Ircam documentation

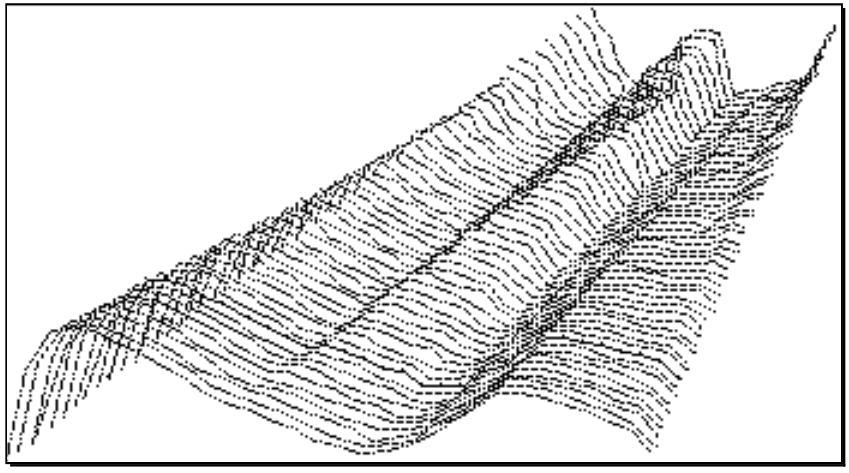
- Research reports
- Musical works
- Software

PatchWork

Profile

Library for the Control of Melodic Profiles

First English Edition, February 1996



IRCAM Centre Georges Pompidou

This manual was written by Mikhail Malt et Jacopo Baboni Schilingi, translated into English by J. Fineberg, and was produced under the editorial responsibility of Marc Battier, Marketing Office, Ircam.

Patchwork was conceived and programmed by
Mikael Laurson, Camilo Rueda, and Jacques Duthen.

The Profile library was conceived and programmed by Mikhail Malt et Jacopo Baboni Schilingi.

First English edition of the documentation, February 1996.

This documentation corresponds to version 1.0 of the library, and to version 2.1 or higher of PatchWork.

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' group

The use of this software and its documentation is restricted to members of the Ircam software users' group. For any supplementary information, contact:

Département de la Valorisation
Ircam
Place Stravinsky, F-75004 Paris

Tel. (1) 44 78 49 62
Fax (1) 42 77 29 47
E-mail: bousac@ircam.fr

Send comments or suggestions to the editor:

E-mail: bam@ircam.fr
Mail: Marc Battier,
Ircam, Département de la Valorisation
Place Stravinsky, F-75004 Paris

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

Résumé

La librairie Profile pour ParchWork est destinée à la manipulation des hauteurs, selon des représentations et des transformations géométriques. Elle utilise la notion de « profil », qui peut être définie comme une succession linéaire de directions d'intervalles de hauteurs.

Les opérations géométriques de représentation et de transformation sont regroupées en six sections :

• Perturbation

• Change

• Reflexions

• Deriv/integr

• Interpolation

• Utilitaires

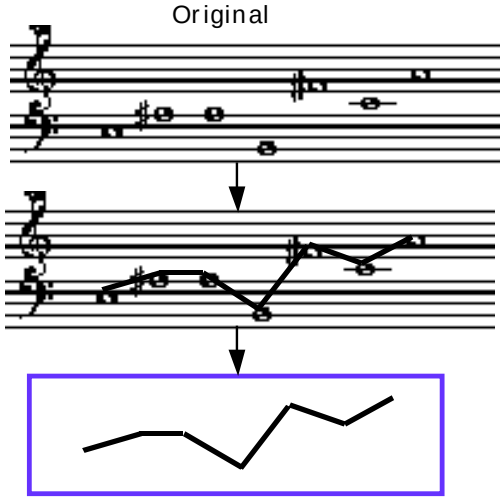
Des références bibliographiques viennent en renfort pour éclairer les concepts et les algorithmes utilisés dans Profile.

1 Overview of the Profile library

The notion of *profile*

Profiles originated as the first part of a project in musical development. It uses a geometric representation and transformation of musical pitches. Of the various perceptual qualities of a "melody," (for our purposes here; a melody may be defined as a succession of musical notes in time, which taken together constitute a single musical line) the profile is probably the most important characteristic for the graphic representation, recognition and memorization of a melodic musical idea.

For use within this library a profile is defined as a linear succession of intervallic directions. These directions are considered, for the moment, equidistant in time[1] and may be represented graphically as a succession of line segments, as shown below:



A natural outgrowth of the parametric conception of musical composition that originated with serial music, the disassociation of musical objects into their constituent parameters is an extremely widely used technique in contemporary music. Thus the Profile, along with the other parameters of a musical object, manages to separate itself from the other constituents and take on an important role in the musical techniques and conception of many contemporary composers. Thus the profile has become an important aspect of a musical composition in its own right . Once a composer wishes to graphically represent the variation of a parameter the notion of a profile becomes important. Other conceptions of this same idea use terms such as envelope, contour, line, etc.

1. Time is not explicitely taken in account by Profile, which is a library devoted to processing pitch.

Finally, this library allows the profile to be used in various ways: as a basic compositional parameter, as a musical process, or as the controlling element in a various musical evolutions and transformations.

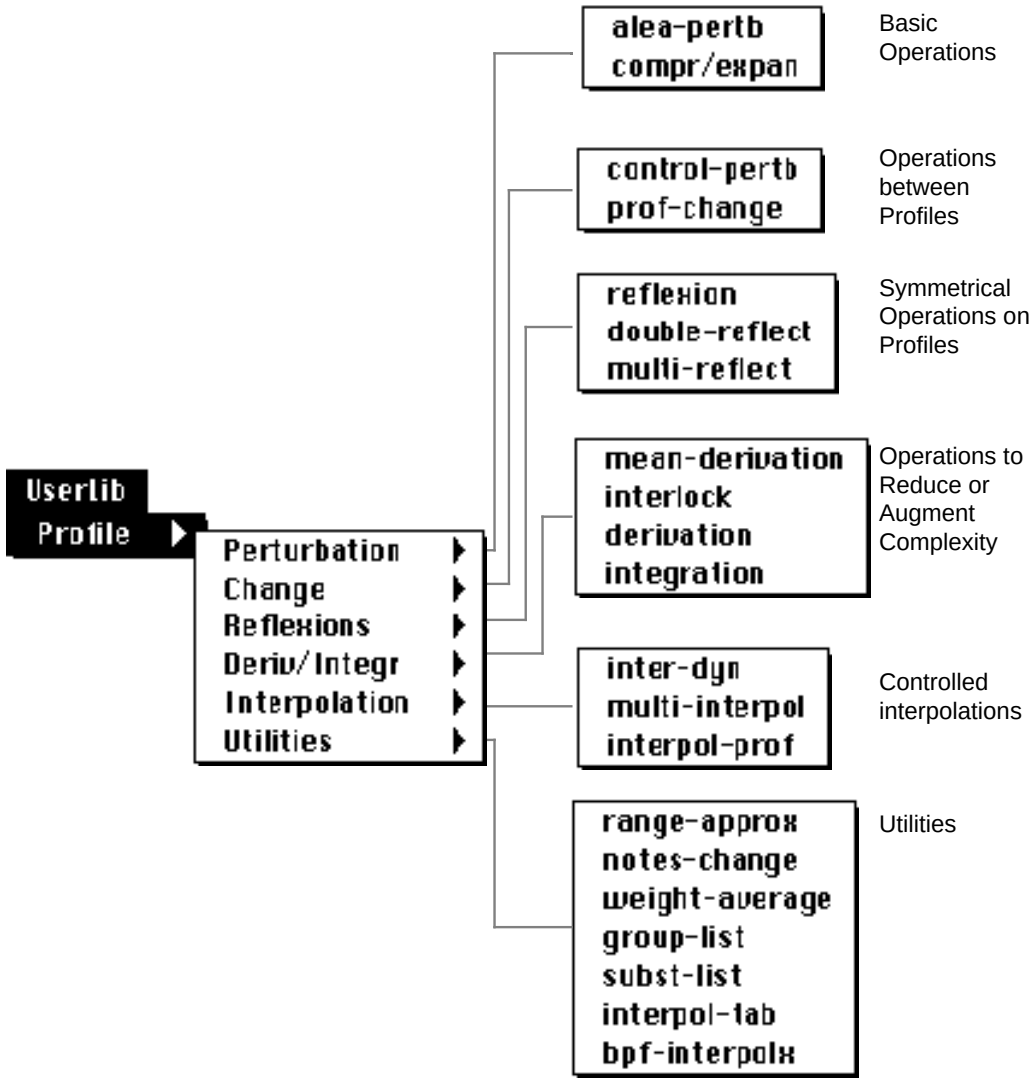
The purpose of Profile

Technically speaking, this library was conceived with the goal of making the generation and control of compositional material as close as possible to musical intuition. Additionally, we wanted to propose certain processes for treating melodic lines in as generalized a manner as possible.

From a practical point of view, the user has direct control of all of the following aspects of a profile: interval directions, the intervals themselves, the absolute pitches (vertical or horizontal: harmonic control), the global direction of each process through a break-point-function as well as the depth to which each process is applied.

2 The structure of the library

Profiles is subdivided in to six types of functions, each type offers specific methods of manipulating melodic profiles.

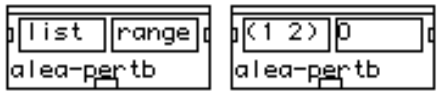


There are a series of patches demonstrating the use of Profiles that may be accessed by simply selecting within PatchWork any module from Profiles and typing the letter ‘t’ on the keyboard.

3 Basic Operations : menu Perturbation

This sub-grouping of modules is contains functions which may be used to perform basic operations on a melodic profile including random perturbations, as well as intervallic compression and expansion.

alea-pertb



Syntax

```
(profile::alea-pertb list range)
```

Inputs

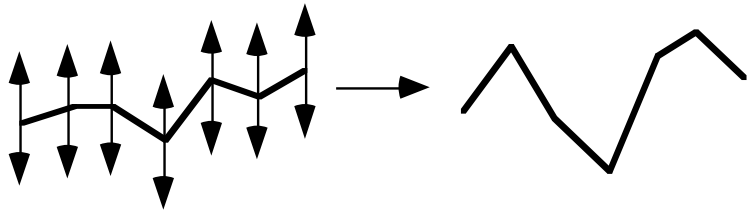
list list

range whole or floating-point number greater than or equal to zero

Output

list

Applies a random disturbance to a list of pitches *list*.



The perturbation is performed on the absolute value of the pitch; each note is modified by the addition of a random value between -*range* and +*range*.

list a simple list with only one level of parentheses, representing the succession of pitches in midi-cents.

range quantity of the variation. If *range* is a whole number, the amount of perturbation will be the addition or subtraction of whole number quantities. If the *range* is in floating point, the perturbations will be likewise calculated in floating-point values.

compr/expan

Syntax

(profile::compr/expan *list value note?*)

Inputs

list list

value whole or floating-point number

note? list

Output

list

This module compresses or expands the intervals derived from the list of notes (*list*) through multiplication by the factor entered as *value.*

list a simple list with only one level of parentheses, representing the succession of pitches in midi-cents.

value multiplication factor, may be either a whole number or floating-point.

note? optional input allowing the generated form to be adapted so as to conform to the harmonic field entered. This harmonic field attached to the input *note?* may be either a chord or a scale.

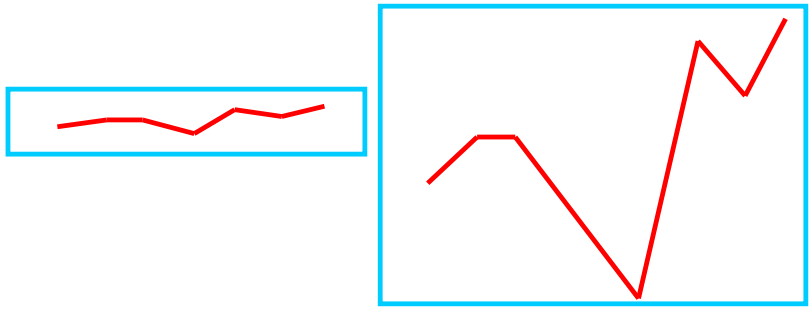
examples:

If *value* equals '1' the intervals contained in the list will remain unchanged.

If *value* is less than '1' the intervals contained in the list will be compressed.

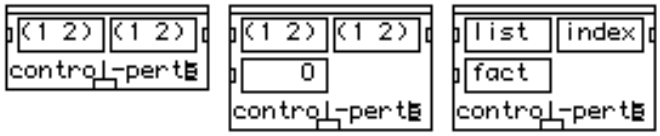
If *value* is greater than '1' the intervals contained in the list will be expanded.

If *value* is negative the intervals contained in the list will be inverted.



4 Operations between Profiles: Change

control-pertb



Syntax

(profile::control-pertb *list fact index*)

Inputs

list list

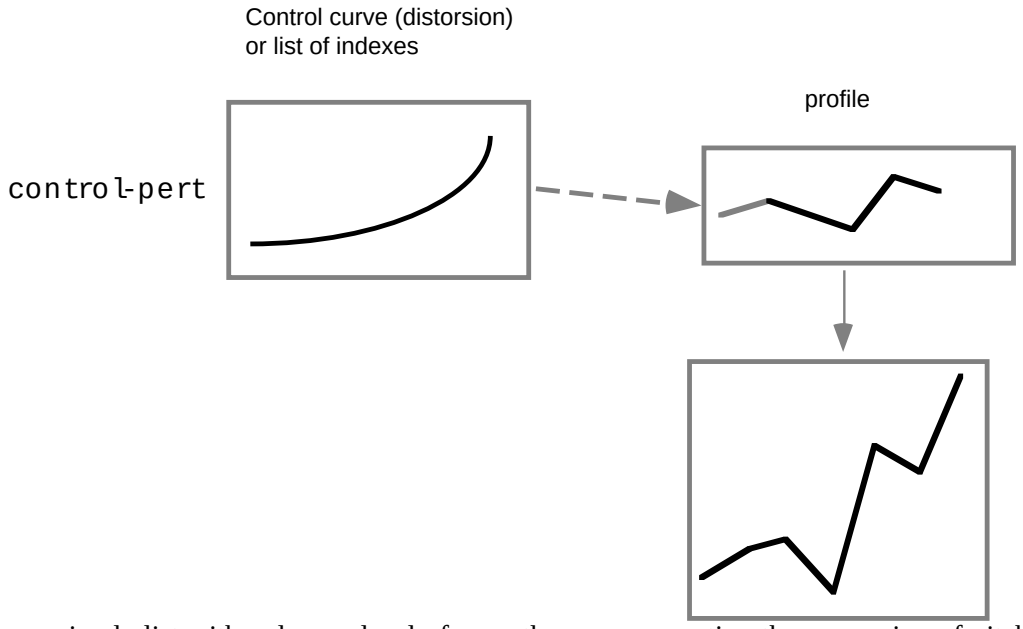
fact list or BPF object
`patch-work::c-break-point-function`

index whole number

Output

list

This module allows the application of a controlled distortion on certain pitches of the list *list* though the addition of the value '*fact***index*' to those pitches.



list a simple list with only one level of parentheses, representing the succession of pitches in midi-cents.

index either a list of index values or a **multi-BPF** module

fact a list of multiplicative factors

The index list determines which elements will be "perturbed."

For example if the entry to the input list is as follows:

```
->>(6000 6500 7100 6400 6100 5500 5800 5300 5800 5100)
```

and the input to index is:

```
->>(0 10 -10 0 0 5 5 -5 0 0)
```

and factor is equal to '10 ;.

The result will be

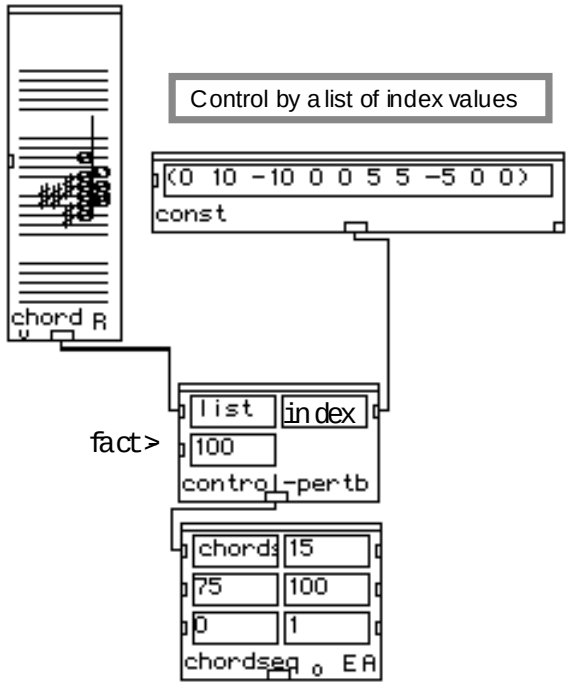
```
->>(6000* 6600 7000 6400* 6100* 5550 5850 5250 5800* 5100*)
```

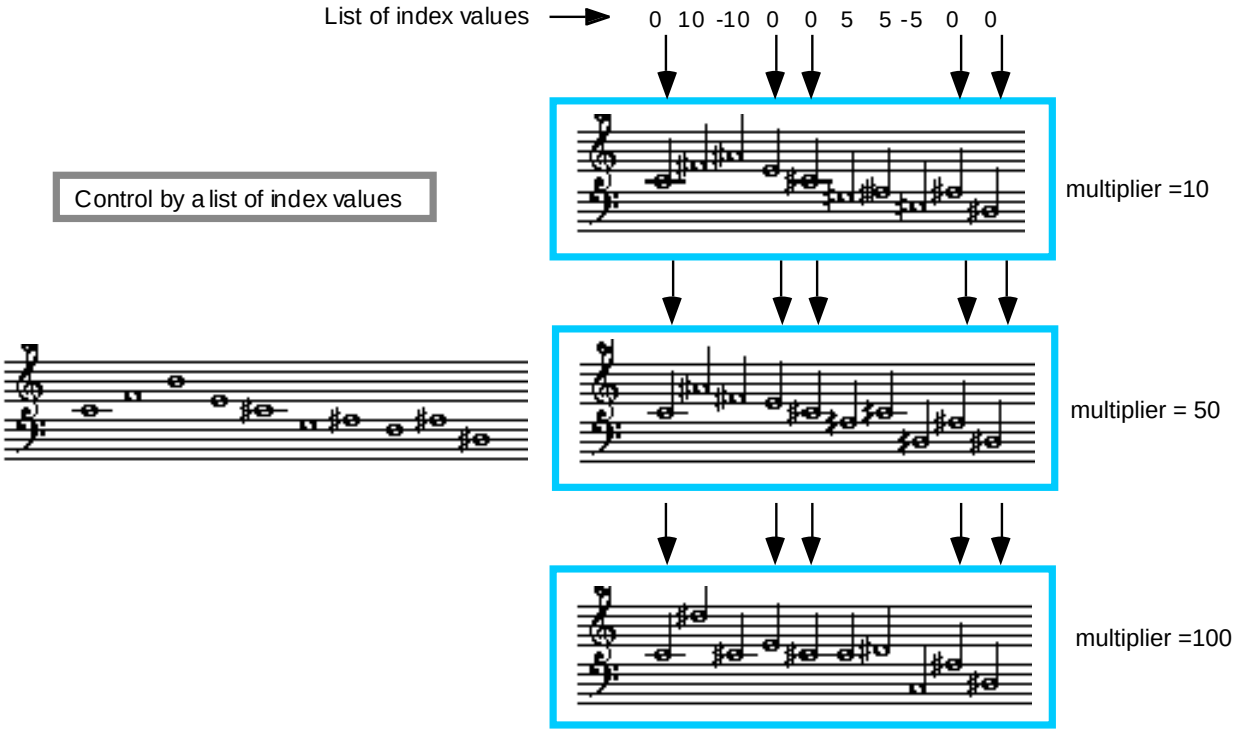
*unchanged elements

A zero in the list *index* indicates that the corresponding element of the list *list* will remain unchanged.

The input *fact* is used as a multiplier, it will cause the effect of the index parameters to be either amplified or attenuated.

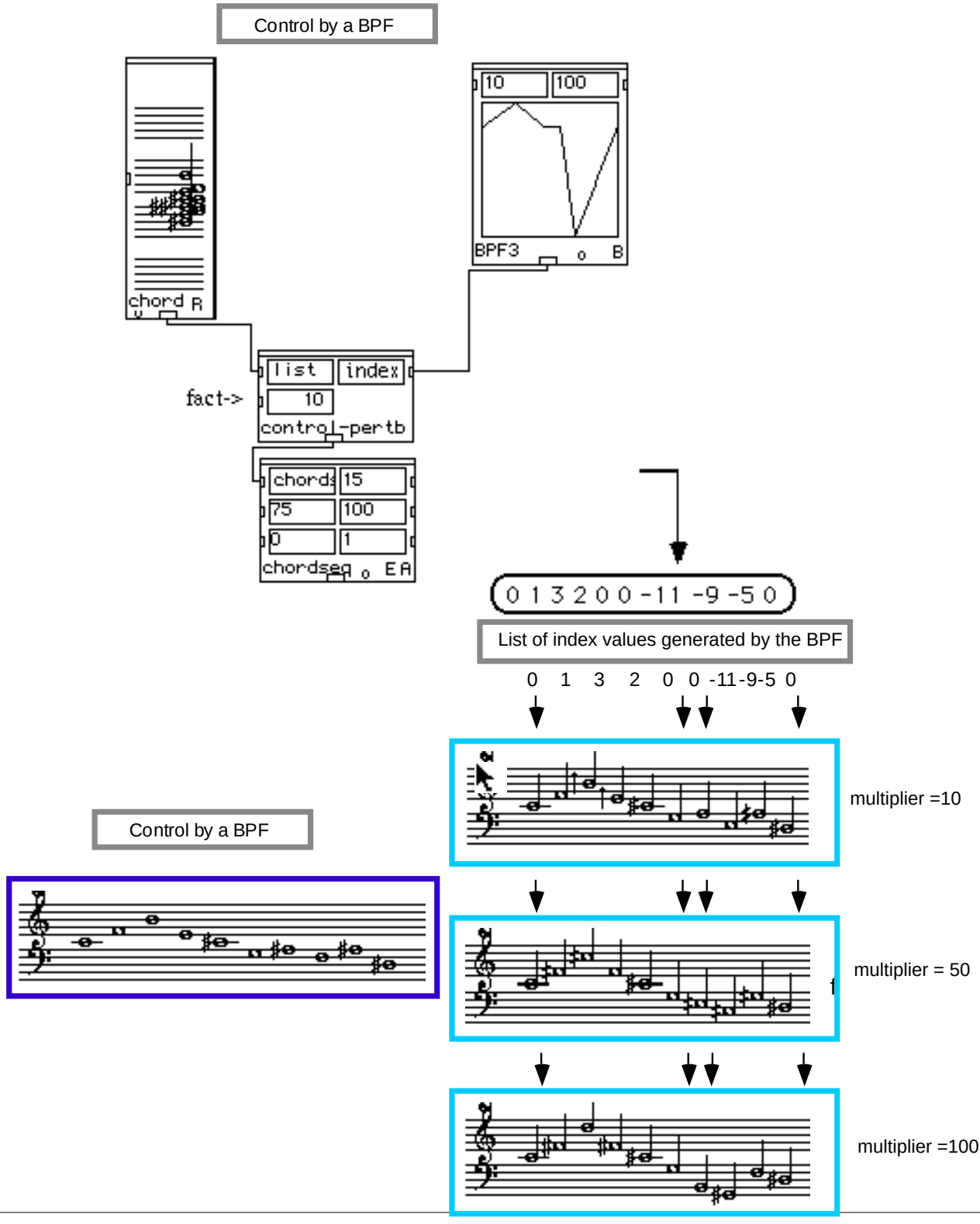
Example of a perturbation (distortion) controlled by a list of index values.





The index input will also accept a **multi-BPF** module. In that case, the **control-pertb** module will sample the BPF (with the same number of steps as the length of list). The absolute values of the results from this sampling are used as the index values, these values will still be scaled by the parameter *fact*.

Example of a perturbation (distorsion) controlled by a table (BPF)


Control by a BPF
chord
10
100
BPF3
B
list
index
10
control-pertb
fact->
chords
15
75
100
0
1
chordseq
EA
0 1 3 2 0 0 -11 -9 -5 0
List of index values generated by the BPF
0 1 3 2 0 0 -11 -9 -5 0
multiplier =10
Control by a BPF
multiplier = 50
multiplier =100

prof-change

Syntax

`(profile::prof-change` *prof pitch mode?*`)`

Inputs

prof	list
pitch	list
mode?	menu options

Output

list

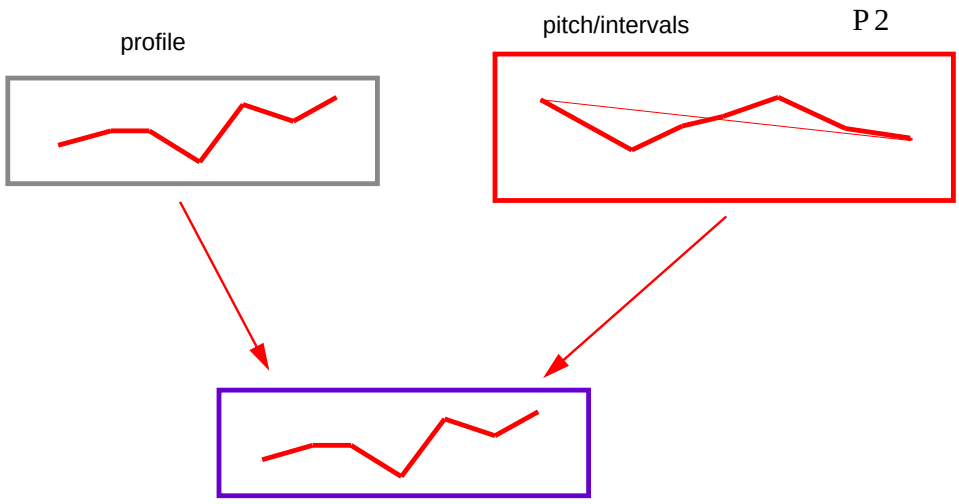
This module transforms the melodic contour of a list of notes pitch by the profile of a second list of notes *prof*.

prof a simple list with only one level of parentheses, representing the succession of pitches in midicents that define a profile.

pitch a simple list with only one level of parentheses, representing pitches in midicents that define either a reservoir of notes or a reservoir of intervals.

mode? menu allowing the user to select the way in which the module will perform.
If *mode?* is set to 'note' the list pitch will be used as a reservoir of notes.
If *mode?* is set to 'intrv' the list pitch will be used as a reservoir of intervals.

In practical terms the structure that is generated is the combination of the interval directions (profile) of *prof* using either the notes or intervals of *pitch* (depending on the input *mode?*).

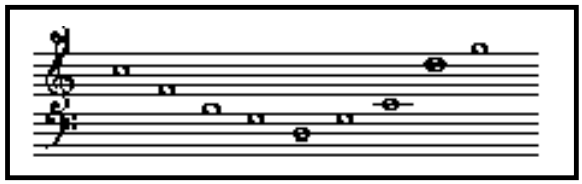


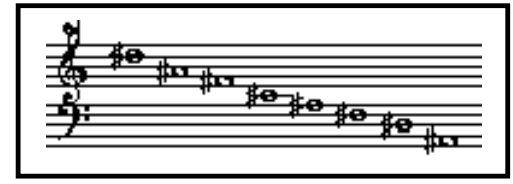
The resulting structure contains the interval directions of P1 and the pitches (or the intervals) of P2.

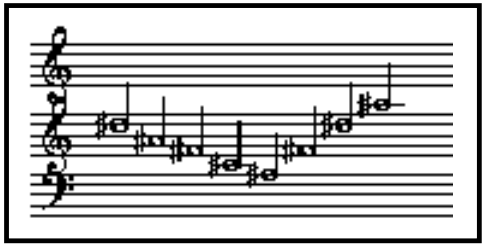
profile P1

Action bewteen profiles. prof-change

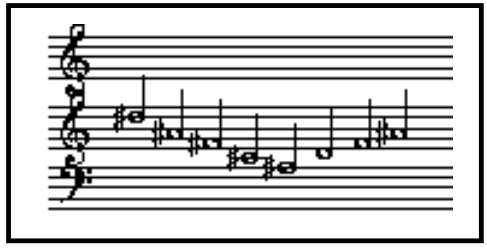
pitch/intervals P2







keeping the pitches



keeping the intervals

5 Symmetrical Operations : Reflexions

reflexion

Syntax

`(profile::reflexion` *list axis mode? up/down*`)`

Inputs

list list

axis whole number in midicents

mode? menu options

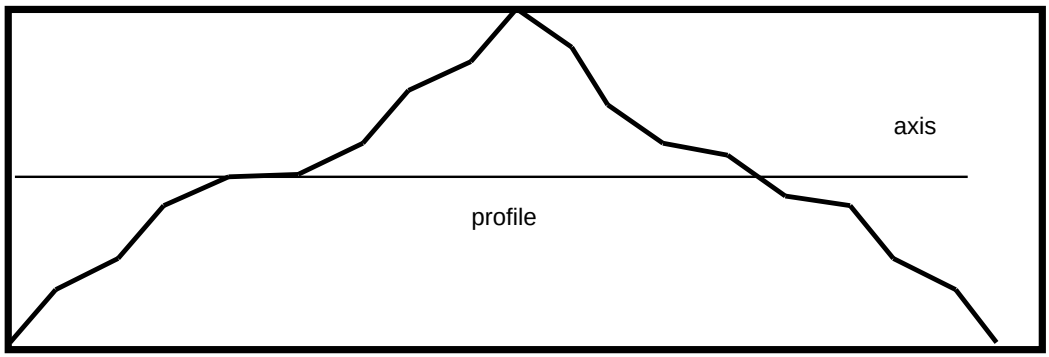
up/down menu options

Output

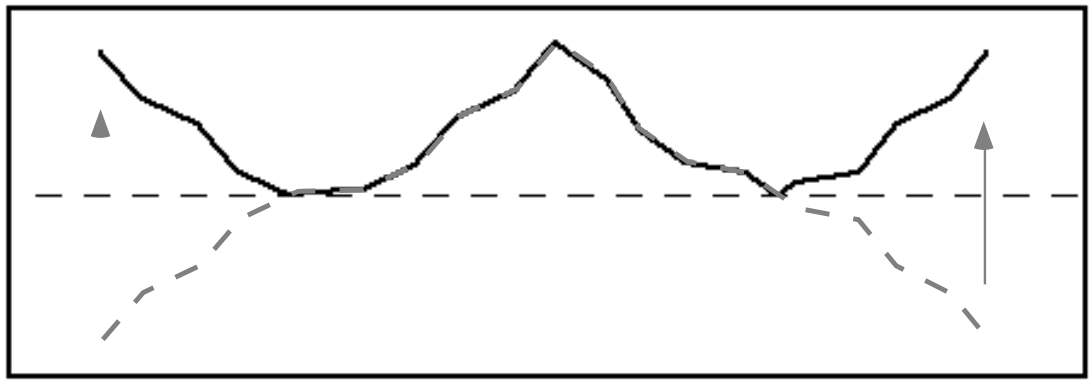
list

Performs a symmetrical projection around the value axis. This operation considers all of the pitches contained in list as a geometric profile.

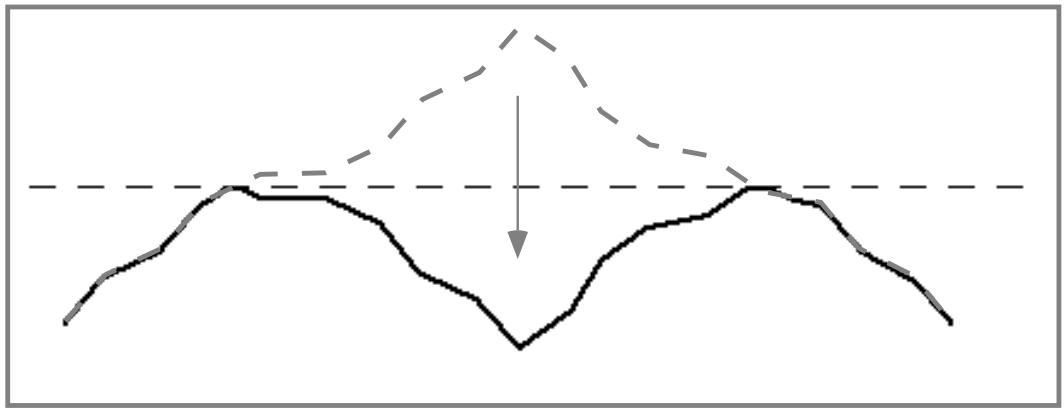
Take, for example, the following profile:



It is possible to reflect a portion of these notes from below the axis to above:



or from above to below:



list a simple list with only one level of parentheses, representing the succession of pitches in midicents.

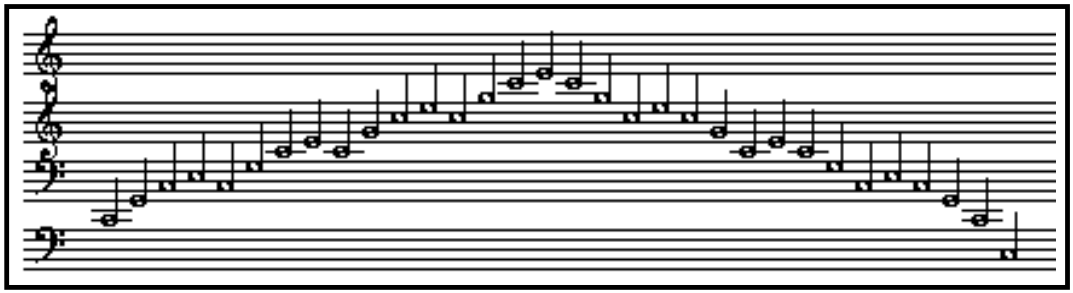
axis Value in midicents to determine the axis of symmetry.

mode? menu allowing the user to select the way in which the module will perform.
If *mode?* is set to 'note' the list pitch will be used as a reservoir of notes, in other words, the reflection around the axis will respect the values of the notes of list.
If *mode?* is set to 'intrv' the list pitch will be used as a reservoir of intervals, in other words, the reflection around the axis will respect the values of the intervals of list.

up/down menu allowing the user to select the direction of the reflection.
If *up/down* is set to 'up' the reflection around the axis will be from below to above.
If *up/down* is set to 'down' the reflection around the axis will be from above to below.

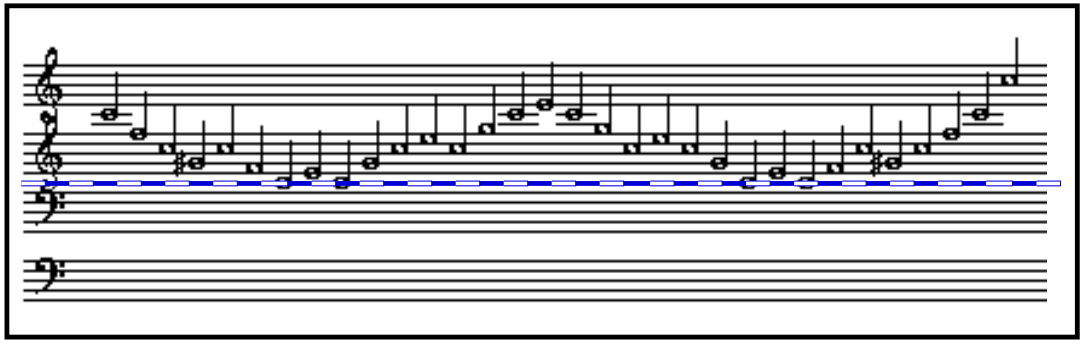
Example:

Chopin



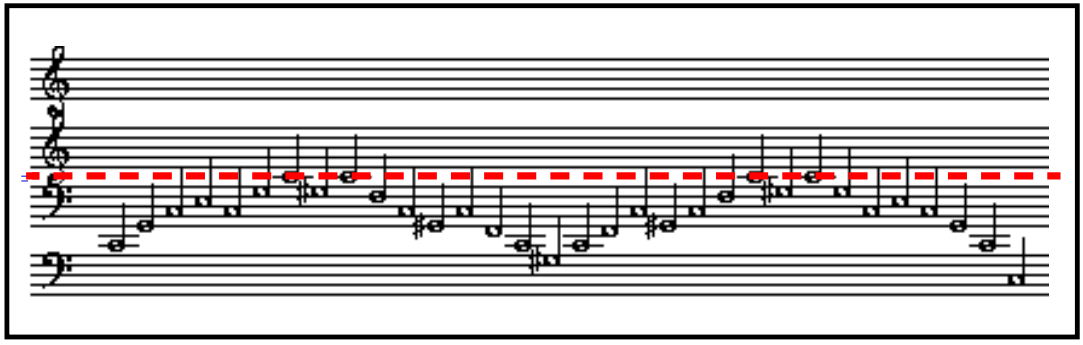
Reflection around the axis will be from below to above

Axis = C3

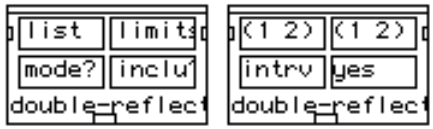


Reflection around the axis will be from above to below

Axis = C3



double-reflect



Syntax

(profile::double-reflect *list limits mode? inclu?*)

Inputs

list list

limits list

mode? menu options

inclu? menu options

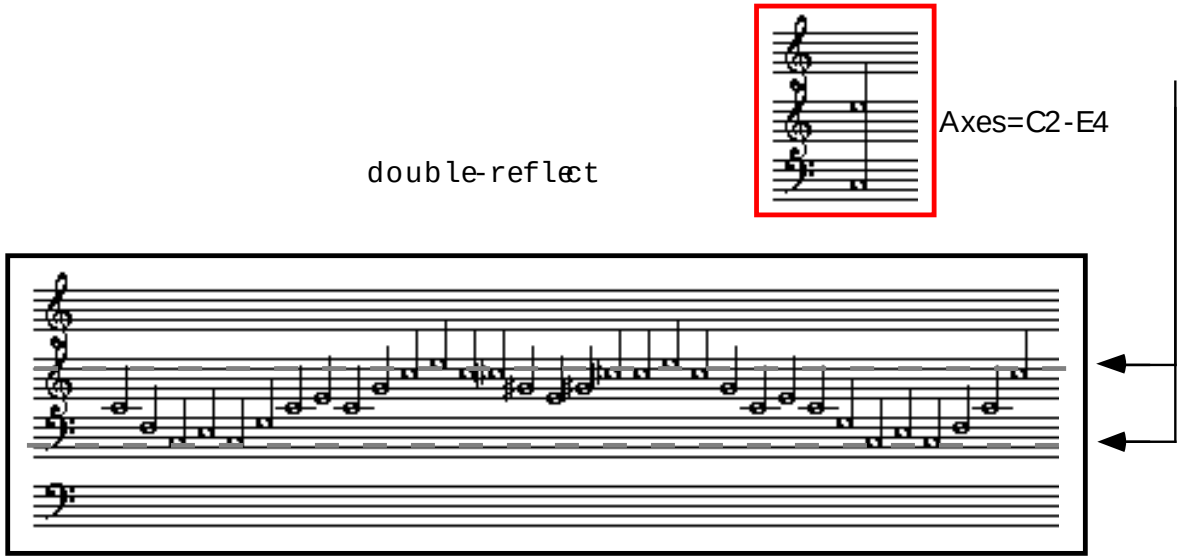
Output

list

Performs a symmetrical projection in relation to two range limits. This operation considers all of the pitches contained in list as a geometric profile.

Example:

It is possible to reflect a portion of the notes from the preceding Chopin example in relation to two range limits:



list a simple list with only one level of parentheses, representing the succession of pitches in midicents.

limits a list of two values in midicents which determine the lower and upper range limits into which outlying notes will be reflected.

mode? menu allowing the user to select the way in which the module will perform.
If *mode?* is set to 'note' the list pitch will be used as a reservoir of notes, in other words, the reflection around the upper or lower value of limits will respect the values of the notes of list.
If *mode?* is set to 'intrv' the list pitch will be used as a reservoir of intervals, in other words, the reflection around the upper or lower value of limits will respect the values of the intervals of list.

inclu? menu allowing the user to select whether or not the notes which do not exist in any octave within the range defined by limits will be included in the output.
If *inclu?* is set to 'yes' the notes outside of the range defined by limits are placed as close as possible to one of the limits.If *inclu?* is set to 'no' the notes outside of the range defined by limits are eliminated.

multi-reflect

Syntax

(profile::multi-reflect *list limits note?*)

Inputs

list list

limits list

note? list

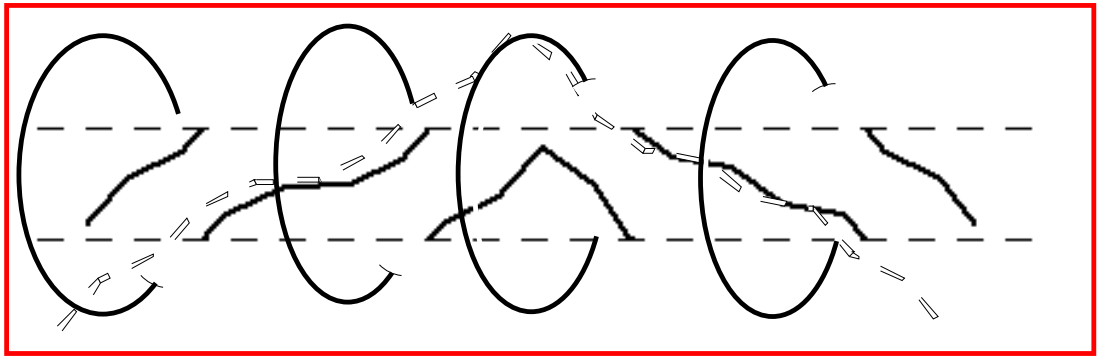
Output

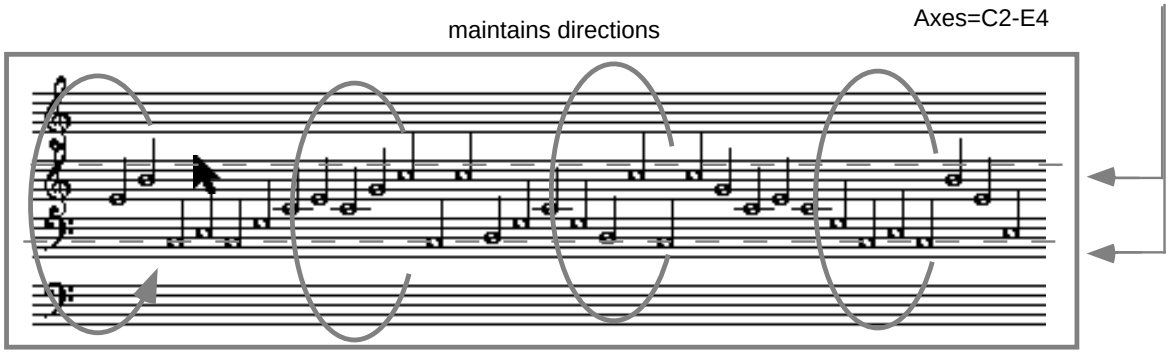
list

Performs a symmetrical projection in relation to two range limits. This operation considers all of the pitches contained in list as a geometric profile.

Example:

It is possible to reflect a portion of the notes, still using Chopin fragment example from the two preceding examples, in relation to two range limits:





The difference between multi-reflect and the double-reflect module is that this module performs the reflection in such a way as to preserve the relative directions of the elements at the moment they are reflected.

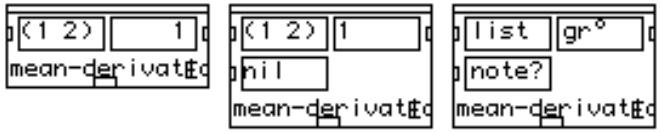
list a simple list with only one level of parentheses, representing the succession of pitches in midicents.

limits a list of two values in midicents which determine the lower and upper range limits into which outlying notes will be reflected.

note? optional input allowing the generated form to be adapted so as to conform to the harmonic field entered. This harmonic field attached to the input *note?* may be either a chord or a scale.

6 Operations on Complexity : Deriv/integr

mean-derivation



Syntax

(profile::mean-derivation *list gr° note?*)

Inputs

list list

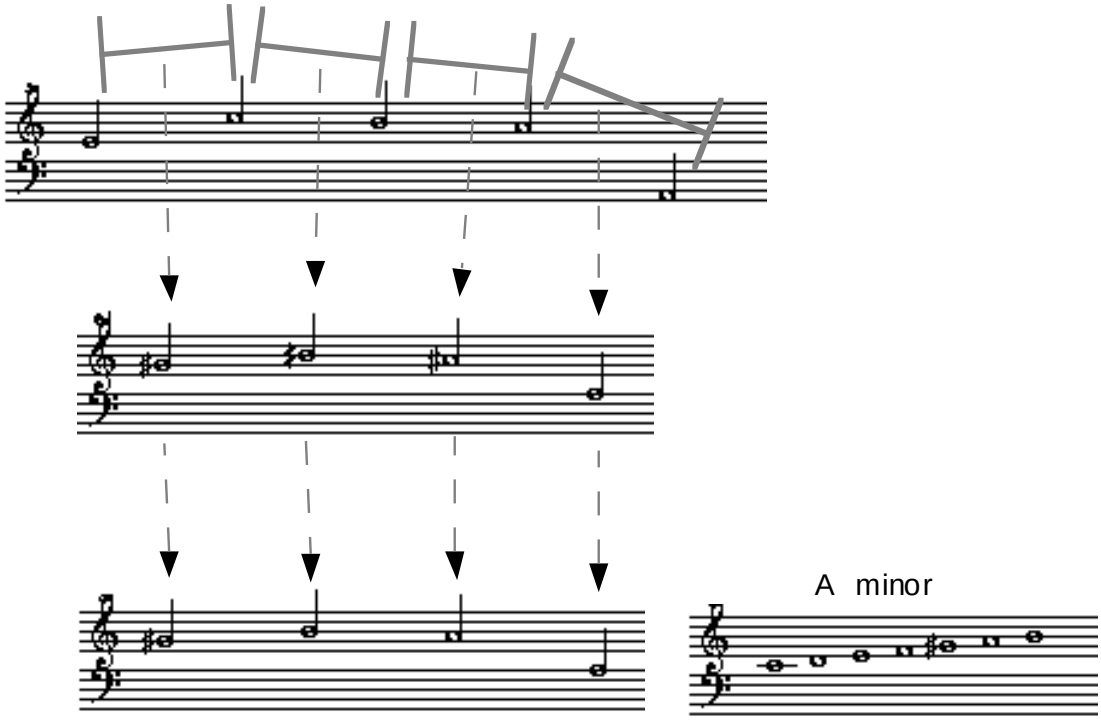
gr° whole number greater than or equal to one

note? list

Output

list

This module simplifies melodic profiles. The resulting profile is determined by calculating the mean value of consecutive notes from list.



Example:

with a list = `(6000 5700 5100 5200 4900 5800 6400 7200 7800 7500 6600 6800)` and *gr°* = 1.

The result is the following:

`(5850 5400 5150 5050 5350 6100 6800 7500 7650 7050 6700)`

as can be seen the value 5850 is the mean between 6000 and 5700,

5400 is the mean between 5700 and 5100, and so on.

The number of elements in the output profile will always be one less than in the input.

If we keep the same list, but use a value of *gr°* = 2, the result is the following:

`(5625 5275 5100 5200 5725 6450 7150 7575 7350 6875)`

thus 5625 is the mean between 5850 and 5400,

5275 is the mean between 5400 and 5150,

and so on.

It should be kept in mind that the list (`5850 5400 5150...`) is the result of the process with *gr°*= 1.

This process may be calculated with varying depth (the depth determines number of iterations), but remember that for each additional level of depth the resulting list is reduced by one element.

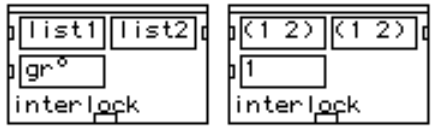
It is also possible to use the optional input *note?* which allows the generated form to be adapted so as to conform to the harmonic field entered.

list a simple list with only one level of parentheses, representing the succession of pitches in midi-cents.

gr° depth (number of iterations).

note? optional input allowing the generated form to be adapted so as to conform to the harmonic field entered. This harmonic field attached to the input *note?* may be either a chord or a scale.

interlock



Syntax

(profile::interlock *list1 list2 gr°*)

Inputs

list1 list

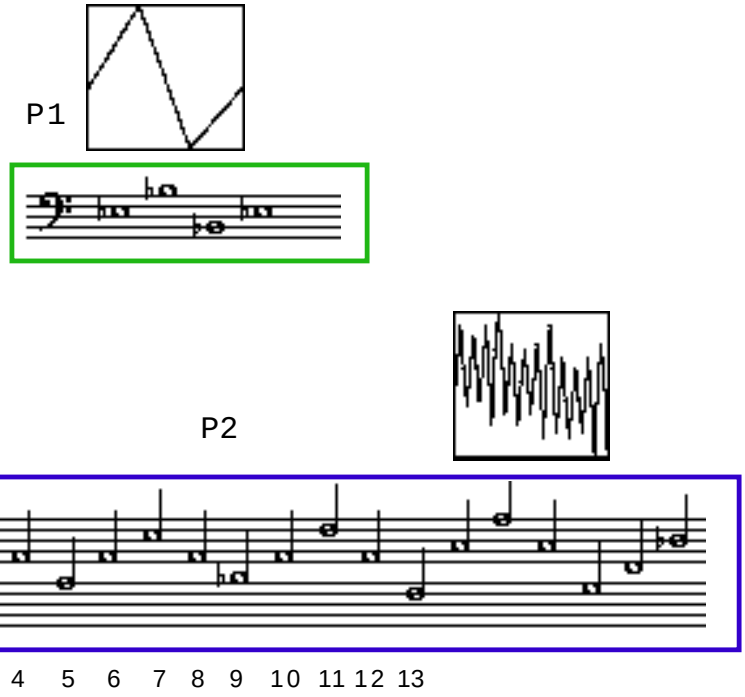
list2 list

gr° whole number greater than or equal to one

Output

list

This module intersperses the notes of *list2* between those *list1*, with a variable depth *gr°*. The usefulness of this module lies in its ability to change the octavation of the notes of *list2* so as to always place them between two notes of *list1*.



Example :

If the note F#2 from *list2* must be interspersed between the notes E4 and A3 from *list1*; the F#2 from *list2* will be transformed into an F#4 producing the new sequence:

E4 F#4 A4.

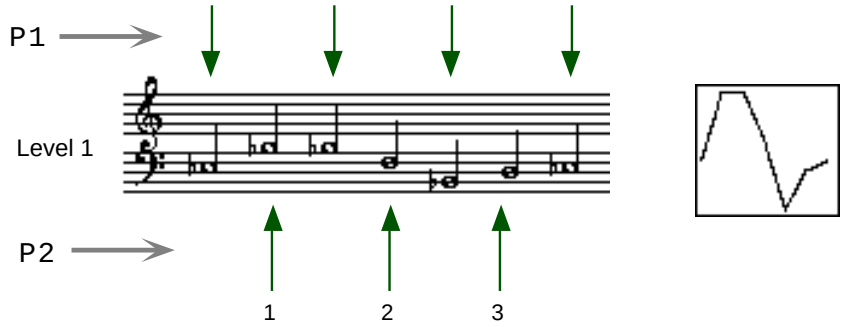
If the note to be interspersed does not exist between the two notes of *list1* regardless of the octave, the note from *list2* will be transposed to the octave closest to one of the two notes in question.

Example :

If the same F#2 from *list2* must be inserted between G4 and A#4 from *list1*; the F#2 from *list2* will be transposed to an F#4 producing the following sequence:

G4 F#4 A#4.

The F#2 was transposed to the octave where it would be as close as possible to one of the two *list1* notes, in this case the F#4 is closest to the G3.



gr° depth (number of iterations).

Example :

if one uses *list1* = `(5100* 5800* 4600* 5100*)`

and (for the sake of clarity)

list2 = `(5800 6500 7400 6500 5700 6500 7200 6500 5800 6500 7400 6500 5300 6900 7700 6900 5500)` and *gr°* = 1.

the results are as follows :

`(5100* 5800 5800* 5300 4600* 5000 5100*)`

[the symbol * has been added to mark the notes from *list1*]

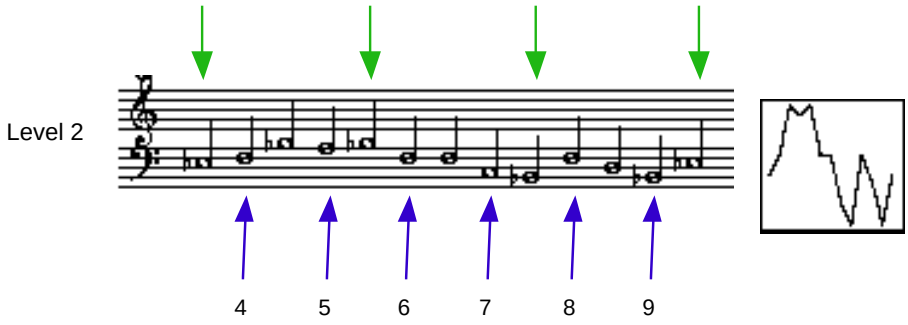
Two things should be noted :

1. The length of list one determines the end of the process

2. The F4 (6500) was transposed to F3 (5300) so that it could be inserted between the second pair of notes from *list1* :

`(5800* 4600*)`

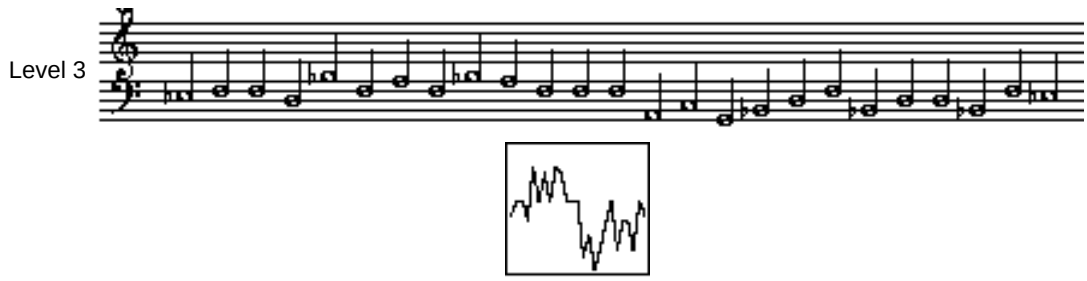
using the same *list1* and *list2* but setting *gr°* to 2 the results are as follows:



`(5100* 5300 5800! 5700 5800* 5300 5300! 4800 4600* 5300 5000! 4600 5100*)`

[the symbol * has been added to mark the notes from *list1* and the ! to mark the notes already added in the first iteration]

Notice that the added notes are 5300 (F3 transposed from F4 6500, so as to be inserted between 5100 and 5800), 5700 (A3 inserted between 5800 and 5800) and so on. At each step of the process, the module reads the first note of *list2* which has not yet been in the preceding operations. If all the notes of *list2* are used up before all of the necessary interlacing has occurred *list2* is reused in a circular manner until the process is complete.



This process is a musical transcription of the algorithm called "Midpoint- Displacement," used in the construction of fractal curves.

derivation

Syntax

(profile::derivation *list start note? gr°*)

Inputs

list list

start menu options

note? list, second element of the output from the module integration

gr° whole number greater than or equal to one

Output

list of two lists

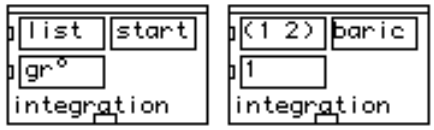
This module performs the musical transcription of a derivation, as applied to a melodic profile. In the current implementation the time interval between pitches is considered as equal to one (1). The output from this module is given in the form of a list of lists, where the first element list is the resultant derivation presented as a list of notes in midicents. The rest of the elements present the center of gravities of the derived structures. In a first degree derivation this second element list will contain a single element; in a second degree derivation it will contain two, and so on.

list either a simple list with only one level of parentheses, representing the succession of pitches in midicents or a list of lists from the output of an integration module.

start menu allowing the user to select the way in which the module will perform.
• If start is set to 'first' the input list must be connected to a simple list of pitches in midicents, representing a profile. In this case, the output will be the derivation of this profile.
• If start is set to 'orig' the input list must be connected to a list of lists, from the output of an integration module. This mode is used to reconstruct a profile that has gone through multiple successive integrations.

gr° depth or degree of the derivation.

integration



Syntax

(profile::integration *list start value gr°*)

Inputs

list list

start menu options

gr° whole number greater than or equal to one

Output

list of two lists

This module performs the musical transcription of an integration, as applied to a melodic profile. In the current implementation the time interval between pitches is considered as equal to one (1).

The output from this module is given in the form of a list of two lists, where the first element list is the resultant integration presented as a list of notes in midicents. The second element list contains the center of gravity of the integrated structure. Regardless of the degree of integration this second element list will always contain only a single element.

list either a simple list with only one level of parentheses, representing the succession of pitches in midicents or a list of lists from the output of an integration module.

start menu allowing the user to select the way in which the module will perform.
• If start is set to 'baric' he input list must be connected to a simple list of pitches in midicents, representing a profile. In this case, the output will be the integration of this profile.
• If start is set to 'orig' the input list must be connected to a list of lists, from the output of an derivation module. This mode is used to reconstruct a profile that has gone through multiple successive derivations.

gr° depth or degree of the derivation.

7 Controlled Interpolations : Interpolation

inter-dyn

Syntax

`(profile::inter-dyn` *`begin end steps tab inclu? note?`*`)`

Inputs

begin whole number, floating-point number or a list.

end whole number, floating-point number or a list.

steps whole number

tab a BPF object
`patch-work::c-break-point-function`

inclu? menu options

note? list

Output

list

Dynamic interpolation between two points, with the possibility of defining the trajectory of the interpolation.

begin initial value, either a single value or a list of values.

end terminal value, either a single value or a list of values.

steps number of steps in the interpolation

tab this module can be connected to a BPF module causing the trajectory of the interpolation between begin and end to follow the table in the BPF. If no multi-bpf module is connected to this input the interpolation will be linear.

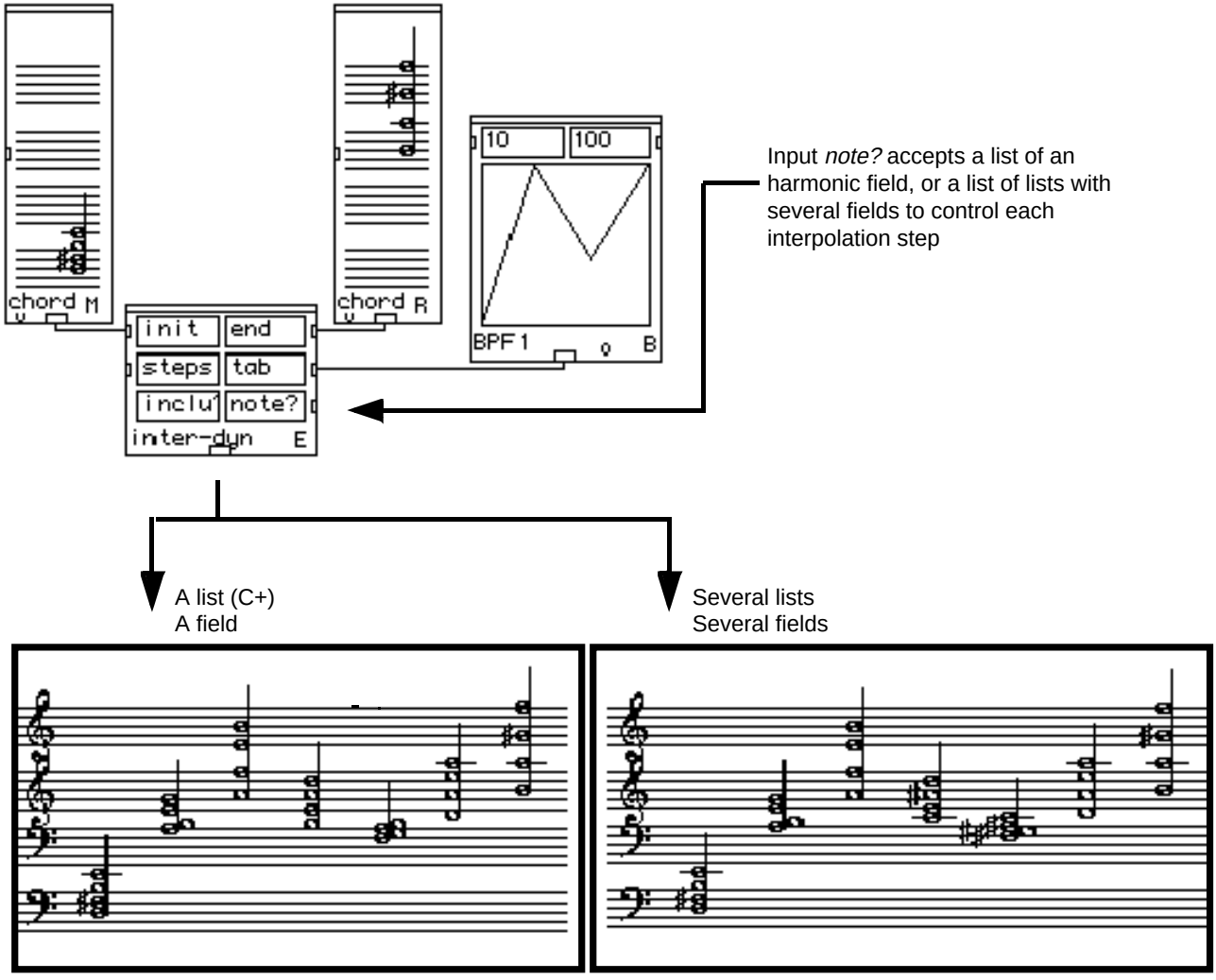
inclu? menu allowing the user to select whether or not to include the values for begin and end in the output list.
If *inclu?* is set to 'yes' the endpoints of the interpolation will be included in the output.
If *inclu?* is set to 'no' the endpoints of the interpolation will not be included in the output.

Optional Input

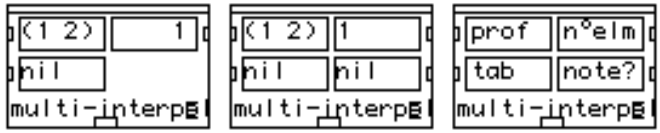
note? optional input allowing the generated form (except for the values of begin and end) to be adapted so as to conform to the one or more entered harmonic fields. *note?* may be either a simple list or a list of lists.

• If *note?* is a simple list all intermediate steps in the interpolation will be altered to conform to the notes in that list.

• If *note?* is a list of lists each intermediate step will be made to correspond to one of the sub-lists. If the number of sub-lists is smaller than the number of steps in the interpolation, the list of lists *note?* will be read circularly.



multi-interpol



Syntax

(profile::multi-interpol *prof n°elm tab note?*)

Inputs

prof list or list of lists

n°elm whole number or a list

tab a BPF object or a list of BPF objects
patch-work::c-break-point-function

note? list or list of lists

Output

list

Dynamic interpolation between the elements of a profile. This module allows the interpolation between the elements of a list of either notes or chords.

prof either a simple list with only one level of parentheses (for a series of notes), or a list of lists (for a series of chords), in either case the notes are to be given in midicents.

n°elm either a whole number or a list. This argument allows the user to chose the number of interpolative steps that will be calculated between each element (note or chord) contained in *prof*. If *n°elm* is a whole number, for example '3', three intermediate steps will be added between each adjacent element of *prof*. In this case the argument *n°elm* is applied globally to the entire sequence. However, it is also possible to connect a list to the argument *n°elm*, thus allowing the definition of a different number of interpolation to be specified between each adjacent element of *prof*. In this case the argument *n°elm* becomes a local control. For example, if *n°elm* = (3 4 5) there will be three interpolations calculated between the first pair of values, four between the second and five between the third. If the number of elements in the list connected to *n°elm* contains fewer elements than *prof* minus one, the list *n°elm* will be read circularly. In the above example, if *prof* has more than four elements, the module will go back to the beginning of the list *n°elm*, thereby calculating three interpolations between the fourth pair of elements, four for the fifth, etc.

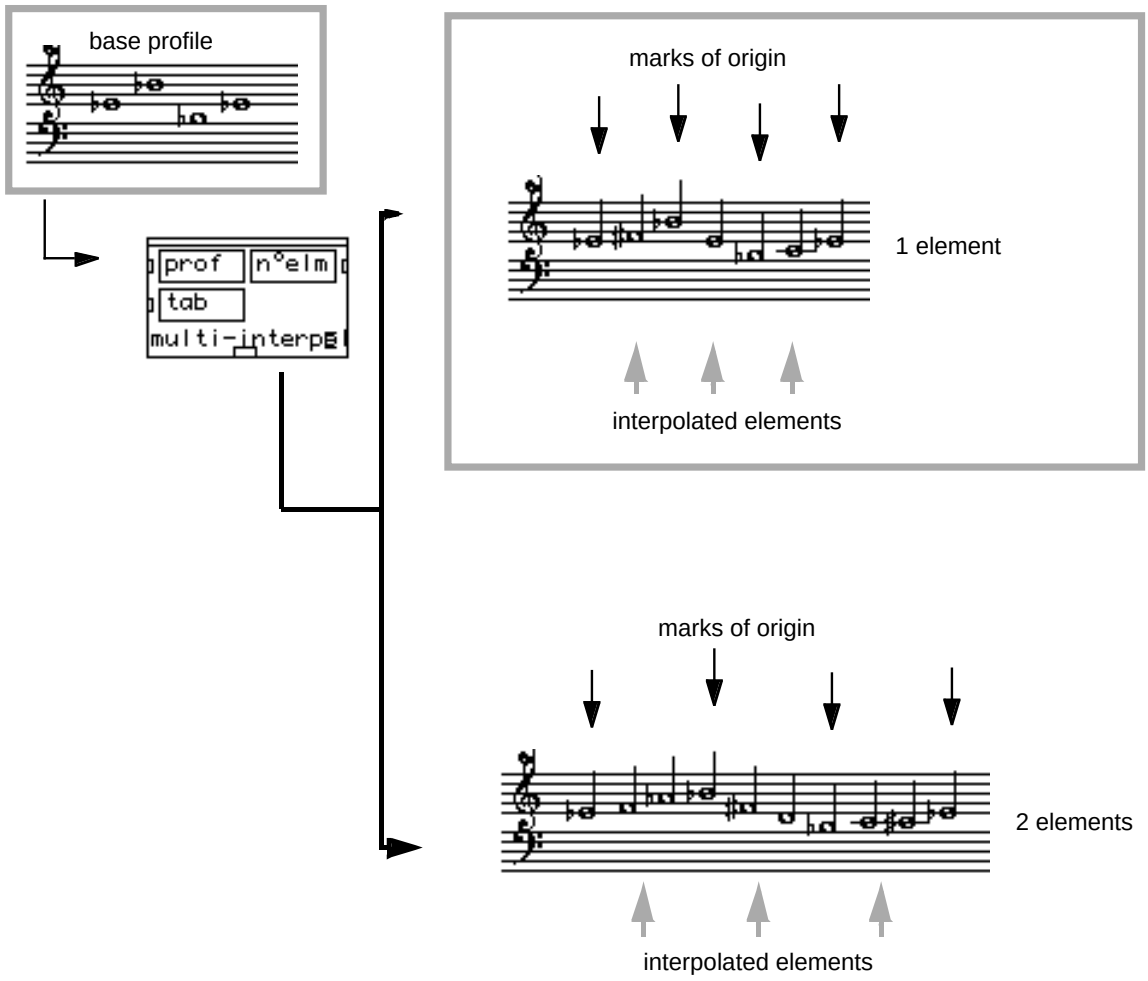
tab this input can be connected to a BPF module causing the trajectory of the interpolation between begin and end to follow the table in the BPF. If no multi-bpf module is connected to this input the interpolation will be linear. As with the input *n°elm* it is possible to connect to this input either a single multi-bpf objet, or a list of several multi-bpf objets. If *tab* is connected to a single multi-bpf object, the interpolation between each adjacent element of *prof* will follow the trajectory of that table. In this case the input to *tab* is applied globally to the entire sequence. However, it is also

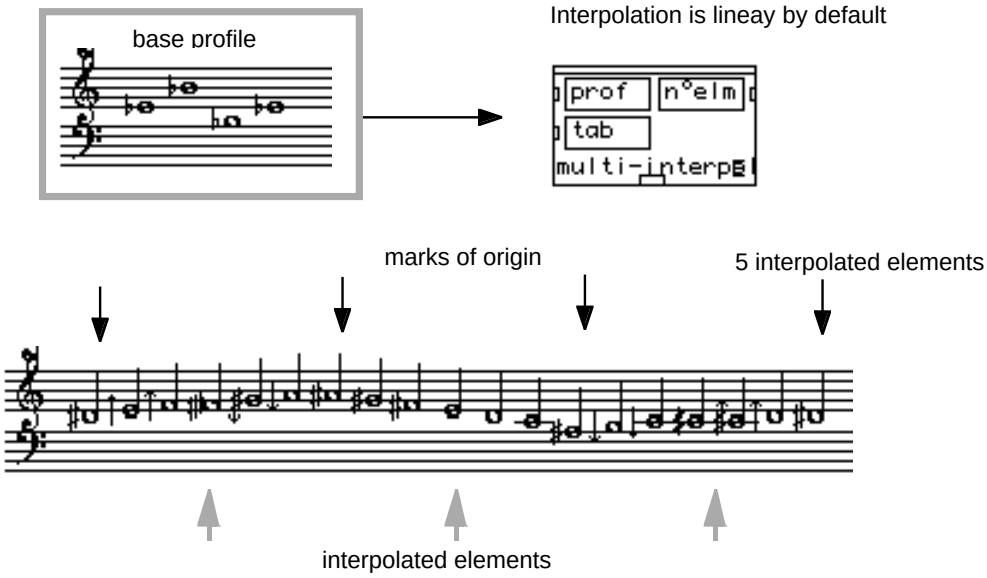
possible to connect a list to the input *tab* which will define a separate trajectory of interpolation for each adjacent element of *prof*. In this case the argument *tab* becomes a local control. If the number of elements in the list of BPFs connected to *tab* contains fewer elements than *prof* minus one, the list *tab* will be read circularly.

note? optional input allowing the generated form (except for the values of begin and end) to be adapted so as to conform to the one or more entered harmonic fields. *note?* may be either a simple list or a list of lists.

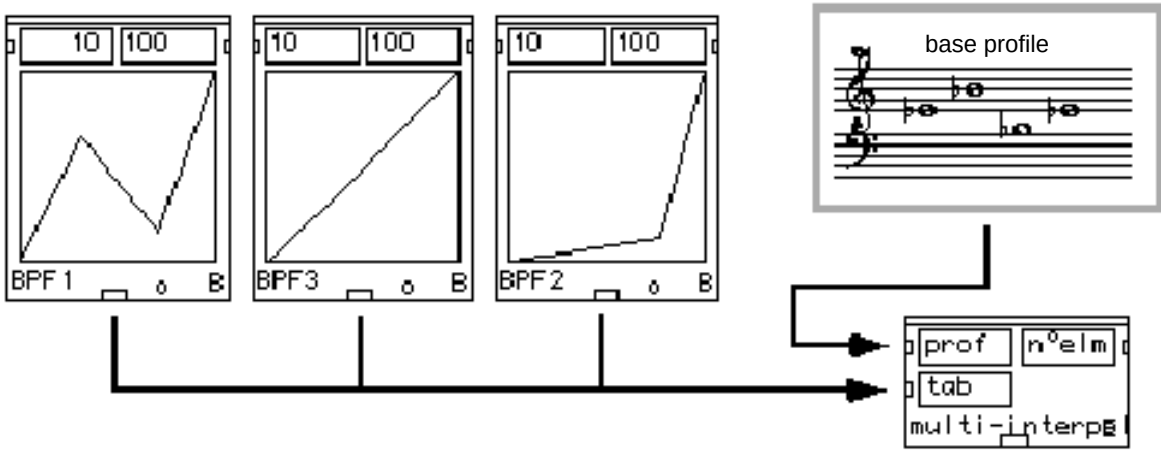
• If *note?* is a simple list all intermediate steps in the interpolation will be altered to conform to the notes in that list.

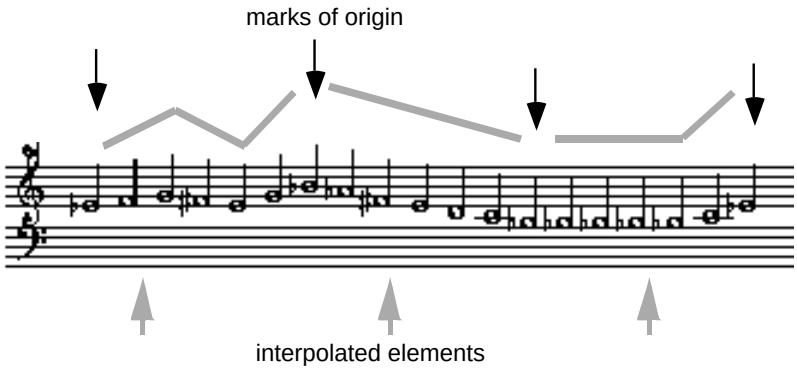
• If *note?* is a list of lists each intermediate step will be made to correspond to one of the sub-lists. If the number of sub-lists is smaller than the number of steps in the interpolation, the list of lists *note?* will be read circularly.





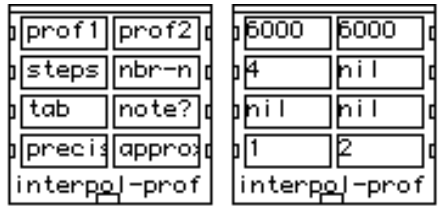
Interpolation controlled by a table (or a list of tables):





N.B. Marks of origin (Original reference points) are not affected by the harmonic control.

interpol-*prof*



Syntax

(profile::interpol-prof *prof1 prof2 steps nbr-n tab note? precis approx*)

Inputs

prof1	list containing at least three elements
prof2	list containing at least three elements
steps	whole number
nbr-n	whole number or a list
tab	a BPF object `patch-work::c-break-point-function`
note?	list or list of lists
precis	whole number greater than or equal to 1
approx	whole number greater than or equal to 2

Output

list

Interpolation between two melodic profiles, *prof1* and *prof2*, of independent lengths. This module is especially useful with long profiles. The output of the module is a list of lists with each sub-list corresponding to a profile.

prof1	a simple list with only one level of parentheses, representing the succession of pitches in midi-cents of a melodic profile.
prof2	a simple list with only one level of parentheses, representing the succession of pitches in midi-cents of a second melodic profile.
steps	number of interpolation steps to be calculated.
nbr-n	number of notes to be used in each of the intermediate profiles. If no number or list is given, each intermediate profile will have the number of notes corresponding to a linear interpolation between the number of notes in *prof1* and the number of notes in *prof2*. If *nbr-n* is a whole number, for example '5', all the newly generated profiles (*prof1* and *prof2* will not be changed) will contain five notes. If *nbr-n* is a list, each of the newly generated profiles will contain a number of notes corresponding to an element in the list connected to *nbr-n*. For example, if *nbr-n* is (2 3 4 5 6 1 2), the first profile will have two notes, the second will have three, the third four, and so on.

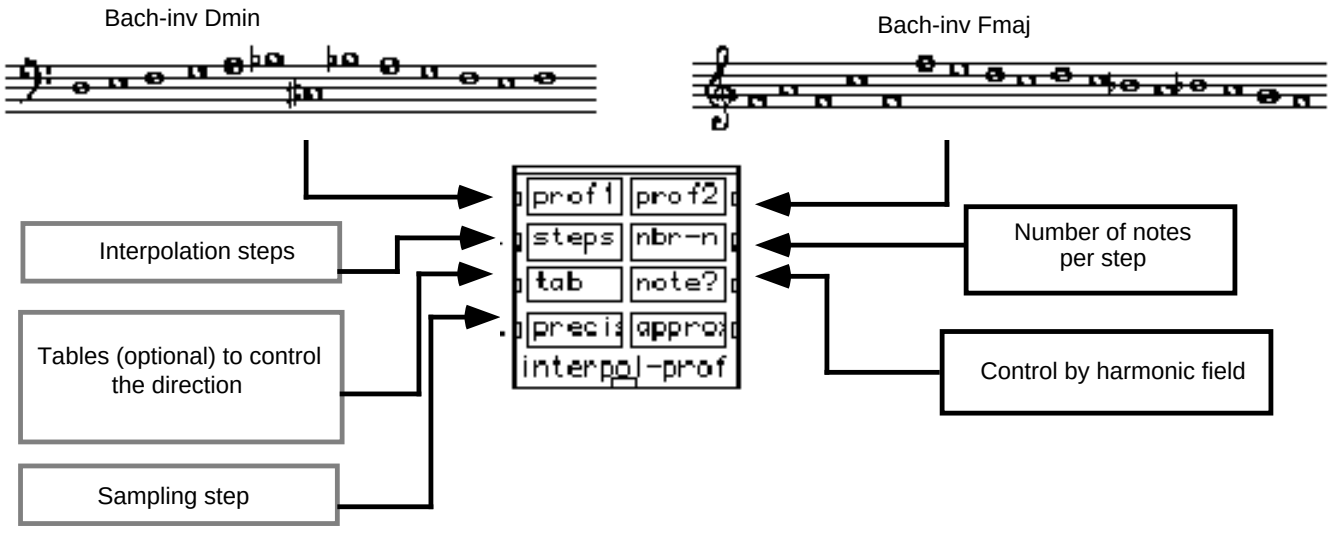
tab this input can be connected to a BPF module causing the trajectory of the interpolation between *prof1* and *prof2* to follow the table in the BPF. If no **multi-bpf** module is connected to this input the interpolation will be linear.

note? optional input allowing the generated profiles (except for *prof1* and *prof2*) to be adapted so as to conform to the one or more entered harmonic fields. *note?* may be either a simple list or a list of lists.
• If *note?* is a simple list all intermediate steps in the interpolation will be altered to conform to the notes in that list.
• If *note?* is a list of lists each intermediate step will be made to correspond to one of the sub-lists. If the number of sub-lists is smaller than the number of steps in the interpolation, the list of lists *note?* will be read circularly.

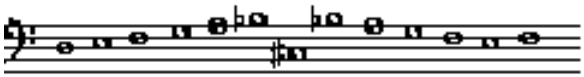
precis this module does not interpolate notes, but rather profiles; one of its steps is to convert the input list, *prof1* and *prof2*, into profiles. To do this it is necessary to establish a "sampling rate." This input (precis) establishes that sampling rate as the value of precis multiplied by the longer of the two profiles, *prof1* or *prof2*. Thus the minimum value for precis is '1.' Depending on how the module is used it may be necessary to adjust this parameter. We have found that a value of '5' seems more than adequate to any applications we have yet encountered.

approx approximation of the results.
• *approx* = 4; results approximated to the nearest quarter-tone,
• *approx* = 2; results approximated to the nearest semi-tone,
• *approx* = 8; results approximated to the nearest eighth-tone, and so on.

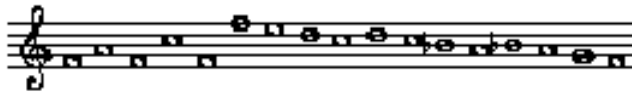
The example below is based on a BPF-interpolx which broadens the process of interpolation by applying it on both directions

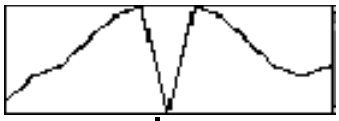


Bach-inv Dmin

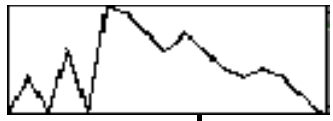


Bach-inv Fmaj



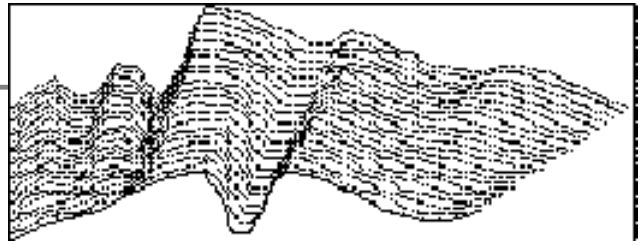


Interpolation between the profiles
with a predetermined sampling rate



(?)

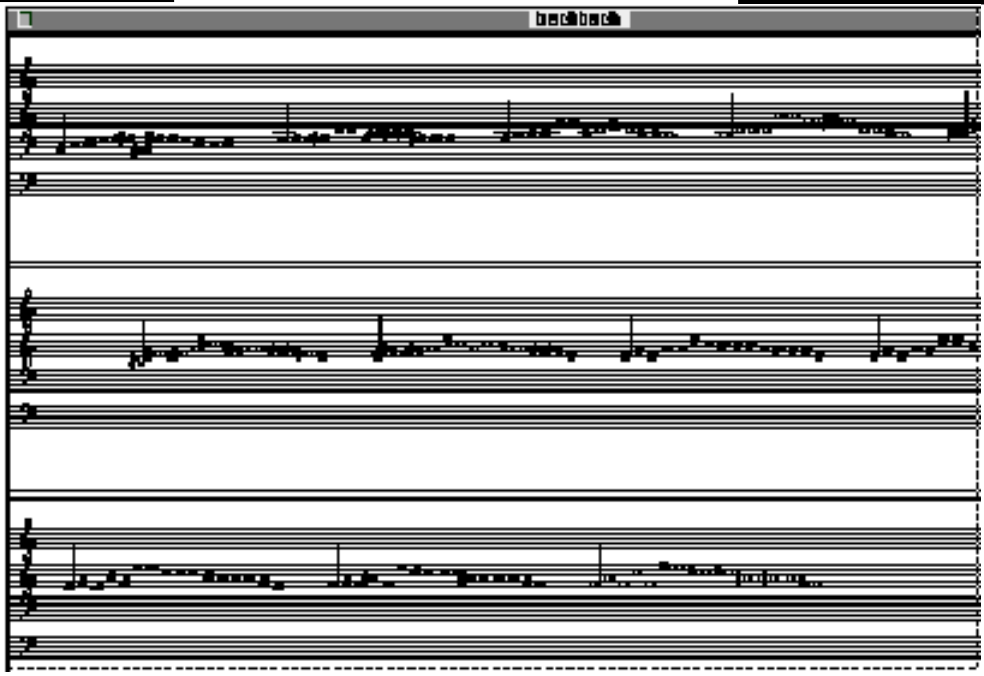
Control by tables



Sampling of profiles for
pitch conversion

(?)

Control by harmonic fields



8 Utilities Menu

range-approx

Syntax

```
(profile::range-approx list limits inclu?)
```

Inputs

list list

limits list

inclu? menu options

Output

list

Transposition of the notes (in midicents) contained in list into the register range defined by the list limits.

list a simple list with only one level of parentheses, representing the succession of pitches in midicents.

limits list containing two notes, in midicents, which define the range limits into which the notes of list will be transposed.

inclu? menu allowing the user to select whether or not the notes which do not exist in any octave within the range defined by limits will be included in the output. If *inclu?* is set to 'yes' the notes outside of the range defined by limits are placed as close as possible to one of the limits. *inclu?* is set to 'no' the notes outside of the range defined by limits are eliminated.

notes-change

Syntax

```
(profile::notes-change list note?)
```

Inputs

list list

note? list

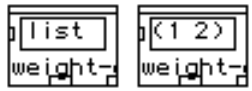
Output

list

notes-change allows a pitch or a list of pitches to be adjusted to correspond to the nearest element or elements of the harmonic field specified as the list of pitches: scale.

This module has an optional third input, mod, which indicates the modulo to be applied to the harmonic field.

weight-average



Syntax

```
(profile::weight-average list)
```

Inputs

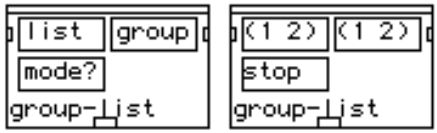
list list

Output

whole or floating-point number

Calculates the center of gravity of the pitches contained in list.

group-list



Syntax

```
(profile::group-list list group mode?)
```

Inputs

list list

group list

mode? menu options

Output

list

Articulation of the list list into segments of variable length. A second list of whole numbers group defines the length of these segments.

list any list

group list of whole numbers, which define the length of the segments.
For example, if one connects the following list to the input list (a b c d e f g h i j k l m)
and for the list group one uses (4 2 1 3 3)
the result will be:
`PW->((a b c d) (e f) (g) (h i j) (k l m))`

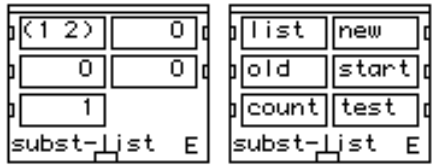
mode? menu allowing the user to select the way in which the module will function.
• If *mode?* is set to 'stop' the segmentation will be performed linearly; in other words, even if the list group contains more elements or if the sum of its elements is greater than the length of list, the segmentation will stop when the end of list has been reached.
Example:
for list = (a b c d e f g h i j k l m) and
group = (4 2 1 3 5)
the result will be: PW->((a b c d) (e f) (g) (h i j) (k l m)). If the list group is even longer, for example: (4 2 1 3 5 2) the result will still be the same:
PW->((a b c d) (e f) (g) (h i j) (k l m)).
• If *mode?* is set to 'circ' the segmentation will treat list circularly. Thus if the number of elements needed for the segmentation specified by group exceeds the number of values contained in list, the additional values will be taken from the beginning of list.

Example:
for list = `(a b c d e f g h i j k l m)`

and group = `(4 2 1 3 5 2)`
the result will be:
`PW->((a b c d) (e f) (g) (h i j) (k l m a b) (c d)).`
If *mode?* is set to 'scal' the segmentation will be performed proportionally. The articulation will take into consideration the proportions between the different elements of group and reconstruct those proportions with the number of elements in list.
Example :
If list contains the following 12 elements `(a b c d e f g h i j k l)`
and the list group is `(5 3 4)`.
the result will, of course be `PW->((a b c d e) (f g h) (i j k l))` since group asked for segments containing the same 12 elements.

However if we use the same input for list, but replace the value of group with `(10 6 8)` the proportions are the same and thus so is the result:
`PW->((a b c d e) (f g h) (i j k l)).`

subst-list



Syntax

`(profile::subst-list` *list new old tart count*`)`

Inputs

list list or list of lists

new whole number, list or list of lists

old whole number, list or list of lists

start whole number

count whole number

optional input

test symbol

Output

list

This module replaces all the instances of old from list with the element new.

list list of elements

old the element to be removed from list; it may be a number, list or symbol

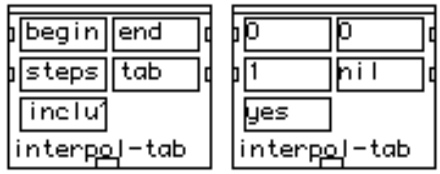
new the element which is to be placed in the position or positions formerly occupied by old; it may be a number, list or symbol

start index which specifies the earliest point in the list from which the substitution of new for old may take place. '0' (zero) indicates the first element of list.

count index which specifies how many the element old from list will be replaced.

test option argument allowing the user to specify the function of comparison to be used on the elements of list. For certain types of applications, the elements to be replaced are of diverse types; thus the elements to be compared are also of diverse types. This situations may require the use of special comparative functions. The default function of comparison is 'equalp' which is a weak equality function that is effective for comparing numbers, lists and symbols.

interpol-tab



Syntax

(profile::interpol-tab *begin end steps tab inclu?*)

Inputs

begin whole number, floating-point number or list

end whole number, floating-point number or list

steps whole number

tab a BPF object
patch-work::c-break-point-function

inclu? menu options

Output

list

Dynamic interpolation between two points, with the possibility of defining the trajectory of the interpolation.

begin initial value, either a single value or a list of values.

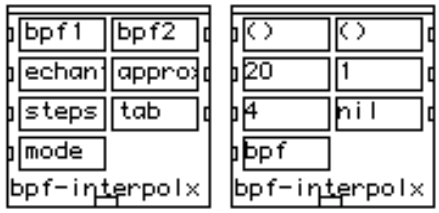
end terminal value, either a single value or a list of values.

steps number of steps in the interpolation

tab this module can be connected to a BPF module causing the trajectory of the interpolation between begin and end to follow the table in the BPF. If no **multi-bpf** module is connected to this input the interpolation will be linear.

inclu? menu allowing the user to select whether or not to include the values for begin and end in the output list.
If *inclu?* is set to 'yes' the endpoints of the interpolation will be included in the output.
If *inclu?* is set to 'no' the endpoints of the interpolation will not be included in the output.

bpf-interpolx



Syntax

(profile::bpf-interpolx *bpf1 bpf2 echant approx steps tab mode*)

Inputs

bpf1 a BPF object
patch-work::c-break-point-function

bpf2 a BPF object
patch-work::c-break-point-function

echant whole number

approx whole number

steps whole number

tab a BPF object
patch-work::c-break-point-function

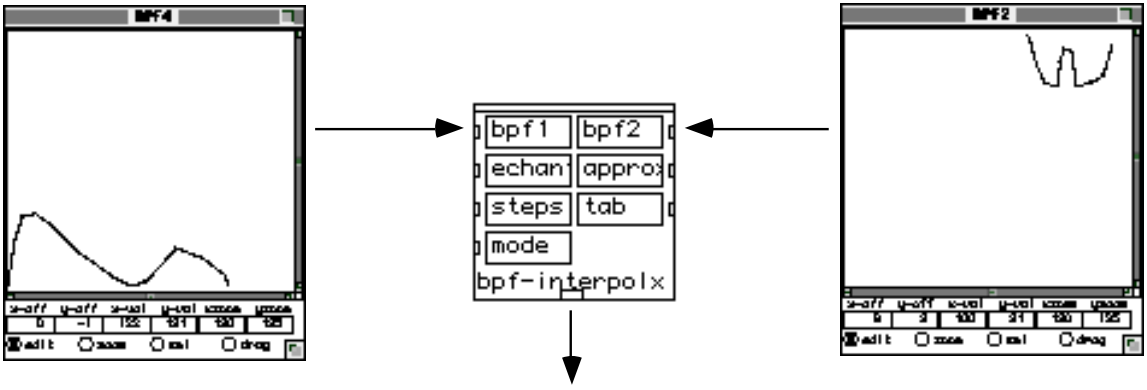
mode menu options

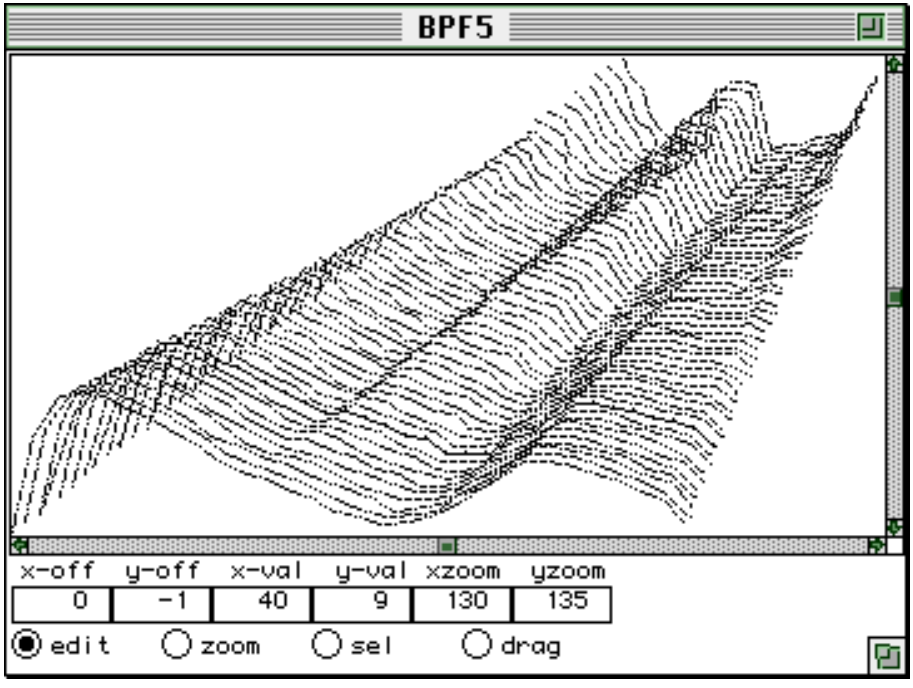
Output

list

Interpolation between two tables represented as break-point functions bpf1 and bpf2 (these tables may be of any size). This module is most effective when the tables contain a large number of points. The output from the module is either a list of lists, where each sub-list contains the coordinates of a different intermediate table between *bpf1* and *bpf2*, or a list of BPF objects.

The BPF-interpolx module performs a complete interpolation in two dimensions. The trajectory of the interpolation may be specified as anything other than linear, through the use of a BPF object attached to the input *tab*.





Note that BPF-interplox does not interpolate points, but two-dimensional profiles.

bpf1 a simple list with only one level of parentheses, representing a melodic profile of pitches in midi-cents.

bpf2 a simple list with only one level of parentheses, representing a second melodic profile of pitches in midicents.

echant since this module does not interpolate notes, but rather profiles; one of its steps is to convert the inputs, bpf1 and bpf2, into profiles. To do this it is necessary to establish a "sampling rate." This input (échant) establishes that sampling rate (taux d'échantillonnage, in French). The value of

échant defines the number of points needed in the sampling. We have found that a value of for échant of 5 times the length of the longest profile seems adequate to any applications we have yet encountered.

steps number of interpolative steps to be calculated.

tab this input can be connected to a BPF module causing the trajectory of the interpolation between bpf1 and bpf2 to follow the form of the table in the BPF. If no multi-bpf module is connected to this input the interpolation will be linear.

mode menu options which define the format of the module's output.
If mode is set to 'bpf,' the output will be a list of BPF objects.
If mode is set to 'list,' the output will be a list of lists, where each sub-list contains in turn two other sub-lists: the fist containing the horizontal points and the second the vertical points of each intermediate profile.

9 Bibliography

Baboni Schilingi, J., *Composizione per Modelli Interattivi*, publication currently in progress, 1995

Barrière, Jean-Baptiste, « Chréode I: chemins vers une nouvelle musique avec ordinateur », in *L'Ircam: Une pensée musicale*, éditions des archives contemporaines, Paris, 1984.

Kandinsky W., *Punto, linea, superficie*, Adelphi 16, 1968.

Malt M. and J. Baboni Schilingi, « Profile- libreria per il controllo del profilo melodico per Patchwork, », XI Colloquio di Informatica Musicale, Bologna, 1995.

Mc Adams, Steve et A. Bregman, « L'audition des flux musicaux », *Marsyas*, Institut de pédagogie musicale et chorèographique, La Villette, Paris (3-4) December 1987.

Murail T., « Spectres et lutins », in *L'Ircam: Une pensée musicale*, éditions des archives contemporaines, Paris, 1984.

Murail T., « Questions de cible », *Entretemps* n° 8, 1989.

Saariaho K., « Timbre et Harmonie », in Le *timbre, métaphore pour la composition*, Jean-Baptiste Barrière (ed.), Christian Bourgois Éditeur, Ircam, Paris, 1991.

Index