Ircam
documentation

- Research reports
- Musical works
- Software

PATCHWORK

Csound/Edit-sco

Library of Modules for Generating
Csound scores

r e f e r e n c e

Second Edition, April 1996

IRCAM Centre Georges Pompidou

This manual was written by Laurent Pottier,
translated by Tom Johnson,
revised and produced under the editorial responsibility of
Marc Battier - Marketing Office, Ircam.

The software was conceived and programmed by
Laurent Pottier and Mikhael Malt.

Second edition of the documentation, April 1996.
This documentation corresponds to version 1.2 of the software, and to
PatchWork version 2.5 or higher.

Ircam
1, place Igor-Stravinsky
F-75004 Paris
Tel. (33) (1) 44 78 49 62
Fax (33) (1) 42 77 29 47
E-mail ircam-doc@ircam.fr

IRCAM Users Groups

The use of this program and its documentation is strictly reserved for members of the IRCAM Software Users Groups. For further information, contact:

Marketing Office
Ircam
1, Place Stravinsky
F-75004 Paris
France

Telephone (1) 44 78 49 62
Fax (1) 42 77 29 47
Electronic mail: bousac@ircam.fr

Please send suggestions and comments on this documentation to

mail: Marc Battier, Documentation, Marketing Office, Ircam,
1, Place Stravinsky, F-75004 Paris, France
Electronic mail: ircam-doc@ircam.fr

To see the table of contents of this manual, click on the Bookmark Button located in the Viewing section of the Adobe Acrobat Reader toolbar.

Contents

1 Résumé

La librairie appellée **Csound-edit-sco** (ou « PW-Csound ») sert à générer des « partitions » pour Csound à partir de PatchWork. Cela permet par exemple d'employer la capacité algorithmique et graphique de PatchWork pour manipuler des données (hauteurs, amplitudes, durées, fonctions et tables).

La librairie ne permet pas de concevoir les « instruments » de Csound, mais concentre ses fonctions sur la production de « partitions ». Des instruments-types, « ex1.orc » et « ex2.orc » sont présentés à l'annexe 2 de ce manuel.

Le logiciel Csound est disponible auprès de Barry Vercoe, Media Lab, Massachussetts Institute of Technology (accessible par FTP ou World Wide Web). Il est également distribué par le Forum Ircam.

1 Introduction

The **Csound-edit-sco** library includes a group of PatchWork modules intended for preparing Csound scores in a precise and convenient way. It allows one to create parameters and tables either by preparing them graphically (with the graphic tools of PatchWork), by means of algorithms, or by programing notes in traditional music notation.

It is assumed that the reader is familiar with PatchWork and with the operation of Csound. This is not a Csound manual, and readers should refer to the manual of Barry Vercoe[1] for information on the use of Csound itself. Csound involves two different elements, the score and the orchestra, each of which may vary in complexity, according to the user. Some prefer to use very sophisticated orchestras and a reduced number of score parameters, while others may wish to place most of the information in the score. The instruments (orchestra) are in this case rather simple. It is this second option which has been chosen in this manual, which shows how to generate and control parameters "p" of statements "f" and "i" of Csound scores.

The examples in this manual refer principally to two types of simple orchestras: "ex1.orc" and "ex2.orc" [2] (see appendix A2).

---

1. **Csound - A Manual for the Audio Processing System and Supporting Programms with Tutorials**, by Barry Vercoe, Media Lab, M.I.T. The manuel, as well as the software, are freely accessible through internet; they are also available through Ircam's Marketing Office.

2. These files are located in the **User-library->Csound-edit-sco->getstarted** folder.

Loading the library

The library is loaded from a **Csound-edit-sco** folder, which must be present in the **User-Library** folder[3]. It contains a loading file (selected from the menu **Load-library** in PatchWork): **edit-sco.lib**. This is the file that should be selected in order to proceed with the loading of the library. Once loaded, the library displays four main sub-menus, shown here. Please refer to the end of this manual for the complete list of sub-menus.

This folder includes the library for programing Csound scores and also some functions permitting one to program permutations, interpolations, and aleatoric data.

3. The **Csound-edit-sco** library is available as part as the regular PatchWork distribution.

Basic Structure of Csound Scores

A score file normally consists of three basic statements: *tables*, *notes*, and the end of score declaration. Each category of statement is identified by a specific *opcode*, respectively "f", "i", and "e". Other score statements are defined in Csound, but they are not explicitely used in this library. Please consult MIT's Csound manual for details on these statements, or see modules **editsco** and **edit-sco-obj** later in this manuel. The list of parameters consists of "p-fields" : a "p-field" (short for parameter field) is the place where Csound expects to find a certain parameter. For example, the first p-field of a "i" note statement is always the instrument number, while the two following p-fields are reserved for the action time and the duration of the note. The first parameter, which is found in the first p-field, is hence called p1, and so on.

Tables

Tables are defined by the opcode "f" at the beginning of a line, followed by various function parameters, depending on the subroutine used**.** Such a statement calls a specific function, defined in Csound as a GEN subroutine. The purpose of a "f" statement is to fill a table with data, which could be for example a cycle of a wave, or an envelope. Here is the general syntax of a "f" statement.

```
f p1 p2 p3 p4 ->  p1      table number
                  p2      moment of activating the table
                  p3      size of the table (2^n ou 2^n+1,max. 2^24)
                  p4      number of the GEN subroutine
                  p5      …
                  p6      …
                  ...parameters depending on p4
```

Note Statements

Notes statements are defined by a opcode "i" at the beginning of the line, followed by the relevant parameters. This declaration serves to trigger off an instrument (p1) at a specific time (p2) for a defined duration (p3).

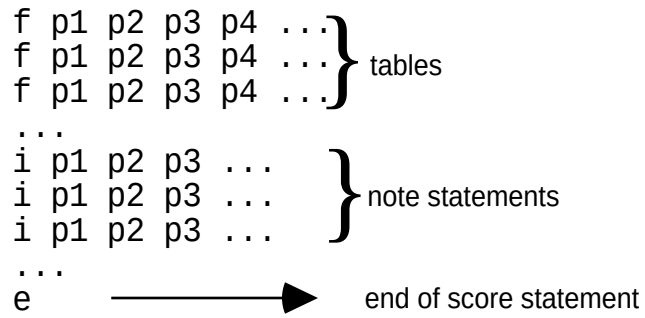
```
i p1 p2 p3  ->    p1      instrument
                  p2      attack time
                  p3      duration
                  p4      …
                  p5      …
                  ... other parameters
```

The "e" Statement

Declaring "e" indicates the end of a file**.** All subsequent note statements are ignored.

The complete score

We can summarize this structure as follows:

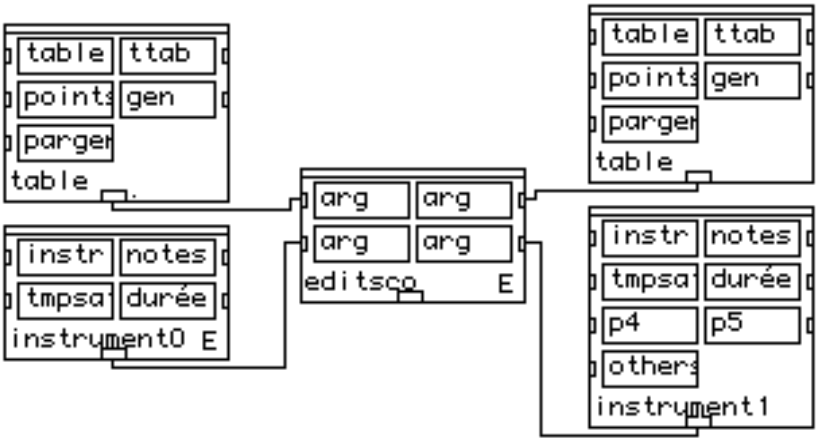


Example of a score producing phasing[4]:

```
; function 1 calls GEN07              (this is a comment)
f  1  0  1024  7
0.0 82.0 0.78 72.0 0.93 153.0 0.98 185.0 1.0 143.0 1.1.4.2. 194.0 0.94  82.0
0.82 113.0 0.0
; function 2 calls GEN11              (this is a comment)
f  2  0  1024  11
128  1  0.95
; p1      p2       p3      p4          p5        (this is a comment)
i 1       0        3       32          3000
i 1       0        3       32.032      3000
i 1       0        3       32.064      3000
i 1       0        3       32.096      3000
i 1       0        3       32.128      3000
i 1       0        3       32.16       3000
i 1       0        3       32.192      3000
i 1       0        3       32.224      3000
i 1       0        3       32.256      3000
i 1       0        3       32.288      3000
i 1       0        3       32.32       3000
i 1       0        3       32.352      3000
i 1       0        3       32.384      3000
i 1       0        3       32.416      3000
i 1       0        3       32.448      3000
i 1       0        3       32.48       3000
i 1       0        3       32.512      3000
i 1       0        3       32.544      3000
i 1       0        3       32.576      3000
i 1       0        3       32.608      3000
e
```

4. This score can be used with the "ex1.orc" orchestra file.

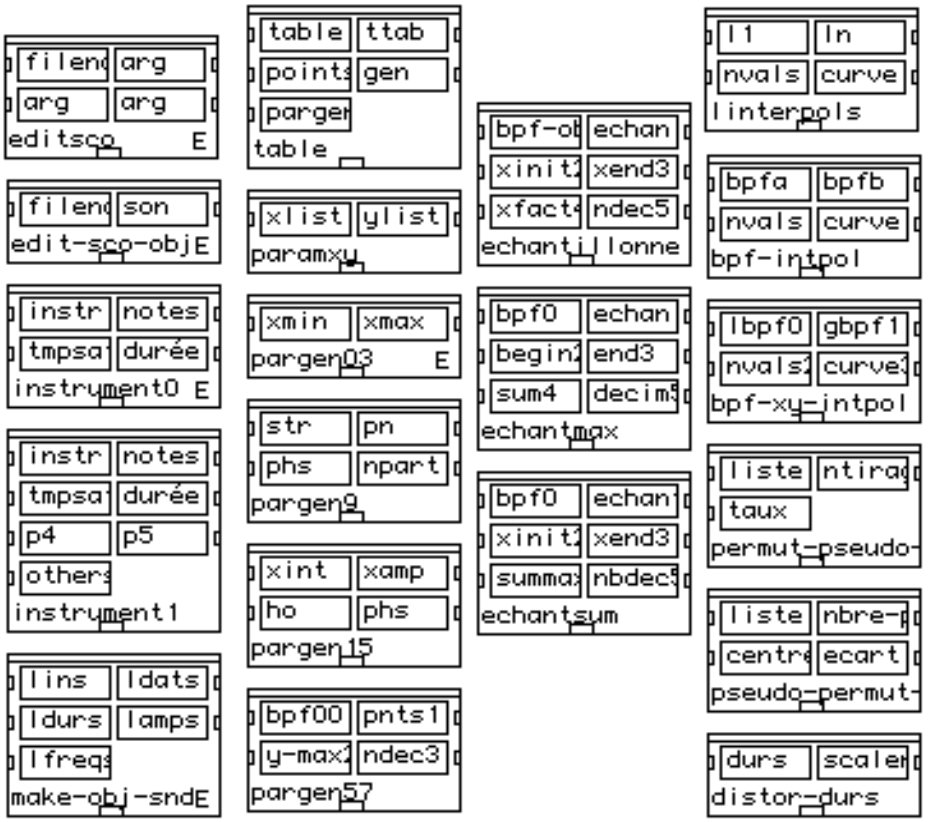
To execute this score in PatchWork, one may construct the following patch :



in which the **table** modules are used to format and edit the data for the "f" statements (envelopes, waveforms, etc.). The **instrument0** or **instrument1** modules are used to format and edit data for the "i" statements, and the **editsco** module puts this information together and prints it in a file of the user's choice.

The **instrument0** module, like the **editsco** module, can be extended to increase the number of entries (either to have more parameters for "i" statements, or to have more tables or instruments).

The **Csound-edit-sco** library contains the following menus and modules:



See appendix A3 for a complete **Csound-edit-sco** menu layout.

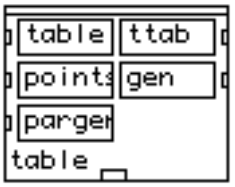
* In the illustration of modules, some modules appear in several forms, depending on how they have been extended (by option-click on the *E* at the right of the module).

2 Constructing Tables

Csound has a number of GEN functions, that generate complete tables. The reader is invited to refer to Barry Vercoe's Csound manual for a comprehensive description of all GEN subroutines. This chapter assumes that the reader is familiar with these subroutines.

table

formatting data for a function



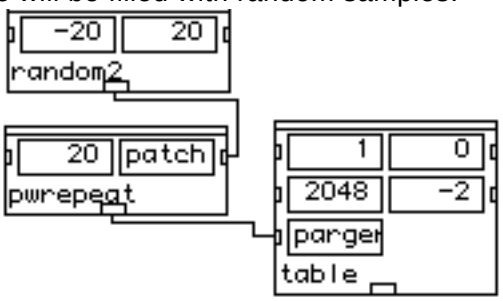
Inputs

table	(number)	table number (p1)
ttab	(number)	moment, in seconds, for triggering off the table (p2)
points	(number)	points in the table (p3)
gen	(number)	number of the GEN subroutine (p4)
pargen	(number)	list containing auxiliary parameters (p5, p6, p7, ...), specifying values for the GEN subroutine

The **table** module is used to generate Csound's function parameters. These tables are input data to GEN subroutines. The input *pargen* requires a particular format for its parameters, according to the number of the GEN subroutine used.

This format may also be defined via standard PatchWork modules, particularly the specialized modules whose names start with "pargen".

These parameters are simply a list of values obtainable with any coherent algorithm. Here is an example on how to generate a GEN02 table. The table will be filled with random samples:



output[5]

```
PW->((epw::f 1 0 2048 -2)
(-14 0 -7 -3 9 -15 13 -1 10 -9 -8 7 -16 10 1 3 -6 8 3 -10))
```

The *gen* value must be negative if one does not wish to normalize the parameters.

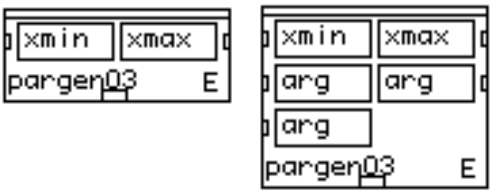
There are special cases of GEN subroutines, that are treated in PatchWork by specialized "pargen" modules; they are listed below.

5. The following examples of outputs show only the generation of "tables", and the output format does not yet match the format required by the Csound score. The final format will be realized by the module **editsco**, to which are connected the tables and the instruments.

pargen03

formatting data for a GEN03

(extensible module)



Inputs

xmin	(number) minimal values (x_{min}-p_5)
xmax	(number) maximal values (x_{max}-p_6)
args	(numbers) coefficients of a polynomial

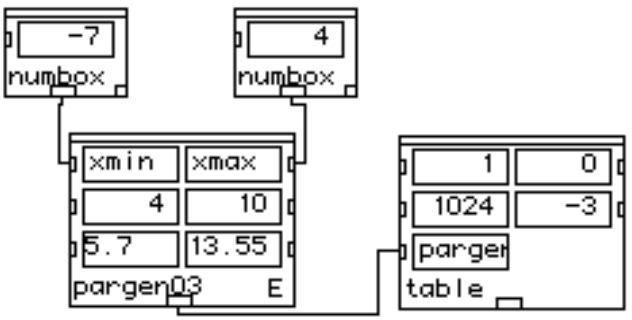
The GEN03 subroutine generates a table by evaulating a polynomial curve. The parameters used by GEN03 are the minimal values (x_{min} - p_5) and maximal values (x_{max} - p_6) of "x" (abscissa), that is, of the interval in which the polynomial is defined, and the parameters p7, p8, p9, etc. are the coefficients of the chosen polynomial:

$$p_7 + p_8*x + p_9*x^2 + p_{10}*x^3 + ... + p_n * x^{(n-7)}$$

One can have as many inputs *arg* as one wishes, depending on the degree of the polynomial. A second degree polynomial ($p_7 + p_8*x + p_9*x^2$) requires three *arg* inputs, a third degree polynomial requires four, etc.

Example: using the third degree polynomial for constructing a polynomial table:

$$4 + 10*x + 5.7*x^2 + 13.55*x^3$$



Output of the patch :

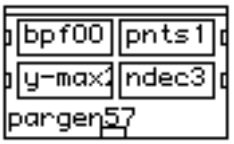
```
PW->((epw::f 1 0 1024 -3) (-7 4 4 10 5.7 13.55))
```

The same approach is applied for programing GEN13 and GEN14 routines.

* Csound's GEN05 and GEN07 can be generated by a specialized module: **pargen57**. This module can also be used for GEN08. These GEN subroutines construct functions by linear segments (GEN07) or by exponential segments (GEN05). For these routines, **pargen57** formats data from a PatchWork BPF (Break Point Function).

pargen57

formatting data for a GEN05 or a GEN07



Inputs

bpf-ob	(BPF)	input from a BPF table
pnts	(number)	number of points in the table (p3)
y-max	(number)	parameter for formatting, which must have the same *y-max* value as the BPF table (scaling parameter), and which gives the degree of precision specified by the ordinates
ndec	(number)	number of decimals in the newly formed parameters

To use a BPF connected to this module, it is necessary to format the BPF in this way:

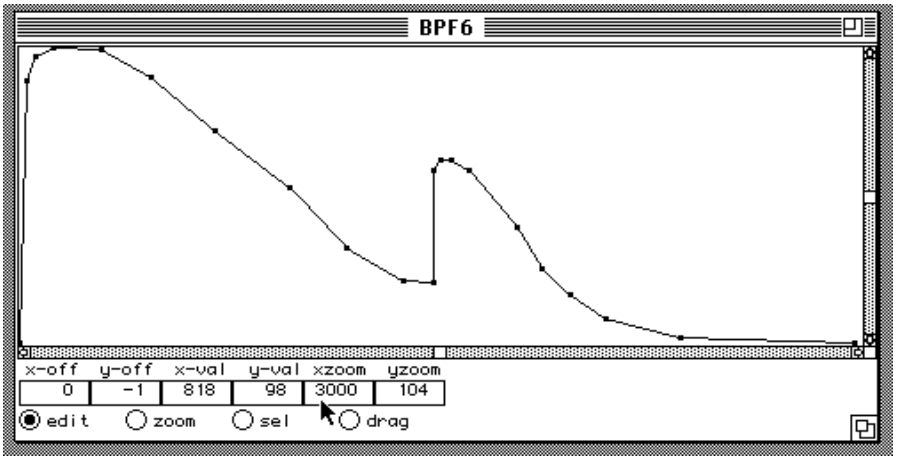
- the *x-off* and *y-off* values must be zero.
- *yzoom* must equal *y-max*. It is important to know wether one wants to define the edited values or not, because, if the 'Gen' parameter of the module **table** is negative, the *y-zoom* and *y-max* values will indicate precisely the maximum value edited, i.e. that values will be rescaled to the maximum (this is called normalization). If the 'Gen' of the module **table** is negative, the *y-zoom* and *y-max* values will indicate the maximum value on a proportional scale, i.e. no normalization will take place.
- *xzoom* can be formated according to the needs of the user. For example, if one wishes to construct envelopes, the *xzoom* should be defined in p3.

Example:

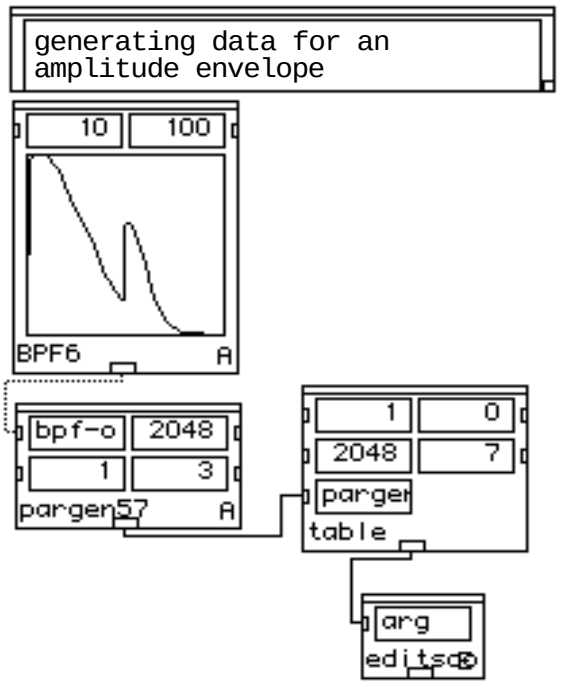
For formatting the data of an amplitude envelope for a sound three seconds long (p3 = 3 seconds), one would indicate :

x-off :	0 (zero)
xzoom:	3000, signifying a precise envelope defined in milliseconds
y-off:	should generally equal zero, but might equal -1 so as to have, for example, more space for editing
yzoom:	could be defined quite precisely, or simply left equal to 100 (thus having the amplitude in percentage) opting for a normalization ('Gen' is positive)[6] ; one might also give *yzoom* a value of 104 for convenience in editing

6. *ndec* should be consistant with the desired precision.



Here is a patch that generates an amplitude envelope. Note that it uses a GEN07 (specified in the **table** module). The data are prepared by the **pargen57** module.



output of the patch:

```
PW->((epw::f 1 0 2048 7)
       2.3.1. 18.0 0.889 20.0 0.97 43.0 1.0 121.0 0.99 122.0 0.899 155.0
0.717 181.0 0.525 143.0 0.323 140.0 0.212  71.0 0.202 0.0 0.586 16.0 0.616
28.0 0.616 44.0 0.586 115.0 0.394 65.0 0.253 69.0 0.162 84.0 0.081 184.0
0.02 429.0 0.0))
```

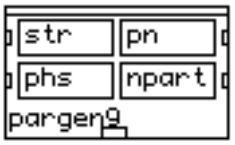
Programing the parameters for the GEN05 subroutine is done in the same way as for the GEN07 subroutine, except that the line segments, drawn with the BPF, will actually be exponential segments.

* Only values **strictly** positive are accepted as ordinates.

The generation of a GEN08 table is done much like the GEN07 and GEN05 modules.
Let's now study a last example of the **table** module: the generation of wave forms composed of the sum of sine waves. In Csound, GEN09 and GEN10 are used for this purpose. We will explain how to generate data for GEN09**.** This subroutine allows one to define three parameters of the harmonics: the relative intensities, the phases, and the ranges of the harmonics, not necessarily in whole numbers. The module **Pargen9** formats the data.

pargen9

formatting data for a GEN09



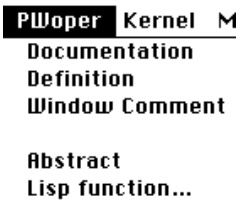
Inputs

str	(list or number)	intensites of the harmonics
pn	(list or number)	proportions of the harmonics (in comparison with the first harmonic)
phs	(list or number)	values of the phases
npart	(number)	number of decimals for the new parameters.

The input values of the pargen9;[7] module may be specified in lists, in tables, or by particular algorithms. The following example generates relative intensities *(str)* of the harmonics through sampling a BPF, as in the case of the **GEN10** subroutine. The series of 24 rows of harmonics, with a degree of 1.15 of distortion, is generated by the module **inharm-ser.** This module produces a series of frequencies following the formula:

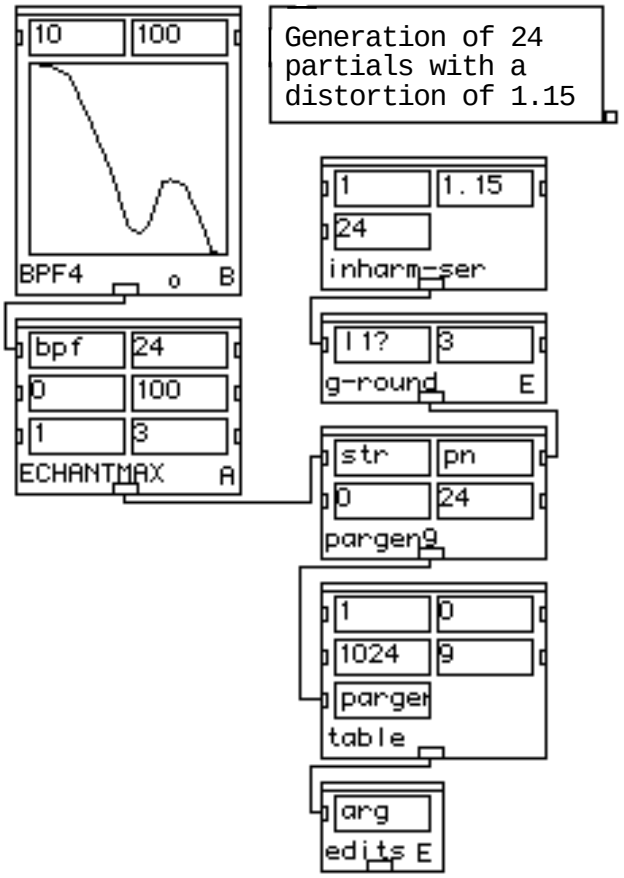
harmonic = fundamental * row distortion.

In our example the frequency of the fundamental will be "1", so that the generated list falls in "rows". At the same time, the module **g-round** is used to limit the number of decimals (here three) so as not to overload the score. Note also the presence of a module called **inharm-ser**. This module is not readily available from any PatchWork library menu. It might be of interest to see how to load such modules. Because it lives only as Lisp code, it has to be loaded through the **Lisp function...** option of the **PWoper** menu.



This opens a dialog window, in which the following command line must be entered : `cs-e::inharm-ser.` The **inharm-ser** module will appear in the current PW window, and it is possible to realize this patch.

7. The name of this module will be changed to **pargen09** in the future, to better follow the usual Csound syntax. Nevertheless, the forthcoming **pargen09** will remain compatible with the older module name, **pargen9**.

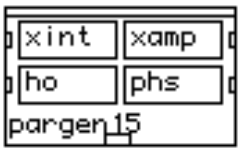


Output of the patch:

```
PW->((epw::f 1 0 1024 9)
2.4.1. 0.074 0 2.219 0.074 0 3.537 0.073 0 4.925 0.072 0 6.365 0.071 0 7.85
0.067 0 9.373 0.062 0 10.928 0.057 0 12.514 0.05 0 14.125 0.043 0 15.762
0.036 0 17.42 0.027 0 19.1 0.021 0 20.799 0.02 0 22.517 0.024 0 24.251 0.03
0 26.003 0.036 0 27.769 0.036 0 29.551 0.035 0 31.346 0.029 0 33.155 0.022
0 34.977 0.016 0 36.812 0.013 0 38.658 0.012 0))
```

pargen15

formatting data for a GEN15

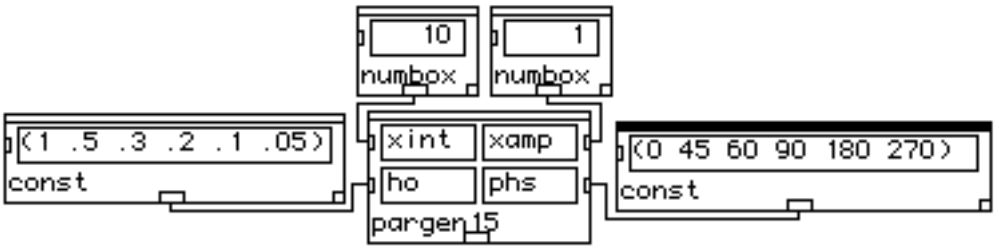


Inputs

xint	(number)	lower limit of the interval defining the polynomial (-*xint*)
xamp	(number)	upper limit of the interval defining the polynomial (+*xint*)
ho	(list or number)	list of relative intensities of the harmonics
phs	(list or number)	list of phases

The module **pargen15** is used to generate a table using for GEN15, which creates two polynomial tables.

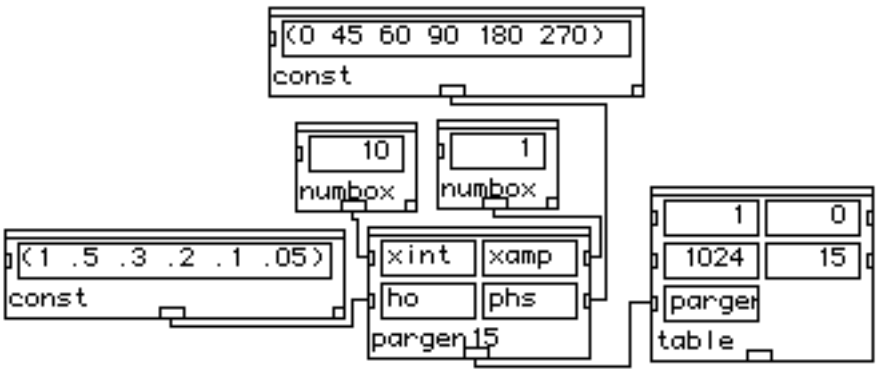
• First stage



output

```
PW->(10 1 1 0 0.5 45 0.3 60 0.2 90 0.1 180 0.05 270)
```

• Second stage



Output of the patch :

```
PW->((epw::f 1 0 1024 15) (10 1 1 0 0.5 45 0.3 60 0.2 90 0.1 180 0.05 270))
```

ho and *phs* can be generated either by a BPF table or by specific algorithms.

In cases where there are many harmonics, one can use the module **echantmax**[8] which samples a BPF and declares the values of the ordinates in conjunction with the parameter *max.*

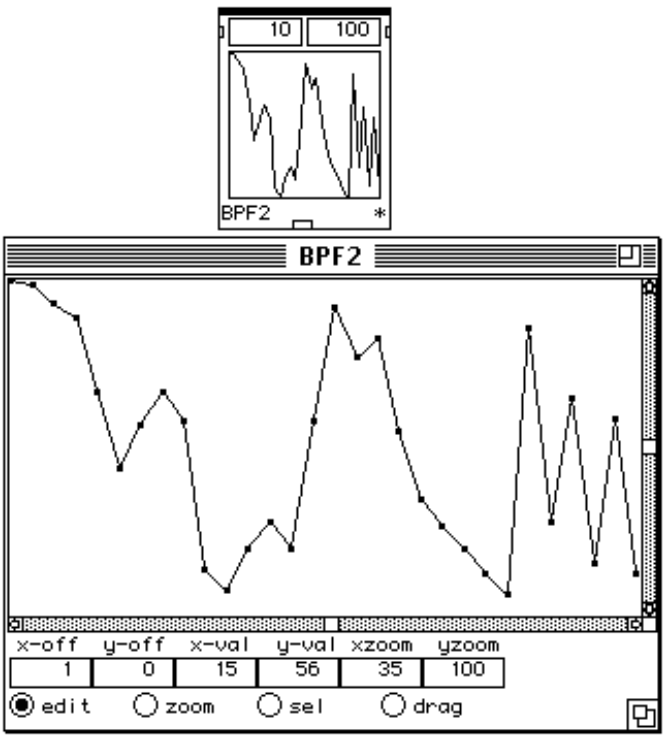
echantmax

sampling a BPF between O and max

Inputs

bpf-ob	(BPF)	input from a BPF table
echant	(number)	number of samples desired
xinit	(number)	minimum value of "x" used in the table (linked with *x-off* parameter of the BPF)
xend	(number)	maximum value of "x" (linked with the *xzoom* parameter of the BPF)
max	(number)	maximum value of the ordinates
nbdec	(number)	number of decimals in the samples for printing score files

Example: to indicate the amplitudes of 30 harmonics, the BPF table would have the following configuration:



in which are indicated

-off = 1 (beginning with the first harmonic)

---

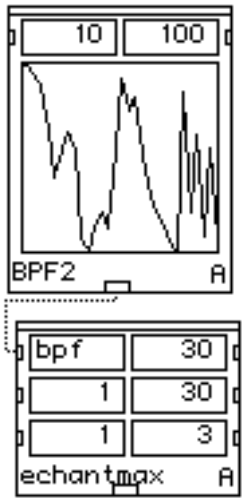
8. The **echantillonneur** module can also be used here.

y-off = 0 (the minimum value of the amplitude is zero)

xzoom = 30 (number of harmonics to be defined)

yzoom = 100 (values of amplitudes in relation to a reference amplitude)

The BPF is connected directly to the module **echantmax**:



Please note : the number of points to be defined *must be equal* to the number of samples.

Output of the patch:

```
PW->(1.0 0.99 0.93 0.89 0.67 0.44 0.57 0.67 0.58 0.14 0.08 0.2 0.28 0.2
0.58 0.92 0.77 0.83 0.55 0.35 0.27 0.2 0.13 0.07 0.86 0.28 0.65 0.16 0.59
0.13)
```

echantsum

sampling a BPF with a fixed sum of values

Inputs

bpf	(BPF)	input from a BPF table
echant	(number)	number of samples one wishes to obtain
xinit	(number)	minimum value of "x" in the table (specified by *x-off*)
xend	(number)	maximum value of "x" (specified by *xzoom*)
summax	(number)	maximum value for the sum of the ordinates
nbdec	(number)	number of decimals in the samples

Output

a list of redistributed abscissas

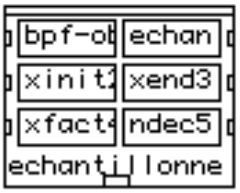
echant = 64	number of samples (échantillons) desired
xinit = 1	sampling begins with x = 1 (*x-off* = 1 in **bpf-edit**)
xend = 64	sampling ends with x = 64 (*xzoom*= 64 dans **bpf-edit**)
sum max= 32000	maximum value for the sum of the amplitudes
nbdec = 0	number of decimals in the output

The amplitude of the output can not be exactly the same as the parameter *sum*, since this value would only be obtained if all the harmonics had maximum amplitude. The amplitude of the output is directly related to the area under the BPF curve. Practice has shown that the total amplitude of the output is generally about half of the *sum*.
The module **echantsum** is similar to the module **echantmax**: the sampled list is given new values in such a way that the sum of these values equals the parameter *sum*[9].
One last module permits one to obtain data by means of a BPF: the module **echantillonne**. This module resembles **echantmax** and **echantsum**. The input parameter *fact* is a factor which multiplies the ordinates: If the format of the table used is *y-off*=0 and *yzoom*= 1000, and if *fact* is equal to 1, the output will range between 0 and 1000. If *fact* is equal to two (2), the output will fall between 0 and 2000.

9. This module is most useful for controlling individual amplitudes within additive synthesis.

echantillonne

sampling a BPF



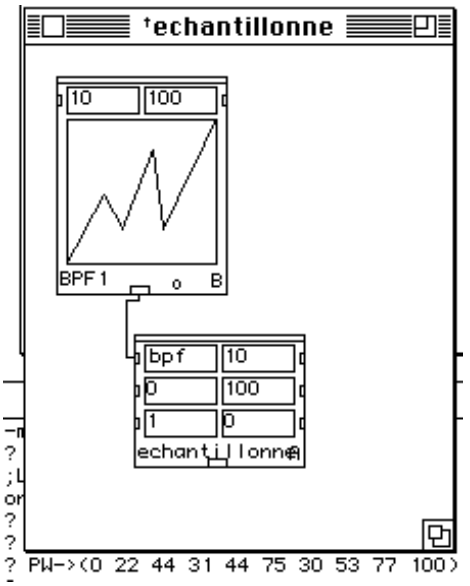
Inputs

bpf-ob	(BPF)	input from a table BPF
echant	(number)	number of samples (*échantillons*) to be obtained
xinit	(number)	minimum value of "x" in the table (specified by *x-off*)
xend	(number)	maximum value of "x" (specified by *xzoom*)
fact	(number)	factor multiplying the ordinates
nbdec	(number)	number of decimals in the samples

Output

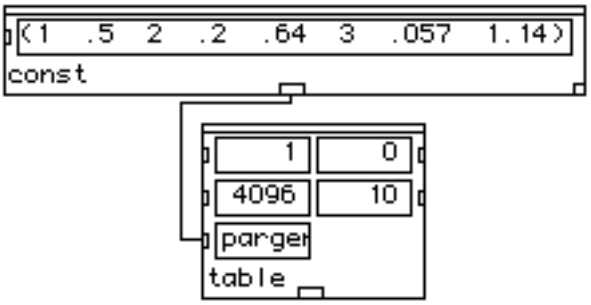
a list of rescaled ordinates

Here is a simple example of a BPF function, and the generation of parameters in the Listener window through the **echantillonne** module:



An example with GEN10

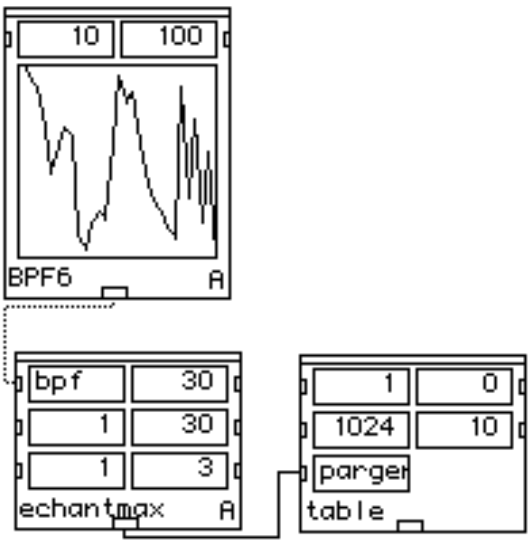
The supplementary parameters of GEN10 are the relative intensities of the harmonics. They can be formatted in a simple list or from a table. The example uses the modules **table** and **echantmax**.



Output of the patch:

```
PW->((epw::f 1 0 4096 10) (1 0.5 2 0.2 0.64 3 0.057 1.14))
```

Here is a patch for indicating the amplitudes of 30 overtones of a sound wave with a BPF:



Output of the patch :

```
PW->((epw::f 1 0 2048 10)
 2.6.1. 0.99 0.93 0.89 0.67 0.44 0.57 0.67 0.58 0.14 0.08 0.2 0.28 0.2 0.58
 0.92 0.77 0.83 0.55 0.35 0.27 0.2 0.13 0.07 0.86 0.28 0.65 0.16 0.59 0.13))
```

param-xy

formats two lists in the form of table parameters

Inputs

xlist	list	gives the values on the abscissa (temporary values or action time)
ylist	list	gives the ordinate values (amplitudes, for example)

Output

list where y values are separated with time values. these time values are intervals between x values (action times).

Here is an example on the use of param-xy.
Given the inputs :

xlist (0, x1, x2, x3, ..., xn). The first element of xlist is always 0
ylist (y1, y2, y3, ...,yn), any values

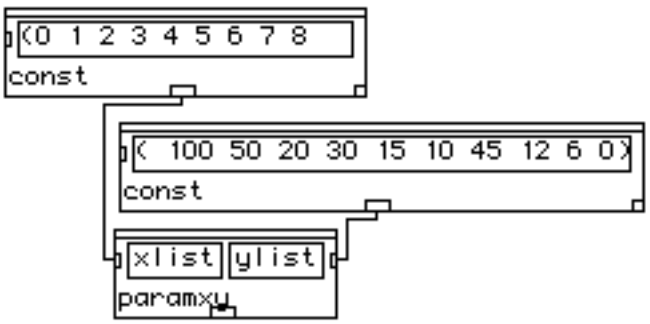
the output will be:

(0. y1 x1 y2 D(2_1) y3 D(3_2) y4 ... D(n_n-1) yn)

where

D(n_n-1) = Xn - Xn-1 (duration between two values of data)
D(2_1) = the interval between x2 and x1.

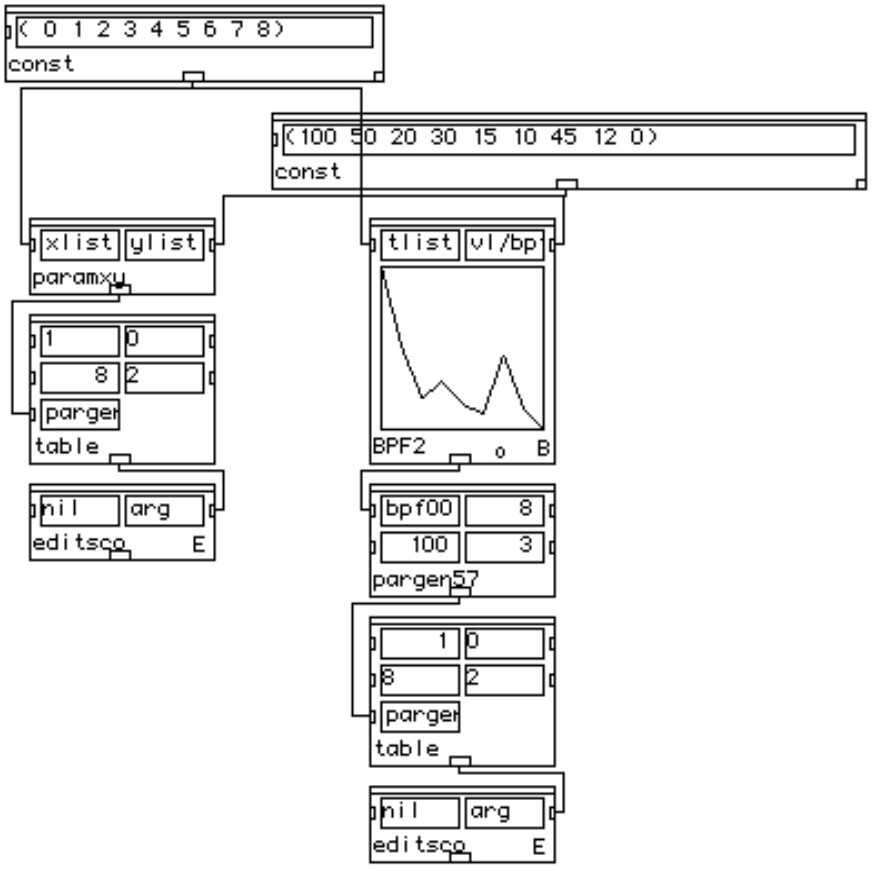
Example: list "x" is a regular data list and list "y" is another list of associated values: :



Output of the patch :

PW->(100 1 50 1 20 1 30 1 15 1 10 1 45 1 12 1 6 1 0)

As a demonstration on how to build tables, we'll construct a complete patch from the example above. The patch is shown next. The BPF displays graphically the pair of data, and **pargen57** is used to build the Csound table through GEN02. Another possible path to accomplish the same result is shown on the right-hand side of the patch; it uses a **paramxy** module linked to a **table** module.

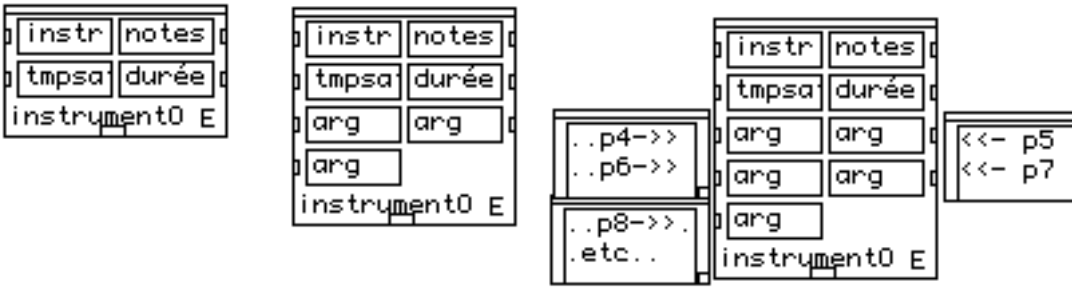

(0 1 2 3 4 5 6 7 8)
const
(100 50 20 30 15 10 45 12 0)
const
xlist
ylist
paramxy
1
0
8
2
parger
table
nil
arg
editsco
E
tlist
vl/bp
BPF2
o
B
bpf00
8
100
3
parger
1
0
8
2
parger
table
nil
arg
editsco
E

3 Note Statements

There are three modules for preparing note lists: i.**instrument0**;, **instrument1**, and **make-obj**- **snd**.

instrument0

formats data for note statements



(extensible module)

Inputs

instr	(number)	number of the instrument
notes	(number)	number of notes to be generated
tmpsa	(list or number)	list of attack times. If this parameter is a number, and not a list, then all notes will be played at the same time
durée	(list or number)	list or number indicating the duration of the event
args	(list or number)	supplementary parameters (p4, p5, ...)

Output

a list for printing

instrument1

formats data for note statements

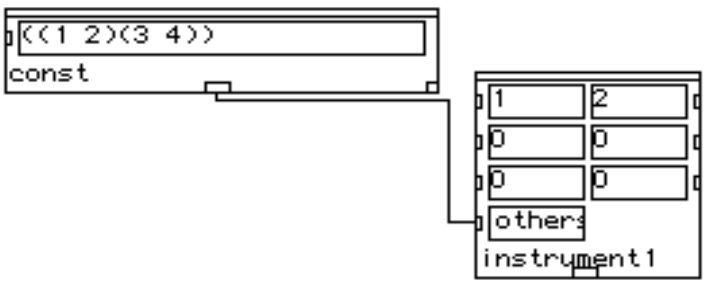
Inputs

instr	(number)	number of the instrument
notes	(number)	number of notes to be generated
tmpsa	(list or number)	list of attack times. If this parameter is a number, and not a list, then all notes will be played at the same time
durée	(list or number)	list or number indicating the duration of the event
p4	(list or number)	parameter p4
p5	(list or number)	parameter p5
others	(list or number)	supplementary parameters (p6, p7, etc.) given together either as a simple list or as a list of lists.

Output

a list for printing

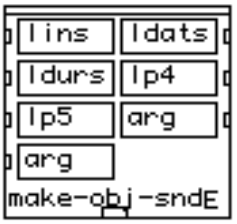
Instrument1 prepares 'i' note statements requiring many different parameters as, for example, in the case of filtering data. Note that the fourth and fifth parameters are called 'p4' and 'p5', although Csound does not provide any special meaning to these p-fields. The use of **Instrument1** implies that the corresponding orchestra has at least six p-fields.
Here is an example patch of list processing by **instrument1**. In this case, parameters 6 and 7 are set by the lists from the **const** module.



make-obj-snd

formats data for note statements

extensible module



Inputs

lins	(list)	number of the instrument
ldats	(list)	list of data initiating the notes (in seconds)
ldurs	(list)	list of note durations (in seconds)
args	(lists)	supplementary parameters (p4, p5, p6, etc.)

Output

an "object" to be connected to an **edit-sco-obj**

make-obj- snd is an extensible module delivering output in the form of a CLOS object, containing all the information necessary for "i" statements. With this module the number of notes is calculated automatically according to the number of elements in the list *lfreqs.* It works similarly to the PatchWork module **chordseq**, depending on the nature of the data provided in *lp4.*

* The fourth and fifth parameters were previously named *lamps* and *lfreqs*, respectively; it implied that there parameters were expressing amplitude and frequency data. Because Csound does not provide such definition, as only the first three parameters have special, fixed function, the fourth and fifth parameters are now simply called in a general fashion, *lp4* and *lp5*.

Parameter p5, however, has a retained a special role : it uses data structure as lists of lists, similar to the structure used by PatchWork's **chordseq** module. For instance :

- a simple list '(440 660 880) indicates three successive notes;
- a double list '((440 660 880)) indicates a chord;
- a list of lists '((440 660 880)(550 1100)(660 1320 1980)) indicates three successive chords;

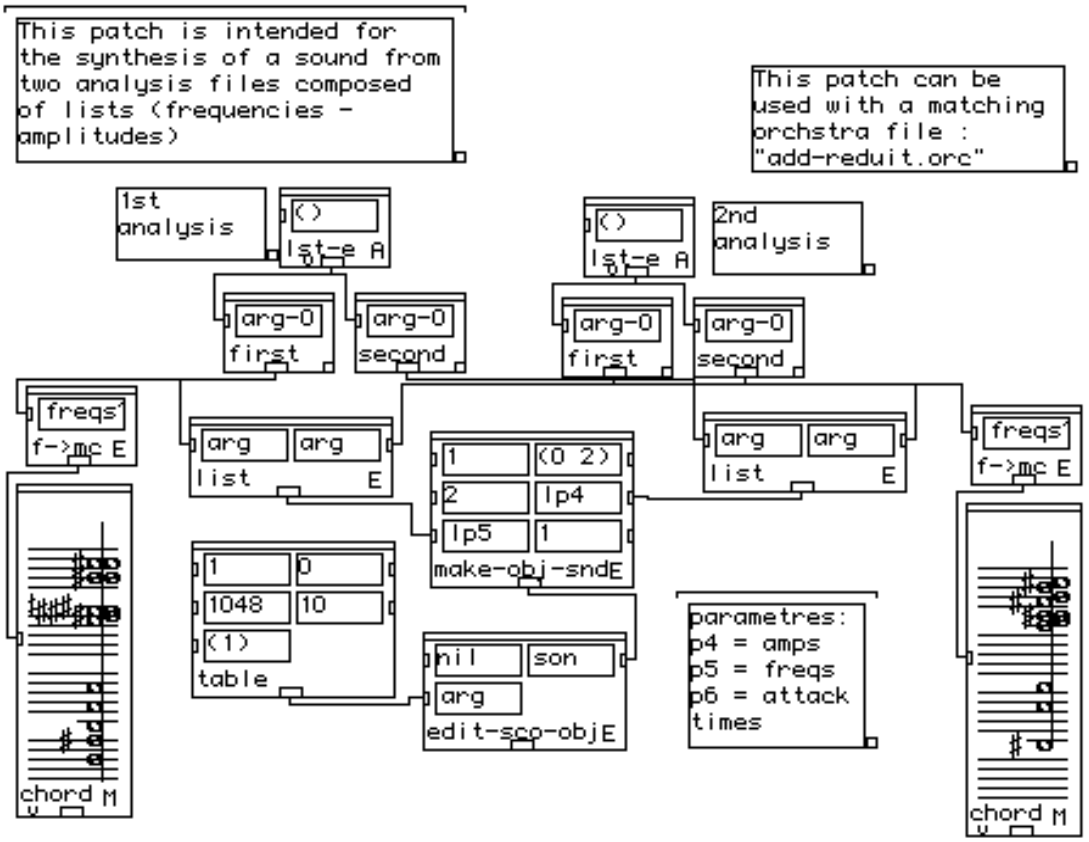
For this last example, other parameters than p5 can be expressed in four different manners:

- a value : 1 or '(1) - this value will be attributed to each note of a each successive chord;
- a list : '(1 2 3) - each element of the list will be attributed in this fashion :
  '((1 1 1)(2 2)(3 3 3))
- a double list : '((1 2 3)) - this lists is attributed to each chord :
  '((1 2 3)(1 2)(1 2 3))
- a list of lists : '((1 2 3)(4 5)(6 7 8)) - the elements of the lists are dispatched on the notes of the chords :
  '((1 2 3)(4 5)(6 7 8))

We suggest now methods of using these different modules in two different situations:

- programing sequences
- .i programing chords or spectra;

As an example, here is a patch called **iana-> csound.pw**:


This patch is intended for the synthesis of a sound from two analysis files composed of lists (frequencies - amplitudes)
This patch can be used with a matching orchstra file : "add-reduit.orc"
1st analysis
2nd analysis
lst-e A
arg-0 first
arg-0 second
freqs
f->mc E
arg arg list E
1 (0 2) 2 lp4 lp5 1 make-obj-sndE
1 0 1048 10 (1) table
nil son arg edit-sco-objE
parametres: p4 = amps p5 = freqs p6 = attack times
chord M

4 Generating the score

Their are two modules at the heart of this library: it is from **editsco** that all the data coming from the other modules are formatted and printed in the score file. A second module **edit-sco-obj** has equivalent functions but more specific utilisations, since it permits some "notes" to be printed by extracting the data contained in the module **make-obj-snd**;.

editsco

prints data in a disk file

(extensible module)

Inputs

filename (text) name of the new file

args (lists) lists of data to be printed

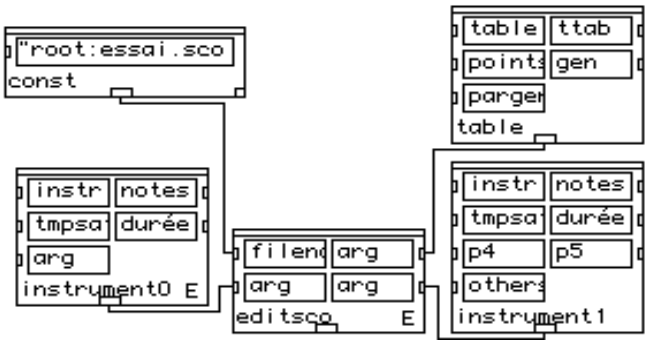
output

transfer of the data to a disk file

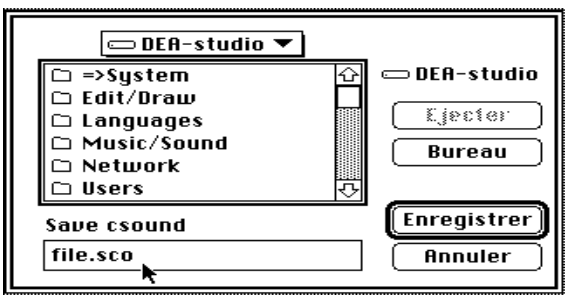
editsco gets its input from modules which process note statement, such as **instrument0** or **instrument1**, and from modules that provide note function statemenrs (**table**, **pargen**, etc.). **editsco** also permits printing the "e" statement at the end of the score.

* Note that it is possible to send **editsco** any score statements defined in Csound, such as comment opcodes (';'), advance statements ('a'), tempo statements ('t'), and section statements ('s').

Here is a patch example on how to create a complete score file unto the hard disk.



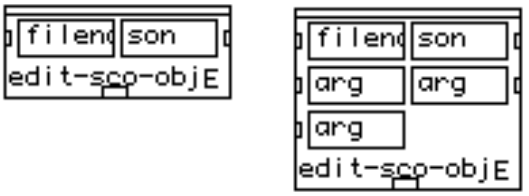
After having assembled and connected various modules to **editsco**, one evaluates the module**,** and a file whose name is marked in the first entry *(filename)* is created to receive the data produced by the patch. If a file by that name existed already, it is now erased, and if no file name is given, a Macintosh dialogue window appears, in which one must name the file, indicate its destination, and click on the option **enregistrer** (save).



edit-sco-obj

prints data in a disk file

(extensible module)



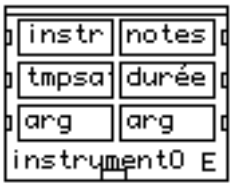
Inputs

filename (text)	name of the new file	
son	(sound -obj)	output from the module **make-obj-snd**
args	(lists)	lists of data to be printed

Its input *son* (sound object) can receive the output from a module **make-obj-snd** or from a list of these modules.

Programing Notes Statements [10]

Given a score for the instrument ex1.orc (see appendix), the note statements are composed of five p-field (p4 = frequency and p5 = amplitude), **instrument0** module would be used in this configuration.



It is important to mention that one has the possibility of controlling each parameter separately, and that it will be necessary to declare:

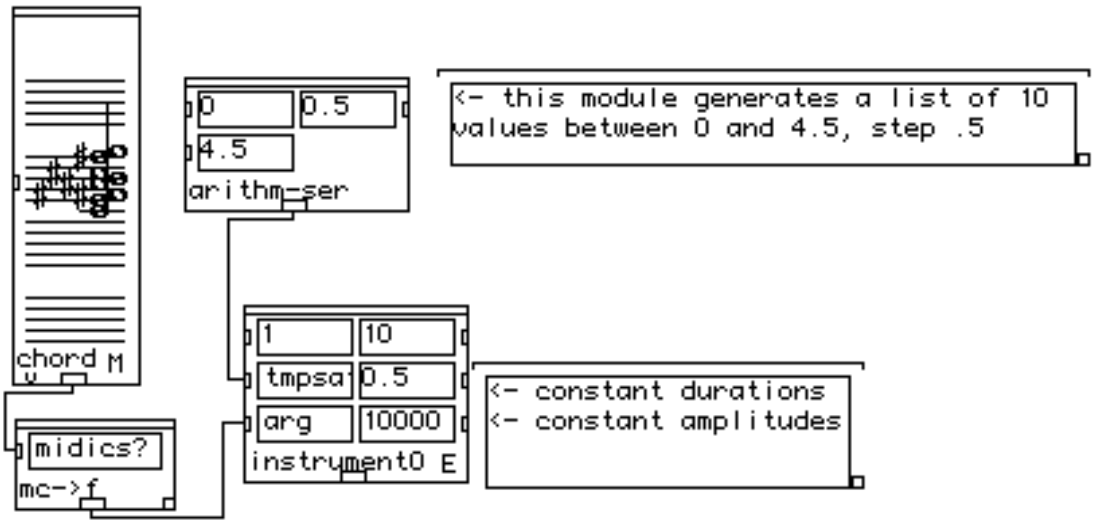
- a list of attack times (p2)
- a list of durations (p3)
- a list of frequencies (p4)
- a list of amplitudes (p5)

When one parameter must be the same for the whole sequence, as for example, the amplitude or the duration, it is not necessary to provide the entire list. The desired value can be simply entered once in the corresponding input.

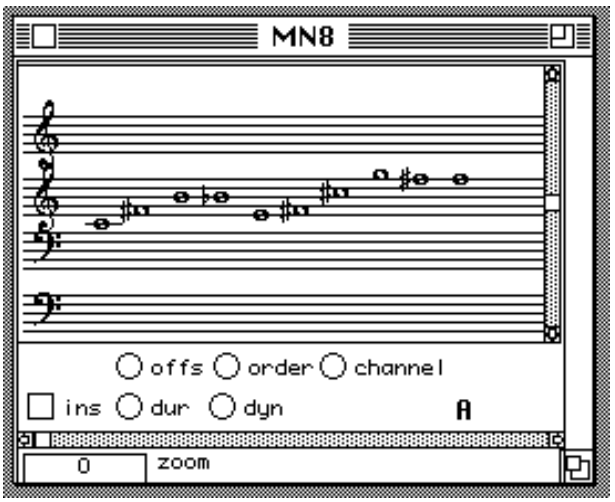
10. The following examples show only the editing of note statements. See note 5.

Example 1

Here is a sequence of 10 frequencies taken from a module **chord**, with amplitudes and durations constant and the rhythm regular:



The frequencies can be programed in musical notation within the module **chord**, when it is open:



These pitches in midicents values must then be converted into frequencies by the module **mc->f**.

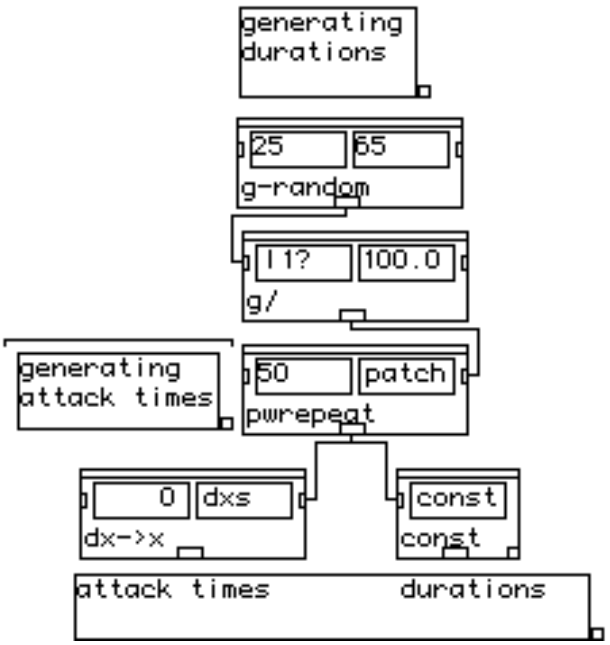
output:

```
PW->((epw::i 1 0 0.5 261.6255653005986 1000)
(epw::i 1 0.5 0.5 329.6275569128699 1000)
(epw::i 1 1.0 0.5 369.9944227116344 1000)
(epw::i 1 1.5 0.5 369.9944227116344 1000)
(epw::i 1 2.0 0.5 466.1637615180899 1000)
(epw::i 1 2.5 0.5 493.8833012561241 1000)
(epw::i 1 3.0 0.5 554.3652619537442 1000)
(epw::i 1 3.5 0.5 698.4564628660078 1000)
(epw::i 1 4.0 0.5 739.9888454232688 1000)
(epw::i 1 4.5 0.5 783.9908719634985 1000))
```

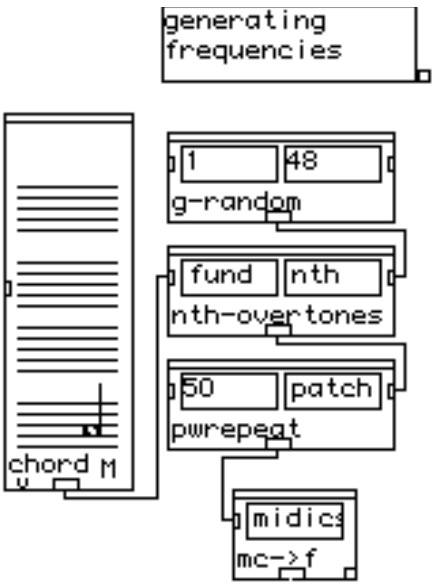
Example 2

Here is a more complex situation for notating a sequence of 50 sounds chosen among the first 48 harmonics of C0 (2400 midicents), with some variations:

- durations are a list of values between 250 and 750 milliseconds with an equal distribution and a mean average of 500 milliseconds (or 0.5 seconds)
- attack times are calculated from a list of durations using the module **dx->x**, which increments the values of a given list



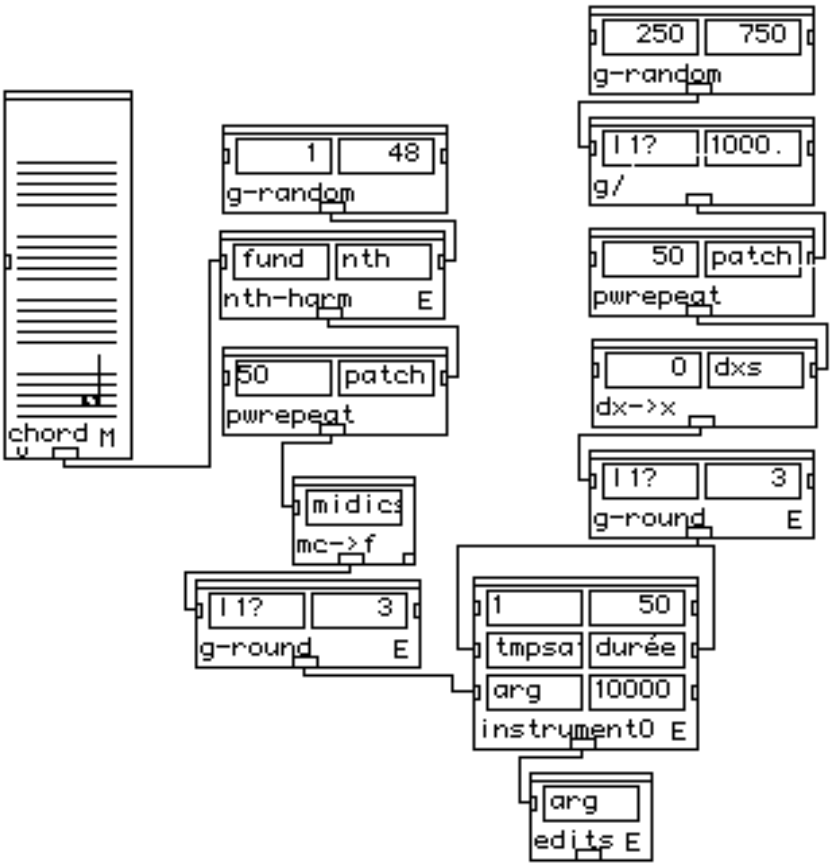
Here is the section of the patch generating the frequency list:



The module **nth-harm**[11] calculates a harmonic with an random position from 1 to 48 (chosen by **g-random**) over a fundamental at 2400 midicents. This operation, repeated 50 times using the module **pwrepeat**, generates the following list,

```
PW->(6204 9066 6840 8168 9066 3600 7304 3600 8652 8346 4302 6204 7828 8870
8454 5768 8346 8288 8168 7752 7586 8830 8504 8454 6840 7498 8230 4800 7498
8870 7972 7752 6386 8400 5768 2400 8830 8652 8912 6000 7404 8504 8830 6552
8742 6702 7088 7752 5186 9102)
```

This list must then be converted into frequencies by the module **mc->f**. The complete patch is then connected to one of the inputs of the module **editsco**:



In this example, the number of auxiliary parameters of **instrument0** can be augmented as desired. The modules **g-round** have been added so as not to overload the score.

11. You'll find the module **nth-harm** in the Esquisse library menu. If you want to try this patch, load first the Esquisse library.

Example 3

Now, let's see a more sophisticated example. This is the "instrument1-exemple.pw" file that resides in the **Csound-edit-sco->Patches** folder. Given a score in which

- p4 is the frequency of some source
- p5 is the amplitude of this source

The parameters p6 to p20 contain the formant data for five "reson" type filters, with

- amplitudes from p6 to p10
- frequencies from p11 to p15
- bandwidths from p16 to p20

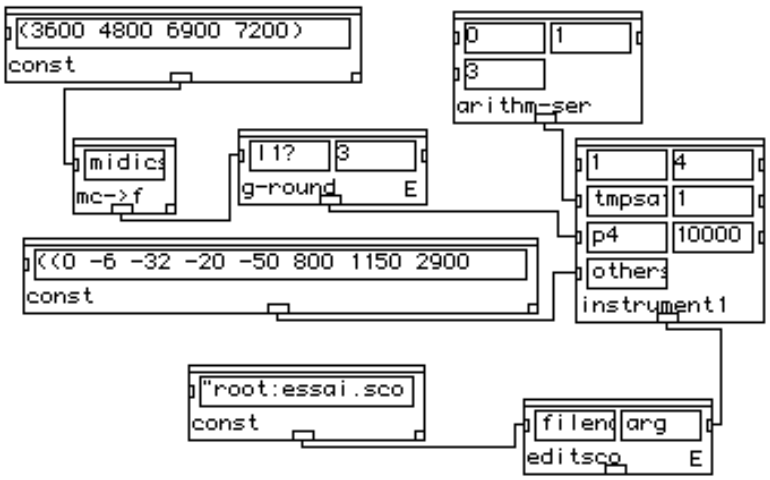
The orchestra used is of the following form:

```
a1   reson   iat*ampdb(80+p6)*am , p11+kv , p16
a2   reson   iat*ampdb(80+p7)*am , p12+kv , p17
a3   reson   iat*ampdb(80+p8)*am , p13+kv , p18
a4   reson   iat*ampdb(80+p9)*am , p14+kv , p19
a5   reson   iat*ampdb(80+p10)*am , p15+kv , p20
```

where

iat	is a factor for the amplitudes,
am	is the source to be filtered,
kv	is an added vibrato.

The notes are declared with the following instrument:



where the input *others* is connected to the module **const** containing a list of lists with the characteristics of the vowel formants (a, i, u and e) for a soprano voice:

```
(((0 -6 -32 -20 -50 800 1150 2900 3900 4950 80 90 120 130 140))
((0 -12 -26 -26 -44 270 2140 2950 3900 4950 60 90 100 120 120))
((0 -16 -35 -40 -60 325 700 2700 3800 4950  50 60 170 180 200))
((0 -20 -15 -40 -56 350 2000 2800 3600 4950  60 100 120 150 200)))
```

The formatted output will be :

```
i 1 0 1 65.406 10000 0 -6 -32 -20 -50 800 1150 2900 3900 4950 80 90 120 130 140
i 1 1 1 130.813 10000 0 -12 -26 -26 -44 270 2140 2950 3900 4950 60 90 100 120 120
i 1 2 1 440.0 10000 0 -16 -35 -40 -60 325 700 2700 3800 4950 50 60 170 180 200
i 1 3 1 523.251 10000 0 -20 -15 -40 -56 350 2000 2800 3600 4950 60 100 120 150 200
```

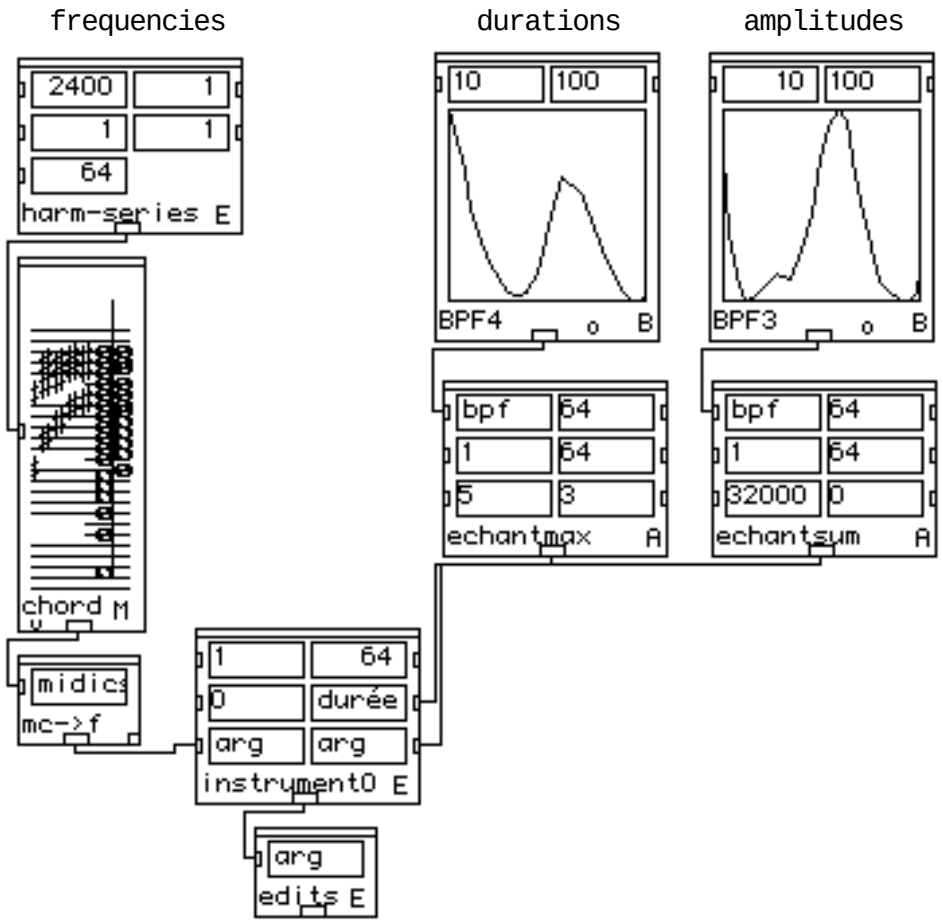
Programing chords and spectra

Example: specifying a spectrum with 64 partials over a fundamental C0 (2400 midicents) where

• the **frequencies** are generated by the module **harm-series**[12] (constructed from a list of 64 harmonics of C-0, or 2400 midicents, ranked from 1 to 64) and converted from midicents into frequencies by the module **mc->f.** . The module **chord** has been connected simply for clarity and is not necessary.

• the **durations** are entered from a table (BPF4) and from the module **echantmax**, which has the following characteristics:

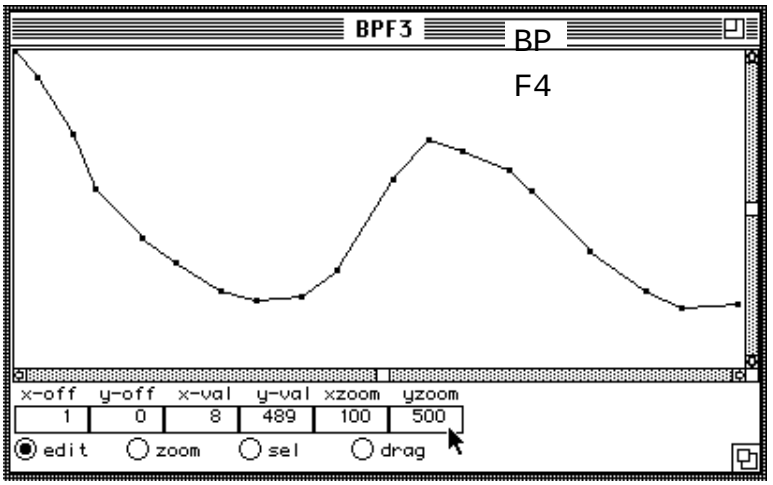
echant = 64	64 samples (*échantillons*) will be taken
xinit = 1	the sampling begins with x = 1 (*x-off* = 1 in **bpf-edit**)
xend = 64	the sampling ends with x = 64 (*xzoom* = 64 in **bpf-edit**)
max = 5	the BPF data are rescaled between 0 et 5
nbdec = 3	the output is accurate to three decimals



12. This module **harm-series** is part of the Esquisse library menu. See note 11.

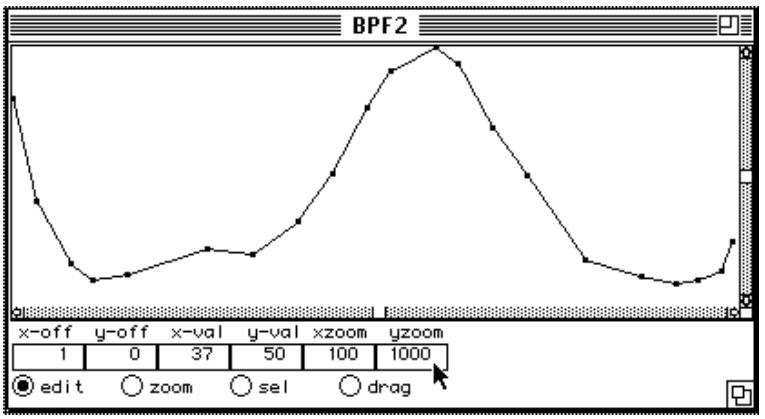
The table of durations, BPF4, can be opened with a double click. Its parameters are the following:

x-off = 1
y-off = 0 (minimum duration of 0 seconds)
xzoom = 64
yzoom = 500 the durations are determined with a precision of 1/100 of a second with a maximum duration of 5 seconds, specified by max = 5)



The **amplitudes** are also taken from a table (BPF3), along with the module **echantsum**. The table BPF3 used to define the amplitudes is structured as follows:

x-off = 1
y-off = 0 (minimum duration of 0 seconds)
xzoom = 64
yzoom = 1000 (the amplitudes are defined in percentage, in relation to a reference amplitude (the 38th harmonic, in this case), with a precision of two decimals.



This is a good example of how this library can facilitate a task requiring control over at least 192 parameters (without counting the data for the phases and the envelope) with:

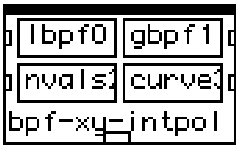
- the manipulation of musical notation for the pitches
- the rapid graphic notation of durations and amplitudes via the PatchWork BPF tables

5 Csound Tools

This supplement to the principal library includes functions for interpolations, permutations and for the generation of random data.
Interpolating BPFs (break-point-functions) for constructing tables can be done with the help of a compiled abstraction: **bpf-xy-intpol**, and a module derived from it : **bpf-intpol**. This collection of interpolating modules also includes **linterpols**, for interpolating lists.
Three modules, **permut-pseudo-rdm**, **pseudo-permut-centre** and **distor-durs** permit one to format permutations and interpolations and to generate aleatoric input.

bpf-xy-intpol

interpolating two BPFs



Inputs

lbpf (BPF) input from a BPF

gbpf (BPF) input from a BPF

nvals (number) number of interpolated values (minimum =1, giving *lbpf)*

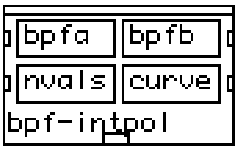
curve (number) interpolation curve, 0 = linear

Output

two lists in one: a list of x values and *nvals* lists of y values

bpf-intpol

Interpolating two BPFs



Inputs

bpfa	(BPF)	input from a BPF table
bpfb	(BPF)	input from a BPF table
nvals (number)	number of interpolated values (minimum = 1, giving *lbpf)*	
curve (number)	interpolation curve, 0 = linear	

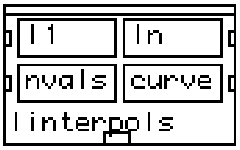
Output

nvals	lists of y values

The module **bpf-intpol** also allows one to make interpolations of BPFs, but this only produces interpolations on the ordinates (values of y) and gives coherent results only for curves having the identical x values.

linterpols

Interpolating two lists



Inputs

l1	(list)	list of values
ln	(list)	list of values
nvals (number)	number of values interpolated (minimum = 1, giving *l1)*	
curve (number)	interpolation curve, 0 = linear	

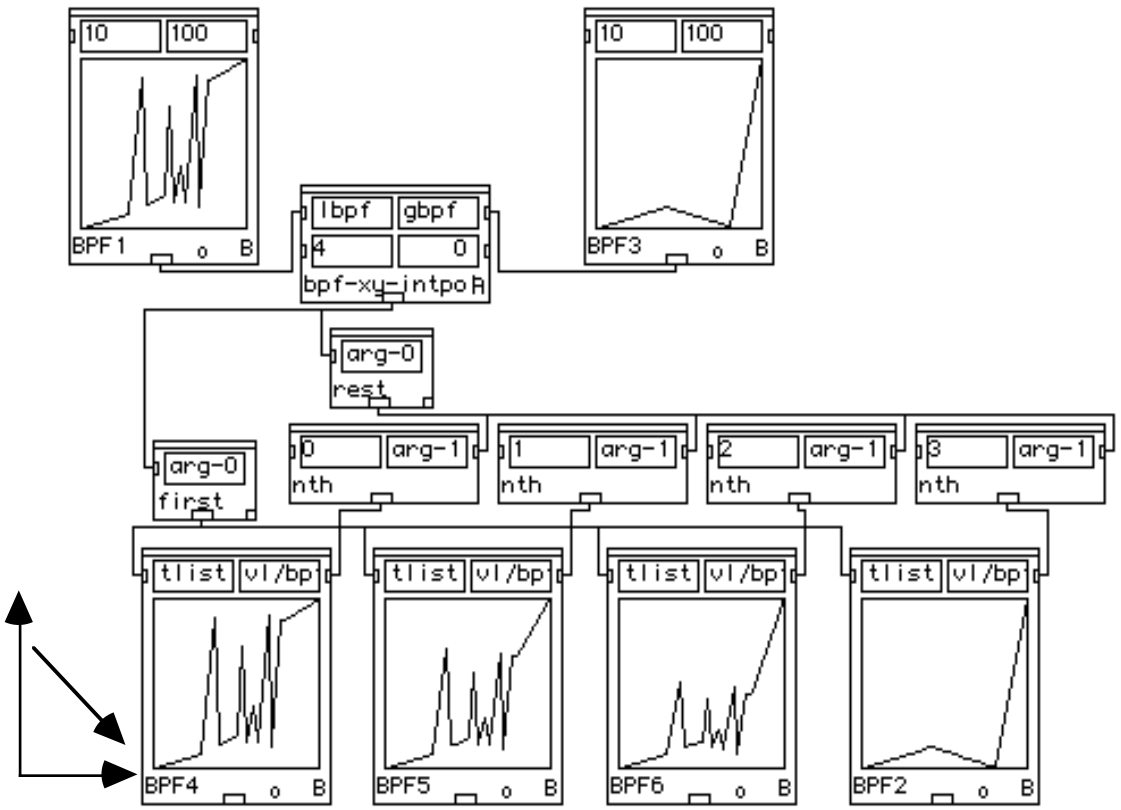
Output

nvals	lists	interpolates between *l1* and *ln*

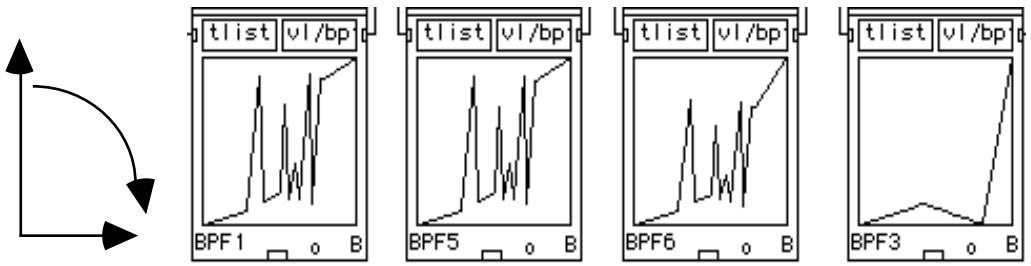
linterpols, does permit interpolations of lists producing a list of n lists interpolated between two lists of initial values.

An example of interpolating tables

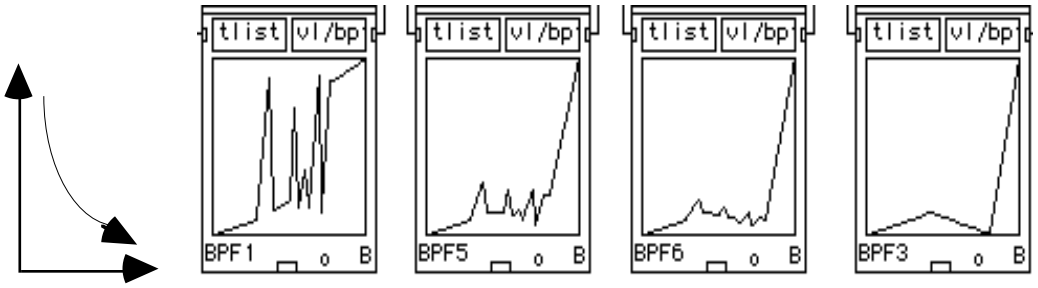
One merely connects two BPFs to this abstraction and indicates the number of BPFs desired at the output, along with the interpolation curve (see below). The output of the abstraction is a list of lists: The first list gives the the ordinate values of all the BPFs, and the following lists give the abscissa values of the consecutive BPFs. Here is the patch "interpol-bpf.pw":



Positive curve (here equal to 2):



Negative curve (here equal to -2):



permut-pseudo-rdm

gives ntirages (*n samples*) of the initial list with more or less numerous permutations, according to the value of the* taux *(rate). This variable can be anywhere between 0 (no permutations) and 1 (only aleatory permutations)

Inputs

liste	(list)	list to be modified
ntirages	(number)	number of samples
taux	(number)	rate of permutations

Example :

```
(permut-pseudo-rdm '(1 2 3 4 5 6 7 8) 4 0.3)
->((1 7 3 4 5 6 2 8) (1 2 6 4 7 3 5 8) (8 2 3 4 1 6 7 5) (1 7 3 4 6 5 2 8))
```

pseudo-permut-centre

function that permutes values around one central value (*centre*) according to variable degrees of dispersion (*ecart*) and intensity (*nbre-permut*)

Inputs

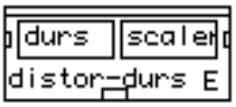
liste	(list)	list to be modified
nbre-permut	(number)	statistical number of permutations
centre	(number)	centre of the permutations
ecart	(number)	dispersion des permutations

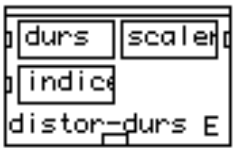
Example :

```
(pseudo-permut-centre '(1 2 3 4 5 6 7 8) 3 2 3)->(1 3 4 5 2 6 7 8)
```

distor-durs

function distorting values within a list (*ldurs*) without changing their sum

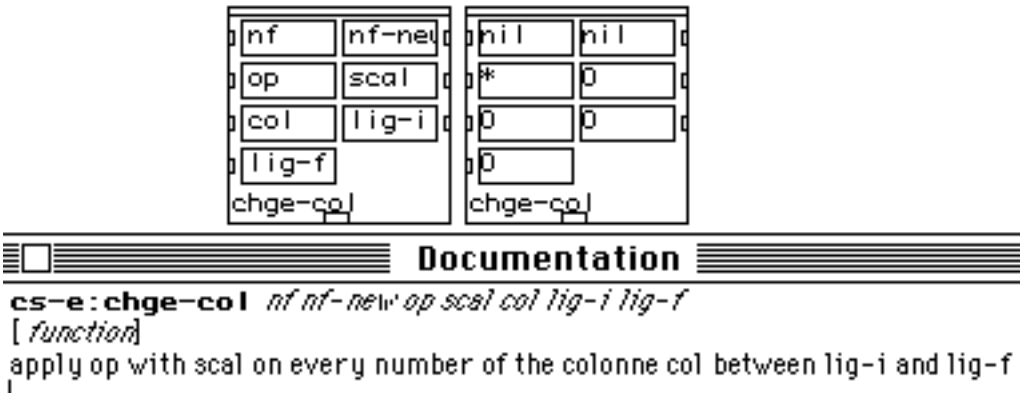




Inputs

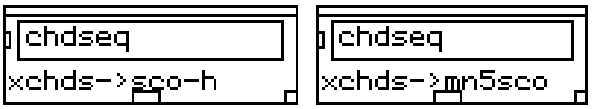
ldurs	(list)	list to be modified
scaler	(number)	degree of distorsion (between 0 and 100)

chge-col



With the module **chge-col**, you can modify an already written score by applying a PatchWork function on one of its columns.

xchds->sco-h, xchds->mn5sco

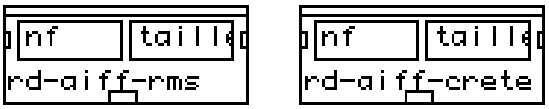


The new modules **xchds->sco-h** and **xchds->mn5sco** allow you to write a score from a chordseq.

With **xchds->sco-h**, the score is written horizontally (as a sequence of chords).

With **xchds->mn5sco**, the score is written vertically as a path of frequencies. It is then necessary to have chords with a similar number of notes, and not to have more than 24 chords (a Csound limit).

rd-aiff-rms, rd-aiff-crete

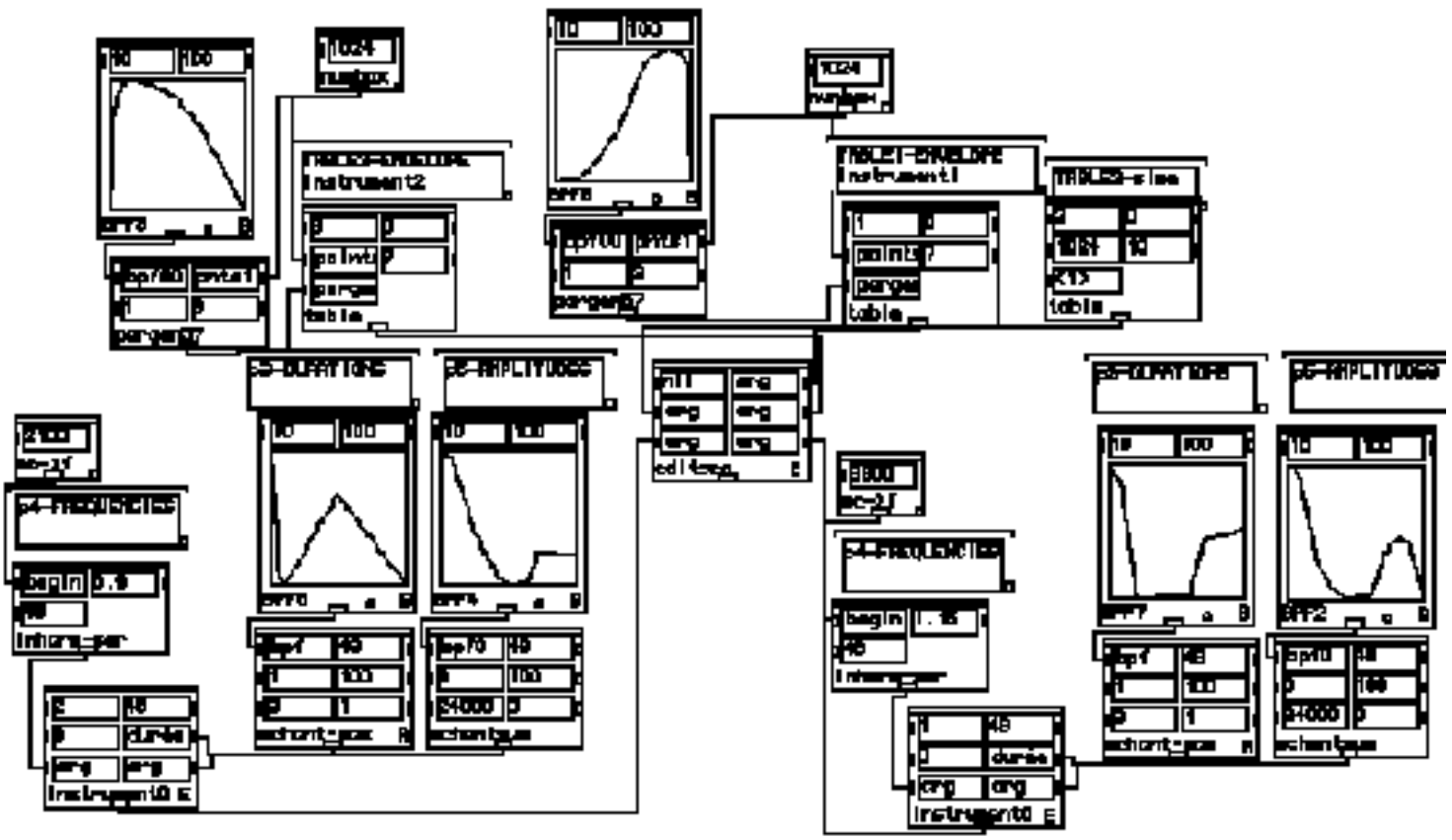


The two modules compute the amplitude envelope of mono AIFF soundfiles. The result can be displayed in a BPF. The taille parameter specifies the size in samples for the window which is used to scan the soundfile.

Appendix 1: Examples of patches

A complete example

This is a complete patch for notating two spectra with 48 harmonics each, in which :
• table 1 constructs the envelope for instrument 1
• table 3 constructs the envelope for instrument 2
• table 2 constructs a sine wave to be read by the oscillators of the two instruments
• instrument 1 has a spectrum of 48 harmonics with a distortion degree of 1.15 over pitch D1 (3800 in midicents). It uses the **inharm-ser** module that has been introduced in § 2.4.
• instrument 2 has a spectrum of 48 harmonics and a distortion level of 0.90 over pitch G0 (1800 in midicents)
Each spectrum has particular configurations of durations and amplitudes of its harmonics.
* This score was conceived for use with the orchestra found in the file "ex2.orc" (see appendix A2)



Score output by this patch

```
f 1 0 1024 7 0.0000 164 0.0300 133 0.1310 164 0.3840 133 0.6770 51 0.8380 103 0.9600 92 1.0000
92 0.9700 61 0.8690 31 0.0000
f 2 0 1024 10 1
f 3 0 1024 7 0.0000 31 0.6840 41 0.8880 51 0.9900 102 1.0000 256 0.8880 215 0.6220 328 0.0000
i 2 0 9.0000 48.9994 1628
i 2 0 5.0000 91.4362 1596
i 2 0 1.3000 131.7044 1563
i 2 0 0.4000 170.6259 1531
i 2 0 0.3000 208.5759 1433
i 2 0 0.5000 245.7690 1335
i 2 0 0.8000 282.3445 1205
i 2 0 1.1000 318.3992 1107
i 2 0 1.5000 354.0049 977
i 2 0 1.8000 389.2163 879
i 2 0 2.3000 424.0767 749
i 2 0 2.6000 458.6212 651
i 2 0 2.9000 492.8787 586
i 2 0 3.4000 526.8734 521
i 2 0 3.7000 560.6259 456
i 2 0 4.1000 594.1540 391
i 2 0 4.4000 627.4730 326
i 2 0 4.8000 660.5965 261
i 2 0 5.2000 693.5364 195
i 2 0 5.5000 726.3033 130
i 2 0 6.0000 758.9067 98
i 2 0 6.3000 791.3552 33
i 2 0 6.6000 823.6564 0
i 2 0 6.8000 855.8175 0
i 2 0 6.6000 887.8448 0
i 2 0 6.3000 919.7442 0
i 2 0 6.1000 951.5211 33
i 2 0 5.7000 983.1805 33
i 2 0 5.4000 1014.7270 33
i 2 0 5.2000 1046.1649 33
i 2 0 4.9000 1077.4982 98
i 2 0 4.6000 1108.7305 195
i 2 0 4.3000 1139.8654 326
i 2 0 4.0000 1170.9060 391
i 2 0 3.7000 1201.8555 391
i 2 0 3.5000 1232.7167 391
i 2 0 3.1000 1263.4922 391
i 2 0 2.8000 1294.1847 391
i 2 0 2.6000 1324.7965 391
i 2 0 2.3000 1355.3299 391
i 2 0 2.1000 1385.7870 358
i 2 0 1.7000 1416.1700 358
i 2 0 1.4000 1446.4807 358
i 2 0 1.2000 1476.7209 358
i 2 0 0.9000 1506.8926 358
i 2 0 0.5000 1536.9972 358
i 2 0 0.3000 1567.0364 358
i 2 0 0.0000 1597.0118 358
i 1 0 9.0000 73.4162 1632
i 1 0 8.5000 162.9207 1615
i 1 0 8.3000 259.7056 1582
i 1 0 8.0000 361.5437 1516
```

```
i 1 0 7.3000 467.3125 1434
i 1 0 6.2000 576.3228 1319
i 1 0 4.6000 688.1048 1121
i 1 0 3.1000 802.3158 923
i 1 0 1.7000 918.6937 725
i 1 0 0.1000 1037.0313 544
i 1 0 0.0000 1157.1601 462
i 1 0 0.0000 1278.9403 396
i 1 0 0.0000 1402.2541 313
i 1 0 0.0000 1527.0002 247
i 1 0 0.1000 1653.0912 181
i 1 0 0.1000 1780.4502 165
i 1 0 0.1000 1909.0096 132
i 1 0 0.1000 2038.7090 115
i 1 0 0.1000 2169.4942 82
i 1 0 0.1000 2301.3165 66
i 1 0 0.2000 2434.1315 33
i 1 0 0.2000 2567.8990 16
i 1 0 0.2000 2702.5819 16
i 1 0 0.2000 2838.1463 33
i 1 0 0.2000 2974.5609 33
i 1 0 0.2000 3111.7966 49
i 1 0 0.3000 3249.8266 49
i 1 0 0.3000 3388.6256 49
i 1 0 0.3000 3528.1704 66
i 1 0 1.0000 3668.4390 99
i 1 0 1.7000 3809.4107 231
i 1 0 2.6000 3951.0664 346
i 1 0 3.3000 4093.3877 478
i 1 0 3.9000 4236.3576 560
i 1 0 3.9000 4379.9596 610
i 1 0 3.9000 4524.1785 659
i 1 0 4.0000 4668.9996 692
i 1 0 4.0000 4814.4091 709
i 1 0 4.1000 4960.3938 725
i 1 0 4.2000 5106.9410 725
i 1 0 4.2000 5254.0390 692
i 1 0 4.3000 5401.6761 676
i 1 0 4.3000 5549.8415 610
i 1 0 4.4000 5698.5247 495
i 1 0 4.4000 5847.7156 379
i 1 0 4.5000 5997.4048 264
i 1 0 4.5000 6147.5829 132
i 1 0 4.6000 6298.2410 0
e
```

Appendix 2: The basic orchestras

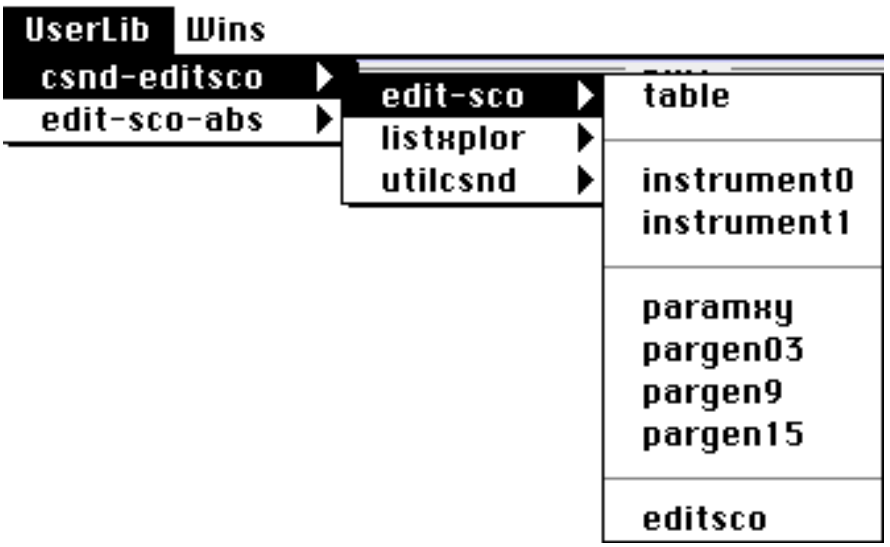
EX1.ORC

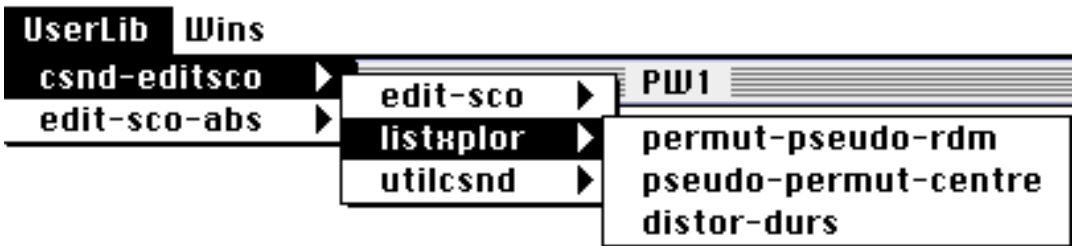
```
sr=16000
kr=1000
ksmps=16
nchnls=1

instr1
kenvoscil10,1,p3,1
a1   oscilip5,p4,2
     outa1*kenv
endin
```

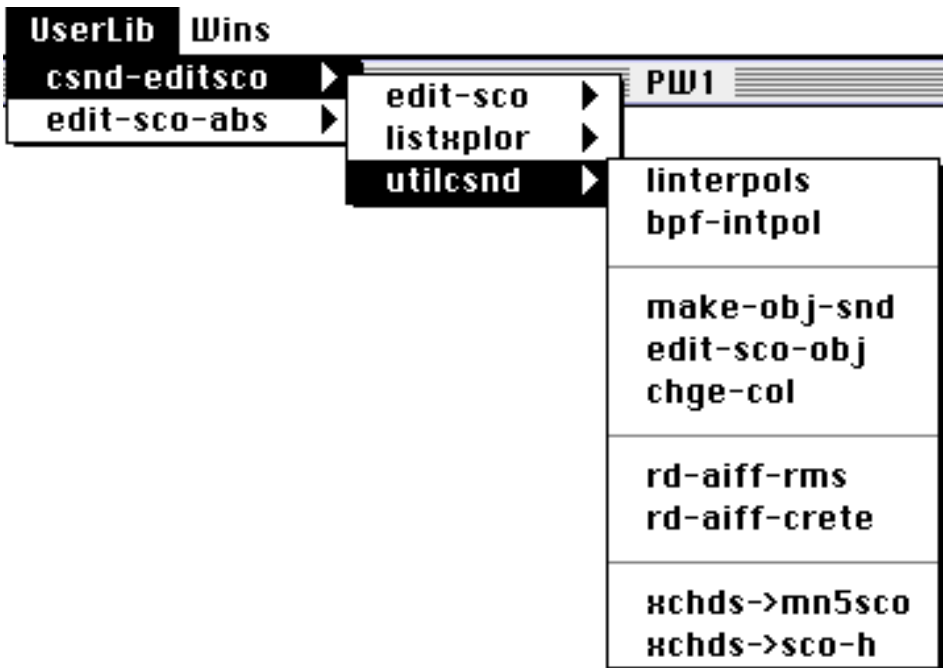
EX2.ORC

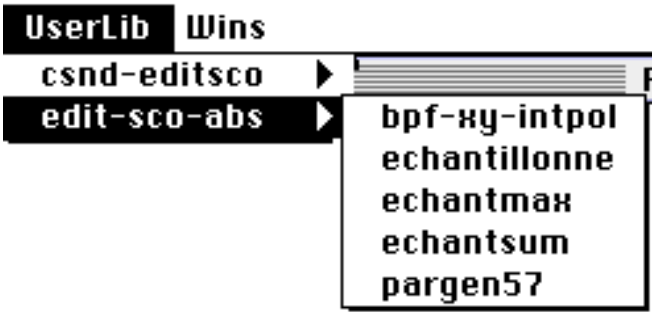
```
sr=16000
kr=1000
ksmps=16
nchnls=1

instr1
kenvoscil10,1,p3,1
a1   oscilip5,p4,2
     outkenv*a1
endin


instr2
kenvoscil10,1,p3,3
a1   oscilip5,p4,2
     outkenv*a1
endin
```

Appendix 3: The Menus








UserLib
Wins
csnd-editsco
edit-sco-abs
bpf-xy-intpol
echantillonne
echantmax
echantsum
pargen57

Index

P

R

S

T

V

X

Y