SOAL

SONIC OBJECT ANALYSIS LIBRARY

OpenMusic Tools for analyzing musical objects structure

Software conception by Didier Guigue [1]. Developed for OpenMusic by Ernestro Trajano [2], Ana Carolina Barbosa, Nazareno Ferreira, Bruno Jefferson, André Lira Rolim, Max Porfírio and Hildegard Paulino Barbosa [3]. MacIntel first portage by Jean Bresson. Documentation by Didier Guigue. Doc. version: 300, 06/02/2010.
SOAL is part of the *Mus³* software research project.
Mus³ – Musicologia, Sonologia & Computação – is a Research Group of the UFPB – *Universidade Federal da Paraíba* (Brazil) – funded and directed by Didier Guigue. It has the support of IRCAM, France, and research grants and funds from the CNPq – Brazilian Council for Research.
Contact: didierguigue@gmail.com –
News & updates: www.cchla.ufpb.br/mus3.

WHAT'S NEW IN SOAL 3.0

SOAL 3.0 offers a new set of functions designed to assist the segmentation of MIDI files, an important step onward for SOAL functionality for analysis, as they allow to segment the music into smaller units that make musical (structural) sense. The resulting segments are saved into small MIDI files ready to be analyzed by the SOAL other analytic functions (see folder "SOAL segmentation").

COMPATIBILITY

SOAL 3.0. was developed and tested on (a) non-Intel Macintoshes with OM version 6, (b) on MacIntel Macintoshes with OM version 6 and (c) on PCs Windows with OM version 6. It should also work with the Classic OM versions (not tested with versions older than 4.5).

---

1 A former version of this library was made for *Patchwork* by Didier Guigue, with the collaboration and help of Mikhail Malt, Camilo Rueda & Gérard Assayag.
2 Part of a Master degree research, with a grant from the CAPES, Brazil.
3 Under-graduate one-year research projects since 2001, with grants from the CNPQ, Brazil.

INSTALLATION INSTRUCTIONS

Put the “SOAL 3.0” folder in the UserLibrary folder. Put the folder “examples” into your own Workspace’s “elements” folder, in order to open the demo patch and demo MIDI files, which work as tutorial files. The demo patch may ask for you to download the MIDI files we put into the folder SOAL examples/SOALmidi examples. Obviously you can use any other MIDI file of your own, or open the patch without downloading any MIDI file. Please report us bugs, opinions and suggestions!

STRUCTURE OF THE LIBRARY

SOAL functions analyze sequentially any number of MIDI files attached to the *Multi-Midi-Reader*. The files are supposed to be segments of a single piece of music, although *SOAL* can also be useful to compare samples from different works, or whole pieces of music. The outputs allow the user to get consistent quantified data of how some music’s statistical dimensions, such as densities, progress during the piece (s).
SOAL evaluates separatedly the “achronic” (i.e. spatial, vertical or *out of time*) content of each individual MIDI file, and its “diachronic” (temporal) organization. This is why *SOAL* is splitted into two main folders.

The achronic analysis folder deals only with spatial (vertical) distribution of pitches: all the pitches of the input file (s), accompanied by the available information on their relative intensity (MIDI velocity) and other relevant data (for example, pedals in piano music) are placed at point 0 on the time vector.

For the diachronic analysis folder, which includes several time-distribution models, the input file must also provide the time-related data (onsets and durations).

The piano specific folder contains analytic tools optimized for piano music, while the stats & utils folder offers a collection of more generic functions for statistical analysis purposes, among others, various deviation and distribution models. The soal midi reader folder contains the main *SOAL* interface between the MIDI file and the analytic functions.

THE *SOAL* OUTPUT : THE “RELATIVE COMPLEXITY” VECTOR

The basic *SOAL* rule for evaluating how important is a given component in the design of the structure of a given file, is the rating of its relative “complexity”, namely the *complexity weight* (Guigue 2009). The maximum value for “complexity” corresponds to the configuration that would produce as “complex” a sonority as possible. In all *SOAL* algorithms, this maximum complexity is the *paradigm* the input file is compared to. Thus all results are calibrated on a vector from (0.00), which means “maximum simplicity”, to (1.00), which means “maximum complexity”. So, when a function returns a *complexity weight* of (1.0), it means that the input file is as “complex” as the maximum complexity paradigm. The specific meaning of “simplicity” and “complexity” varies according to the nature of the parameter. They may in some cases be described as vectors of poorness/richness, weakness/strongness, emptiness/fullness, consonance/dissonance, etc...

The *SOAL* relativity rule seems to be the best way to compare and process together heterogeneous parameters, to correlate them e.g. using standard statistic algorithms, to study how diachronic behaviour can modulate achronic structures, and to compare different files in different musical situations. Besides the complexity weight, some *SOAL* tools may also output other optional useful informations, some of them not relative.

In this documentation, unless otherwise specified, all output are relative and appear on a numeric scale from (0.00) to (1.00).

THE SOAL INPUT

All *SOAL* inputs are intended to receive symbolic data in MIDI format. Namely notes (in midicents), onsets (in milliseconds), durations (in milliseconds), velocities (an integer from 0 to 127), and any kind of other MIDI control data (an integer from 0 to 127). Do [Command-click] upon the function's input button to read what data the *SOAL* input is expecting. That means *SOAL* tools can directly read *OpenMusic* outputs from chordseqs, chords, Midi factories. Notes in common MIDI format (an integer from 0 to 127), as they are for instance when imported by the MIDI factory, are automatically transformed into midicents by the *SOAL MidiReader* (see below). Musical data in other formats (Herz, dBs, etc) may yet be usefully analyzed by some *SOAL* tools.

IMPORTING AND EVALUATING MIDI FILES FOR ANALYSIS

For people which want to work directly with MIDI files, *SOAL* package provides a ***MultiMidiReader*** function (in the folder *Midireader*) which imports as many MIDI files the user put in and extracts and isolates five data (from left to right): "notes", "onsets", "durations", "velocities", and "continuous control #4" (usually affected to pedal). The reader also allows, through its first (left) input, to quantize (to round) the onsets and durations data. So DON'T CONNECT A MIDI FILE OUTPUT INTO THE FIRST INPUT!

To add a MIDI file input on the *Reader*, click option-right arrow and connect an *OpenMusic* MIDIfile standard box (previously feeded in with the MIDI file you want to connect, and locked). A demo *SOAL sample patch* with *SOAL MultiMidiReader* prompted to work is available in the *SOAL* Examples folder. *MultiMidiReader* also reads a single MIDI file.

SOAL'S FUNCTIONS DESCRIPTION

Achronic analysis folder

Spatial analysis folder

Spatial-analysis

This tool mainly informs the input file's *RELATIVE RANGE. Range* (ambitus) means a pitch space defined by two boundary pitches. *Relative range* compares the input file's pitch range to a predefined global pitch range (which can be for instance the whole music pitch range, the instrument or ensemble global pitch range, or the human ear pitch range). The output is obtained by dividing the interval between the input file's lowest and highest pitches by the global pitch range.

The patch can also provide complementary *REGISTER* informations, particularly the *REGISTER DISTRIBUTION*, which informs how pitches are distributed along the file's pitch range. The user must declare in how many slices (registers) the global pitch range is to be partitioned — the default partition, by octaves, can be useful in many cases. Moreover, if he wants so, the user may define a quality weight to each of these registers. Thus the algorithm informs how many notes are in how many registers, and can give to the input file a sonic quality weight according to the way the pitches are distributed in the different registers.

Input (from left to right)

Midicents: a list of lists (in midicents).
Range: a single value in midicents, which informs the interval of the global range (ambitus) of the instrument or music or whatever global pitch range the input data is to be compared. Enter the value with a standard ‘numbox’, without parentheses. The default value (8700) means an instrument which range is 88 notes, e.g. a piano. Thus if the global range is to be 12 notes (an octave) one must enter ‘1100’, and so on.
Registers: a list of sub- lists of two elements in midicents, which indicates the lowest and higest pitch boundaries of each of the registers (slices) in which the instrument or music is to be partitioned. The partition can be complete or not, continuous or not. For continuous and complete partition, the first element of the first sub-list must match the lowest pitch of the instrument or music, and the second element of the last sub-list, its highest pitch. The default values partition the input file into 9 registers of 1 octave each, from C to B.
Reg(ister) weights list: a list of values which sum must be 1. Each value gives a qualitative weight - scaled 0-1 - to the corresponding register. There must be as weight values as there is register partitions, and their sum must always be 1. The default value (1) allows to bypass this parameter.
Remove-duplicates: if YES (1) (the default, recommended), removes the duplicates values of the midicents list.

Output (from left to right)

[1] Gives a list of lists of the following values in this order*: relative range, absolute range, register-filling, register distribution, average spatial quality*;
[2] Returns the file’s *average spatial quality* — i.e. the average value of [3] (relative pitch range) and [10] (register distribution).
[3] Gives the *relative range* of the input file, compared to the predefined global pitch range; a value (1.0) means the input file occupies the whole global range;
[4] Gives the *absolute range* of the input file, i.e. the difference between the highest and the lowest notes.
[5] Gives the number of notes per register - a list of integers, ordered from lowest to highest register; no note in a register returns 0;
[6] Gives the *register-filling*, a relative value which gives the filling rate of the registers, compared to their total number; a value (1.0) means each register contains at least one note;
[7] Is the list of the numeric label of the felt registers; the label is an integer from 0 upwards (0 is the lowest register);
[8] Returns the lowest value of [6].
[9] Returns the highest value of [6]. These two outputs are useful when evaluating a sequence of files (midifiles), because they can be used to draw the graphic envelope of register-filling;
[10] Gives the *register distribution*, a relative weight which is based on the user-defined 'register-weight' input.

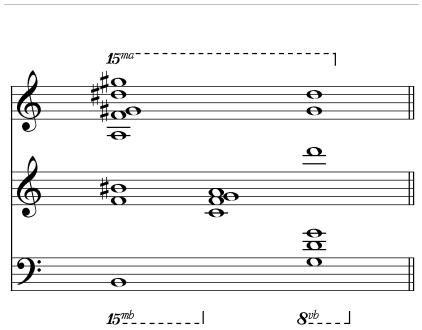

Fig. 1: Three chords — a “large” one, a “narrow” one, and a typical chord from Debussy’s “La Cathédrale engloutie” — analyzed by the *Spatial Analysis* algorithm. All inputs are default, except for the “range” input which has been extended up to 107 notes (‘range’ value 10600). These are some of the results:

OUTPUT	“Large” chord	“Narrow” chord	“Cathédrale” chord
Relat. Range	(0.75)	(0.07)	(0.63)
Register filling rate	(0.56)	(0.11)	(0.44)
n. of notes p. reg.	(0 1 0 0 2 1 2 2 0)	(0 0 0 0 4 0 0 0 0)	(0 1 2 0 0 0 2 1 0)
reg. labels	(1 4 5 6 7)	(4)	(1 2 6 7)

Spatial filling analysis folder

While *Spatial Analysis* gives informations about the overall spatial structure of the input file*, Spatial Filling Analysis* provides complementary and more detailed data about the way the pitches are distributed inside this spatial configuration, through three independent functions.

Spatial-density

Gives the relative density of the input file, which is obtained by dividing the total number of pitches by the theoretical maximum possible number of them, within the actual range of the file. For instance, a musical file in a typical chromatic *cluster* form would receive the maximum weight (1.00).

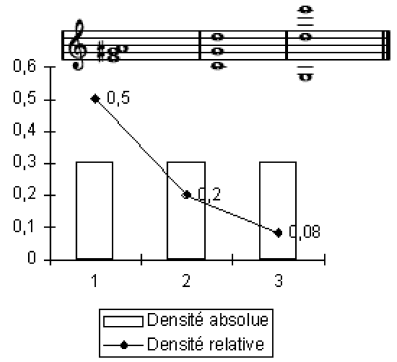

Fig. 2 : These 3 chords have the same absolute density (i.e. the same number of notes) but their relative density is not the same, due to their different ambitus. *SPATIAL-DENSITY* returns respectively (0.5), (0.2) and (0.08) — that means 50%, 20% and 6% of their respective saturation.

Input

Midicents: a list of lists (in midicents).
Remove-duplicates: if YES (1) (the default, recommended), removes the duplicates values of the midicents list

Output

[1] Gives the relative density value of each input file.

Spatial-linearity

Measures how equidistant are distributed the pitches. The interval between the contiguous pitches is compared to a paradigmatic interval which corresponds to the interval that would be necessary for pitches to be exactly equidistant (chromatically approximated). The higher, the weight, the less linear, the pitch distribution and, consequently, the more “complex” the file.

Input

Midicents: a list of lists (in midicents).
Remove-duplicates: if YES (1) (the default, recommended), removes the duplicates values of the midicents list.

Output

[1] Gives the spatial-linearity weight of each input file. Caution: (0.0) means *maximum* linearity, (1.0) means *minimum* linearity (i.e. *maximum* complexity).
[2] Gives the paradigmatic interval (in mcs).

Harmonicity

Evaluates the closeness of the input file's achronic pitch distribution to a paradigmatic harmonic spectrum-like structure, deduced from a real or virtual fundamental, and constructed inside a user-given spectrum bandwidth. The fundamental may be the actual (i.e. real) lowest pitch of the file, or a downward transposition (i.e. virtual) of it, one or two octaves below. The algorithm choose automatically after evaluating the distance between the first two lowest pitches of the file, but the user can force for the only real fundamental calculation through the "use-virtual-fundamental?" optionnal input. The algorithm takes also into account the actual pitch of the lowest tone of the object: a "threshold" input allows the user to force the fundamental detection algorithm not to transpose the lowest pitch if it is situated below this threshold, and to transpose it only an octave lower if it is located within an octave of the threshold. The user can also enter a fixed fundamental ("user-fundamental" input). The spectral bandwidth is computed according to the rank of the upper partial of the paradigmatic harmonic structure. This rank, an integer, is given by the user ("range" input).

Input

Midicents: a list of lists (in midicents).
Range: a single integer, which informs the number of partials (harmonics) to be taken into account for the building of the paradigmatic harmonic spectrum the input file (s) will be compared to. Note: a unique value for *all* input files. Recommended values: 10 to 16 (default).
Use-virtual-fundamental?: if YES(1) (the default), the virtual fundamental detection algorithm is activated. If NO (0), the fundamental is the lowest pitch of the input file(s).
User-fundamental: a pitch (in midicents) to be used as a fixed fundamental (for all the input files). If different of 0 (the default), all other fundamental calculation options are thus by-passed.
Threshold: a single value in midicents (default 3500, well suited for piano music), used by the virtual fundamental algorithm; avoids transposing any pitch below it, and other restrictions.
Round-harmonic-object?: if YES(1) (the default), the paradigmatic harmonic spectrum is chromatically rounded. If NO (0), no.
Transpose-pitches? if YES(1) (the default), the *E-deviation* algorithm (see above) transposes down the input pitches which stay outside the user-defined harmonic range (input 2), in such a way they are also included in the harmonicity weighting. If NO (0), these pitches are excluded. Caution: with the "NO" option, it may remain very few pitches for the algorithm to compare with the paradigmatic spectrum.
Remove-duplicates: if YES (1) (the default, recommended), removes the duplicates values of the midicents list.

Output

[1] Harmonicity: the average value of [2] and [3]. Caution: (0.0) means maximum harmonicity, or a single note object, and (1.0) means maximum inharmonicity.
[2] E-Deviation: The deviation between the file's pitches and the deduced harmonic series from the fundamental.
[3] Harmonic Weight: The relative weight of the position of the file's pitches in the harmonic series. The higher the position, higher the weight.

[4] Fundamental: A list with the fundamental note used by the algorithm (C4 = mcs 6000) and its midicent corresponding value.
[5] Coincident harmonics: the harmonic ranges' list of the file's pitches that match the harmonic series.

Cognitive Sonance

A vector from “maximum consonance” to “maximum dissonance”, evaluates the contiguous dyads of the input file (s) on the basis of a weighting of their relative dissonance, according to a pre-defined cognitive/cultural background, and extracts an average value for the whole MIDI file, its “sonance” rate. The user has to enter a list of his/her own ranking of the twelve prime-intervals (integers from 1 to 12), ordered from the most “consonant” (usually 12, first value of the list) to the most “dissonant” (last value). In our model, and according to the literature, the sonance-weight of any of that prime intervals linearly decreases as a function of the distance, expressed in octave-units, between the two pitches, until it reaches a null weight (that is, until it becomes fully consonant) when the higher pitch assumes a harmonic relation to the lower one. The default sonance weights, implemented in the 2d input, based upon Guigue 1996, 1997, are shown in the following table. The *sonance* rate (3d output) corresponds to the mean value of all dissonances weights [4].

The default sonance weights, implemented in the 2d input as a (strong) suggestion, are based upon Guigue 1996, 1997. They are shown in the following table, which shows also the “decreasing” functions.

The “interval” column show the prime intervals, ranged top-down from the most consonant to most dissonant (according to Guigue 1996, 1997). The “ƒH (G)” column informs the partial range which corresponds to the dyad’s upper pitch, taking the lower pitch as its fundamental. This range determines the sonance weight of the corresponding prime interval (col. “Oct. 0”), again according to Guigue 1996, 1997. “Oct. 1” to “Oct. 4” show the decreasing of the prime weight, as a function of the distance from the fundamental, in modulo-12 (octaves). The “zero” weight means that the prime interval reached a harmonic ratio with the lowest pitch of the dyad.

Interv.	ƒH (G)	Oct.0	Oct.1	Oct.2	Oct.3	Oct.4
12	2	0.00	0	0	0	0
7	3	0.09	0	0	0	0
5	4	0.18	0.09	0	0	0
4	5	0.27	0.135	0	0	0
9	5	0.36	0.18	0	0	0
8	5	0.45	0.225	0	0	0
3	6	0.54	0.27	0	0	0
6	7	0.63	0.315	0	0	0
10	7	0.72	0.36	0	0	0
2	9	0.81	0.54	0.27	0	0
11	15	0.90	0.60	0.30	0	0
1	17	1.00	0.75	0.5	0.25	0

Input

Midicents: a list of lists (in midicents).
Intervals: a single list of 12 integers. ordered from the most consonant to the most dissonant

[4] The *SOAL* team is actually working on complementar psycho-acoustic-oriented functions for evaluating the “sensorial” sonance.

interval. Default: the values in the above table, “interv.” column.

Output

[1] List of lists containing the sequence of each file intervals (integers starting from 1 for minor second), from the lowest pitch up.
[2] The sonance weight of each on these intervals.
[3] Average value of each [2] list, which is the sonance weight of each input file."

Velocity Rate

From the input file(s)’s velocity list, extracts a global velocity rate. This is done by calculating the average (or mode) value of the list, then dividing it by a paradigmatic value which uses to be the common MIDI highest velocity value (i.e. 127). However the user can enter another divider. Another input allows the user to choose between the average of mode options.

Input

Velocity: a list of lists (in midi).
Velo-parag: paradigmatic value (a divider).
Output choice: average (0) or mode (1).
Rounding: num

Output

[1] A list of lists of velocity rates
[2] A list of absolute average (or mode) velocity of each input list.

Diachronic analysis folder

Besides pitches and velocity, time-related analysis need also an *onset* list, which informs the position of the beginning of each note event, and a *duration* list, which gives the duration of each note event. Lists must be in ms (milliseconds).

Span-Analysis folder

Smaller-Impulse

Looks for and returns the smaller duration value of the file(s).

Input

Onsets: the “onsets” list (in ms) of each file.
Durs: the “durations” list (in ms) of each file.

Output

The smaller impulse of each list, in ms.

File duration

Returns the absolute and relative duration of the input file(s), in ms.

Input

Onsets: the “onsets” list (in ms) of each file;
Durs: the “durations” list (in ms) of each file.

Output

[1] Absolute duration of each input file, in ms.
[2] Relative duration of each file of the list. The longer file weights (1.0).

Relative Span

Returns the relative duration of the input file(s), in ms, allowing the user to enter (in the 3rd input) an absolute value (in ms) as reference (The default value 0 bypass this function).

Input

Onsets: the "onsets" list (in ms) of each file.
Durs: the "durations" list (in ms) of each file.
Ref. Duration: an absolute reference duration (in ms) (the divider of file's durations).

Output

[1] Relative duration of each file of the list. The longer file weights (1.0), except when 3d input is used.
[2] A list containing ((the real duration of the longer file of the list) (a list containing the real duration of all files of the entry)).

Analysis-per-onset folder

The "per-onset" functions look at the evolution of the achronic qualities, onset by onset, file by file.

Notes-per-onset

Is a multi-purpose tool which informs how many and what notes are present at each onset. It allows an analysis of the pitch content evolution along the duration of the input file(s). The outputs are lists formatted ((onset 1 midi data)(onset 2 midi data) etc…), where the *midi data* info for each onset depends on the output option. These options are: the list of notes, the number of notes, the lowest note, the highest note, the average note, the most frequent (*mode*) note (returns the average note for list with no duplicates). When patched with standard *OM* functions *flat-once*, *mat-trans, second* and *flat*, in this order, the outputs return a single list which can produce a curve showing the sequential evolution of the pitch content.

Inputs

Midicents: the "midicents" list of each MIDI file;
Onsets: the "onsets" of each MIDI file.

Outputs

[1] A list of the notes (midicents) on each onset (of each input file);
[2] The number of notes per onset;
[3] The lowest note per onset (smallest midicents value) (for each file);
[4] The highest note per onset (greatest midicents value) (for each file);
[5] The notes' average value (in mcs) per onset (for each file);
[6] The notes' mode value (in mcs) per onset, i.e. the most frequent note on each onset. If not existing, the algorithm returns back the average note (as in the 5th output).

Velocity-per-onset

Makes the same as “notes-per-onset”, for the velocities on the input files.

Inputs

Velocities: the “velocity” list of each MIDI file;
Onsets: the “onsets” of each MIDI file.

Outputs

[1] List of the velocities per onset (of each input file);
[2] The number of velocities per onset;
[3] The lowest velocity value on each onset (for each file);
[4] The highest velocity value on each onset (for each file);
[5] The velocities’ average value per onset (for each file);
[6] The velocities’ mode value per onset. If not existing, the algorithm returns back the average velocity (as in the 5th ouput).

Spatial-linearity-per-onset

Measures how equidistant are distributed the pitches. See *spatial-linearity* for more details.

Input

Midicents: a list of lists (in midicents).
Remove-duplicates: if YES (1) (the default, recommended), removes the duplicates values of the midicents list.

Output

[1] Gives the spatial-linearity weight per onset. Caution: (0.0) means *maximum* linearity, (1.0) means *minimum* linearity (i.e. *maximum* complexity).
[2] Gives the paradigmatic interval (in mcs), per onset.

Harmonicity-per-onset

Returns the harmonicity rate per onset according to a choice of criteria (see outputs below). See *harmonicity* function for more details.

Inputs

Notes: list of notes (in mcs) of each input file;
Onsets: the “onsets” of each input file.
Other inputs: see *harmonicity* function.

Outputs

[1] Harmonicity per onset: the average value of [2] and [3]. Caution: (0.0) means maximum harmonicity, or a single note object, and (1.0) means maximum inharmonicity.
[2] E-Deviation per onset: The deviation between the file's pitches and the deduced harmonic series from the fundamental.
[3] Harmonic Weight per onset: The relative weight of the position of the file's pitches in the harmonic series. The higher the position, higher the weight.
[4] Fundamental per onset: A list with the fundamental note used by the algorithm (C4 = mcs 6000) and its midicent corresponding value.
[5] Coincident harmonics per onset: the harmonic ranges' list of the file's pitches that match the harmonic series.

Density-per-onset

Returns the density qualities per onset according to a choice of criteria (see outputs below). See *density* function for more details.

Inputs

Notes: list of notes (in mcs) of each input file;
Onsets: the "onsets" of each input file.

Outputs

[1] Relative densities per onset;
[2] Average density value from [1] (one value per file);
[3] Weighted index of [2] (1.0) corresponds to the higher value of [2]);
[4] Absolute densities per onset (an integer corresponding to the number of notes per onset);
[5] Average value from [4];
[6] Weighted index of [5] (1.0) corresponds to the higher value of [5]).

Cognitive-Sonance-per-onset

Returns the cognitive sonance rate per onset. See *Cognitive-Sonance* function for more informations.

Inputs

Midicents: a list of lists (in midicents).
Intervals: a single list of 12 integers. ordered from the most consonant to the most dissonant interval. Default: the values in the above table, "interv." column.

Output

[1] List of lists containing the sequence of each file intervals (integers starting from 1 for minor second), from the lowest pitch up, per onset.
[2] The sonance weight of each on these intervals, per onset.
[3] Average value of each [2] list, which is the sonance weight per onset.

Time Filling Analysis

This folder provides qualitative data about the way the pitches are distributed along the time, by means of six different and complementary tools. The user may choose the most relevant ones, according to the analytic aim.

Events-Density

An *event* can be one single pitch or any number of simultaneous pitches. The relative density is obtained by dividing the number of events by the maximum possible number of them, within the actual duration of the file, as informed by the enclosed *file-duration* function. This maximum is computed from the smallest observed pulse unit, provided by the enclosed *smaller-impulse* function, or by a fixed value (in ms) entered by the user (3d input) [5].

When a collection of files of the same music is to be analyzed, the smaller impulse is to be searched in the *whole collection*, and not from the current file. If a printed score of the music is available, it appears to be more straightforward to manually enter this impulse value into the 3d input, for a quick glance at the score is usually enough to detect the

[5] Both *File-Duration* and *Smaller-Impulse* data can be obtained separatedly by the respectives patches, described above.

smallest unit, and the conversion of this unit into milliseconds uses to be easy. But this can also be automatically done, connecting the *Smaller Impulse* patch to the ouput of a single MIDI or *chordseq* file of the *complete music* (or complete file collection under analysis) [6].

Input

Onsets: the "onsets" list of each file;
Duration: : the "durations" list (in ms) of each file;
Smaller-impulse: a fixed value (in ms) corresponding to the smaller impulse of the whole collection of input files; the default value (0) bypass this function, for the algorithms to calculate the smaller impulse of each input file.

Output

A list of the relative events' density of each file.

Time-Linearity

Measures how equidistant are distributed the events along the time. This evaluation is independent of the duration of individual events, but includes the duration of the whole file. This tool works in the diachronic domain in the same way *SPATIAL LINEARITY* works in the achronic domain (see above). The time gap between each adjacent onset, and between the last onset and the last offset of the file, is compared to a paradigmatic distribution which corresponds to the value of milliseconds it would be necessary for the onsets and the last offset to be exactly equidistant. The higher the weight, the less linear the event distribution. The *step* input allows the user to work with bigger units than one ms (the default value).

Inputs

Onsets: the "onsets" of each MIDI file;
Step: the time-unit to work with. Default (1) means one ms.
File-duration first ouput: the "file-duration" function's first output.

Outputs

[1] Gives the time-linearity weight of each file. Caution: (0.0) means *maximum* linearity, (1.0) means *minimum* linearity.
[2] Gives the paradigmatic interval of equidistance (in ms), for each file.

Pitch-Direction

Gives the file's *directional profile*, i.e. the global direction (descending and/or ascending) of the file's sequence of events. This outlines a kind of *kynetic analysis*, by observing the deviation rate between the higher and lower pitches of the first and last events, respectively. Both rates are divided by the actual range of the object. The more unidirectional the global pitch profile (as a scale-object would be), the higher the weight. The output can be relative, resulting in a negative value if the global direction is descending. But in most of cases, for analytic purposes, it is better to choose the absolute output option, as it is a better indicator of the rate, that is the *strength*, of the pitch-directionality [see samples Fig. 3 below].

---

6 In both cases (a scanning "by eye" or a computer-assisted one), one must be aware of eventual tempo changes during the music, which can turn, say, a eight-note an absolute smaller unit than a sixteenth-note. However, if the complete music's file have been encoded taking in account that global tempo variations, the *Smaller Impulse* tool will return the correct data.

Inputs

Midicents: the “midicents” list of each MIDI file;
Onsets: the “onsets” list of each MIDI file.

Outputs

[1] The relative pitch-direction rate (negative value if the profile is descending);
[2] The absolute pitch-direction rate (always positive).

Pitch-Deviation

Completes the *PITCH-DIRECTION* analysis, by giving the relative dispersion rate of the input MIDI files’ pitches sequence of onsets. The dispersion rate is produced by calculating the *relative deviation*. The reader must read first (below, “Stats&Utilities/Deviations”) the detailed description of standard deviation function and *SOAL* costumized algorithms for *relative deviation*.

Deviation algorithms work only with sequences of single elements. As input pitch lists may contain more than one note by onset (chords etc), the user has to decide what pitch in each *onset* he want to be considered to build a single-pitch sequence. This is done through the last input.

Input

Midicents: the “midicents” list of each MIDI file;
Onsets: the “onsets” list of each MIDI file;
What-Reference? if (1) (default), uses the reference entered in the 4th input; if (0), uses the reference entered in the 5th input. This “reference” is used by the algorithm to calculate the deviation, the same way *relative-deviation* do (see below, “Stats & Utils” section).
Ref-for-deviation-calc.1: reference data from the input midicents list:
0: average midicent value of each onset (this is the “standard” deviation calculation);
1: the first midicent value of each onset;
2: the smallest midicent value (i.e. lowest note) of each onset;
3: the greatest midicent value (i.e. highest note) of each onset;
4: the most frequent midicent value (i.e. *mode*) of each onset; returns average if no mode value.
Ref-for-deviation-calc2: user-reference value (in midicent): the deviation is thus calculated for all the lists from this reference pitch.; the (0) (default) value passby the function.
Min., and
Max: The minimal (5th input) and maximal (6th input) values must inform to the algorithm the boundaries of the music or instrument. The default values — (2100) and (10800), respectively — are preset for piano music.
Choice-for-input-pitches: The user’s choice for input lists pitches: “lowest”, “highest”, “average”, or “mode” (returning average pitch if no mode).

Output

[1] the relative pitch-direction rate (negative value if the profile is descending);

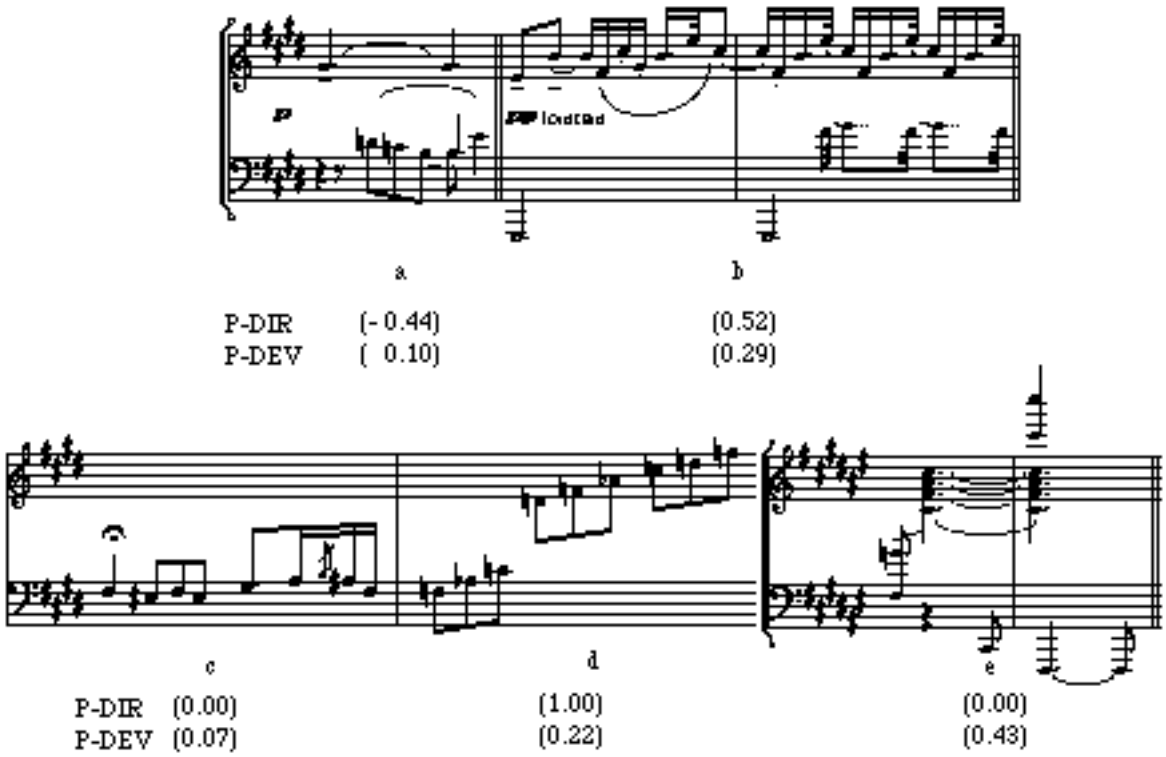

Fig. 3: *PITCH-DIRECTION* and *PITCH-DEVIATION* analyses of sample files extracted from Debussy's piano music.
PITCH-DIRECTION:
(a) and (b): oblique directionalities, downwards (i.e. negative) and upwards;
(c) and (e): the local directionalities are annulated by symmetric contrary motions; these are consecutive in (c), with a return to the starting point, and simultaneous in (e);
(d): absolute unidirectionality. Note that the sequence c-d represents the maximum structural contrast in this dimension.
PITCH-DEVIATION (with *mode-list* as *note-by-onset* output and *mode* as reference deviation value):
(a) small deviation around the average (no mode) pitch D#4; (b): medium deviation around the mode pitch B4;
(c) very small deviation around F#3; (d) and (e): deviations around the average (i.e. central) notes F4 and B3, respectively.

Amplitude Deviation

Applies to MIDI *velocities* the *RELATIVE-DEVIATIONS* algorithm [see below, *Stats&Utilities/Deviations*], with the addition of an evaluation of the direction and slope rate of the attack and decay curves. This is obtained by measuring the velocity differences between the two first (attack) and two last (decay) events. The stronger the attack portion of the file (e.g. ***sfp***), the higher the *positive* value of the attack algorithm. The softer this attack, the higher the *negative* value. And reversely for the decay. Then the tenth of the square-root of the absolute sum of these two data is summed to the relative dispersion rate in order to produce the relative amplitude-deviation value. See some examples below.

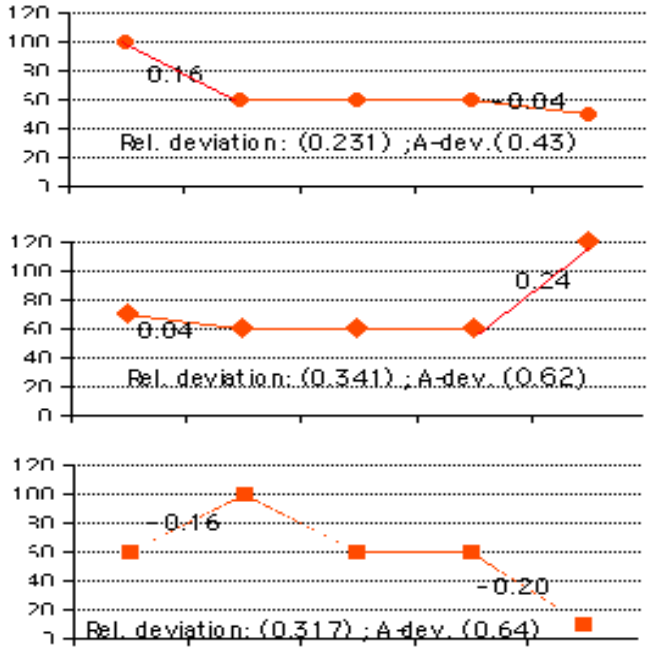

Fig. 4 : Three examples of velocity sequences extracted from MIDI files, and the *AMPLITUDE-DEVIATION* (A-dev) evaluation. The sequence is represented as a graph, where the Y-axis gives the velocity values (0-127) and the X-axis, a sequence of 5 MIDI velocity events. Observe how the *attack* and *decay* algorithms work. The shown results are: "rel. deviation": the standard evaluation returned by the *SOAL Relative-Deviations* algorithm; and "A-dev.": the *AMPLITUDE-DEVIATION* result.

Input

Velocities: the "velocities" list of each MIDI file.

Outputs

[1] A list of formated lists : (relative-deviation (attack decay) amplitude-deviation);
[2] Amplitude-deviation (a single value for each input file).

Piano Specific

This folder contain costumized tools for analyzing piano music.

Sonic Quality Analysis

Concept: MIDI pitches vs piano sound

The specific sonic structure of an object is first determined by the *relative timbral complexity* of each of its pitches. When writing sonic objects for the piano, the composer has to deal — formally or not — with these timbral relations. Thus, the quantification of the relative timbral complexity of each written piano note would be the entry point for a sonic analysis of any piano music, because it gives a basic "sonic weight" to each object. This is the purpose of this patch, which was firstly developped to analyse Debussy's music [Guigue *opp. cit.*]. The sonic basic quality *Q* depends fundamentally on two parameters : the absolute pitches (i.e. MIDI notes) the object contains, and their relative intensity — the so-called musical "dynamics" (i.e. MIDI velocities).

It is possible to synthesize the relative timbral complexity of a piano object into the general principle of *a weighted decline in the timbral complexity of each note in proportion to the pitch of the fundamental frequency* (that is, the written pitch), in such a way that the higher the pitch, the less complex its sonic quality. The three main acoustic factors of this decline are (1) the decreasing number of audible partials for a given fundamental, (2) the

decreasing rank of the loudest partial(s), and (3) the shortening of the decay rate of sound [see sources for these assertions in Guigue 1996, 1997 & 2009]. Each of these factors was modeled by a function.The correlations between the observed declines and our model are shown [fig. 5].

Each written note is given a relative weight *k*, according to the values returned by the three functions.This weight is intended for a *mf* level intensity. The graph [fig. 6] represents the evolution of the value of *k* throughout the entire range of the piano. The two inflexion points of the curve are placed near the pitches B1 and C4 (C4 = Middle C = MIDI 60). The first corresponds to the change in the string's material and the number of strings per note. The model reproduces fairly well the slowing down in the loss of timbral quality that is experienced in this area of the piano. This is due to the effect of *detuning*, which is characteristic of the bi-cording that begins at that point. The steepening in the slope in the second half of this segment (starting from D3) corroborates the observations that have shown the values to diminish by nearly one half, in this register. Starting from the middle of the keyboard this decline stabilizes, becoming nearly linear.

The original timbral complexity of the written note is modulated, up to a certain point, by the attack speed of the piano's hammer as it strikes the string. According to the same quoted sources as above, this modulation can be grossly defined as linear : the fastest the attack, the strongest the modulation. This *velocity modulation*, called *v*, is modeled by a complementary equation. The combination of the two values produces the factor *q* and the sum of the *q* factors for all the pitches of a given object provides a weight for its relative intrinsic sonic quality.

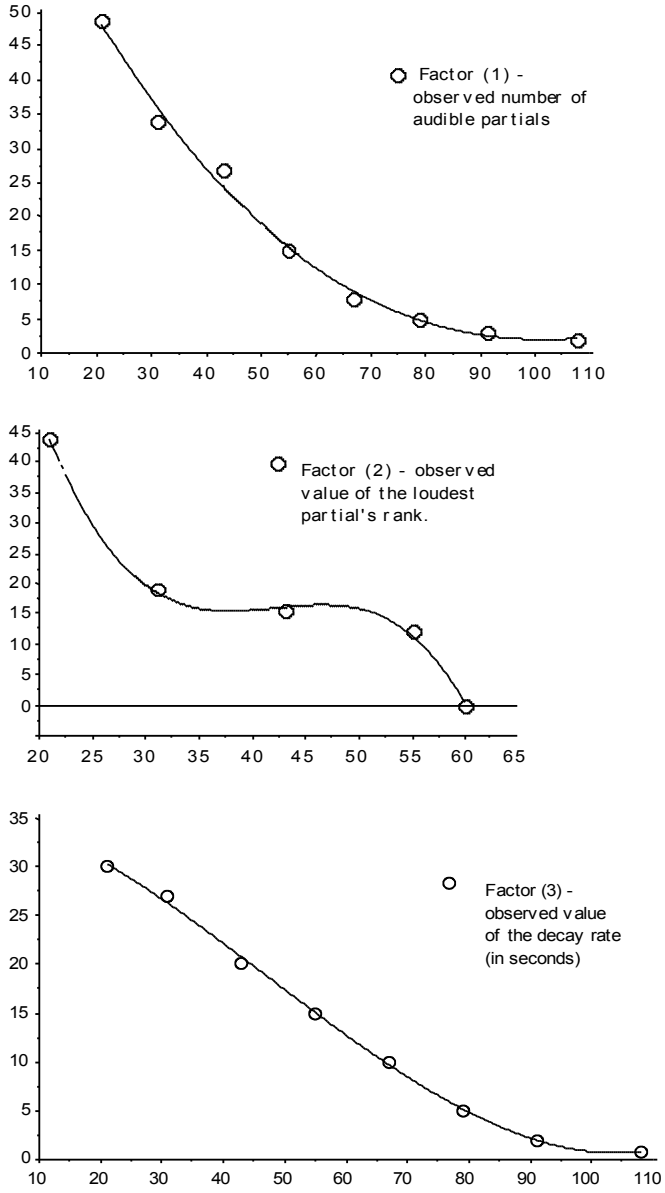

Fig. 5: Correlation between the observed values of three factors of the piano's sonic complexity decline and our model. The observed values correspond to the acoustical experiments reported by the sources refered in [Guigue 1996, 1997a]. The curves are produced by the equations described in the main text. Thus the figure shows that the mathematical model reproduces quite accurately the experiments. *x-axis* : Midi notes; *y-axis*: factor's values.

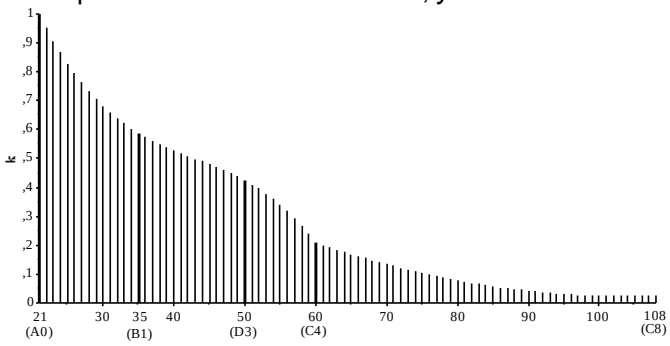

Fig. 6: A representation of the relation between absolute pitch (x-axis, in Midi note numbers) and the amount of relative timbral complexity (y-axis, *k* factor) of the piano.

Finally, the global sonic quality of a piano object can still be modulated by the action

of the pedals (damper or *una corda* pedal). For *SOAL* to be able to weight this parameter, the input MIDI files must have a “Pedal” MIDI information formatted according to the following table,and routed to the MIDI Continuous Foot Controller # 4 [7]. *SOAL MultiMidiReader* routes these data to its 5th output.

PEDAL	MIDI VALUE
“una corda”	32
“a due corde”	48
“ordinario” (no Pedal)	64
1/4 Ped	80
1/3 Ped	86
1/2 Ped and Ped + Uc (simult.)	96
3/4 Ped	112
Ped	127

Intermediate values, or continuous increment or decrement may be used. The “sostenuto” pedal is the same as “Ped” value, but only for the selected notes.

By simply factorizing these values by 64, the *SOAL* algorithm transforms them in a multiplier of the initial *Q* values, in such a way a *una corda* object will have its global sonic quality multiplied by (0.5) — that means, divided by 2 —, while a full *Ped* object, by (2.00). This gives the final *Q* value for each single note of the MIDI file.

Finally, the mean of the sum of Q values is weighted by a function which gives more weight to the highest values.

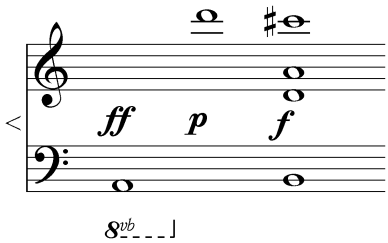

Fig. 7: Three examples of *Q* analyses.
Low A, *ff*; no pedal, Q = (0.74); with damper pedal, Q = (1.00)
Hi D, *p* ; no pedal, Q = (0.02); with U.C. pedal, Q = (0.01)
Chord *f* , no pedal; individual Qs (from lowest pitch upwards) = (0.28 0.11 0.08 0.04);
weighted mean value of Q for the whole chord = (0.20).

Inputs

Midicents: the “midicents” list of each MIDI file;
Velocities: the “velocities” list of each MIDI file;
Ped: the “control #4” output (5th *MidiReader* output). It is a list of integers (from 0 to 127). The MIDI file source should be prepared according to the table of *Pedal* MIDI values (above).
Ped. Output Choice: the user decides wether the algorihm retains the *mean* (1) or the *mode* (0) of the list of Pedal values.
Rounding: an integer, the number of decimals for rounding.

Outputs

[7] The MIDI controler #64, e.g. the standard “Damper Pedal” controller, can usually receive data with the Damper Pedal of most of MIDI digital pianos. But doing this way, these data are encoded only in a “switcher” format, i.e. “Pedal On” (MIDI value 127) or “Pedal Off” (MIDI value 1). As most contemporary piano music uses a far more refinated pedalling, e.g. with progressive opening of the pedal, or “semi-pedal” effects, etc., the data must be entered by hand and routed to a continuous control output, the reason why we have chosen #4 instead.

[1] Returns a list containing the following results in this order: *Q sum*, average *k* value, *q* gaps, velocities mode.
[2] Gives the list of the *q* value of each input note, in their onset order;
[3] Returns ***"Q sum"*** (the most significant information) — the weighted average of *q* values, including pedal multiplicator;
[4] Gives the list of the *k* value of each note;
[5] Returns the average *k* value for all notes (for each file);
[6] Returns the ***q gaps*** — i.e.the average value of intervalic gaps between adjacent *q* values;
[7] Return***s*** the velocities mode — i.e. most frequent value (or, if not applicable, the average value) of the input file(s)'s velocities; scale 0-127.

Piano Spatial Analysis

This patch is a version of *Spatial Analysis* optimized for piano music [see the *Spatial Analysis* description for general concept and main information]. It has no user-defined parameters input because all are preset the following way:

- The *range* is preset to '8700', corresponding to a standard 88-notes piano [8].
- For the *REGISTER* analysis, seven piano registers have been defined according to some global timbral differences caused by physical or mechanical variations, such as the number of strings per note (producing a correlated detuning or « chorus » effect), the strings' material — they can be wound or unwound —, the presence of dampers or not, the average stretching rate of the tuning, etc [9].
- The *REGISTER DISTRIBUTION* quality weight increases from (0.0) for the central register (F3-F5, label 0) — which is thus considered to be the most « neutral » or « poor » register on a sonic point of view — to (0.25) for both highest and lowest regions — respectively D#7-C8, register's label +3, where there is no more dampers and a significative amount of noise, and A0-A#1, label –3, where each note is produced by only one big wound string. The greater the number of registers an file occupies, and/or the more extreme the register, the greater the weight. The table below shows the *PSA* register filling model.

Reg. Label	Bound.	Weight
- 3	A0-A#1	0.250
- 2	**B1**-G2	0.166
- 1	G#2-E3	0.083
0	**F3**-F5	0.000
+ 1	F#5-D6	0.083
+ 2	**D#6**-D7	0.166
+ 3	D#7-C8	0.250

The piano register filling used in *Piano Spatial Analysis*. The 7 piano registers are labelled from –3 to +3 (0 is the central register). The 2nd column shows each register's pitch boundaries. The bold pitches mark the most significant steps of the evolution of the sonority along the keyboard: B1 is the most common switch place from 1 to 2 strings per note, F3 from 2 to 3, and D#6 is the beginning of the no damped region [10].

Input

Midicents: the "midicents" list of each MIDI file, inside the piano range (2100-10700).
Remove-duplicates: if YES (1) (the default, recommended), removes the duplicates values of the midicents list. If (0), they are not removed.

---

[8] If dealing with a non-standard piano, the user must switch back to the generic *Spatial Analysis* patch.

[9] See references section for more details.

[10] These are average boundaries, which can slightly vary according to the manufacturer or model.

Output lists

[1] Gives a list of the following values in this order: relative range, absolute range, register-filling, register distribution, average spatial quality;
[2] Returns the file's average spatial quality — i.e. the average value of [3] (relative pitch range) and [10] (register distribution).
[3] Gives the relative range of the input file, compared to the piano global pitch range; a value (1.0) means the input file occupies the whole piano range;
[4] Gives the absolute range of the input file, i.e. the diference between the highest and the lowest notes.
[5] Gives the number of notes per register - a list of integers, ordered from lowest to highest register; no note in a register returns 0;
[6] Gives the register-filling, a relative value which gives the filling rate of the registers, compared to their total number; a value (1.0) means each register contains at least one note;
[7] Is the list of the numeric label of the felt registers; the label is an integer from -3 to 3 (0 is the central register);
[8] Returns the lowest value of [7].
[9] Returns the highest value of [7]. These two output are useful when evaluating a sequence of files, because they can be used to draw the graphic enveloppe of register-filling;
[10] Gives the register distribution.

Stats & Utils

A collection of generic purpose analytic tools, some of them already internally used by some former described algorithms. Most of them accept any kind of numeric data as input.

Deviations

The three deviation functions in this folder are all based on the *standard deviation* calculation, which corresponds to the square root of the average of the squares of the distances between each value of the input list and a *reference value*. Conventionally used in scientific statistics, it has already shown its effectiveness in spectral analysis. It offers various musically useful choices for measuring and weighting the "dispersion" of a list, that is, the gaps between the values. It applies, typically, to time-related lists. It helps to grasp their relative dynamic complexity through time.

In order to satisfy the wider analytic contexts as possible, *SOAL* deviation tools offer to the user a choice of 6 types of *reference value*: the average value of the input list (which gives the "original" *standard deviation* as used in common statistics); the 1st list's element; the smallest list's element; the greatest list's element; the mode (i.e. the most frequent element of the list — if not existing, the algorithm goes back to the average value); and also a "number" to be entered by the user in an additional "val" input. The *mode* option generally brings very satisfactory results for musical analysis.

Absolute-deviations & Relative-deviations

Absolute-Deviations is the *standard deviation* function, with the 6 different options of reference value, as described above.

Relative-Deviations

In order to return the *relative deviation* of the input list, the absolute deviation, calculated as described above, is divided by a paradigmatic maximum deviation. This value is calculated from a paradigmatic list of 4 elements, which is ordered the following way: 1st element of the input list, a *minimal* value, a *maximal* value (both given by the user), and the last element of the input list. The minimal and maximal values must inform

the algorithm the boundaries of the music or instrument. The default values — (2100) and (10800), respectively — are preset for the MIDI pitch range of piano music. If, for instance, the user wants to analyse the deviation of a sequence of velocities or any kind of MIDI data other than pitch, he must enter (0) and (127) as minimal and maximal values, respectively. According to the *relativity* main *SOAL* rule, *RELATIVE DEVIATIONS* analyses should generally bring more useful informations than *ABSOLUTE DEVIATIONS*.

Inputs (for both tools)

List: list of lists of values;
Use-reference: if 1 (default), uses the reference value in the 3d input; if 0, uses the user-defined reference value in the 4th input;
Reference: the reference from which the deviation will be calculated: 0 = average value of the input list; 1 = first element; 2 = smallest element; 3 = greatest element; 4 = mode (or average if no mode available) (default);
Value: a user-defined value from which the deviation will be calculated; only used if the **use-reference** input is on 0.
The two remaining inputs are only available in relative-deviations:
Min: the minimal value for building the paradigmatic list.
Max: the maximal value for building the paradigmatic list

Output (for both tools)

The absolute or relative deviation of each input list.

Deviations-between-two-lists

Returns the "dispersion rate" between the relative standard deviation of two lists of same length. It functions with any number of pair of lists, collected into two *MultiMidiReaders*: the first list of the first MIDI-reader's output is compared to the first list of the second MIDI-reader's, and so on until the last pair of lists.

Inputs 1 and 2

Any output from MultiMidiReader 1 and 2, respectively. Caution: lists of same length! Same kind of data to compare with (do not attempt to compare apples with oranges)!

Output

A multiple list containing the relative deviation between each pair of lists.

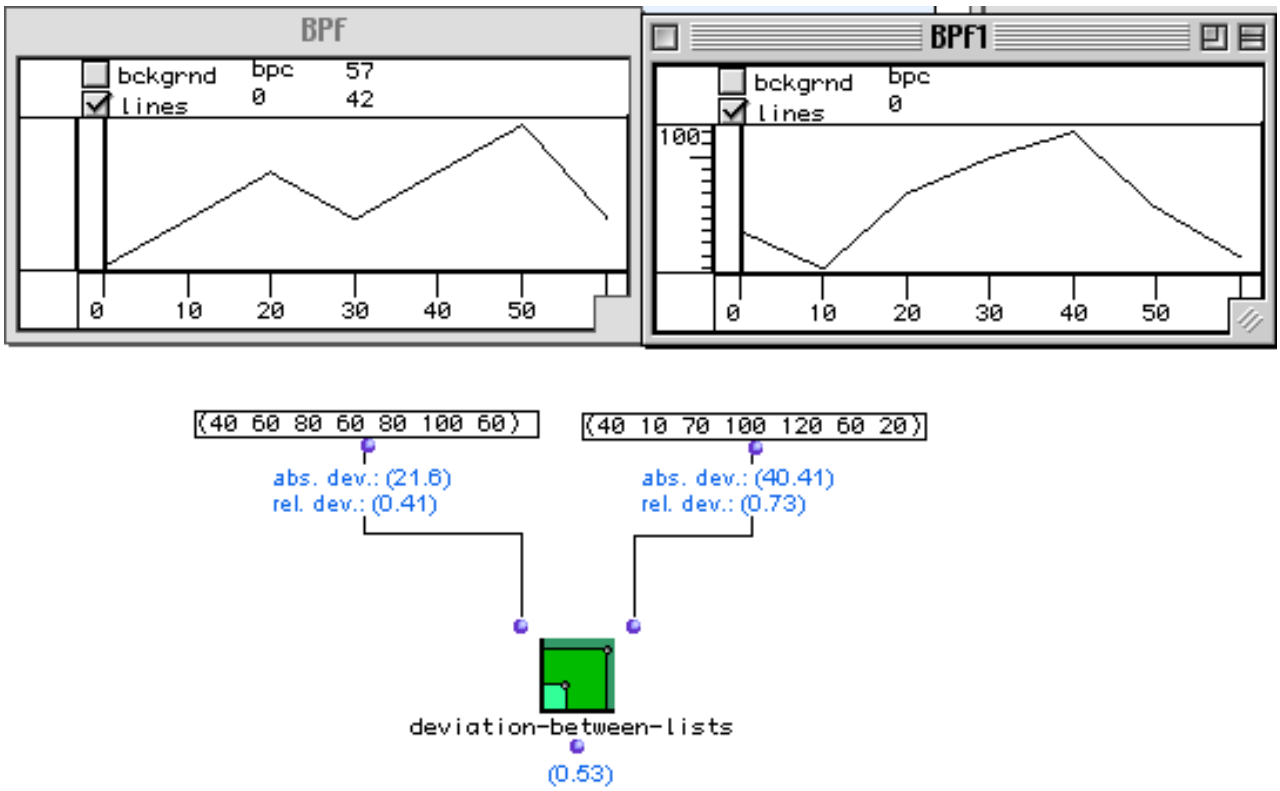

Fig. 8: Two lists (e.g. velocity lists) first analyzed by *ABSOLUTE* and *RELATIVE* deviation tools, both with *mode* as the reference value. The *min* and *max* values for relative deviation calculations are respectively (1) and (127).
Below, the "divergence rate" value of the two lists (0.53).

Lists-Deviation

Returns the deviation rate between a reference list (left input, must be a one-level list, use the *flat* function if necessary) and *n* number of lists (right input, in list of lists format). Lists should better have the same number of arguments.

Input

List1: a single list of values;
Lists: a list of any number of lists.

Output

The deviation rate beteween the reference list (left input) and the other lists (right input).

Soal-Relative-Entropy

Entropy is a way to measure the level of disorder or chaos of a list of values. The higher the entropy, the less predictive, the less «patterned» the order of values. The formula SOAL uses is the following :

$H = - (P1 \log_2 P1 + P2 \log_2 P2 + P3 \log_2 P3 + \ldots + Pn \log_2 Pn)$

$= - \sum (Pi \log_2 Pi)$, for i, a variable from 1 to n.

«Relative-Entropy» matches the input lists' entropy and orders them from the most entropic one, weighted (1.0).

Input :

List of lists

Output :

[1] Relative entropy of each list
[2] Absolute entropy of each list (in « bits »)

Distribution Stats

Range Filling Rate

Returns the percentage of the filling of a "range" (or "ambitus") entered by the user. Useful to analyse the evolution of this filling in a sequence of lists. Default (8700): the range of the piano, in midicents.

Input lists

List: list of values
Range: range of values

Output lists

Range filling rate

Ranges Filling Density

Counts how many data are in each user-defined “ranges” or “bands”. Default ranges: 9 octaves, in midicents.

Input lists

Elements: list of elements to be checked;
Ranges: a list of sub- lists of two elements, which indicates the lowest and higest pitch boundaries of each of the ranges. The ranges can be continuous or not. The default ranges values correspond to 9 registers of 1 octave each, from C to B, in midicents.

Output lists

Range filling rate

Active Ranges

Informs what ranges are filled with some data. The ranges are counted from 0 upwards. This tool completes or may substitute the *Ranges Filling Denstiy* evaluation.

Input lists

Elements: list of elements to be checked;
Ranges: a list of sub- lists of two elements, which indicates the lowest and higest pitch boundaries of each of the ranges. The ranges can be continuous or not. The default ranges values correspond to 9 registers of 1 octave each, from C to B, in midicents.

Output lists

The active ranges, a list with the index (an integer) of the ranges which contain some element. Ranges are indexed from 0 upwards.

Density Rate

Returns a density rate, which factorizes the number of elements of a "list" by the

maximum number of elements this list could contain. This maximum is determined by the boundaries of the list and the "step" the user enters. This step gives the division unit of the list's range. Density rate is a good tool for both space and time related musical events, as it gives a picture of vertical or horizontal textures evolution.

Input lists

List: list of values;
Step: step-unit for density evaluation:
Remove-dups: if a non-zero value (the default, recommended), removes the duplicates values of the list.

Output lists

The density rate

Rate

Returns the percentage of "value (s)" in "max". The user may enter a custom "base" (default: 100) and a number of "decimals" for results.

Inputs

Value: a number or list;
Max: the maximum value for rate calculating;
Base: the base for calculation;
Decimals: decimals for result.

Output

The rate.

Linear Distribution Rate

Evaluates the way the elements of a list are distributed, according to a linearity paradigm. "step" gives the unit base for calculation. Returns two values: the first one gives the linearity rate, where the most regular the distribution (i.e. the more equidistant the elements of the list), the smaller the rate; the second one gives the paradigm, that is, the exact interval it would have to exist between each element to have the most regular distribution.

Input lists

Elements: list of values;
Step: step-unit for evaluation:
Remove-dups: if a non-zero value (the default, recommended), removes the duplicates values of the list.

Output lists

[1] The linear distribution rate. Caution: (0.0) means *maximum* linearity, (1.0) means *minimum* linearity
[2] The *paradigm*.

List Stats

List-stats
Gives the minimum, the average, the maximum and the mode (i.e. most frequent) value of the input list [11].

Relative-list-stats
Divides the 'list' by 'divisor' and returns the minimal, maximal, average and mode value of the 'list' input, relative values between 0 and 1.

List Index
Indexes a list. Useful to quickly go to or check the value at a certain point of big lists.

List Weight
Re-lists a list, returning the number of occurencies of each value, and ranks the list values in descending weight order. For instance, the list (40 60 80 60 80 100 60) is relisted as follows: ((60 3)(80 2)(40 1)(100 1)).

How Many Elements?
Says how many “elements” (percent) a “list” contains.

Keep Duplicates
Shows the duplicated data in a list, removing the others. The output also informs how many times is repeated each data.

Find-elements
Find the elements (in <list of elements to find>), in a <list-of-lists> of numbers or strings. Returns a list with the found element(s) and its (their) position (s).

SOAL substitute
In all list (s) of lists, substitutes a value by one another.

Input
[1] The list(s)
[2] The value to be substituted by [3]
[3] The substituting value

Output
The original list(s) with substituted values.

SOAL-included?
Searches a collection of values in the input list(s)

Input
[1] The list(s)
[2] A list of values to be searched. Note: The function of this algorithm is to detect the presence of the whole list, not each element of the list (for this task, use “find-elements”).

[11] If more than one most frequent value is present, returns the higher one.

Output

A list of “true” (t) or “false” (nil) values.

SOAL-multilists-flat

This function extracts the most important data -- the second of each pair of data -- and flats it into a single list. Applies especially to 'notes-per-onset' and 'velocity-per-onset.

MISC

Midi->notes

Converts midicents into notes, with 6000 = C4 or C3, according to the 2d input.

Midi->pc

Converts midicents into pitch-classes, with C = 0.

SOAL Segmentation

A collection of functions designed to assist the segmentation of MIDI files. There are supposed to be applied to music which can be segmented into smaller units (“time-units”, “sonic units”, “sonic objects”, “*momente*”, etc…) that make musical (structural) sense. The resulting segments are saved as small MIDI files ready to be analyzed by the *SOAL* other analytic functions.

Segmentation-MIDI

Offers three ways to segment the MIDI file(s), to be selected in the 10th input pop-up menu (option[right]-click with the mouse):

(1) through the scanning of the intervals of the list of values of one of the 4 basic parameters (midicents, onsets, durations, velocities) [12]. The user indicates what parameter is to be used in the 6th input pop-up menu. It also indicates, in the 7th input pop-up menu (“operation-option”), what will be the reference interval for the segmentation process: mode, mean, lowest value, highest value, or a fixed value entered by the user in the 11th input. Through the 5th input, the user can also add a percentage upon that reference value (default: 20%). The higher the percentage, the coarser the segmentation.
Any interval greater than the reference value of the chosen parameter’s list of values marks a new segment.

(2) works the same way as the 1st option, but segments instead the files through the scanning of the intervals of one of the “analysis-per-onset” values output, to be connected in the 8th input. This option allows to segment a music according to density oscillations, harmonicity variations or velocities (intensities) gaps, for instance. This input only admits a single-level list per file. As some of “per-onset” functions return embedded lists, the user must insert the ad hoc *SOAL-Multilists-Flat* complementary function (see description below) to reduce the data into a single-level list before connecting to the 8th input.

(3) the music is segmented according to a user-defined fixed duration in ms,

[12] The *midicents*, *durations* and *velocities* options are better used on single-voiced files. As far as multi-voiced music is concerned, one will rather use the *note-per-onset* or *velocity-per-onset* functions, available through the 2d segmentation option.

9th input. This option segments the music into equal time units. In this case, the remaining options are bypassed.

As an example for the 2d option, let say we choose the number of notes as the segmentation criterium. The original music (MIDI file) as 12 onsets, each one containing the following number of notes: (2 1 4 3 4 2 3 5 1 5 4 3). The user chose the “mode” as the reference interval, which is 1. If we do not apply any additional percentage to this mode, the algorithm will segment this list the following way: ((2 1)(4 3 4)(2 3)(5)(1)(5 4 3)). Then it will resume the original music, segment all the corresponding MIDI data according to this pattern, and form 6 MIDI files, each one corresponding to a segment, ready to be saved through the *save-soal-segmentation* function (see below). The patch should look as the fig. 9.

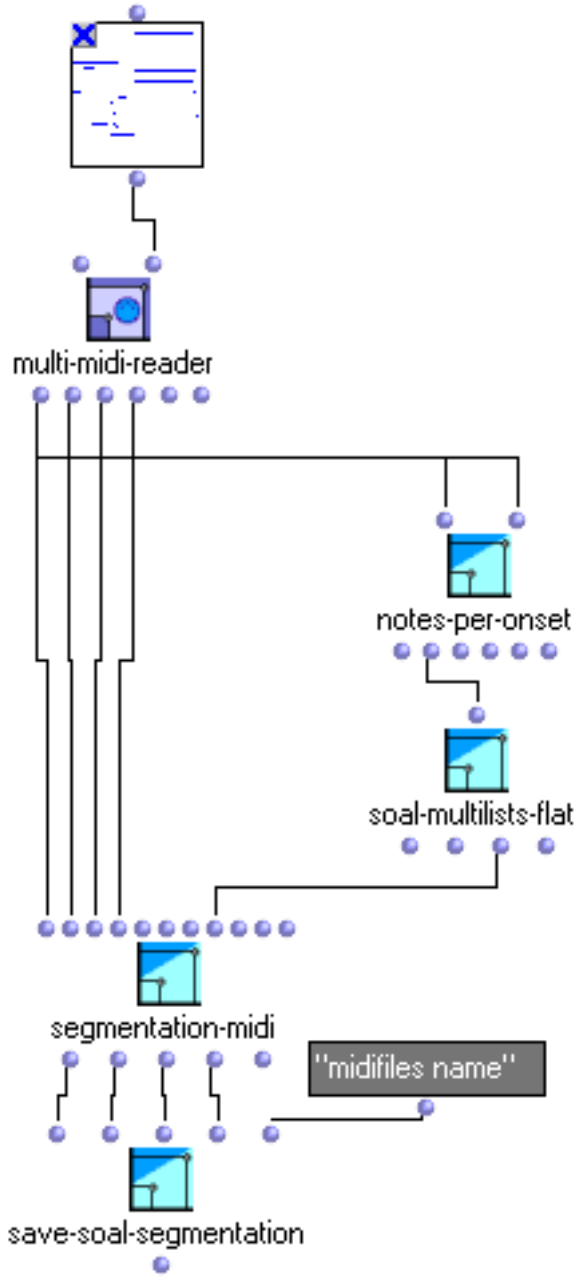

Figura 1: A patch using the SOAL-segmentation functions.

Inputs

Obs: The inputs 1-4 *must always* be fed in.

Midicents: the “midicents” list of each MIDI file (from Multi-midi-reader 1st output);

Onsets: the “onsets” of each MIDI file (from Multi-midi-reader 2nd output);
Durations: the “durations” of each MIDI file (from Multi-midi-reader 3rd output);
Velocities: the “velocities” of each MIDI file (from Multi-midi-reader 4th output);
Percentage: the percent value to be add upon the *mode* of the reference list;
Option (pop-up menu): to choose what basic parameter is to be used for the 1st way of segmentation;
Operation-option (pop-up menu): the operation used to get the reference value (mode, mean, lowest value, highest value, fixed user-defined value;
Reference: the output of a “per-onset” function (2nd way of segmentation): a single list per file; may need the *SOAL-Multilist-flat*;
Time: the fixed duration for the 3rd way of segmentation;
Use-option: pop-up menu to select the segmentation main option (way) the algorithm will work with.
Chosen-value: if “fixed user-defined value” is selected in the 7th input, the reference value is the value entered in this field. Caution: this value must be the same kind of the chosen segmentation parameter (e.g. integer if integer, relative if relative, milliseconds if milliseconds, etc).

Note: once selected the segmentation main option (10th input), the algorithm automatically bypasses the optional input it not need.

Outputs

Obs: The output 1-4 are intended to be connected to the *save-soal-segmentation* respective inputs. The 5th output is for user’s information only.

[1] The original(s) midicents list(s), segmented according to the chosen criterium;
[2] The original(s) onsets list(s), segmented according to the same chosen criterium;
[3] The original(s) durations list(s), segmented according to the same chosen criterium;
[4] The original(s) velocities list(s), segmented according to the same chosen criterium;
[5] Informs the reference value the algorithm used for the segmentation; returns a null value in case of segmentation by “fixed duration”.

SOAL-Multilists-Flat

From embedded lists formatted (((onset)(data)…))), extracts the “data” and flats it. Intended to adapt the ouput of 'notes-per-onset' and 'velocity-per-onset’ functions, before connecting them to the ‘segmentation-midi’ function. When outputs 1 and 2 merge all the MIDI files data, outputs 3 and 4 preserve the ‘multi-list’ standard SOAL format ((…)).

Inputs

Information: the list(s) to be flated; expects ‘notes-per-onset’-type output format (e.g. (((onset) (data)…))).

Outputs

[1] A single list (all MIDI files are merged);
[2] A list of data per onset (all MIDI files are merged);
[3] A single list for each MIDI file;
[4] A list of data per onset, for each MIDI file.

Save-SOAL-Segmentation

Transforms the lists of lists from the Segmentation-midi output into a number of MIDI files, according to the segmentation result. When command-clicking on “save”, a standard system dialogue appears, for the user to choose the directory/folder for saving the files. The 5th input is a text field where the user can write a name for the files. The segments will be numbered after this name (*namenumber.mid*).

Inputs

Midicents: “midicents” ouput of *segmentation-midi*;
Onsets: the “onsets” ouput of *segmentation-midi*;
Durations: the “durations” ouput of *segmentation-midi*;
Velocities: the “velocities” ouput of *segmentation-midi*;
Namefile: the name for the files.

Outputs

[1] command-click brings a standard dialogue window for choosing the directory/folder to save the files.

SOME REFERENCES

GUIGUE, D. *Esthétique de la sonorité. L’héritage debussyste dans la musique pour piano du XXe siècle*. Paris, L’Harmattan, 2009.

GUIGUE, D., & ONOFRE, M. « Sonic complexity and harmonic syntax in *Sequenza* IV for piano », in : Janet K. Halfyard (ed.), *Berio's Sequenzas – Essays on Composition, Performance, Analysis and Aesthetics*, Aldershot, Ashgate, 2007, Chap. 12, p. 209-232.

GUIGUE, D. , *et al*.. « SOAL For Music Analysis : A Study Case With Berio’s *Sequenza IV* », *Actes des JIM05 — Journées d’Informatique Musicale*, Paris : CICM, 2005. <http://jim2005.mshparisnord.net/articles.htm>.

GUIGUE, D. “Sonic object : a Model for Twentieth Century Music Analysis”, *The Journal of New Music Research*, 1997, Vol. 26, pp. 346-375.

_____ . *Une Etude ‘pour les sonorités opposées’ — Principes méthodologiques d’une analyse ‘orientée objets’de la musique du XXe siècle*, Paris, Mediathèque IRCAM, 1996. <http://mediatheque.ircam.fr/articles/textes/Guigue96a>.