The *stft* package

Managing short time Fourier transforms

and

The *mp* package

Managing Matching Pursuit decompositions

Rémi Gribonval
IRISA, Campus de Beaulieu, F-35042 Rennes Cedex, France
email : remi.gribonval@inria.fr
web : http://www.cmap.polytechnique.fr/˜ bacry/LastWave

The *stft* package and *mp* packages were co-written by

- **Rémi Gribonval** *IRISA, Campus de Beaulieu, F-35042 Rennes Cedex, France.*
  e-mail: remi.gribonval@inria.fr
  web: http://www.irisa.fr/metiss/gribonval/
- **Emmanuel Bacry** *CMAP, Ecole Polytechnique, 91128 Palaiseau Cedex France.*
- **Javier Abadia** *formerly at CMAP, Ecole Polytechnique, 91128 Palaiseau Cedex France.*

Contents

Part I

Documentation

Chapter 1

Using the Short Time Fourier Transform (*stft*) package, version 2.0.

1.1 Loading the *stft package*

The *stft* package allows to compute Short Time Fourier Transforms of real signals. It actually allows to compute other time-frequency representations that we will describe in more details in section 1.4 of this chapter. In order to use the *stft* package you must first be sure it is loaded (using the `package` command, see chapter 2). If it is not then you should load it by typing

```
wtrans a> package load 'stft'
```

1.2 A simple example

Let us start by a simple example: we will describe here the demo script in the file 'scripts/stft/DemoStft'.

WARNING : For Macintosh (Classic/OS 9) users. This demo needs a lot of memory, in order to run it you must allocate at least 30Mo to LastWave$^{\mathrm{TM}}$.

In order to compute a Short Time Fourier Transform, we need to create a *stft* structure (no one is created when the package is loaded. We will call it *s1* :

```
wtrans a> s1=[new &stft]
```

This creates a variable of type '`&stft`'. Then we need to create a signal *bonjour* in which we will load the signal to be analyzed :

```
wtrans a> bonjour=[new &signal]
```

The signal will be a speech signal that I have recorded, it is in the file 'scripts/sound/-sounds/bonjour.aiff16':

```
wtrans a> sound read bonjour 'scripts/sound/sounds/bonjour.aiff16'
```

You can display it and hear it

```
wtrans a> disp bonjour
wtrans a> sound play bonjour
```

Then, in order to compute the Short Time Fourier Transform of the signal, we need to specify the *stft* structure where the result will be stored, the signal to be analyzed, the size of the window (in this example the window will be 256 samples) :

WARNING : There has been a major change since version 1.2 : we now specify the actual window size instead of its logarithm in base 2. For example, we use 256 instead of 8.

```
wtrans a> stftd s1 bonjour 256
```

You can display the energy of this Short Time Fourier Transform (that is to say the squared magnitude of the complex coefficients) along with the signal using the `disp` command and specifying that the *stft* structure should be displayed using an inverse grey colormap :

```
wtrans a> disp bonjour s1 -..2 -cm '_grey'
```

1.3 The *stft* structure in LastWave^{TM}

The *stft* package allows to compute various time-frequency representations of real signals that we will describe in more details in section 1.4 of this chapter. These time-frequency representations are stored in variables of type `&stft`: such variables essentially contain a 2D-array that corresponds to a map on a regular time-frequency grid with a time axis and a frequency axis.

1.3.1 Real units *vs* indexes

The coordinates on these two axes can be specified either in “real units” (second, Hertz) or as an “index”. The conversion between real units and indexes is made possible by the '*dx*' and '*x0*' fields of the `&stft` structure.

- *dx* : the *dx* of the signal which has been analyzed,
- *x0* : the *x0* of the signal which has been analyzed,

Time units

Whenever the user needs to specify, as an argument of a command of the *stft* package (or of a derived package such as the *mp* package), a point in time, it will be either a <time> which would mean that “real units” should be used (e.g., 1.3 for 1.3 seconds after the beginning of a signal when *x0* is zero), or a <timeId>. The *signalSize* field of a `&stft` variable limits the range of allowable <timeId> values:

- *signalSize* : the *size* of the signal which has been analyzed, that is to say its number of samples.

A <timeId> is in general a number ranging from 0 (beginning of the signal) to *signalSize*-1. The relation between real time units and time indexes is

$$< time >= x0+ < timeId > \times dx. \tag{1.1}$$

Frequency units

Whenever the user needs to specify, as an argument of a command, a frequency value, it will be either a <freq> which would mean that "real units" should be used (e.g., 440 for 440Hz) or a <freqId>. There is a field called *freqIdNyquist* in a `&stft` variable:

- *freqIdNyquist* : the value of the Nyquist frequency, in 'index' coordinates.

This field allows one to convert between <freq> and <freqId> values:

$$< freq >= \frac{1}{2}\frac{< freqId >}{freqIdNyquist} \times (dx)^{-1}. \tag{1.2}$$

WARNING : in version 1.2 of the *stft* package, there was a field called *fftSize* which corresponded to $2 \times freqIdNyquist$.

Thus, a <freqId> is in general a number ranging from 0 included (corresponding to 0Hz) to *freqIdNyquist* included.

The different fields can be accessed using the usual syntax

```
wtrans a> s1.signalSize
= 10871
wtrans a> s1.freqIdNyquist
= '32768'
```

Note that the *signalSize* and *freqIdNyquist* fields are read-only: they are set when the `stftd` command is invoked to compute the time-frequency representation.

1.3.2 Window-related fields

Two (read-only) fields of `&stft` variables characterize the analysis window (or, equivalently, the enveloppe of the Gabor atoms associated to the time-frequency representation, see Section 1.4:

- *windowSize* : the number of samples of the analysis window;

- *windowShape* : the type of window used (could be either 'blackman', 'hamming', 'hanning', 'gauss', 'spline0' (rectangular), 'spline1' (triangle), 'spline2', 'spline3', 'exponential' or 'FoF'.),

To see what a given window looks like, you can use the `stft window` command:

```
wtrans a> disp [stft window 'hanning' 256]
```

Remark : in the current implementation of LastWave$^{\mathrm{TM}}$, *windowSize* must be a power of 2 that divides *freqIdNyquist*. This limitation should be removed in future versions.

All windows $w[n], 0 \leq n < windowSize$ are of unit energy $\sum_n w[n]^2 = 1$. Symmetric windows (*i.e.* of all but the 'exponential' and 'FoF' shapes) satisfy $w[0] = 0$, their maximum is at $n = windowSize/2$ and the symmetry corresponds to $w[windowSize/2 - n] = w[windowSize/2 + n]$.

WARNING : The 'exponential' and 'FoF' shapes are still alpha-features, there may be bugs left if you use them.

Controlling the window shape and its size. The *windowSize* and the *windowShape* fields are read only. You can control the shape of the window using the arguments in the `stftd` command.

1.3.3 The structure of the time-frequency grid

The Short Time Fourier Transform is subsampled both in time and in frequency. Thus both the <freqId>'s and the <timeId>'s do not take all possible values. The (read-only) field *grid* describes how they are subsampled.

- *grid* : a `&listv` of 4 numbers {*timeRate*, *timeLength*, *freqRate*, *freqLength*}.

The <timeId>'s are subsampled on the grid <n> ×*timeRate* where <n> varies from 0 to *timeLength-1* and the <freqId>'s are subsampled on the grid <m> ×*freqRate* where <m> varies from 0 to *freqLength-1*.

Controlling the time grid. By default the `stftd` command overlaps the window so that each point is exactly covered by 4 windows, thus by default *timeRate*=*windowSize*/4. In the last example, *windowSize*=256 so *timeRate*=64. The *timeLength* is the smallest integer so that there are enough windows to cover all the signal. You can get its value using the standard syntax :

```
wtrans a> s1.grid
= {64 170 32 129}
```

Thus we checked that *timeRate*=64 and we got *timeLength*=170. We can check that $170 * 64 = 10880$ which is larger than the signal size (10871).

You can change the time resolution (i.e., the overlapping of the windows) using the option '-T <T>' in the `stftd` command. An argument <T>= 4 is the default, an integer value of <T> will set

$$timeRate = \frac{windowSize}{< T >}. \tag{1.3}$$

If the resulting value of *timeRate* is not an integer, then the integer part is used instead. If the resulting value of *timeRate* is smaller than 1 then 1 is used instead, hence <T> should be between 1 and *windowSize*.

Thus, in order to increase the time resolution, we could type

```
wtrans a> stftd s1 bonjour 256 -T 16
wtrans a> disp bonjour s1 -..2 -cm '_grey'
wtrans a> s1.grid
= {16 680 256 129}
```

Controlling the frequency grid. In the same way we can control the frequency grid. By default, the `stftd` command sets the *freqLength* to *windowSize/2+1*: this is the number of independent coefficients of a complex FFT performed on the windowed real-valued signal (without zero padding). The <freqId> varies by steps of *freqRate*, from 0 to *freqIdNyquist* (included), thus by default *freqRate*=2×*freqIdNyquist/windowSize.*

In our case, we can check that *freqIdNyquist*=32768 and *windowSize*=256 thus *freqLength*=129 and *freqRate*=256.

You can increase the frequency resolution, *i.e.* the number of bins actually considered in each Fourier transform, by using the option '-F <F>' in the `stftd` command. Technically, zero padding is used before a FFT is performed. An argment <F>= 2 is the default, an integer value of <F> will set

$$freqLength = 1 + <F> \times windowSize/4 \tag{1.4}$$

thus *freqRate* = 4×*freqIdNyquist*/(<F> ×*windowSize*). If the resulting value of *freqRate* is smaller than 1, then 1 is used instead. Else the value of <F> should be such that <F> ×*windowSize* divides 2×*freqIdNyquist.*

For example, in order to increase the frequency resolution, we could type

```
wtrans a> stftd s1 bonjour 256 -T 16 -F 16
wtrans a> disp bonjour s1 -..2 -cm '_grey'
wtrans a> s1.grid
= {16 680 16 1025}
```

WARNING : There has been a major change since version 1.2 : -T <T> and -F <F> used to set *timeRate* = *windowSize*/$2^{<T>}$ and *freqLength* = 1+$2^{<F>}$*windowSize/2.*

1.3.4 Border effects

The time-frequency analysis of a finite length signal is subject to some border effects, depending how we model the signal beyond its borders.

Controlling the treatment of the borders. You can control how an analyzed signal is extended beyond its borders using the '-b' option in the `stftd` command. Let us note that you can have a look at the different border treatments available using the help of the command `extract`.

The (read-only) field *border* of `&stft` variables describes what border treatment was done.

- *border* : the type of border effects (either 'pad0', 'pad'', 'per' or 'mir'), *i.e.* how is the signal extended beyond its boundaries.

When some further treatment is done based on an `&stft` variable, it can be useful to know which values of the time-frequency representation were not affected by border effects. This is indicated by two fields

- *firstp* : the index (<timeId>) of the first time-frequency "column" that is not affected by border effects.
- *lastp* : the index (<timeId>) of the last time-frequency "column" that is not affected by border effects.

Remark : Commands performed on a `&stft` variable generally use the entirety of its content. Some commands –such as the `stftmax` command– can act on or use only the part of the content that is not affected by border effects. This behaviour is usually selected with the '-c' option (for "causality"). It is also possible to use the `disp` command with the '-causal' option in order to display only the representation not affected by border effects.

```
wtrans a> disp bonjour s1 -..2 -cm '_grey' -causal 1
```

1.3.5 Type

The default type of time-frequency representation that can be stored in a `&stft` variable is a standard Short Time Fourier Transform, but there are several other types. There is a corresponding (read-only) field:

- *type* : the type of the stft. For Short Time Fourier Transform, this type is set to 'complex' but later we will see two other types: 'real', 'phase', ... that correspond to other time-frequency representations.

Controlling the type. You can control the type using an optional parameter of the `stftd` command. This will be discussed with examples in Section 1.4.

1.3.6 Data slices and signal names

It is possible to easily access the values of a `&stft` variable. A first way to do that is to use the *sig* field. For instance, if you want to have a look at the real and imaginary part of *s1*, you should type

```
wtrans a> disp s1.sig
```

This would display two signals. Each one is the concatenation of *timeLength* signals of size *freqLength* that you may consider as "vertical slices" of the stft: each vertical slice corresponds to the values of the real (resp. imaginary) part of the Short Time Fourier Transform at a given <timeId> and for all <freqId>. You could also superimpose the display of the real part and the imaginary part using

```
wtrans a> disp {s1.sig}
```

Alternatively, you could get directly either a vertical slice or an horizontal slice of *s1* using the signal extraction syntax. To obtain the vertical slice corresponding to <timeId>=512 simply type

```
wtrans a> disp {512s1}
```

For the horizontal slice corresponding to <freqId>=128 you should type

```
wtrans a> disp {.128s1}
```

When the type of the stft is not *complex*, the slices and the *sig* field return a single signal instead of the `&listv` corresponding to the real and imaginary parts.

Remark : you can combine the above commands with the `stats` command (see the *signal* package) to get statistics on the value of a Short Time Fourier Transform.

WARNING : so far, the content of the time-frequency array of a stft is only accessible in read-only mode, which means that modifying the content of the signals obtained through *s1.sig = ...*, *<freqId>s1= ...* or *<timeId>s1= ...* does not change the corresponding content of the `&stft` variable. This behaviour may change in a future release of LastWave^{TM}.

1.4 Other Time-Frequency representations

We have already mentioned that a `&stft` variable can contain various types of time-frequency representations other than a standard Short Time Fourier Transform. A standard Short Time Fourier Transform corresponds to a subsampling (both in time and frequency) of the following function

$$G(t,f) = \int s(u)w(u-t)e^{-2i\pi fu}du,$$

where $s(t)$ is the analyzed signal and $w(t)$ is the window function. One can rewrite the latter expression as

$$G(t,f) =< s, g_{t,f} >,$$

where $< .,. >$ corresponds to the scalar product of two signals and $g_{t,f}(u)$ is a time and frequency localized function whose expression is

$$g_{t,f}(u) = w(u-t)e^{+2i\pi fu}.$$

Thus, computing the Short Time Fourier Transform consists in selecting a subsampled family of all the atoms $\{g_{t,f}\}_{t,f}$ and computing all the scalar products of the original signal with each *atomic* function of the family. These *atomic* functions (referred to as *atoms* in the following) behave like *test* functions we want to test the signal with.

LastWave^{TM} allows to work with the 'real-valued' version of this complex valued representation. Without going into the details it basically consists in finding for each time-frequency location (t,f) the best phasis so that the scalar product with $< s, w(t-u)\cos(2\pi(fu+\phi)) >$ is maximum for that $\phi \in [0,1)$. Then for each (t,f), instead of storing the real and imaginary part of the scalar product $< s, g_{t,f} >$ (which is what is done in the case of a regular Short Time Fourier Transform), a `&stft` variable stores either the best phase ϕ (if its type is 'phase') or the associated optimized real-valued scalar product $< s, w(t-u)\cos(2\pi(fu+\phi)) >$ (if its type is 'real').

This type of representation is mainly used in the Matching Pursuit package (*mp*). However you can compute and display such a representation by setting the type of the `&stft` variable (in the `stftd` command) to 'phase' or 'real' instead of the default value ('complex'). If we use the same example as in the previous sections, we would type :

```
wtrans a> stftd s1 bonjour 'hamming' 256 'real' -F 16 -T 16
wtrans a> disp bonjour s1 -..2 -cm '_grey'
```

It would display the signal and the square of the real valued scalar product (in deciBel by default).

Remark : When the type of the `&stft` is 'phase', the display does not depend on the value of the 'db' flag.

1.5 Display and mouse interaction

When you display a *stft* structure, you can, while moving the mouse on the image, see (at the bottom of the window) the coordinate of the time frequency point and the energy (or the phase) of the *stft* at this point. You can zoom the image using the left button and go back to the original image using the middle button. If you type the 'z' key you switch to a different zooming mode (just try it!). If you hit 'z' again you go back to the original mode. In the same way, hitting the 'c' key allows you to get a cursor on the screen.

Moreover, if you hit the middle button while the 'control' key is down, you display a vertical cut (in dB) of the Short Time Fourier Transform. With the 'shift' key down instead of the 'control' key, you obtain an horizontal cut. If you drag the mouse (while holding the button down) it changes dynamically !

A *stft* structure is displayed using *GraphStft* graphic class. This class inherits from the basic class *GObject* and has 5 new fields :

- *cm* : that allows to set the colormap used to display the stft.
- *causal* : which is a flag which indicates whether we display or not the part affected by border effects (default value=0)
- *db* : which is a flag which indicates whether the energy is coded in deciBel or not (default value=1)
- *expo* : the dynamic range of the display in deciBels (default value=$70dB$). When *db* is non zero, the colormap corresponds to the energy in deciBels between $-$<expo> dB and $0dB$ relatively to the maximum energy in the window. When *db* is zero, the *expo* is not used.
- *graph* : which allows to get/set the $stft$ structure that will be displayed,

1.6 Miscelaneous

Let us list a few additional features of the *stft* package.

1.6.1 Arithmetic operations on `&stft` variables

Arithmetic operations on `&stft` variables are possible through the `stft` command.

- `stft + <stft1> <stft2> [<stftOut>=<stft1>]`
- `stft - <stft1> <stft2> [<stftOut>=<stft1>]`
- `stft * <stft1> <stft2> [<stftOut>=<stft1>]`
- `stft / <stft1> <stft2> [<stftOut>=<stft1>]`
- `stft ln <stft1> [<stftOut>=<stft1>]`
- `stft log <stft1> [<stftOut>=<stft1>]`
- `stft log2 <stft1> [<stftOut>=<stft1>]`
- `stft conjugate <stft1> [<stftOut>=<stft1>]`

The stfts must be of the same type and have the same time-frequency structure.

WARNING : this is still an alpha version feature, we now there are some bugs left. For instance, the stft resulting from the above operations may not display correctly with the `disp` function.

1.6.2 Wish list

- a `stftr` command to perform overlap-add reconstruction using a `&stft` variable.
- write access to the content of `&stft` variables using the fields extraction syntax
- optimization of the analysis algorithm (direct convolution/FFT) depending on the oversampling and the size of the window.
- allow any window size (not only powers of two) and more arbitrary grids.

Chapter 2

The Matching Pursuit (*mp*) package.

The *mp* package allows to compute various types of Matching Pursuit decompositions of real-valued signals. The *mp* package makes an intensive use of the *stft* package, so in order to understand this chapter you should first read the chapter about the *stft* package.

WARNING : In this chapter we will assume that you know about the Matching Pursuit method. If you do not know it, you should have a look at the article *Matching pursuit with time-frequency dictionaries*, S.Mallat and Z.Zhang, IEEE Trans. on Sig. Proc. 41 (12), 3397 (1993) or the corresponding chapter in the book *A wavelet tour of signal processing*, S.Mallat, Academic Press. (1998).

2.1 Loading the *mp package*

In order to use the *mp* package you must first be sure it is loaded. If it is not then you should type

```
wtransa> package load 'mp'
```

2.2 A simple example

As for `&wtrans` variables, when loading the *mp* package, it creates (look in the script file 'scripts/mp/mp.pkg') two `&book` variables named *m* and *n*. We will explain in Section 2.3 the structure of `&book` variables, but for now we will just look at an example to see how it works. You can set the current object *objCur* to be *m* by simply typing the command `m` :

```
wtrans a> m
book m>
```

The current object is now the variable *m* which is of type `&book`. Again, as for `&wtrans` structures, the numbers 0 to 9 (or 0<bookName> to 9<bookName> if you want to refer to a book which is not the current object) refer to signals for your personal use. They are automatically allocated by LastWave^{TM}(in every book) and never used.

The example we are going to present here is very much inspired from the demo which is in the file 'scripts/mp/DemoMP' (commands `DemoMPRegAlgo` and `DemoMPFastAlgo`). Let us analyze a signal which corresponds to a sinusoid plus a Dirac (in the middle) and some white Gaussian noise. It will be of size 1024 :

```
m> 0 = sin(2*pi*40*I(1024)/1024)+1.2*(I==512)+0.1*Grand
```

Actually, we are going to smooth it by a gaussian window in order to avoid discontinuities at the borders :

```
book m> 0 = 0m*exp(-.00002*(I-512)*(I-512))
```

You can display the signal

```
book m> disp 0
```

Then we want the matching pursuit to find the first 30 Gabor atoms that approximate the signal :

```
book m> mpd 30
0.4599908
={<&dict;0xa1b9428> <size=31>}
```

As indicated by the value 0.4599908 displayed after the `mpd` command has been performed, the duration of the computation on my system (a Pentium III 750 Mhz with 256 Mb of RAM) is about 0.45 second. We can build the so-obtained approximation using the 30 Gabor atoms and store it in the signal 1

```
book m> mpr 1
= '1 1023'
```

Then we can display both the original and the approximation along with the error of approximation

```
book m> disp 0 1 0m-1m
```

You can reconstruct using only some specific atoms. This can be done using a “masking signal” for arbitrary selection of atoms, or using options of the `mpr` command.

WARNING : in the previous release of the *mp* package, a '-r' flag was needed before the first option of the `mpr` command. This is no longer needed.

For instance if you want to reconstruct using only the atoms with a large scale, you must use the *-s* syntax :

```
book m> mpr 1 -s 2^7 2^15
book m> disp 0 1
```

This allows you to extract only the sinusoidal behavior and get rid of both the noise and the Dirac. If you want to extract the Dirac you can type

```
book m> mpr 1 -s 2^1 2^2
book m> disp 0 1
```

WARNING : in the previous release of the *mp* package, the '-o' option was used to specify an 'octave' range for the atoms used in the reconstruction. It has been replaced by the '-s' option which specifies a 'windowSize' range.

If you want to denoise the whole signal (sinusoidal component + Dirac) you can reconstruct using the first atoms only, i.e., the (10) most energetic atoms

```
book m> mpr 1 -n 1 10
book m> disp 0 1
```

Last, but not the least, you can display the time frequency representation of the book *m* using the regular `disp` command

```
book m> disp 0 m -..2 -db 1 -expo 34 -cm '_grey'
```

2.3 The `&book` structure in LastWave^{TM}

The *mp* package allows to perform various types of Matching Pursuit decomposition of real-valued signals. The decomposition is performed using the `mpd` command and its variants (`fastmpd`, `hmpd`, `fasthmpd`), and the reconstruction is performed with the `mpr` command. We will describe in Sections 2.6, 2.7 and 2.8 of this chapter the various types of Matching Pursuit decompositions that are available and how to invoke the variants of the `mpd` command to perform these various decompositions.

The result of a pursuit is stored in a variable of type `&book`, the fields of which are accessed through the standard syntax. A few fields are identical to those of `&stft` variables, they specify the correspondance between "real units" (seconds,Hertz,. . .) and "index" units.

- *signalSize*, *freqIdNyquist*, *dx*, *x0* : these fields have the same meaning as those defined in the structure `&stft` (see Chapter 1).

The result of a matching pursuit with a multiscale Gabor dictionnary is a list of Gabor atoms. In order to store these atoms (using `&atom` variables), we actually use another variable type called a `&mol` Thus you have to see a `&book` variable as containing an array of `&mol` variables each of them containing one or more `&atom` variables.

To access the <i>th molecule in a book (in a read-only mode) you should use the <bookName>[<i>] syntax, where <bookName> is the `&book` variable where the molecule you want to access is, and <i> is the rank number of this molecule in that book (0 is the first molecule that has been selected by the pursuit, 1 is the second one and so on until <bookName>.size-1 for the last one)

```
bookm> print m[1]
m[1] =
<&mol[1][1];0xa20dc68>
```

WARNING : in the previous version of LastWave^{TM}, molecules were numbered between 1 for the first one and <bookName>.size for the last one. Check your scripts!

You can add a `&mol` to a book using the *book+=molecule* syntax. To know the (read-only) number of `&mol` variables stored in the `&book` you should use the field *size*.

2.4 The `&mol` structure in LastWave^{TM}

Each `&mol` structure actually can store *several* atoms (this feature is used essentially for Harmonic Matching Pursuit). You can access (read-only) the <k>th atom of a molecule with the <moleculeName>[<k>] syntax. The main fields of `&mol` variables are

- *dim*: (read-only) number of `&atom` variables contained in the molecule.
- *coeff2*: (read-only) sum of the *coeff2* fields of the `&atom` variables in the molecule.

As a very good example of how to use molecules you should read the 'scripts/mp/mp.pkg' file and particularly the arrow key behavior definitions.

2.5 The `&atom` structure

Gabor time-frequency atoms are represented in LastWave^{TM} by `&atom` variables. The fields of `&atom` variables are accessed using the standard syntax. Besides the *signalSize*, *freqId-Nyquist*, *dx*, *x0*, *windowShape* fields (*cf.* the corresponding fields of `&stft` variables), the main fields are

- *time/timeId* : the time where the atom is localized at (in real or "index" units). In "index" coordinates, it can range from 0 to *signalSize-1*.
- *freq/freqId* : the frequency where the atom is localized at (in real or "index" units). In "index" coordinates, it can range from 0 (for unmodulated atoms) to *freqIdNyquist* (for atoms at the Nyquist frequency).
- *chirp/chirpId* : the chirprate of the atom (in real or "index" coordinates)

2.5.1 Chirp units

Starting in LastWave^{TM} version 2.0, as an extension to Gabor atoms, `&atom` variables can be used to represent *chirplets* through the field *chirp(Id)*. Whenever the user needs to specify, as an argument of a command of the *mp* package, the frequency slope of a chirp atom, it will be either a <chirp> which would mean that "real units" should be used (e.g., 0.01 for 0.01 Hz/second), or a <chirpId>. The relation between real chirp units and chirp indexes is

$$< chirp >= \frac{< chirpId >}{2freqIdNyquist} \times (dx)^{-2}. \tag{2.1}$$

The *freqIdNyquist* field of a `&atom` variable limits the range of allowable <chirpId> values: a <chirpId> is in general a number with absolute value less than or equal to *freqIdNyquist*/2, because a larger value of the chirprate would mean that the atom is necessarily aliased.

2.5.2 Time-frequency concentration

One of the interesting aspects of decompositions in the Gabor multiscale dictionary is the possibility to use atoms at different scales and shapes. The corresponding fields are

- *windowSize* : the size of the time support of the atom (number of samples).

WARNING : In the previous version of LastWave^{TM}, there was an *octave* field which was the $\log_2$ of the *windowSize*. In this version, *windowSize* is limited to being a power of 2, this should change in a future version.

- *support/supportId* : support {*timeMin,timeMax*} of the atom in real or index coordinates.
- *dt,df* : (read-only) time/frequency spread of the atom, in real coordinates.

2.5.3 Coefficient information

As briefly explained in Section 1.4, one can consider either complex-valued or real-valued normalized Gabor atoms. An `&atom` variable stores the inner product (complex or real) of the residual with the atoms $g_{t,f}$ (complex valued scalar product) or $w(t-u)\cos(2\pi(fu+\phi))$ (real valued scalar product), as well as the phase of real-valued atoms.

- *coeff* : the real part and imaginary parts of the complex-scalar product between the analyzed signal and a 'complex' atom with the `&atom`'s time-frequency-shape-chirp coordinates.
- *coeff2/phase* : the square of the modulus and the phase of the 'real' atom that best matches the analyzed signal with the `&atom`'s time-frequency-shape-chirp coordinates.
- *gg* : the inner-product between the 'complex' atom and its conjugate. It is used to convert between *coeff* and *coeff2/phase*.

2.5.4 Controlling the content of an `&atom`

To have some good example of manipulation of atoms you should have a look at the 'scripts/mp/mp.pkg' file and in particular the mouse interaction with atoms to move them around.

2.5.5 Building an `&atom` into a signal

There are two "methods" of `&atom` variables that make it possible to build a signal with the corresponding waveform. If you type

```
book m> disp m[0][0].buildr
```

it will display the waveform of the 'real' atom, taking into account its *coeff2* and *phase*. By typing

```
book m> disp m[0][0].buildc
```

you will display two signals corresponding to the real and the imaginary part of the waveform of the normalized 'complex' atom.

2.6 The `mpd` command

There is only *one* argument that the `mpd` command requires, and it is the number <nIter> of iterations of the pursuit. If you invoke `mpd` with no additional argument, *i.e.*,

```
book m> mpd 20
```

what happens is:

- it will assume you want to put the result in *objCur* (which should be a `&book` variable);
- *objCur* will be cleared and a new pursuit will start using the signal 0 of this book. The dictionary for the pursuit will be with 'gauss' (Gaussian) windows and $2^2 \leq windowSize \leq 2^n$ where $2^n \leq signalSize < 2^{n+1}$.

If you want to, you can specify explicitly to the `mpd` command which book to put the result in, and/or which signal to analyze.

```
book m> 1m = Grand(1183)
book m> mpd n 20 1m
```

2.6.1 How to change the dictionary ?

By default, `mpd` performs a decomposition using a multiscale Gabor dictionary with real-valued Gabor atoms based on a 'gauss' window and $2^2 \leq windowSize = 2^j \leq 2^n$ where $2^n \leq signalSize < 2^{n+1}$. The nature of the dictionary can be changed by using some additional arguments of the `mpd` command. If you want to use both 'gauss' and 'FoF' windows ('gauss' windows are symmetric in time, 'FoF' ones are not) and window size 4,16 and 1024, you can type

```
book m> mpd 30 '-s' {4 16 1024} '-w' {'gauss' 'FoF'}
0.36
={<&dict;0xa1b9428> <size=31>}
```

Please use the `stft window` command to see the available window shapes.

2.6.2 How to perform a Matching Pursuit with Chirplets ?

For understanding how a Matching Pursuit with Chirplets works, please refer to the paper *Fast matching pursuit with a multiscale dictionary of Gaussian chirps*, R. Gribonval, IEEE Trans. on Signal Proc., Vol. 49, No. 5, mai 2001, pp 994-1001.

You simply have to invoke the `mpd` command with the '-O' {'chirp'} option, but only for Gaussian windows (which is the default).

2.6.3 How to resume a pursuit

A new feature in this version of the *mp* package is that you can now resume a Matching Pursuit decomposition, using the return value of the `mpd` command.

```
book m> {dict decay}=[mpd 30]
0.22000122
={<&dict;0xx87b890> <size=31>}
```

Remark : Notice that the Matching Pursuit decomposition is faster (only 0.22 second) the second time you call it, because some data has been tabulated at the first call.

The return value consist in a `&listv` of two variables : a `&dict` variable (cf the Reference part) and a `&signal` containing the decay of the approximation error. To resume a pursuit, simply use the `mpd` command again but with the alternate syntax

```
book m> {dict decay}=[mpd 30 dict decay]
0.150001
={<&dict;0xx87b890> <size=61>}
```

You can have a look at how the relative error (the ratio of the energy of the residual by that of the original analyzed signal) decays, in deciBels, as a function of the number of iterations of the pursuit

```
book m> disp 10*log(decay)
```

2.7 A faster algorithm for Matching Pursuit

The algorithm presented above to perform the pursuit can be pretty slow when performed on very long signals. You can use a different algorithm which is much faster but that is less optimized in terms of capturing energy. It consists in performing the pursuits only on the atoms which correspond to the most energetic local maxima (both in time and frequency) of the spectrogram (at any scale). When none are left (either because they have all been selected or because after a few iterations their energy is too low), then all the spectrograms are updated (using the residual) and a new set of maximum atoms selected. the algorithm performs the pursuit on this new set and so on.

In order to use this algorithm you need to specify the number of maxima in the set. Let us note that if this number is 1, then this algorithm is exactly equivalent to the previous one. The more maxima you put in the set, the faster the algorithm will be and the less optimum in terms of captured energy. Using the same signal as the one presented in the section above, one could compare the regular and the fast algorithm. The regular decomposition of the signal 0 with 100 Gabor atoms is obtained by typing

```
book m> mpd 100
'1.522'
= {<&dict;0xa23ce28> <size=101>}
```

and takes about 1.5 seconds to compute. The fast algorithm is invoked using

```
book m> fastmpd 100 100
= '0.631'
= {<&dict;0xa32a980> <size=101>}
```

It means that you want to get 100 molecules and that each time a set of 100 maxima should be selected. As you can see it is much faster (only about 0.6 second) than the regular algorithm.

2.7.1 Fast Matching Pursuit with Chirplets

You simply have to invoke the `fastmpd` command with the '-O' {'chirp'} option, but only for Gaussian windows (which is the default).

2.7.2 Resuming a Fast Matching Pursuit

A Fast Matching Pursuit decomposition can be resumed just as a regular one, using the `mpd` command. For that, you would type

```
book m> {dict decay}=[fastmpd 100 100]
0.652
={<&dict;0xx87b890> <size=101>}
book m> {dict decay}=[mpd 30 dict decay]
0.150001
={<&dict;0xx87b890> <size=131>}
```

Next, we will see other types of pursuit. You can resume them in the same way.

2.8 Other types of pursuits

2.8.1 A High Resolution Matching Pursuit

WARNING : High-Resolution MP is no longer supported in LastWave$^{\text{TM}}$2.0. In case of demand and we could help interested people making the necessary ports to re implement it with the new APIs of LastWave$^{\text{TM}}$2.0

2.8.2 A Harmonic Matching Pursuit

For understanding how a Harmonic Matching Pursuit works, please refer to the paper *Harmonic Decomposition of Audio Signals with Matching Pursuit*, R. Gribonval and E. Bacry, IEEE Trans. on Signal Proc., Vol. 51, no. 1, January 2003, pp 101–111.

You simply have to invoke the `hmpd` command instead of the `mpd` one. The `fasthmpd` command corresponding to the fast algorithm with local maxima is also available, but is not recommended. Compared to the regular `mpd` and `fastmpd` commands, the `hmpd` and `fasthmpd` commands take just two additional arguments <freq0Min> and <freq0Max> that indicate the range in which the fundamental frequency of harmonic molecules should be looked for.

2.9 Display and mouse interaction

Zoom and cursor: When you display a `&book` structure, you can, while moving the mouse on the image, see (at the bottom of the window) the coordinate of the time frequency point you are pointing to (the <timeId> and <freqId> are between brackets). You can zoom the image using the left button and go back to the original image using the middle button. If you type the 'z' key you switch to a different zooming mode (just try it!). If you hit 'z' again you go back to the original mode.

In the same way, hitting a first time the 'c' key allows you to get a cross-hair cursor on the screen. If you hit it a second time, you enter a mode where LastWave^{TM}looks for the atom/molecule which is the closest to the point the mouse is pointing to. This atom is circled and its fields are shown at the bottom of the window (the fist number shown is the iteration number the corresponding atom was found at). If you hit again the 'c' key you go back to the no-cursor mode. Actually, there is another way to navigate through the atoms/molecules using the display. It consists in using the arrow keys : the down key circles the first atom which was found by the pursuit and which is currently displayed in the graphic, the up key circles the last atom, the right key circles the next atom and the left key the previous atom.

Playing molecules/atoms: On computers where LastWave^{TM}was set up so that sounds can be played, if the signal you decomposed is a sound signal and if you set the *dx* field of the signal according to the sample frequency, you can play on the computer speaker the sound associated to an atom which has been circled using either the arrow keys of the mouse. For that purpose you just need to type the '=' key. If you type the $' <'$ key then it will play the reconstruction of the sound using all the atoms up to the circled atom and if you type the $' >'$ key it will play the reconstruction of the sound using all the atoms down to the circled atom.

Moving molecules/atoms and changing their parameters: You can change the parameters of a molecule/atom interactivel using the mouse drag and drop. Thus

- left button drag/drop + $shift$ key : allows to move a molecule/atom in the time/frequency plane
- middle button drag/drop + $shift$ key : allows to $transpose$ a molecule/atom, i.e., to change the center frequency (the time position is not changed)
- right button drag/drop + $shift$ key : allows to $translate$ a molecule/atom, i.e., to change the time position (the center frequency is not changed)
- left button drag/drop + *ctrl* key : allows to change the scale of a molecule/atom

The *GraphBook* graphic class: A *book* structure is displayed using *GraphBook* graphic class. This class inherits from the basic class *GObject* and has some new fields :

- *cm* : that allows to set the colormap used to display the book
- *db* : which is a flag which indicates whether the energy is coded in deciBel or not (default value=1)
- *expo* : the dynamic range of the display in deciBels (default value=30dB) information in the manual of the *stft* package.
- *n* : the rank range [<nMin> <nMax>] of the atoms displayed (this allow to select the atoms you want to be displayed).
- *k* : the partial number range [<kMin> <kMax>] of the atoms displayed within a harmonic molecule (this allow to select the atoms you want to be displayed).

- *s* : the scale range [<windowSizeMin> <windowSizeMax>] of the atoms displayed (this allow to select the atoms you want to be displayed).
- *chirp/chirpId*: the chirp range of the atoms displayed (this allow to select the atoms you want to be displayed).
- *fund*: a flag that selects the way harmonic words are displayed. All the atoms within each word (flag='all') or only the most energetic atom with its energy (flag='max') or with the total energy of the word (flag='sum').
- *graph* : allows to set/get the *book* structure that will be displayed,
- ?*closest*: which is a "get-only" field which allows to get the index {<n> <k>} of the displayed atom which is closest to a time-frequency location.

Part II

Reference

Chapter 3

Package `mp` 2.1

Package allowing to perform Matching Pursuit.
*** Authors and Copyright : R.Gribonval, E.Bacry and J. Abadia*

3.1 Defined types

3.1.1 Type `&atom`

This type is the basic type for time-frequency atoms that are used in Short Time Fourier Transform and Matching Pursuit decompositions.

- `&atom.dx [= <dx>]`
  Sets/Gets the abscissa step of the signal which has been analyzed.
- `&atom.x0 [= <x0>]`
  Sets/Gets the first abscissa of the signal which has been analyzed.
- `&atom.signalSize`
  Gets the size of the original signal of the time-frequency transform.
- `&atom.freqIdNyquist`
  Gets the Nyquist frequency in sample coordinates.
- `&atom.windowShape [= <windowShape>]`
  Sets/Gets the windowShape of an atom. The available window shapes are : blackman, hanning, hamming, gauss, spline0 (rectangle), spline1, spline2, spline3, exponential or FoF.
- `&atom.windowSize [= <windowSize>]`
  Sets/Gets the windowSize of an atom, i.e. the number of samples of its window. So far only powers of 2 are allowed.
- `&atom.timeId [= <timeId>]`
  Sets/Gets the time center of an atom in sample coordinates, i.e. an index 0 <= timeId < atom.signalSize.

- `&atom.time [= <time>]`
  Sets/Gets the time center of an atom in real coordinates, i.e. the real time in seconds.

- `&atom.freqId [= <freqId>]`
  Sets/Gets the frequency center of an atom in sample coordinates, i.e. an index 0 <= freqId <= atom.freqIdNyquist.

- `&atom.freq [= <freq>]`
  Sets/Gets the frequency center of an atom in real coordinates, i.e. the real frequency in Hertz.

- `&atom.chirp [= <chirp>]`
  Sets/Gets the chirp rate of an atom in real coordinates, i.e. the real frequency slope in Hertz per second.

- `&atom.chirpId [= <chirpId>]`
  Sets/Gets the chirp rate of an atom in sample coordinates, i.e. an index chirpId with |chirpId| <= atom.freqIdNyquist/2.

- `&atom.coeff2 [= <coeff2>]`
  Sets/Gets the atom squared coefficient.

- `&atom.phase [= <phase>]`
  Sets/Gets the atom phase (which is defined modulo 1).

- `&atom.coeff`
  Gets the atom complex coefficient.

- `&atom.buildr [*opt,...]  [:]`
  Gets a signal where the real atom has been built.
  Options are : `*sizeonly,*bperiodic,...`

  - *sizeonly : the signal(s) where the atom will be built will have exactly the size of the atom. Otherwise the atom.signalSize will be used.

- `&atom.buildc [*opt,...]  [:]`
  Gets a pair {real imag} of signals where the normalized complex atom has been built.
  Options are : `*sizeonly,*bperiodic,...`

  - *sizeonly : the signal(s) where the atom will be built will have exactly the size of the atom. Otherwise the atom.signalSize will be used.

- `&atom.dt`
  Gets the time spread of an atom in seconds.

- `&atom.df`
  Gets the frequency spread of an atom in Hertz.

- `&atom.support`
  Gets the time support {timeMin timeMax} of an atom in seconds

- `&atom.supportId`
  Gets the time support {timeIdMin timeIdMax} of an atom in sample coordinates

- `&atom.gg`
  Gets a listv {real imag} that corresponds to the (complex) inner-product <g,_g> between a (normalized) complex atom 'g' and its complex conjugate '_g'. The value of <g,_g> depends on the windowShape, windowSize, frequency and chirp parameters. It is used to compute the energy ||P_{g,_g}s||ˆ 2 of the projection of a (real valued) signal 's' on the subspace spanned by 'g' and '_g' from the complex inner product <s,g>. WARNING : this is a read-only field. If you type 'atom.gg[0]=1' it will be accepted but what will be performed is similar to 'l=atom.gg; l[0]=1'.

3.1.2 Type `&book`

This type is the basic type for storing the result of Matching Pursuit decompositions as an array of &mol's.
- Operator + : book+molecule, appends a molecule at the end of the book.

- `&book [<n>]`
  Gets the molecule <n> of a book

- `&book.sig [<n>]`
  Gets the signal <n> of a book

- `&book.name [= <name>]`
  Sets/Gets the name of a book

- `&book.dx [= <dx>]`
  Sets/Gets the abscissa step of the signal which has been analyzed.

- `&book.x0 [= <x0>]`
  Sets/Gets the first abscissa of the signal which has been analyzed.

- `&book.signalSize`
  Gets the size of the original signal of the time-frequency transform.

- `&book.freqIdNyquist`
  Gets the Nyquist frequency in sample coordinates.

- `&book.size`
  Gets the number of &mol in a book.

- `&book.sizeAlloc [= <sizeAlloc>]`
  Sets/Gets the allocation size for the array of &mol in a book. In case of a Set, <sizeAlloc> must be larger than book.size, else an error is generated. The previously allocated part is kept (book.size is not changed).

- `&book.dim`
  Gets a &signal of size book.size containing the list of dimensions of the molecules of a

book (i.e. the number of atoms contained in each molecule). The dimension is larger than 1 only for books built using the Harmonic Matching Pursuit.

- `&book.wcoeff2`
  Gets a &signal of size book.size containing the list of 'coeff2' of the molecule in a book.
- `&book.windowSize`
  Gets a &signal of size book.size containing the list of 'windowSize' of the first atom of the molecules in a book.
- `&book.windowShape`
  Gets a &listv of size book.size containing the list of 'windowShape' of the first atom of the molecules of a book.The available window shapes are : blackman, hanning, hamming, gauss, spline0 (rectangle), spline1, spline2, spline3, exponential or FoF.
- `&book.timeId`
  Gets a &signal of size book.size containing the list of 'timeId' of the first atom of the molecules in a book..
- `&book.time`
  Gets a &signal of size book.size containing the list of 'time' of the first atom of the molecules in a book.
- `&book.freqId`
  Gets a &signal of size book.size containing the list of 'freqId' of the first atom of the molecules in a book.
- `&book.freq`
  Gets a &signal of size book.size containing the list of 'freq' of the first atom of the molecules in a book.
- `&book.chirpId`
  Gets a &signal of size book.size containing the list of 'chirpId' of the first atom of the molecules in a book.
- `&book.chirp`
  Gets a &signal of size book.size containing the list of 'chirp' of the first atom of the molecules in a book.
- `&book.acoeff2`
  Gets a &signal of size book.size containing the list of 'coeff2' of the first atom of the molecules in a book.
- `&book.phase`
  Gets a &signal of size book.size containing the list of 'phase' of the first atom of the molecules in a book.
- `&book.gg`
  Gets a &listv {real imag} of two &signal of size book.size containing the list of 'gg' of the first atom of the molecules in a book.

3.1.3 Type `&dict`

This type is the basic type for time-frequency dictionaries for Matching Pursuit decompositions.

- `&dict.channels`
  Gets a &listv containing the channels of a &dict.
- `&dict.signalEnergy`
  Gets the energy of the signal of a &dict.
- `&dict.maximadict`
  Gets the &maximadict sub-dictionary of a &dict.
- `&dict.stft`
  Gets a &listv containing all the &stft sub-dictionaries of a &dict.

3.1.4 Type `&maximadict`

This type is the basic type for local maxima of time-frequency dictionaries for Matching Pursuit decompositions.

- `&maximadict.book`
  Gets a &listv containing all the &book of local maxima of a &maximadict.
- `&maximadict.thresh`
  Gets the threshold of a &maximadict.
- `&maximadict.nmax`
  Gets the total number of maxima of a &maximadict.

3.1.5 Type `&mol`

This type corresponds to subspaces spanned by a few atoms and is used in (Harmonic) Matching Pursuit decompositions.

- `&mol [<n>]`
  Gets the molecule <n> of a molecule
- `&mol.dim`
  Returns the number of atoms in the &mol
- `&mol.coeff2`
  Returns the coeff2 of the &mol (it is the sum of those of its atoms)

3.2 Commands which deal with &dict : dictionaries of atoms

- **setdict**

  - **setdict** `add <dict> '&maximadict' <nmaxima>`

    Adds a sub-dictionary of local maxima.

  - **setdict** `add <dict> '&stft' [('real'|'harmo'|'highres')] <windowSize> [<windowShape>='g`
    `[{<freq0Min> <freq0max>}]`

    Adds a &stft sub-dictionary.

  - **setdict** `channels <dict> {<signali1> ... <signaliN>}`

    Sets the channels of a dictionary.

  - **setdict** `getmax <dict> [ {['causal'] [{'time(Id)' <range>}] [{'freq(Id)' <range>}] [{'windowSize' <range>}]} ] [ {['interpolate'] ['chirp']} ]`

  - **setdict** `optmol <dict> <molecule> [ {['time'] ['freq'] ['chirp'] ['recompute']} ]`

    Optimizes a molecule using a dictionary

  - **setdict** `rmmol <dict> <molecule>`

  - **setdict** `update <dict>`

    Updates all sub-dictionaries to enable a new 'getmax'.

3.3 Commands which deal with &book and &mol variables

- **book**

  - **book** `read [<book>=objCur] <filename>`

    Reads a book from a file.

  - **book** `readold [<book>=objCur] <filename> <forceMaxFreqId> <decay>`

    Reads a book (and the decay signal) from a file in older format. You have to specify what was the MaxFreqId, you may have to make several trials and check the result using 'mpr'.

  - **book** `write [<book>=objCur] <filename> [-b]`

    Writes a book to a file in ascii format (by default) or in binary (option -b).

- **mpr** `[<book>=objCur] <reconsSignal> [<maskSignal>] [-n <nMin> [<nMax>=<nMin>]] [-s <windowSizeMin> [<windowSizeMax>=<windowSizeMin>]>] [-t <timeMin> <timeMax>] [-T <timeIdMin> <timeIdMax>] [-f <freqMin> <freqMax>] [-F <freqIdMin> <freqIdMax>] [-c <chirpMin> <chirpMax>] [-C <chirpIdMin> <chirpIdMax>]`
Builds the reconstructed signal from the molecules of a book. The command returns the number of molecules that were actually used. A masking signal can optionally be used to

specify which molecules are used (the molecule number <n> is used in the reconstruction iff the <n>th sample in maskSignal is nonzero). Additionally, the reconstruction only uses the molecules whose fields lie within the range specified by the following options:
-n : Specifies the molecule number range
-s : Specifies the scale range
-t : Specifies the time range (real units)
-f : Specifies the frequency range (real coordinate)
-c : Specifies the chirp range (real coordinates)
-C : Specifies the chirpId range (id units).
-T : Specifies the timeId range (id units)

3.4 Commands which deal with inner products of &atoms

- **inner**

  - **inner** `auto <atom>`

    Computes the inner product between a complex atom and its conjugate.

  - **inner** `cc <atom1> <atom2> [-n]`

    Computes the inner product between two complex atoms, with a fast computation if possible. Option -n forces exact (slow) numeric computation.

  - **inner** `rc <atomR> <atomC> [-n]`

    Computes the inner product between a real atom and a complex one, with a fast computation if possible. Option -n forces exact (slow) numeric computation.

  - **inner** `rr <atom1> <atom2> [-n]`

    Computes the inner product between two real atoms, with a fast computation if possible. Option -n forces exact (slow) numeric computation.

  - **inner** `sig <signal> <atom>`

    Computes the inner product between a signal and a complex atom.

3.5 Commands which deal with notes from a book

- **notes** `<book> ([-n <nMin> [<nMax>=<nMin>]] | [-s <output> [<attackDuration>] <noteList>])`

Gets a list with the notes from <book> usign molecules between index <nMin> and <nMax>. Options '-s' is used to synthesize in a signal the notes using sinusoids and a simple cosinusoidal attack pattern of <attackDuration> samples.

- **profile** `<book> <signal> <n> <deltaFreq>`

Computes the energy profile at the location of the <n>th molecule of a book and puts it in a signal.

3.6 Script Commands

• **BuildDict** (in file `LastWave_3_0/scripts/mp/MPDalgorithms`) `<book> {<args>}`
Parses the arguments of a *mpd command to derive the correct dictionary and residual energy signals

• **book_convert_format** (in file `LastWave_3_0/scripts/mp/MPDalgorithms`) `<oldfilename> <newfilename> <forceMaxFreqId> [<decayFilename>]`
Reads a book in an old format from <oldfilename> using <forceMaxFreqId> and writes it in the new format to <newfilename>. The residualEnergy, as a signal, is written to <decayFileName>.

• **fasthmpd** (in file `LastWave_3_0/scripts/mp/MPDalgorithms`) `[<book>=objCur] <nIter> <nmaxima> <freq0min> <freq0max> [<signal>=0book] ['-w' {<windowShapes>}={'gauss'}] ['-s' {<windowSizes>}=2^ (2:max)] ['-O' {MPoptimizations}]`
Makes <nIter> iterations of a Fast Harmonic Matching Pursuit on <signal> (by default, we analyze the signal '0' of the <book>) and stores the result in <book>. The book will end up with <nIter> molecules in it, except if the pursuit stops before because the residue is zero.

• **fastmpd** (in file `LastWave_3_0/scripts/mp/MPDalgorithms`) `[<book>=objCur] <nIter> <nMaxima> [<signal>=0book] ['-w' {<windowShapes>}={'gauss'}] ['-s' {<windowSizes>}=2^ (2:max)] ['-O' {MPoptimizations}]`
Makes <nIter> iterations of a Fast Matching Pursuit on <signal> (by default, we analyze the signal '0' of the <book>) and stores the result in <book>. The book will end up with <nIter> molecules in it, except if the pursuit stops before because the residue is zero.
Fast Matching Pursuit uses sub-dictionaries of <nmaxima> local time-frequency maximas (it is MUCH faster than the regular 'mpd' algorithm but the algorithm is more greedy and may sometimes give poorer approximations of the analyzed signal). HINT : as a rule of thumb, you can choose <nmaxima> of the order of <nIter>. The algorithm is optimized for 'gauss' and 'FoF' atoms type.
The default shape of the atoms in the dictionary is Gaussian but you can use, e.g., Hanning and FoF windows using the {'hanning' 'FoF'} syntax for {<windowShapes>}. The default atom sizes in the dictionary are powers of two from 2 to 2^ max, where 2^ max is about the size of the signal, but you can choose other (still powers of two!) sizes using, e.g., the {2 16 32} syntax for {<windowShapes>}. Note that for {<windowShapes>} you can provide either a &range, a &listv or a &signal or &signali.
The possible MP optimizations options are 'chirp' and 'freq' which correspond to extending the dictionary to chirps (only for 'gauss' window shapes) and newton interpolation of the frequency, respectively.

• **hmpd** (in file `LastWave_3_0/scripts/mp/MPDalgorithms`) `[<book>=objCur] <nIter> <freq0min> <freq0max> [<signal>=0book] ['-w' {<windowShapes>}={'gauss'}] ['-s' {<windowSizes>}=2^ (2:max)] ['-O' {MPoptimizations}]`
Makes <nIter> iterations of a Harmonic Matching Pursuit on <signal> (by default, we analyze the signal '0' of the <book>) and stores the result in <book>. The book will end

up with <nIter> molecules in it, except if the pursuit stops before because the residue is zero.

• **m** (in file `LastWave_3_0/scripts/mp/mp.pkg`)
Changes the objCur variable to the book 'm'.

• **mpd** (in file `LastWave_3_0/scripts/mp/MPDalgorithms`)

- **mpd** `[<book>=objCur] <nIter> <dict> <residualEnergy> [{MPoptimizations}]`

  To resume a Matching Pursuit using the pair <dict> <residualEnergy> returned by a previous call, you only have to specify <nIter> and you can specify the optimizations you want to perform.

- **mpd** `[<book>=objCur] <nIter> [<signal>=0book] ['-w' {<windowShapes>}={'gauss'}] ['-s' {<windowSizes>}=2^ (2:max)] ['-O' {MPoptimizations}]`

  Makes <nIter> iterations of a Matching Pursuit on <signal> (by default, we analyze the signal '0' of the <book>) and stores the result in <book>.The book will end up with <nIter> molecules in it, except if the pursuit stops before because the residue is zero.
  The default shape of the atoms in the dictionary is Gaussian but you can use, e.g., Hanning and FoF windows using the {'hanning' 'FoF'} syntax for {<windowShapes>}. The default atom sizes in the dictionary are powers of two from 2 to 2^ max, where 2^ max is about the size of the signal, but you can choose other sizes (powers of two) using, e.g., the {2 16 32} syntax for {<windowShapes>}. Note that for {<windowShapes>} you can provide either a &range, a &listv or a &signal or &signali.
  The possible MP optimizations options are 'chirp' and 'freq' which correspond to extending the dictionary to chirps (only for 'gauss' window shapes) and newton interpolation of the frequency, respectively.

• **n** (in file `LastWave_3_0/scripts/mp/mp.pkg`)
Changes the objCur variable to the book 'n'.

3.7 Graphic class `GraphBook` (inherits from `GObject`)

Graphic Class that allows to display Book
• **setg**

- **setg** `*GraphBook* -?closest <time> <freq>`

  Gets the rank of the atom that is currently displayed and that is the closest to position <time> <freq>. It returns either 'null' if it fails or a listv {<n> <k>} where 0<=n<book.size is the rank of the molecule and 0<=k<molecule.dim is the atom number in the molecule.

- **setg** `*GraphBook* -chirp [<chirpMin> <chirpMax>]`

  Sets/Gets the chirp range of the molecules that are displayed.

- **setg** `*GraphBook* -chirpId [<chirpIdMin> <chirpIdMax>]`

  Sets/Gets the chirpId range of the molecules that are displayed.

- **setg** `*GraphBook* -cm [<colormap>]`

  Sets/Gets the colormap that will be used to display the book.

- **setg** `*GraphBook* -db [<flagOnOff>]`

  Sets/Gets the decibel-display flag. If it is on, the energy of the molecules is displayed in decibel.

- **setg** `*GraphBook* -expo [<exponent>]`

  Sets/Gets the exponent used for display. If '-db' is off, then the energy to the <exponent> of the molecules is displayed. If '-db' is on, then the same quantity is displayed in decibel.

- **setg** `*GraphBook* -fund <flag>`

  Sets/Gets the fundamental flag which can be 'all','max' or 'sum'. This flag allows to display all the atoms within each molecule (<flag>='all') or only the most energetic atom with its energy (<flag>='max') or with the total energy of the molecule (<flag>='sum').

- **setg** `*GraphBook* -graph [<book>]`

  Gets/Sets the book to be displayed by the GraphBook. (The '-cgraph' field is equivalent to that field).

- **setg** `*GraphBook* -n [<nMin> <nMax>]`

  Sets/Gets the minimum and maximum ranks of the molecules that are displayed.

- **setg** `*GraphBook* -scale [<windowSizeMin> <windowSizeMax>]`

  Sets/Gets the windowSize range of the molecules that are displayed.

Bindings

- Shift+Left button = move molecule/atom
- Shift+Middle button = transpose molecule/atom
- Shift+Right button = translates molecule/atom
- Ctrl+Left button = scale molecule/atom
- Ctrl+Middle button = change amplitude molecule/atom
- Down key = Go to First atom
- Up key = Go to Last atom
- Right key = Go to Next atom

- Left key = Go to Previous atom
- '-' = Play a sine at cursor frequency
- '=' = Play Closest atom
- '<' = Play reconstruction up to closest atom
- '>' = Play reconstruction from closest atom
- Type 'c' to change cursor mode
- 'z' : changes the zoom mode just type 'z'
- Left/Right/Middle button : operate the zoom

3.8 Demos

Here is a list of all the Demo files and for each of them all the corresponding Demo commands. To try a Demo command, you should first source the corresponding Demo file then run the command. (When sourcing the Demo file, LastWave tells you about all the commands included in this file).
The Demo files corresponding to this package are :

Demo file **DemoMP**

• **DemoMPFastAlgo** (in file `LastWave_3_0/scripts/mp/DemoMP`)

Demo command that compares the results of both the regular matching pursuit algorithm and the fast one (using 100 atoms) on an artificial signal which consists in the sum of a sinus a dirac and some white noise. It displays the time-frequency representations given by both the fast algorithm and the regular one as well as the so-obtained reconstruction (along with the original and the error). It also displays the decays of the residue energy (after each iteration) for both the regular and the fast algorithms.

• **DemoMPRegAlgo** (in file `LastWave_3_0/scripts/mp/DemoMP`)

Demo command that computes and displays the result of the regular matching pursuit algorithm (30 atoms) on an artificial signal which consists in the sum of a sinus a dirac and some white noise. It also displays the reconstructed signal when using only 'long' atoms (i.e., to recover the sinus) and the reconstructed signal when using only 'short' atoms (i.e., to recover the dirac). It also teaches you how to use the mouse.

• **DemoMPSound** (in file `LastWave_3_0/scripts/mp/DemoMP`)

WARNING : YOU SHOULD FIRST RUN THE DEMO 'DemoSound' OF THE SOUND PACKAGE AND THE OTHER DEMOS OF THE MP PACKAGE BEFORE RUNNING THIS DEMO. This Demo command performs the matching pursuit analysis of a piano sound and teaches you the sound capabilities of the 'mp' package. This demo involves computation that can take 2-3 minutes. This demo is also included in the 'sound' package.

Chapter 4

Package `stft` 2.1

Package allowing the computation of Short Time Fourier Transforms and some derived time-frequency representations.
*** Authors and Copyright : R.Gribonval, E. Bacry and J.Abadia*

4.1 Defined types

4.1.1 Type `&stft`

This type is the basic type for Short Time Fourier transforms and related time-frequency transforms.

- `&stft.dx [= <dx>]`
  Sets/Gets the abscissa step of the signal which has been analyzed.

- `&stft.x0 [= <x0>]`
  Sets/Gets the first abscissa of the signal which has been analyzed.

- `&stft.signalSize`
  Gets the size of the original signal of the time-frequency transform.

- `&stft.freqIdNyquist`
  Gets the Nyquist frequency in sample coordinates.

- `&stft.windowShape`
  Gets the windowShape of a Short Time Fourier Transform. The available window shapes are : blackman, hanning, hamming, gauss, spline0 (rectangle), spline1, spline2, spline3, exponential or FoF.. To see the shape of a window, you can build it using the 'stft window ...' command.

- `&stft.windowSize`
  Gets the windowSize of a Short Time Fourier Transform, i.e. the number of samples of its window. So far only powers of 2 are allowed. To see the shape of a window, you can build it using the 'stft window ...' command.

- `&stft.grid`
  Gets stft 'grid' i.e. returns a &listv {timeRate timeLength freqRate freqLength}.
- `&stft.border`
  Returns stft <border type>('per'=periodic, 'mir'=mirror, 'pad0'=padding with 0 values).
- `&stft.firstp`
  Gets stft <firstp>
- `&stft.lastp`
  Gets stft <lastp>
- `&stft.type`
  Gets stft type (it is either 'complex' for short time fourier transform, 'real', 'phase' or 'highres'
- `&stft.sig`
  Returns a signal containing the stft real data or a listv {real imag} containing its complex data

4.2 Commands which deal with Short Time Fourier Transforms

- **stft**
  - **stft** `* <stftIn1> <stftIn2> [<stftOut>=<stftIn1>]`
    Multiply two stfts and puts the result in a new one.
  - **stft** `+ <stftIn1> <stftIn2> [<stftOut>=<stftIn1>]`
    Adds two stfts and puts the result in a new one.
  - **stft** `- <stftIn1> <stftIn2> [<stftOut>=<stftIn1>]`
    Substract two stfts and puts the result in a new one.
  - **stft** `/ <stftIn1> <stftIn2> [<stftOut>=<stftIn1>]`
    Divides two stfts and puts the result in a new one.
  - **stft** `conjugate <stftIn> [<stftOut>=<stftIn>]`
    Conjugates a (complex) stft and puts the result in a new one.
  - **stft** `ln <stftIn> [<stftOut>=<stftIn>]`
    Takes the natural logarithm of a stft and puts the result in a new one. If the stft is complex, the result is complex with its imaginary part equal to the phase of the input.
  - **stft** `log <stftIn> [<stftOut>=<stftIn>]`
    Takes the logarithm in base 10 of a stft and puts the result in a new one. If the stft is complex, the result is complex with its imaginary part equal to the phase of the input.

- **stft** `log2 <stftIn> [<stftOut>=<stftIn>]`

  Takes the logarithm in base 2 of a stft and puts the result in a new one. If the stft is complex, the result is complex with its imaginary part equal to the phase of the input.

- **stft** `window <windowShape> <windowSize>`

  Returns a signal which contains a copy of a window tabulated in the package 'stft'.

- **stft** `write <stft> (<stream>|<filename>) [-h]`

  Writes a stft to a <file> in ascii format. With option '-h' no header is written.

• **stftd** `[<stft>=objCur] <signal> [<windowShape>='gauss'] <windowSize> [(complex | real | phase)] [-b <borderType>] [-T <time redundancy>=4] [-F <freq redundancy>=2]`
Computes a Short Time Fourier Transform of the <signal>.
- The window size is <windowSize>, its shape is given by <windowShape>. The available window shapes are : blackman, hanning, hamming, gauss, spline0 (rectangle), spline1, spline2, spline3, exponential or FoF.
- You can specify the type of stft that should be computed :
'complex' corresponds to a regular short time fourier transform.
'real', 'phase' correspond respectively to the energy/phase of the best matched real Gabor atoms (which is quite close to, but slightly different from, the magnitude or phase of the complex spectrogram ...).
- The treatment of border effects is determined by the argument of option '-b'. ('per'=periodic, 'mir'=mirror, 'pad0'=padding with 0 values)
- Options '-T' and '-F' determine the time and frequency redundancy factors, that is to say the *time-frequency grid* associated with the spectrogram. This means that :
** a FFT is computed at each 1/<time redundancy>th of the window size.

• **stftmax** `[<stft>=objCur] [-(t,T) <tMin> <tMax>] [-(f,F) <fMin> <fMax>] [-c]`
Gets the point where the maximum of energy of a stft is reached. It returns the list '<maxEnergy> <timeId> <freqId>', or 0 if <maxEnergy> is zero, -1 if the domain is empty. The options '-t' and '-T' allow to restrict the search to points for which the time is in between a given range <tMin> <tMax> which is specified using real time scale ('-t') or timeIds ('-T'). The options '-f' and '-F' allow to restrict the search in the same way in the frequency domain. The option '-c' restricts the search to points not affected by border effects.

4.3 Graphic class `GraphStft` (inherits from `GObject`)

Graphic Class that allows to display Stft
• **setg**

- **setg** `*GraphStft* -causal [<flagOnOff>]`

  Sets/Gets the causal-display flag. If it is on, only the region not affected by border effects is displayed.

- **setg** `*GraphStft* -cm [<colormap>]`

  Sets/Gets the colormap that will be used to display the stft.

- **setg** `*GraphStft* -db [<flagOnOff>]`

  Sets/Gets the decibel-display flag. If it is on, the energy is displayed in decibel.

- **setg** `*GraphStft* -expo [<exponent>]`

  Sets/Gets the exponent used for display. If '-db' is off, then the <exponent> is not used. If '-db' is on, then the same quantity is displayed in decibel.

- **setg** `*GraphStft* -graph [<stft>]`

  Gets/Sets the stft to be displayed by the GraphStft. (The '-cgraph' field is equivalent to that field).

Bindings

- {Shift + Middle button = vertical (decibel) section}
- {Ctrl + Middle button = horizontal (decibel) section}
- Type 'c' to change cursor mode
- 'z' : changes the zoom mode just type 'z'
- Left/Right/Middle button : operate the zoom

4.4 Demos

Here is a list of all the Demo files and for each of them all the corresponding Demo commands. To try a Demo command, you should first source the corresponding Demo file then run the command. (When sourcing the Demo file, LastWave tells you about all the commands included in this file).
The Demo files corresponding to this package are :

Demo file **DemoSTFT**

• **DemoSTFTBonjour** (in file `LastWave_3_0/scripts/stft/DemoSTFT`)

Demo command that computes and displays the short time fourier transform (using a hamming window) of a voice signal saying the french word 'bonjour'. It also prints the coordinates of the most correlated atom. Warning : This demo uses a lot of memory, thus, if you are running on a Macintosh, you should allocate at least 20Mo to LastWave before running it.

Index