The Matching Pursuit Tool Kit
User Manual and Developer's Notes

Sacha Krstulović and Rémi Gribonval

November 21, 2007, *Revision* : 1107

Contents

This document aims at describing the basic principles of the Matching Pursuit Tool Kit, and some basic guidelines about how to use it. The appendix also provides the potential contributor with notes about the insides of our implementation, plus guidelines about some of the development conventions that we have adopted. For a more detailed documentation of the code, see the automatically generated reference manual.

1 General principles

1.1 Terminology

Atom – An elementary piece of signal. An atom is characterized by its class (e.g., Gabor atoms, harmonic atoms etc.) and a set of parameters (e.g., for a Gabor atom, its time location and length, its frequency, its amplitude, a chirp factor and a given shaping window). In our code, the corresponding object knows how to subtract itself from a signal, and how to generate its own waveform.

Block – A block computes the correlations (inner products) between a signal to analyze and a set of atoms for which the computation of the correlations can be factorized in an efficient manner, along the whole support of a signal.

For instance, the inner products can stem from a time-frequency transform, such as the Fourier Transform or a wavelet transform, provided the transform can be interpreted as a set of correlations between the signal and a set of atoms which cover the time-frequency plane.

Each block object can search for the location of the maximum correlation between the atoms and the signal, and can thus deduce which atom contributes the most energy to the analyzed signal. Several blocks corresponding to various scales or various transforms can be concurrently applied to the same signal, thus providing for multi-scale or multiple-basis analysis.

For the moment, the following blocks are implemented:

- Gabor blocks, based on Short-Time Fourier Transforms, which compute the correlations with windowed sinusoids having a "flat" frequency (chirp rate == 0). In this case, one block is conceptually equivalent to one STFT with a given scale;
- Harmonic blocks, based on the harmonic grouping of Gabor atoms, and inheriting from the Gabor block;
- Chirp blocks, based on a post-processing of the Gabor atoms and aimed at optimizing the chirp rate;
- AnyWave blocks, based on a fast convolution algorithm to compute some correlations with arbitrary waveforms.
- MDCT/MDST/MCLT blocks. They are transforms based on local cosine functions.
- Dirac blocks,which provide no particular time-frequency transform, but just find the signal samples which have a high energy
- Constant blocks, which are an extension of the Dirac blocks. Constant atoms are rectangular windows, defined by a length and a shift between atoms. A constant atom catches the mean of a signal on the specified frame.

- Nyquist blocks. They are defined by a length and a shift between atoms, and the waveform is a normalized succession of 1 and -1. A Nyquist atom catches the greatest frequency that can be distinguished (called the Nyquist frequency if the support is even) of the specified frame.

In the future, blocks based on fast wavelet transforms should also be designed.

Dictionary – A dictionary contains a collection of blocks plus the signal on which they operate. It can search across all the blocks (i.e., all the scales and all the bases) for the atom which brings the most energy to the analyzed signal.

Book – A book is a collection of atoms. Summing all the atoms in a book gives a signal.

Figure 1 illustrates the fact that a dictionary contains a signal and a collection of blocks, and that a books contains a collection of atoms. The algorithm linking these objects will be explained in the next section.

1.2 Algorithm

Our implementation of the Matching Pursuit algorithm uses roughly 3 steps, as illustrated in figure 1:

1. update the correlations in the blocks, by applying the relevant correlation computation algorithm to the analyzed signal, and find the maximum correlation in the same loop;
2. create the atom which corresponds to the maximum correlation with the signal (and store this atom in the book);
3. subtract the created atom from the analyzed signal, thus obtaining a residual signal, and re-iterate the analysis on this residual.

At each step, the original signal can be rebuilt exactly by summing all the atoms of the book and adding them to the residual signal.

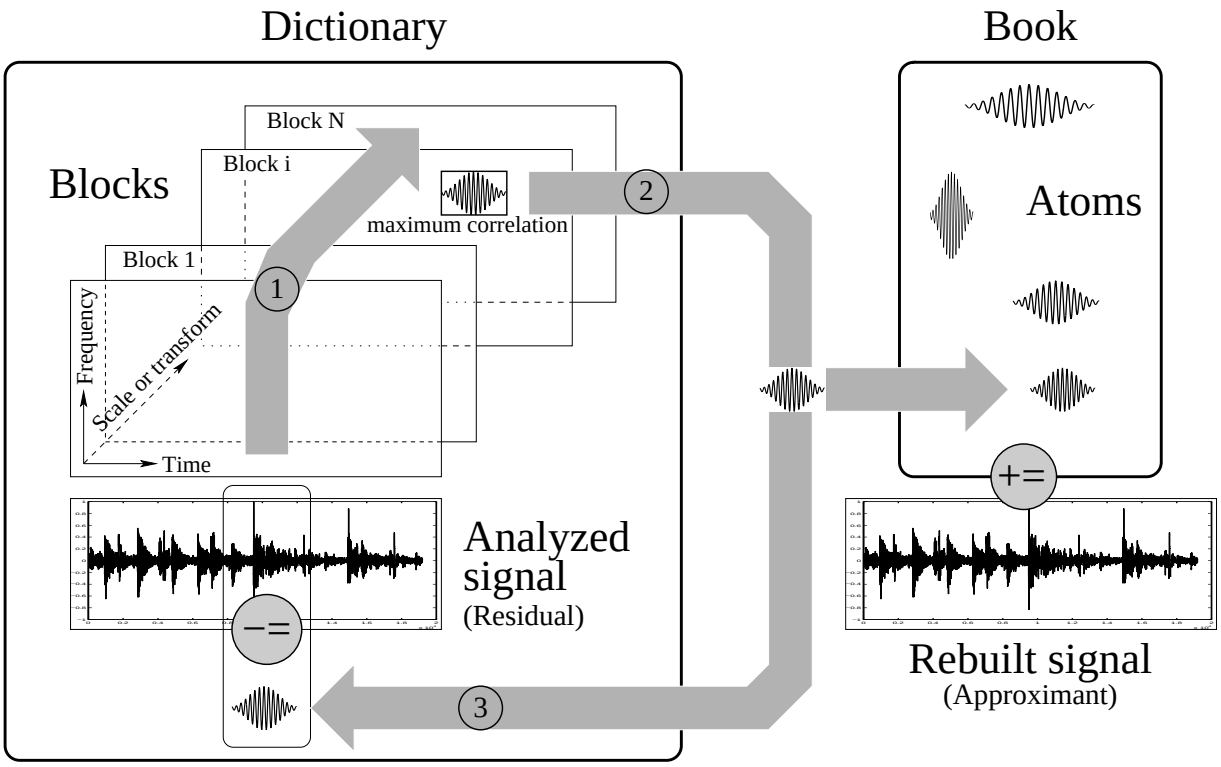

Figure 1: Our implementation of Matching Pursuit. (See sections 1.1 and 1.2 for a detailed explanation.)

2 User manual for the standalone utilities

2.1 Introduction

The Matching Pursuit standalone utilities are designed so that a typical processing task decomposes into a chain of commands which can be connected by pipes. The signal decomposition is performed by `mpd`. The atoms stored in the resulting books can be filtered (i.e., sorted and selected) with the command `mpf`. Several books can then be concatenated with the command `mpcat`. Once a desired book is obtained, a corresponding approximant signal can be rebuilt using `mpr`, with the optional addition of a residual signal (or in addition to any other signal). Alternately, the `mpd_demix` utility provides blind source separation using Matching Pursuit when the mixing matrix is known. Each of the cited utilities have a `-h` switch to get command-line help.

Some typical processing chains include:

- Signal approximation with N atoms:
  ```
  mpd -D dictionary -n N soundFile.wav book.bin;
  mpr book.bin approx.wav
  ```
  – or –
  ```
  mpd -D dictionary -n N soundFile.wav - | mpr - approx.wav
  ```
- Exact reconstruction:
  ```
  mpd -D dictionary -n N soundFile.wav book.bin residual.wav;
  mpr book.bin approx.wav residual.wav
  ```
- Signal approximation, using only the short atoms (e.g., of less than 128 samples) and ignoring the residual:
  ```
  mpd -D dictionary -n N soundFile.wav book.bin;
  mpf --length=[0:128] book.bin bookShort.bin;
  mpr bookShort.bin approx.wav
  ```
- Blind source separation and rebuilding an approximation of one of the sources (e.g., the 3rd one):
  ```
  mpd_demix -D dictionary -M matrix.txt -n N mix.wav bookFile;
  mpr bookFile_3.bin approxOfSource3.wav
  ```

Lots of other combinations are possible. The relevant utilities are described with more detail in the next sections.

Note about the signal file formats – MPTK has been originally designed for sound processing, but it is applicable to any kind of signal.

For input, we have mostly been using the WAV format, but any file format recognized by the `libsndfile` library (i.e., most audio formats, see `http://www.mega-nerd.com/libsndfile/`) should work with the provided MPTK utilities.

The signal output is more specific: the MPTK binaries output signals in the WAV format only. *(WARNING: in the* `libsndfile` *implementation, this format is not protected against clipping, which may happen for some books and is not an artifact of the MPTK analysis.)* Nevertheless, the MPTK API also offers Matlab, raw `double` and raw `float` signal output formats: those could be quickly enabled by simple hacks of the relevant utilities. To enable other output formats, please contribute some code to the `mp_signal.{h,cpp}` API.

2.2 the MPTK environment

In order to define the working environment of Matching Pursuit utilities, An XML file is loaded before using Matching Pursuit. This XML file contains the configuration path `<configpath>` path`</configpath>` the syntax of the path is `<path name="NAME" path="PATH"/>` the path used are:

- `dll_directory` the directory where MPTK library search for plugins Dynamic Link Library (DLL) defining atoms and blocks
- `fftw_wisdomfile` the default path for the FFTW wisdom file which allows to save the setting for FFT computation.

This values can be set by two way, the complete path of the file can be specified with the `-C<FILE>, -config-file=<FILE>` for utilities or MPTK library calls the environment variable `MPTK_CONFIG_FILENAME` to determine which file to use for setting up the MPTK environment.

2.3 mpd: signal decomposition

Usage:

```
mpd [options] -D dictFILE.txt (-n N|-s SNR) (sndFILE.wav|-) (bookFILE.bin|-) ...
              ... [residualFILE.wav]
```

Synopsis:

Iterates Matching Pursuit on signal sndFILE.wav with dictionary dictFILE.txt and gives the resulting book bookFILE.bin (and an optional residual signal) after N iterations or after reaching the signal-to-residual ratio SNR. See section 2.4 for a description of the syntax of the dictionary files.

Mandatory arguments:

```
     -D<FILE>, --dictionary=<FILE>  Read the dictionary from text file FILE.

     -n<N>, --num-iter=<N>|--num-atoms=<N>    Stop after N iterations.
AND/OR -s<SNR>, --snr=<SNR>                   Stop when the SNR value SNR
                                              is reached. If both options are
                                              used together, the algorithm stops
                                              as soon as either one is reached.

     (sndFILE.wav|-)                          The signal to analyze or stdin
                                              (in WAV format).
     (bookFILE.bin|-)                         The file to store the resulting
                                              book of atoms, or stdout.
```

Optional arguments:

```
     -C<FILE>, -config-file=<FILE>  Use the specified configuration file,
                                    otherwise the MPTK_CONFIG_FILENAME environment
                                    variable will be used to find the configuration
                                    file and set up the MPTK environment
     -E<FILE>, --energy-decay=<FILE>  Save the energy decay as doubles
                                      to file FILE.
     -R<N>,    --report-hit=<N>       Report some progress info (in stderr)
                                      every N iterations.
     -S<N>,    --save-hit=<N>         Save the output files every N iterations.
     -T<N>,    --snr-hit=<N>          Test the SNR every N iterations only
                                      (instead of each iteration).

     -p<double>, --preemp=<double>    Pre-emphasize the input signal with
                                      coefficient <double>.

     residualFILE.wav                The residual signal after subtraction
                                     of the atoms.
```

```
-q, --quiet                 No text output.
-v, --verbose               Verbose.
-V, --version               Output the version and exit.
-h, --help                  This help.
```

2.4 Format of the dictionary files

The dictionary files use a XML syntax, with tags enclosed between angle brackets. Strict XML compliance is now mandatory. [1].

General rules

A dictionary should include, in the following order:

1. an optional XML declaration line;
2. a mandatory dictionary opening tag;
3. an optional library version tag;
4. a list of blocks with their parameters (some parameters being mandatory, some admitting default values);
5. a dictionary closing tag.

Any text included after the dictionary closing tag will be ignored. Blank spaces and line breaks are ignored. Any part of the file can be commented out either by enclosing it between `<!--` and `-->` (XML style) The parser will send any text it can't match to stderr with a error message: in the event of syntax errors in a dictionary, some dictionary pieces will therefore show up in stderr.

Valid tags

- `<?xml version="1.0" encoding="ISO-8859-1"?>` [optional]: the XML declaration line.
- `<dict>`List of blocks`</dict>`: the opening and closing tags for the dictionary. Anything coming after the `</dict>` tag will be ignored by the parser.
- `<libVersion>`blah`</libVersion>` [optional]: the library version declaration. This is provided for backward compatibility if we ever change the dictionary syntax. When absent, the library version is taken to be the current one.
- `<blockproperties name="PROPERTIES_NAME">`List of block parameters`</blockproperties>`: the opening and closing tags for a block properties list.this
- `<block uses="PROPERTIES_NAME">`List of block parameters`</block>`: the opening and closing tags for a block. The block will be construct using the list of parameters defined in `PROPERTIES_NAME` block properties.
- `<block>`List of block parameters`</block>`: the opening and closing tags for a block. the list of block parameters should contains all the mandatories parameters for this type of block

[1] Uh, that is, if a DTD get ready someday.

- `<par type="NAME" value="VALUE"/>` : a block parameter. The `NAME` and corresponding `VALUE` value in string. The parameter value is set to the corresponding type when the dictionary file is parsed.

  Note that when using the `<block uses="PROPERTIES_NAME">`List of block parameters `</block>` syntax. The block parameters defined in the block list override the parameters from `PROPERTIES_NAME` block properties.

  The list of parameters for relevant block types (G==gabor, H==harmonic, C==Chirp, A==Anywave, CT==constant, N==nyquist) are given below:

  - (G,H,C,CT,N) `windowLen` with an unsigned long int value: the atom length (i.e., the window length in the STFT).
  - (G,H,C,A,CT,N) `windowShift` with an unsigned long int value: the atom shift (i.e., the window shift in the STFT).
  - (G,H,C) `windowRate` with a double value between 0.0 and 1.0: an alternate way to specify the window shift, as a proportion of the windowLen. If both are present, `windowShift` has precedence over `windowRate`.
  - (G,H,C) `fftSize` with an unsigned long value: the frequency resolution in terms of FFT size. It has to be greater or equal to `windowLen` if `windowLen` is even, or $\geq$`(windowLen+1)` if `windowLen` is odd. If absent, defaults to `windowLen` (or `windowLen+1`).
  - (G,H,C,CT,N) `blockOffset` with an unsigned long value: the block Offset, i.e. the position of the first frame. If absent, defaults to 0.
  - (G,H,C,CT,N) `windowtype` the window specification to be included among the block parameters.
  - (G,H,C,CT,N) `windowopt` the optional parameter for window type:
    * The following `windowtype`s do not require the `windowopt` attribute: `rectangle`, `triangle`, `cosine`, `hanning`, `hamming`, `blackman`, `flattop`, `fof`.
    * The following `windowtype`s do require the `windowopt` attribute: `hamgen`, `gauss`, `exponential`.
      The meaning of the optional parameter varies according to the window type. See the reference manual for more info.
  - (H) `f0Min` with a double value: minimum frequency (in Hz) from which the fundamental frequency of the harmonic atoms is searched. Defaults to the first non-null FFT frequency.
  - (H) `f0Max` with a double value: maximum frequency (in Hz) at which the fundamental frequency of the harmonic atoms is searched. Defaults to the Nyquist frequency of the considered signal.
  - (H) `numPartials` with an unsigned int value: number of partials considered when tracking the harmonic atoms.
  - (C) `numFitPoints` (EXPERIMENTAL) with an unsigned int value: number of polynomial fitting points considered for the chirp optimization algorithm. Defaults to 1.
  - (C) `numIter` (EXPERIMENTAL) with an unsigned int value: number of iterations considered for the chirp optimization algorithm. Defaults to 1.

– (A) `tableFileName` with a string: filename of the table containing the waveforms (ex: —/udd/toto/table.bin—). Note that there is no ” around the string.

Note that the `dirac` blocks don't need any parameter (they just match signal samples).

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>        <= The optional XML
                                                      declaration line.
<dict>                                             <= The dictionary opening tag.
<libVersion>0.2</libVersion>                       <= The optional library version.

        <blockproperties name="GAUSS-WINDOW">      <= A new block properties.
                <param name="windowtype" value="gauss"/>
                <param name="windowopt" value="0.02"/> <= Gauss windows need
                                                          a parameter.
        </blockproperties>
        <blockproperties name="HAMMING-WINDOW">    <= Another block properties.
                <param name="windowtype" value="hamming"/>
                <param name="windowopt" value="0"/> <= The hamming window
                                                       ignores the opt="0"
                                                       attribute.
        </blockproperties>

        <block uses="GAUSS-WINDOW">                 <= A new block using
                                                       GAUSS-WINDOW block properties.
                <param name="type" value="gabor"/>
                <param name="windowLen" value="32"/>
                <param name="windowShift" value="32"/>
                <param name="fftSize" value="32"/>
        </block>

        <blockproperties name="GAUSS-WINDOW-BIS" refines="GAUSS-WINDOW">
                <param name="windowopt" value="0.02"/> <= A new block properties overriding
                                                          the GAUSS-WINDOW block properties.
        </blockproperties>

        <block uses="GAUSS-WINDOW-BIS">             <= A new block using
                                                       GAUSS-WINDOW-BIS block properties.
                <param name="type" value="gabor"/>
                <param name="windowLen" value="32"/>
                <param name="windowRate" value="0.25"/>   <= This is equivalent to
                                                       setting windowShift=16.
                <param name="fftSize" value="64"/>
        </block>
        <blockproperties name="WINDOW-SHIFT">           <= A new block properties.
                <param name="windowShift" value="32"/>  <= A value for the
```

```
                                                    window shift.
        </blockproperties>

        <block uses="WINDOW-SHIFT">
                <param name="windowLen" value="128"/>
<!--            <param name="windowShift" value="32"/>     <= These lines are
                <param name="fftSize" value="128"/>   -->  commented out.
                                                           This block will use
                                                           the default
                                                           windowShift from
                                                           WINDOW-SHIFT, and will default
                                                           fftSize to windowLen.
        </block>
        <block uses="GAUSS-WINDOW">
                <param name="type" value="gabor"/>
                <varparam name="fftSize">                  <= A variable parameter list.
                        <var>64</var>                         a list of block for each value
                        <var>32</var>                         of the variable parameter list.
                        <var>16</var>
                        <var>8</var>
                </varparam>
                <param name="windowLen" value="8"/>
                <varparam name="windowShift">              <= Another variable parameter list.
                        <var>64</var>                         a list of block for each value
                        <var>32</var>                         of the variable parameter list,
                        <var>16</var>                         combined with the list of block
                </varparam>                                   defined above.
        </block>
        <block uses="GAUSS-WINDOW-BIS">
                <param name="type" value="harmonic"/>     <= A harmonic block.
                <param name="fftSize" value="256"/>
                <param name="windowLen" value="256"/>
                <param name="windowShift" value="128"/>
                <param name="f0Min" value="340"/>
                <param name="f0Max" value="1000"/>
                <param name="numPartials" value="5"/>
        </block>
        <block uses="GAUSS-WINDOW">
                <param name="type" value="chirp"/>        <= A chirp block.
                <param name="fftSize" value="1024"/>
                <param name="windowLen" value="1024"/>
                <param name="windowShift" value="128"/>
                <param name="numFitPoints" value="2"/>
                <param name="numIter" value="1"/>
        </block>
        <block>
                <param name="type" value="anywave"/>        <= An anywave block.
```

```
                <param name="tableFileName" value="/udd/toto/table.xml"/>
                <param name="windowShift" value="1"/>
        </block>
        <block>
                <param name="type" value="dirac"/>         <= A dirac block.
        </block>
        <block>
                <param name="type" value="constant"/>      <= A constant block.
                <param name="windowLen" value="512"/>
                <param name="windowShift" value="32"/>
        </block>
        <block>
                <param name="type" value="nyquist"/>      <= A nyquist block.
                <param name="windowLen" value="512"/>
                <param name="windowShift" value="32"/>
        </block>
</dict>
This is that great dictionary I used for obtaining        <= This text is ignored
these wonderful experimental results thanks to the           by the parser.
Matching Pursuit Library ...
```

Anywave table

Waveforms need have been loaded before using anywave atoms. That's the difference between parametric atoms, such as Dirac, Gabor, ... and anywave atoms. Therefore, a different syntax has to be employed. In the dictionary file, the anywave block points to a "anywave table definition file" (/udd/toto/table.bin in the example). This file has a XML syntax, to give all the parameters of the waveforms, and points to a binary "anywave table data file", that gives the data (/udd/toto/table_data.bin in the example).

Note that one table corresponds to one atom length, and one number of channels. To use several atom lengths, one must create several anywave tables, and define several "anywave" blocks.

The "anywave table definition file" is structured as follows:

- `<?xml version="1.0" encoding="ISO-8859-1"?>` [optional]: the XML declaration line. It is ignored by the parser.
- `<table>`List of table parameters`</table>`: the opening and closing tags for the table. Anything coming after the `</table>` tag will be ignored by the parser.
- `<libVersion>`blah`</libVersion>` [optional]: the library version declaration. This is provided for backward compatibility if we ever change the dictionary syntax. When absent, the library version is taken to be the current one.
- `<par type="NAME" value="VALUE"/>` : a table parameter. The `NAME` and corresponding `VALUE` value in string. The parameter value is set to the corresponding type when the table parameter file is parsed. The `NAME` and corresponding `VALUE`s types are given below:

- `numChans` with an unsigned short int value: the number of channels of the atoms.
- `filterLen` with an unsigned long int value: the length of the anywave waveforms in the table.
- `numFilters` with an unsigned long int value: the number of anywave waveforms in the table.
- `data` with a string: name of the file containing the waveforms data (ex: —/udd/toto/table_data.bin—). Note that there is no " around the string.

The "anywave table data file" is binary, and built as follows: all the waveforms are written in double type, one after the other, and for each, one channel after the other. The following scheme is for two channels and four waveforms:

—w1c1—w1c2—w2c1—w2c2—w3c1—w3c2—w4c1—w4c2—

Example of an anywave table

```
<?xml version="1.0" encoding="ISO-8859-1"?>        <= The optional XML
                                                      declaration line.
<table>                                           <= The table opening tag
                                                     (only one table per file
        <libVersion>0.4beta</libVersion>          <= The optional library version.
        <param name="numChans" value="1"/>
        <param name="filterLen" value="64"/>
        <param name="numFilters" value="20"/>
        <param name="data" value="/udd/toto/table_data_001.bin"/>
</table>                                          <= The table closing tag.
```

MDCT/MDST/MCLT

Definition The Modified Discrete Cosine Transform (MDCT) and the Modified Discrete Sine Transform (MDST) are two orthogonal transforms based on local cosine functions. The Modulated Complex Lapped Transform (MCLT) is the complex extension such as $MCLT = MDCT + i * MDST$. The atoms corresponding to the MCLT of a signal of length $N = PL$ and a window length of $2L$, are defined as:

$$x_{p,k}(n) = w(n - pL)exp\left[i\frac{\pi}{L}\left((n - pL) + \frac{1}{2} + \frac{L}{2}\right)\left(k + \frac{1}{2}\right)\right] \quad (1)$$

with $n = 0, .., N-1$, $k = 0, .., L-1$ and $p = 0, .., P-1$. w is a window which is complementary in energy i.e. it verifies $w^2(n) + w^2(n + L) = 1, n = 0, .., L - 1$.

The atoms corresponding to the MDCT are defined as the real part of the previous formula and the atoms corresponding to the MDST the imaginary part of the previous formula.

We also define a generalized MDCT/MDST/MCLT where the window shift, the window shape and the fft size are not constraint by the orthogonality property. The corresponding atoms for a signal of length $N = PS$, a window length of $2L$, a window shift of S, a fft size of $Nfft$ and a window w, are defined as:

$$x_{p,k}(n) = w(n - pS)exp\left[i\frac{2\pi}{Nfft}\left((n - pS) + \frac{1}{2} + \frac{L}{2}\right)\left(k + \frac{1}{2}\right)\right] \text{ if } Nfft = 2L \quad (2)$$

$$x_{p,k}(n) = w(n - pS)exp\left[i\frac{2\pi}{Nfft}\left((n - pS) + \frac{1}{2} + \frac{L}{2}\right)(k)\right] \text{ if } Nfft = 2mL \quad (3)$$

with $n = 0, .., N-1$, $k = 0, .., Nfft/2-1$, $p = 0, .., P-1$ and m is even with $m >= 2$.

Valid tags

- `<?xml version="1.0" encoding="ISO-8859-1"?>` [optional]: the XML declaration line. It is ignored by the parser.
- `<libVersion>`blah`</libVersion>` [optional]: the library version declaration. This is provided for backward compatibility if we ever change the dictionary syntax. When absent, the library version is taken to be the current one.
- `<dict>`List of blocks`</dict>`: the opening and closing tags for the dictionary. Anything coming after the `</dict>` tag will be ignored by the parser.

  The list of parameters for relevant block types (G==gabor, H==harmonic, C==Chirp, A==Anywave, CT==constant, N==nyquist) are given below:

  - `windowLen` with an unsigned long int value: the atom length (i.e., the window length). It has to be even.
  - `windowShift` with an unsigned long int value: the atom shift (i.e., the window shift).
  - `windowRate` with a double value between 0.0 and 1.0: an alternate way to specify the window shift, as a proportion of the windowLen. If both are present, `windowShift` has precedence over `windowRate`.
  - `fftSize` with an unsigned long value: the frequency resolution in terms of FFT size. It has to be equal to `windowLen` or a multiple of 2*`windowLen`.
  - `blockOffset` with an unsigned long value: the block Offset, i.e. the position of the first frame. If absent, defaults to 0.
  - `windowtype` the window specification to be included among the block parameters.
  - `windowopt` the optional parameter for window type:
    - The following `windowtype`s do not require the `opt=""` attribute:
      `rectangle`, `triangle`, `cosine`, `hanning`, `hamming`, `blackman`, `flattop`, `fof`.
    - The following `windowtype`s do require the `opt=""` attribute:
      `hamgen`, `gauss`, `exponential`, `kbd`.
      The meaning of the optional parameter varies according to the window type. See the reference manual for more info.

A default MDCT/MDST/MCLT block is defined using 2 parameters, the window length and the window type. The window type must be `rectangle`, `cosine` or `kbd`. An example of a MDCT dictionnary is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<dict>
<libVersion>0.2</libVersion>
     <block>
          <param name="type" value="mdct"/>
          <param name="windowLen" value="2048"/>
          <param name="windowtype" value="kbd"/>
```

```
          <param name="windowopt" value="5"/>
     </block>
</dict>
```

An example of a generalized MCLT dictionnary is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<dict>
<libVersion>0.2</libVersion>
     <block>
          <param name="type" value="mdct"/>
          <param name="windowLen" value="2048"/>
          <param name="fftSize" value="4096"/>
          <param name="windowtype" value="cosine"/>
          <param name="windowopt" value="0"/>
     </block>
</dict>
```

2.5 mpf: filtering of the book files

Usage:

```
mpf --PROPERTY1=[min:max] ... --PROPERTY_N=[min:max] (bookIn.bin|-) ...
    [bookYes.bin|-] [bookNo.bin]
```

Synopsis:

Filters the atoms contained in bookIn.bin (or stdin), stores those which satisfy the indicated properties in bookYes.bin (or stdout) and the others in bookNo.bin.

Mandatory arguments:

```
(bookIn.bin|-)      A book of input atoms, or stdin.
```

Optional arguments:

```
(bookYes.bin|-)     A file (or stdout) to store the book of atoms which
                    satisfy the indicates properties.
bookNo.bin          A file to store a book of atoms which do not satisfy
                    the indicated properties.
If no output files are given, the atoms are just counted and their number
is reported in stderr.

One or more of the following switches:
--index=[min:max]    / -i [min:max] : keep the atoms ordered from min to max
                                      in the book
--length=[min:max]   / -l [min:max] : keep a specific range of atom lengths
                                      (in number of samples)
--Length=[min:max]   / -L [min:max] : keep a specific range of atom lengths
                                      (in seconds)
--position=[min:max] / -p [min:max] : keep a specific range of atom positions
                                      (in number of samples)
--Position=[min:max] / -P [min:max] : keep a specific range of atom positions
                                      (in seconds)
--freq=[min:max]     / -f [min:max] : keep a specific frequency range (in
                                      normalized values between 0 and 0.5)
--Freq=[min:max]     / -F [min:max] : keep a specific frequency range (in Hz)
--amp=[min:max]      / -a [min:max] : keep a specific range of amplitudes
--chirp=[min:max]    / -c [min:max] : keep a specific range of chirp factors

The intervals can exclude the min or max value by using reverted braces,
e.g. ]min:max] will exclude the min value.
The intervals can be negated with prepending the '^' character,
e.g. ^[min:max].

The atom's type can be tested with:
```

```
 --type=gabor|harmonic|dirac|anywave / -t gabor|harmonic|dirac|anywave
 (The chirp type is not provided: a chirp atom is a gabor atom with
  a non-null chirp rate.)

Other optional arguments are:
 -C<FILE>, -config-file=<FILE>  Use the specified configuration file,
                                otherwise the MPTK_CONFIG_FILENAME environment
                                variable will be used to find the configuration
                                file and set up the MPTK environment
 -q, --quiet          No text output.
 -v, --verbose        Verbose.
 -V, --version        Output the version and exit.
 -h, --help           This help.
```

Example:

Take all the atoms with a frequency lower than 50Hz and higher than 1000Hz among the first 100 atoms of bookIn.bin, store them in bookYes.bin and store all the others in bookNo.bin:

```
mpf --index=[0:100[ --Freq=^[50:1000] bookIn.bin bookYes.bin bookNo.bin
```

Note:

Only one instance of each property is allowed. More complicated domains can be elaborated using pipes.

2.6 mpcat: concatenation of book files

Usage:

```
mpcat (book1.bin|-) (book2.bin|-) ... (bookN.bin|-) (bookOut.bin|-)
```

Synopsis:

Concatenates the N books book1.bin...bookN.bin into the book file bookOut.bin.

Mandatory arguments:

```
(bookN.bin|-)        At least 2 books (or stdin) to concatenate.
(bookOut.bin|-)      A book where to store the concatenated books,
                     or stdout
```

Optional arguments:

```
-C<FILE>, -config-file=<FILE>  Use the specified configuration file,
                               otherwise the MPTK_CONFIG_FILENAME environment
                               variable will be used to find the configuration
                               file and set up the MPTK environment.
-f, --force          Force the overwriting of bookOut.bin.

-q, --quiet          No text output.
-v, --verbose        Verbose.
-V, --version        Output the version and exit.
-h, --help           This help.
```

2.7 mpr: signal reconstruction

Usage:

```
mpr (bookFILE.bin|-) (reconsFILE.wav|-) [residualFILE.wav]
```

Synopsis:

Rebuilds a signal reconsFile.wav from the atoms contained in the book file bookFile.bin. An optional residual can be added.

Mandatory arguments:

```
(bookFILE.bin|-)     A book of atoms, or stdin.
(reconsFILE.wav|-)   A file to store the rebuilt signal, or stdout
                     (in WAV format).
```

Optional arguments:

```
-C<FILE>, -config-file=<FILE>  Use the specified configuration file,
                               otherwise the MPTK_CONFIG_FILENAME environment
                               variable will be used to find the configuration
                               file and set up the MPTK environment.
residualFILE.wav     A residual signal that was obtained from
                     a Matching Pursuit decomposition.

 -q, --quiet          No text output.
 -v, --verbose        Verbose.
 -V, --version        Output the version and exit.
 -h, --help           This help.
```

2.8 mpd_demix: blind source separation with a known mixing matrix

Usage:

```
mpd_demix [options] -D dictFILE.txt -M matrix.txt (-n N|-s SNR) ...
          (sndFILE.wav|-) (bookFILE) [residualFILE.wav]
```

Synopsis:

Performs Blind Source Separation on signal sndFILE.wav with dictionary dictFile.txt and with the known mixer matrix mixFILE.txt. The result is stored in as many books as estimated sources (plus an optional residual signal), after N iterations or after reaching the signal-to-residual ratio SNR.

Mandatory arguments:

```
  -M<FILE>, --mix-matrix=<FILE>  Read the mixer matrix from text file FILE.
                                 The first line of the file should indicate
                                 the number of rows and the number of columns,
                                 and the following lines should give space
                                 separated values, with a line break after
                                 each row.
                                 EXAMPLE:
                                  2 3
                                  0.92  0.38  0.71
                                  0.71  0.77  1.85

  -n<N>, --num-iter=<N>|--num-atoms=<N>    Stop after N iterations.
AND/OR -s<SNR>, --snr=<SNR>                Stop when the SNR value SNR
                                           is reached.
                                           If both options are used together,
                                           the algorithm stops as soon as
                                           either one is reached.

  (sndFILE.wav|-)                          The signal to analyze or stdin
                                           (in WAV format).
  (bookFILE)                               The base name of the files to store
                                           the books of atoms_n corresponding
                                           to the N estimated sources. These
                                           N books will be named bookFILE_n.bin,
                                           n=[0,...,N-1].
```

Optional arguments:

```
  -C<FILE>, -config-file=<FILE>    Use the specified configuration file,
                                   otherwise the MPTK_CONFIG_FILENAME environment
                                   variable will be used to find the configuration
```

```
                                    file and set up the MPTK environment.
-D<FILE>, --dictionary=<FILE>       Read the dictionary from text file FILE.
                                    If no dictionary is given, a default
                                    dictionary is used.
                                    (Use -v to see the structure of the default
                                    dictionary reported in the verbose information.)
-E<FILE>, --energy-decay=<FILE>     Save the energy decay as doubles to file FILE.
-Q<FILE>, --src-sequence=<FILE>     Save the source sequence as unsigned short ints
                                    to file FILE.
-R<N>,    --report-hit=<N>          Report some progress info (in stderr) every
                                    N iterations.
-S<N>,    --save-hit=<N>            Save the output files every N iterations.
-T<N>,    --snr-hit=<N>             Test the SNR every N iterations only
                                    (instead of each iteration).

residualFILE.wav                    The residual signal after subtraction
                                    of the atoms.

-q, --quiet                         No text output.
-v, --verbose                       Verbose.
-V, --version                       Output the version and exit.
-h, --help                          This help.
```

2.9 mpview: generation of a time-frequency map from a book

Usage:

```
mpview [options] (bookFILE.bin|-) tfmapFILE.flt
```

Synopsis:

Makes a time-frequency pixmap fill it with the time-frequency representation of the atoms contained in the book file bookFile.bin and write it to the file tfmapFILE.flt as a raw sequence of floats. The pixmap size is 640x480 pixels unless option –size is used.

Mandatory arguments:

```
Mandatory arguments:
    (bookFILE.bin|-)     A book of atoms, or stdin.
    tfmapFILE.flt        The file where to write the pixmap in float.
```

Optional arguments:

```
    -C<FILE>, -config-file=<FILE>    Use the specified configuration file,
                                     otherwise the MPTK_CONFIG_FILENAME environment
                                     variable will be used to find the configuration
                                     file and set up the MPTK environment.
    -s, --size=<numCols>x<numRows> : change the size of the pixmap.
    -q, --quiet          No text output.
    -v, --verbose        Verbose.
    -V, --version        Output the version and exit.
    -h, --help           This help.
```

Synopsis:

Displays a book in a pixmap of numCols x numRows pixels Returns nonzero in case of failure, zero otherwise

3 The GUI

A wxWidget-based Graphical User Interface is delivered as the `MptkGuiApp` binary, in the `./src/gui/` directory. It is not compiled/installed by default: use `--enable-gui` with the `configure` script if you want to use it. The initial version of this GUI has been developed in the framework of an IFSIC/DIIC3 student project by: Nicolas Bonnet, Benjamin Boutier, Vincent Chapon and Sylvestre Cozic. (Thanks!)

This GUI takes up the functionality of `mpd` and `mpr` (no atom filtering implemented), and adds visualization/audio playback capabilities for the books and signals. Globally, it offers a friendlier control of the decomposition process. In its current state, it remains bug-prone and poorly documented, but it can help vizualizing and understanding the Matching Pursuit decomposition process in a nice way.

4 Extra utilities: visualization of MP books under Matlab

Some Matlab utilities are bundled with the distribution to help loading and visualizing books under Matlab. They can be found in the ./extras/matlab/ directory of the source tree. They are not automatically installed by the configure system: you should copy them manually to a location accessible to your MATLABPATH.

The included files are:

- bookread.m: to load a binary book into Matlab;
- bookplot.m: to plot a book in a spectrogram-like fashion;
- bookover.m: to overlay a book plot on a STFT spectrogram.
- dictread.m: under development. Don't use.

Each of these functions is equipped with some help info, e.g. to get help about bookplot use:

```
>> help bookplot
```

under Matlab.

A Development conventions

A.1 Typographic conventions

Throughout the source code, the following typographic conventions are preferably adopted:

- **myVar** for variables;
- **my_function** for functions;
- **MP_MY_CONST** for constants (#define) and for global variables;
- **MP_My_Type_t** for types;
- **MP_My_Class_c** for classes.

A.2 Memory allocation conventions

The arrays are allocated using malloc (NOT new[]) and freed using free (NOT delete[]). Conversely, the objects are created and deleted using new and delete.

A.3 Source code locations

From the source tree, the doc sources are stored in doc/, and the code sources in src/. Then :

- the bulk of the Matching Pursuit library is stored in src/libmptk/. Specifically, the implementation of blocks and atoms related to specific time-frequency transforms should be located in src/plugin/. As a matter of fact, you should not have to modify other parts of the code when adding new classes of blocks and atoms. Think of it as a sort of plugin logic;
- a standalone library (in C) to compute standard signal processing windows is stored in src/libdsp_windows/;
- the applications are stored in src/utils/;
- a test suite is stored in src/tests/.

For the implementation of each class, we use one .h header file to declare the class, plus a corresponding .cpp source with the same base name to implement its methods. During a build, the .h headers are concatenated in one global mptk.h header which should be the one included in all the individual .cpp files.

A.4 Building and portability

Building system – We use the CMake build system to automatically adapt the building process to your particular flavor of Unix, MAC OS X or Windows Win32. Therefore, every system call should be performed through the "mp_system.h"header file located at the top of the source tree.

Binary format and byte order – The byte-order of any binary output is forced to be the LITTLE_ENDIAN order, native on Intel (⇒ most PCs and the new Macs) and VAX[2], swapped on PowerPC (⇒ Macintosh PPC), SPARC (⇒ SUN workstations) and Motorola.

A.5 Installed files

- Static libraries get installed in `EPREFIX/lib`:
  - `libdsp_windows`: a standalone library to compute standard signal processing windows.
  - `libtinyxml`: a standalone library to parse XML dictionary structure file
- Dynamic Link Libraries (DLL) get installed in `EPREFIX/lib`:
  - `libmptk`: the MPT core library
  - `"ATOMNAME"Atom`: the `"ATOMNAME"` atom plugin, which is dynamically loaded when MPTK environment is loaded.
  - `"BLOCKNAME"Atom`: the `"BLOCKNAME"` block plugin, which is dynamically loaded when MPTK environment is loaded.
- MPTK environment configuration file: `path.xml` gets installed in `PREFIX/bin`.
- Headers: `mpth.h` gets installed in `PREFIX/include`.
- Binaries: `mpd`, `mpd_demix`, `mpf`, `mpr`, `mpcat`, `mpview` and optionally `MptkGuiApp` get installed in `EPREFIX/bin`. The binaries from `./src/test/` do not get installed.
- Documentation: the documentation is delivered in `./doc`, both in built form and source form, but does not get installed. Copy it manually to your favorite location.
- Extras: matlab interface files are delivered in `./extras/matlab`, but do not get installed. Copy them manually to a location accessible from your MATLABPATH.

[2]Dude, if you have a VAX alive, we would sure like to know if MPTK runs on it.

B Development tricks and notes

B.1 Class interfacing and code factorization

To allow the dictionary to manage every class of blocks (i.e., every different time-frequency transform) in the same way, every specific block class inherits from a generic block class which implements some mandatory methods. Similarly, all the atoms inherit from a generic atom class, so that they can perform a certain number of standard operations and so that they can be stored in books. Please refer to the reference manual and code headers for more details.

B.2 Speed tricks

Our general philosophy is to avoid as much as possible to compute the same operation twice. For example, in the blocks, the update of the time-frequency transforms is executed only along the part of the signal that were modified by subtracting an atom at the previous step.

Tree for the max search – The outcome of the time-frequency transforms is stored as a single dimension vector. After each update of the time-frequency transforms, the location of the maximum correlation can be anywhere along the vector. To avoid browsing the whole length of it each time we perform a max correlation search, we use a tree structure which keeps track of local maxima (figure 2).

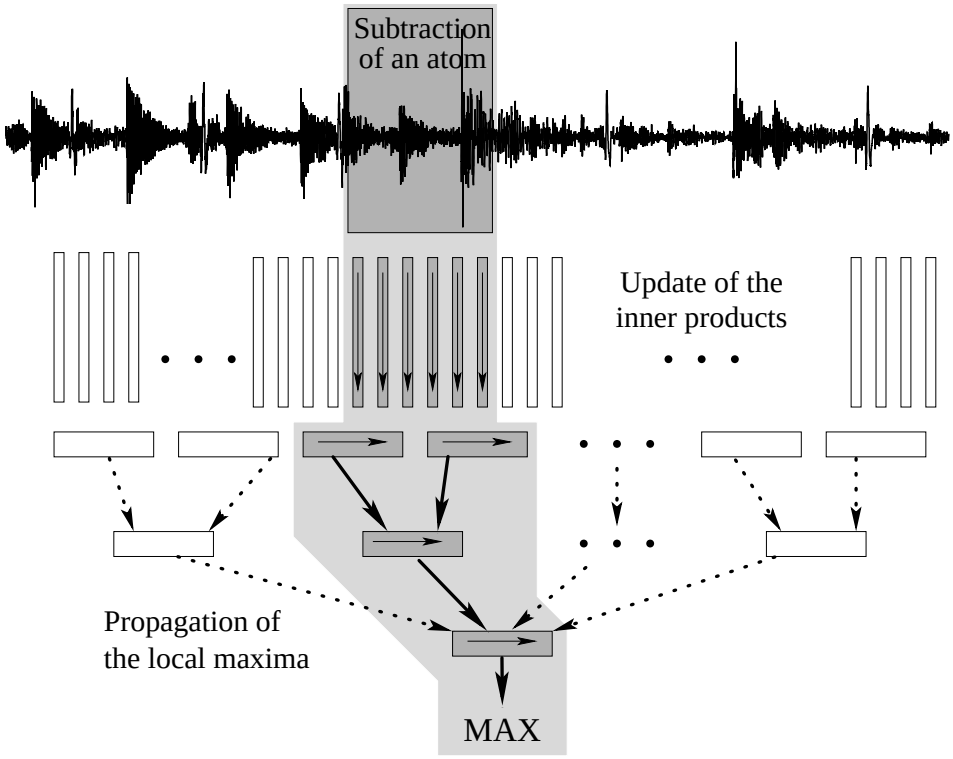

Figure 2: Use of a tree to speed up the max search: at each iteration, only the grey-shaded parts, modified by the previous atom subtraction, are browsed.

Tabulations – The signal processing windows are tabulated in a window server (called by the blocks and the atoms) which has global scope over the whole code. The atom cross-correlations are computed and tabulated at block creation time.

B.3 The parser for XML dictionaries: TinyXML

The parsing of the XML dictionary is implemented using tinyXML. TinyXML is a simple, small, C++ XML parser that can be easily integrating into other programs. TinyXML has evolved from community feedback and become the work of many contributors. It is a simple, stable, basic XML parser used by many open source and commercial products.

B.4 Experimental hacks and limitations – !!READ THIS!!

Harmonic block – In its current implementation, the harmonic block allows for an increase of the signal energy for certain "ill choosen" values of f0Min (the lower authorized base frequency).

As a matter of fact, for low f0 values, the various Gabor atoms making the harmonic components are not strictly orthogonal, although they are treated as such when removing the whole harmonic atom from the analyzed signal. Various solutions are possible, such as subtracting the harmonic atom in a truly orthogonal way (implying the computation of the Gram-Schmidt matrix of the Gabor components), removing the Gabor components individually in an iterative, stepwise orthogonal way, or limiting f0Min on the basis of tabulated values (related to the window types and dimensions). None of these solutions has been implemented yet.

Chirp block – The chirp optimization method is well understood at the theoretical level, but its practical behaviour needs to be studied in more detail. It has been observed that high chirp rates may lead to wrong correlations and an increase of the signal energy. To avoid pathological cases, three noteworthy tricks have been used:

- after each iteration of chirp rate estimation, it is verified that the inner product (IP) with the signal is augmented. If not, the last valid "IP increasing" chirp value is returned;
- at the relocation step in the chirp estimation process, the atom can not jump further than 10 FFT bins from its current location (lines using `MP_FREQ_RELOC_RANGE`);
- the chirp rate estimate is arbitrarily bounded by a value of $0.5 \cdot 10^{-5}$ (code labelled `/* BORK */` in `chirp_block.cpp`). This value should be made a block parameter and correlation behaviour at high chirp rates should be better explored.

In addition, the effect of the number of iterations and number of fit points deserves a deeper study. So far, 1 iteration with numFitPoints=1 seems to give acceptable results, so the usefulness of an iterative process is questionable.

C References

Algorithmic aspects For more information about the algorithmic aspects of the Matching Pursuit algorithm, see:
[Zhang, 1993], [Mallat and Zhang, 1993], [Pati et al., 1993], [Davis, 1994], [Bergeaud, 1995], [Bergeaud and Mallat, 1996], [Jaggi et al., 1998], [Gribonval et al., 1996a], [Gribonval et al., 1996b], [Davis et al., 1997], [Goodwin and Vetterli, 1999], [Gribonval, 1999], [Gribonval, 2001b], [Gribonval, 2002], [Gribonval and Bacry, 2003], [Gribonval, 2003], [Krstulović and Gribonval, 2006].

Theoretic aspects For more information about the theoretic aspects of the Matching Pursuit algorithm, see:
[Gribonval, 2001a], [Gribonval and Nielsen, 2001], [Gribonval, 2001c], [Gribonval and Nielsen, 2003b], [Gribonval and Nielsen, 2003c], [Gribonval and Vandergheynst, 2004], [Gribonval and Nielsen, 2003a], [Gribonval et al., 2004].

Experimental results and applications Examples of experimental and applicative results obtained with MPTK can be found in:
[Krstulović et al., 2005], [Lesage et al., 2006].

References

[Bergeaud, 1995] Bergeaud, F. (1995). *Représentations adaptatives d'images numériques, Matching Pursuit*. PhD thesis, Ecole Centrale Paris.

[Bergeaud and Mallat, 1996] Bergeaud, F. and Mallat, S. (1996). Matching pursuit : Adaptive representations of images and sounds. *Computational and Applied Mathematics*, 15(2):97–109.

[Davis, 1994] Davis, G. (1994). *Adaptive Nonlinear Approximations*. PhD thesis, New York University.

[Davis et al., 1997] Davis, G., Mallat, S., and Avellaneda, M. (1997). Adaptive greedy approximations. *Constr. Approx.*, 13(1):57–98.

[Goodwin and Vetterli, 1999] Goodwin, M. and Vetterli, M. (1999). Matching pursuit and atomic signal models based on recursive filter banks. *IEEE Trans. on Signal Proc.*, 47(7):1890–1902.

[Gribonval, 1999] Gribonval, R. (1999). *Approximations non-linéaires pour l'analyse de signaux sonores*. PhD thesis, Université Paris IX Dauphine.

[Gribonval, 2001a] Gribonval, R. (2001a). A counter-example to the general convergence of partially greedy algorithms. *Journal of Adaptive Theory*, 111:128–138. doi:10.1006/jath.2001.3556.

[Gribonval, 2001b] Gribonval, R. (2001b). Fast matching pursuit with a multiscale dictionary of Gaussian chirps. *IEEE Trans. Sig. Proc.*, 49(5):994–1001.

[Gribonval, 2001c] Gribonval, R. (2001c). Partially greedy algorithms. In Kopotun, K., Lyche, T., and Neamtu, M., editors, *Trends in Approximation Theory*, pages 143–148, Nashville, TN. Vanderbilt University Press.

[Gribonval, 2002] Gribonval, R. (2002). Sparse decomposition of stereo signals with matching pursuit and application to blind separation of more than two sources from a stereo mixture. In *Proc. Int. Conf. Acoust. Speech Signal Process. (ICASSP'02)*, Orlando, Florida.

[Gribonval, 2003] Gribonval, R. (2003). Piecewise linear source separation. In Unser, M., Aldroubi, A., and Laine, A., editors, *Proc. SPIE '03*, volume 5207 Wavelets: Applications in Signal and Image Processing X, pages 297–310, San Diego, CA.

[Gribonval and Bacry, 2003] Gribonval, R. and Bacry, E. (2003). Harmonic decomposition of audio signals with matching pursuit. *IEEE Trans. Sig. Proc.*, 51(1):101–111.

[Gribonval et al., 1996a] Gribonval, R., Bacry, E., Mallat, S., Depalle, P., and Rodet, X. (1996a). Analysis of sound signals with high resolution matching pursuit. In *Proc. TFTS'96*, pages 125–128, Paris, France.

[Gribonval et al., 1996b] Gribonval, R., Depalle, P., Rodet, X., Bacry, E., and Mallat, S. (1996b). Sound signals decomposition using a high resolution matching pursuit. In *Proc. ICMC'96*, pages 293–296.

[Gribonval et al., 2004] Gribonval, R., Figueras i Ventura, R. M., and Vandergheynst, P. (2004). A simple test to check the optimality of sparse signal approximations. Technical report, IRISA, Rennes, France. submitted to EURASIP Signal Processing J.

[Gribonval and Nielsen, 2001] Gribonval, R. and Nielsen, M. (2001). Approximate weak greedy algorithms. *Advances in Computational Mathematics*, 14(4):361–378.

[Gribonval and Nielsen, 2003a] Gribonval, R. and Nielsen, M. (2003a). Highly sparse representations from dictionaries are unique and independent of the sparseness measure. Technical Report R-2003-16, Dept of Math. Sciences, Aalborg University.

[Gribonval and Nielsen, 2003b] Gribonval, R. and Nielsen, M. (2003b). Sparse decompositions in "incoherent" dictionaries. In *Proc. IEEE Intl. Conf. Image Proc. (ICIP'03)*, Barcelona, Spain.

[Gribonval and Nielsen, 2003c] Gribonval, R. and Nielsen, M. (2003c). Sparse decompositions in unions of bases. *IEEE Trans. Inform. Theory*, 49(12):3320–3325.

[Gribonval and Vandergheynst, 2004] Gribonval, R. and Vandergheynst, P. (2004). On the exponential convergence of Matching Pursuits in quasi-incoherent dictionaries. Technical Report PI-1619, IRISA, Rennes, France. submitted to IEEE Trans. Inf. Th.

[Jaggi et al., 1998] Jaggi, S., Carl, W., Mallat, S., and Willsky, A. (1998). High resolution pursuit for feature extraction. *Applied Computational Harmonic Analysis*, 5(4):428–449.

[Krstulović and Gribonval, 2006] Krstulović, S. and Gribonval, R. (2006). Matching pursuit made tractable. In *Proc.ICASSP 2006*.

[Krstulović et al., 2005] Krstulović, S., Gribonval, R., Leveau, P., and Daudet, L. (2005). A comparison of two extensions of the matching pursuit algorithm for the harmonic decomposition of sounds. In *Proc.WASPAA'05*.

[Lesage et al., 2006] Lesage, S., Krstulović, S., and Gribonval, R. (2006). Underdetermined source separation: comparison of two approaches based on sparse decompositions. In *Proc.ICA'06*.

[Mallat and Zhang, 1993] Mallat, S. and Zhang, Z. (1993). Matching pursuit with time-frequency dictionaries. *IEEE Trans. Sig. Proc.*, 41(12):3397–3415.

[Pati et al., 1993] Pati, Y., Rezaiifar, R., and Krishnaprasad, P. (1993). Orthonormal matching pursuit : recursive function approximation with applications to wavelet decomposition. In *Proceedings of the* 27^{th} *Annual Asilomar Conf. on Signals, Systems and Computers*.

[Zhang, 1993] Zhang, Z. (1993). *Matching Pursuit*. PhD thesis, New York University.