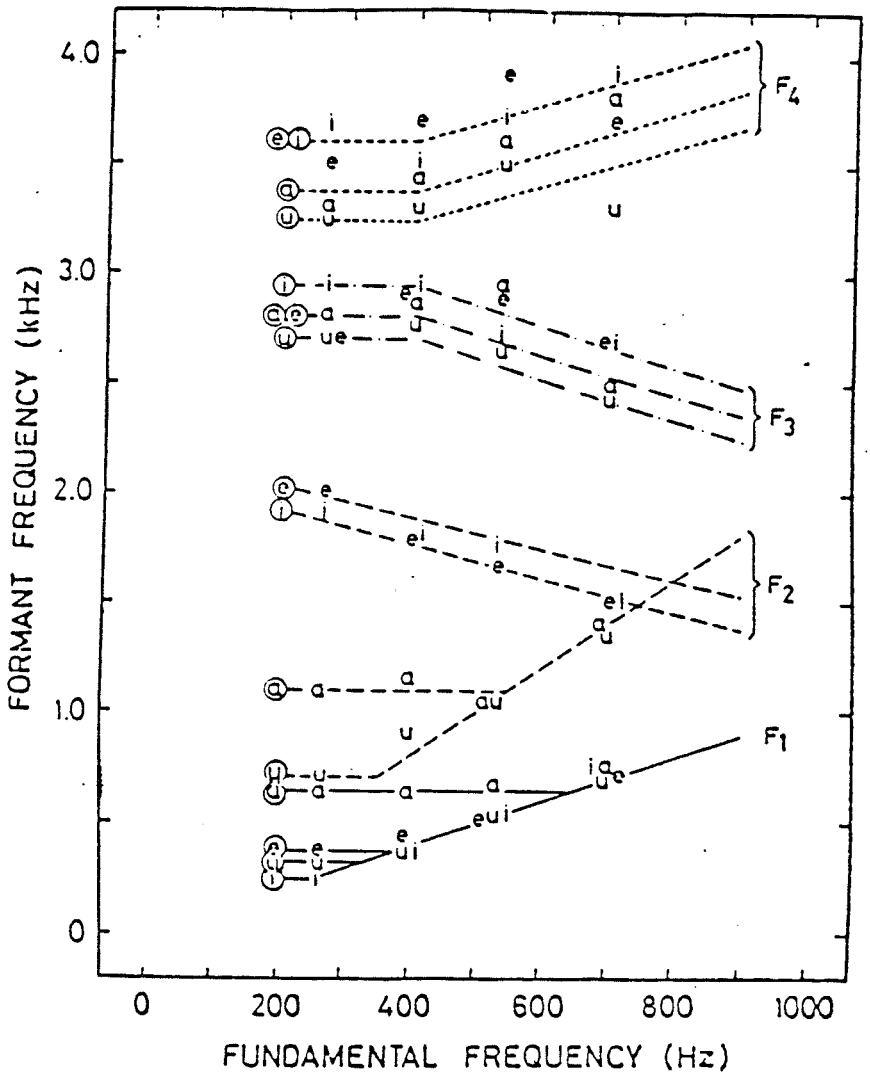


Figure 17. Formant frequencies in relation to fundamental frequency.(from: Johan Sundberg, Synthesis of Singing, Swedish Journal of Musicology, 1978, 60.)