Speech Signal Processing Toolkit User and Programmer Manual
SPro 3.3.1

Guillaume Gravier

Last update, Jan. 03

Contents

1 Warning

For distribution reasons, the FFT algorithm has been changed in SPro 3.3. Some minor changes have also been made in the code. **Those changes result in slightly different features between SPro 3.3 and greater and the previous versions**. Those differences are minor but it may slightly change the results in any automatic recognition system trained with some features computed from previous versions of SPro.

Some option switches have also changed so that the same letters mean the same thing accross all tools. Please pay attention to your old scripts when upgrading to the current version. However, long option names have not been changed so it may be a good idea in the future considering to use the long option names...

2 Introduction

2.1 What is SPro?

This user manual briefly describes the main features of the speech processing toolkit SPro 3.3.1. Basically, SPro was designed for variable spectral resolution analysis of the speech signal (see section 2.2 for more details on this topic). It provides a library for the standard (frame-level) analysis techniques and for the signal and feature I/O functions. Classical feature extraction algorithms are also provided as runtimes, developped using the library.

In section 3, the file formats used by SPro are described. In section 4, the basic feature extraction tools are described. Finally, an overview of the functions provided by the library is given in section 5.

2.2 About variable resolution spectral analysis

This technique is fully described in [1] where it is applied to speech for coding purposes. Attempts to use this technique in speech and speaker recognition are described in [2, 3].

Briefly, it consists in applying a bi-linear transformation of the frequency scale defined by

$$\overline{\omega} = \arctan \left| \frac{(1-\alpha^2)\sin\omega}{(1+\alpha^2)\cos\omega - 2\alpha} \right|$$

Spectral transformations for various values of α are plotted in figure 1 where the x-axis is the original frequency scale (ω) and the y-axis, the transformed frequency scale ($\overline{\omega}$).

3 File formats

3.1 Signal files

SPro currently supports three signal file formats: RAW, WAVE and SPHERE. The default format is the raw 16 bits/sample PCM format, that is, raw PCM

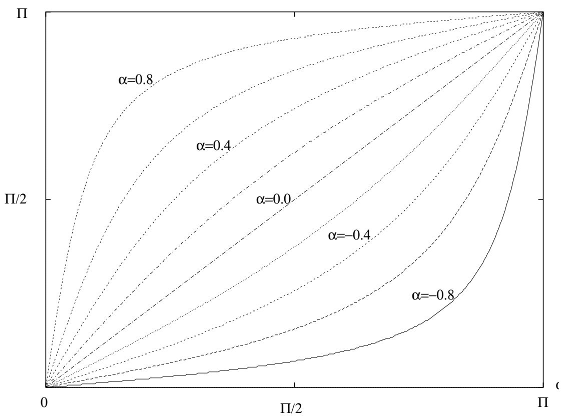

Figure 1: Bi-linear transformation of the frequency scale

data with 16 bits per samples. Note that the sample rate must be specified when using the raw PCM signal format. By default, the sample rate is assumed to be 8 kHz.SPro also supports WAVE files and can optionnally be linked to the SPHERE 2.6a library, in order to directly read SPHERE signal files[1]. In the two last cases, the sample rate is directly read form the signal file header. All the information in the NIST header are lost.

3.2 Feature files

A data file is the file that contains the feature vectors. It consists of a 16 bytes header followed by the concatenated feature vector. The header is defined by the four following fields:

- data kind (4 bytes), where data kind is an enumeration of the following: `OTHER`, `FBANK`, `FBCEPSTRA`, `LPCEPSTRA`, `LPCOEFF`, `PARCOR` and `LAR`
- basic feature vector dimension (4 bytes)
- number of feature vectors (4 bytes)
- qualifier (4 bytes)

The qualifier field is a bit field used to describe in more the details the content of the file. The bits have the following meanings:

1. log energy appended (E)
2. long term mean of the data subtracted (Z)
3. absolute log energy suppressed (this bit makes no sense without bit 4) (N)
4. Δ coefficients appended (D)

[1]SPHERE 2.6a is freely distributed by the American National Institute of Standards and Technologies (NIST) and can be found at URL *http://www.nist.gov/speech/tools.*

5. $\Delta\Delta$ coefficients appended (A)

The letter in parenthesis for each bit is used to refer to a given qualifier (*e.g.* in the `--convert` option of `scopy`). It must be noted that the second header field contains the basic feature vector dimension (*i.e.* the dimension of the static coefficients regardless of the energy) and not the real feature vector dimension which must be retrieved according to the qualifier field. The feature vectors are stored as *float* (4 bytes) with the static coefficients first, optionally followed by the log-energy, the Δ and $\Delta\Delta$ coefficients.

4 User guide

In this section, the runtimes are briefly described.

4.1 Common options

A short on-line description can always be obtained using the `--help` option. Every runtime also have a `--verbose` switch to run in verbose mode and a `--version` switch to print out the version number. The four runtimes which take a signal as input (`sfbank`, `sfbcep`, `slpc` and `slpcep`) shares some common options described below:

- `--sample-rate=n` – set sample rate of the input signal (default is 8 kHz).
- `--sphere` – indicates the input signal file is a SPHERE file.
- `--window=s` – set the analysis window type. `s` can be `none`, `haming` or `hanning` (default is `haming`).
- `--length=f` – set the analysis window length (default is 20.0 ms).
- `--decay=f` – set the analysis window decay (default is 10.0 ms).
- `--alpha=f` – set the spectral resolution factor α (see sec. 2.2).
- `--pre-emphasis=f` – set coefficient of the $1-kz^{-1}$ pre-emphasis filter (default is 0.95).
- `--energy` – output log-energy
- `--scale-energy=f` – scale log-energy according to $\ln e = (\ln e - \ln e_{\max})\, f + 1$.

In all the programs, either the input file or the output file can be omitted in which case input is made from `stdin` and output to `stdout`. In addition, waveform input from `stdin` can be explicitely specified using a dash '-' as the waveform input filename, thus making it possible to pipe the output of a command (*e.g.* an MPEG audio decoder[2] into the input of the SPro command.

[2]As it comes to this, check out the madplay decoder if you're interesting in processing MPEG audio streams (*http://www.mars.org/home/rob/proj/mpeg*)

4.2 sfbank

`Usage: sfbank [options] [ifile [ofile]]`

Filter-bank analysis of a speech signal. This program takes as input a signal and outputs a `FBANK` SPro data file. For each frame in the signal, a FFT is performed. The FFT size defaults to 512 points but can be specified using the `--fft-length=n` option. A triangular filter-bank is applied to the module of the FFT and the log-energy in each filter is returned. If the energy is less than 1.0, the returned value is 0.0. The central frequencies of the filters are equally spaced along the transformed frequency scale if the `--alpha=f` option is used (along the linear scale if $\alpha = 0.0$, which is the default) or along the MEL scale if the `--mel` option is used. Both options are exclusive. Band-limiting can be specified with the `--freq-min=f` and `--freq-max=f` options. In this case, filters are distributed in the range $[f_{\min}, f_{\max}]$. The number of filters can be set with the `--num-filter=n` option and defaults to 24.

4.3 sfbcep

`Usage: sfbcep [options] ifile ofile`

Filter-bank based cepstral analysis of a speech signal. This program takes as input a signal and outputs a `FBCEPSTRA` SPro data file. For each frame in the signal, a filter-bank analysis is performed as in `sfbank`. A discrete cosine transform is applied to the output of the filter-bank according to:

$$c_i = \frac{1}{\sqrt{2\pi}} \sum_{j=1}^{N} e_j \cos\left(\frac{i(j-0.5)\pi}{N}\right) \qquad i = 1, \ldots, P \ ,$$

where e_j denotes the output of the j'th filter. The `--num-ceps=n` option sets the order of the DCT (default is 16). The `-cms[=n]` option turns on the long term cepstral mean subtraction. If a value $n > 0$ is specified, CMS is performed on n frames consecutive windows. Otherwise, the CMS window length is the entire file. Liftering can be obtained with the `--lifter=n` option. The lifter is defined by

$$lifter[i] = 1 + \frac{l}{2} \sin\left(\frac{i\pi}{l}\right) \qquad i = 1, \ldots, P \ .$$

4.4 slpc

`Usage: slpc [options] ifile ofile`

Linear predictive analysis of a speech signal. This program takes as input a signal and outputs a `LPC`, `PARCOR` or `LAR` SPro data file. For each frame of the input signal, a generalized LPC analysis is performed using the Levinson-Robinson-Durbin procedure to solve the Yule-Walker equations (implemented as in [5]). Option `--order=n` controls the order of the LPC analysis. The default behavior is to output the linear prediction coefficients but option `--parcor` (resp.

`--lar`) enables to save reflexion (resp. log-area ratio) coefficients to the output file.

4.5 slpcep

`Usage: slpcep [options] ifile ofile`

LPC-based cepstral analysis of a speech signal. This program takes as input a signal and outputs a `LPCEPSTRA` SPro data file. For each frame in the input signal, the prediction coefficients a_i are computed as in `slpc`. Option `--order=n` sets the LPC analysis order which defaults to 16. The cepstral coefficients are then computed using the recursion

$$c_i = -a_i + \frac{1}{i} \sum_{j=1} i - 1 \left((i-j) a_j c_{i-j} \right) \text{ for } i = 1, \ldots, N$$

where P is the order of the cepstral analysis defined using the `--num-ceps=n` option (default is 16). As for `sfbcep`, the `--cms[=n]` option turns on the long term cepstral mean subtraction and `--lifter=n` sets the liftering value.

4.6 scopy

`Usage: scopy [options] ifile ofile`

Copy the data file `ifile` into `ofile`. The input data can be converted according to the following transformations:

1. mean normalization and delta computation
2. multiplicative scaling
3. linear transformation
4. bin extraction

If several transformation options are specified, the transformations are applied in the order given above.

Mean normalization and delta computation is obtained using the `--convert=s` option. The argument to this option is a string composed of one or zero occurrence of the letters D, A, N and Z. The meaning of the letters is:

- D: add Δ coefficients
- A: add $\Delta\Delta$ coefficients (only valid if the input already have deltas or if D is also specified)
- N: suppress static log-energy (only valid if the log-energy is in the input and if either D is specified or the input file already contains the Δ coefficients).
- Z: long-term mean subtraction

The specified qualifiers are added to the input file qualifiers. This means that you cannot use the `--convert` option to remove qualifiers from a stream. The `--scale=f` option may be used to multiply the features by an arbitrary value `f`. Furthermore, a linear transformation can be applied to the feature vectors using the `--transform=fn`, where `fn` is the path to a file containing the transform matrix A such that the new feature vectors $y = Ax$ assuming x is the original feature vector. The tranformation matrix file is a text file with the following syntax

```
nrows ncolumns
A[0][0]         A[0][1]     .........   A[0][ncolumns-1]
                            .........
A[nrows-1][0]               .........   A[nrows-1][ncolumns-1]
```

Finally, it is possible to extract some components or bins of the feature vectors using the `--extract=s` option, where `s` is an extraction pattern. For example, the option `--extract=1-9,14` will extract components 1 to 9 and component 14 of each input feature vectors. Note that the order in which the components are specified in the patter does not matter.

By default, `scopy` output file format is the SPro format but this can be changed using the `--output-format=s` option where `s` can be `ASCII`, `HTK`[3] or `Sirocco`. Byte-swapping before writing the output file can be turned on with the `--swap` option. It is also possible to copy only parts of the input file by specifying the start and/or end frame using the `--start=n` and `--end=n` options.

Finally, `scopy --info` prints the input file header information.

[3]Only work with HTK release 2.0 or greater. HTK output data type is OTHER and a frame rate of 10 ms is assumed.

5 Programmer guide

All the type and constant definitions, the function prototypes are defined in the header file `spro.h`[4].

5.1 Type definitions

5.1.1 Waveform related types

There are three main types related to waveforms. A *waveform* is stored in a structure defined as

```
typedef struct {
        unsigned long n;        /* number of samples                    */
        sample_t *s;            /* array of samples                     */
} spsig_t;
```

where `sample_t` is a `float`. To allow for a control of the amount of memory used for I/O functions, I/Os are made using a *waveform buffer* defined as

```
typedef struct {
        unsigned long m;        /* maximum number of samples            */
        unsigned long n;        /* number of samples                    */
        sample_t *s;            /* array of samples                     */
} sigbuf_t;
```

Finally, waveform I/Os rely on the use of a *waveform stream* whose corresponding structure is given by

```
typedef struct {
        char *name;             /* stream file name (NULL if stdin)  */
        void *f;                /* input stream (FILE or SP_FILE)    */
        int format;             /* stream format                     */
        unsigned long nsamples; /* total number of samples in stream */
        unsigned long nread;    /* number of samples read            */
        float Fs;               /* sample rate                       */
        unsigned short nchannels;/* number of channels               */
        int nbps;               /* bytes per samples and channel     */
        int swap;               /* to swap or not to swap?           */
        sigbuf_t *buf;          /* input buffer                      */
} sigstream_t;
```

5.1.2 Feature related types

Features, also called data whithin SPro, are stored in a structure defined as:

```
typedef struct {
        datakind_t kind;        /* feature kind                     */
        unsigned long dim;      /* vector dimension                 */
        unsigned long n;        /* number of feature vectors        */
        unsigned long m;        /* maximum number of vectors        */
        flag_t flag;            /* qualifiers                       */
        feature_t *s;           /* array of features                */
} data_t;
```

[4] Actually, reading the header file might prove more easy than reading this manual and I strongly recommend the reader to do so...

where `feature_t` is a `float`, `flag_t` is a bit field of type `unsigned int` and the feature kind is defined as an enum of `OTHER`,`FBANK`, `FBCEPSTRA`, `LPCEPSTRA`, `LPCOEFF`, `PARCOR` and `LAR`.

The library also defines a feature buffer similar to what is done for waveforms, the library defines a feature buffer as

```
typedef struct {
        unsigned long dim;      /* vector dimension                */
        unsigned long n;        /* number of feature vectors       */
        unsigned long m;        /* maximum number of vectors       */
        feature_t *s;           /* array of features               */
} spfbuf_t;
```

5.2 Waveform allocation and input functions

5.2.1 Opening a waveform stream

Since SPro is only intended for reading waveforms, a waveform stream is always considered as an input stream. *sig_stream_open() and sig_stream_close().*

5.2.2 Basic allocation functions

Signal and buffers alloc/free functions.

5.3 Feature allocation and input functions

5.3.1 Data I/Os

The following functions are used to load data from a file:

```
int ReadSPData(data_t **, FILE *);

int LoadSPData(data_t **, flag_t, FILE *);
```

The difference between those two functions is that the later also makes conversions. Actually, the second argument of `LoadSPData()` is the qualifier that the returned data should have. Therefore, `LoadSPData()` first invoke `ReadSPData()` and then convert the data using the `ConvertData()` routine. An error is returned if the conversion is impossible. The conversion function (`int ConvertData(data_t **,data_t *,flag_t)`), is in charge of simple conversions such as subtracting the long term mean, adding the Δ and $\Delta\Delta$ coefficients ...

The output function is:

```
int WriteSPData(data_t *, FILE *);
```

5.3.2 Basic allocation functions

CreateSPData() and FreeSPData().

5.3.3 Feature buffer realted functions

All the spfbuf_t related functions.

5.4 Waveform and feature structure handling functions

5.4.1 Related to waveforms

`int get_next_sig_frame(sigstream_t *f, int c, int l, int d, float a, sample_t *s)`
return in `s` the next frame after pre-emphasis by $1-az^{-1}$, reading channel `c` in the waveform stream `f` and assuming a frame length of `l` samples and a frame shift of `d` samples; the sample array `s` must remain unchanged between two successive calls to the function; returns 1 upon succes and 0 otherwise.

`float sig_normalize(spsig_t *s, int flag)`
returns the energy in signal `s` and, if flag is not null, normalize the waveform samples to unit energy.

`double *set_sig_win(unsigned long n, int type)`
allocate and initialize memory for the specified window of length `n` samples, returning a pointer to the allocated vector or `NULL` in case of error; the valid window types are `SPRO_XXX_WINDOW` where `XXX` is one of `HAMMING`, `HANNING` and `BLACKMAN`.

`int sig_weight(spsig_t *s, double *w, sample_t *p)`
apply the specified window to the signal, storing the result in `p`..

5.4.2 Related to data

`unsigned long DataVecSize(data_t *data)`
Returns the real length of the feature vectors.

`param_t *DataGetVec(data_t *data, int n, unsigned long nelem)`
Returns a pointer to vector `n` in the data. The third argument is the real feature vector length to avoid recomputing it at each invocation of this function. Vectors are numbered starting with index 0 and NULL is returned if `n` has an invalid value.

`void DataZeroMean(data_t *data, unsigned long n)`
Suppress the long term mean of the static coefficients in `data`. If `n` is not null, the mean is removed over consecutive windows of `n` frames.

`void CepLifter(vector_t *v, int l)`
Apply cepstral liftering to coefficients in `v`.

`void ScaleEnergy(data_t *p, float s)`
Scale energy according to s.

5.5 Linear predictive analysis

`int ARGModel(sig_t *sig, float a, datakind_t kind, int p, vector_t *v, float *s)`
Perform generalized linear predictive analysis of order `p` on signal `sig`,

using the spectral resolution defined by `a`. Output can be The energy of the filter $\frac{\sigma}{A(z)}$ is returned in `s`. Note that *sig* is usually a frame.

`int ARConvert(vector_t *v1, datakind_t t1, datakind_t t2, vector_t *v2)`
LAR $\leftrightarrow$ PARCOR $\leftrightarrow$ LPC conversions. `v1` contains the source coefficients of type `t1` which are converted in `t2` type coefficients returned in vector `v2`. `v2` is NOT allocated in the function.

`int LPCepstrum(vector_t *a, int p, vector_t *c)`
computes `p` cepstral coefficients returned in vector `c` from the prediction coefficients of vector `a`.

5.6 Fourrier analysis

`int FFTInit(unsigned long n)`
Initialize an n point FFT. Must be called prior to any invocation of `FFTransform()`.

`int FFTransform(sig_t *x, vector_t *m, vector_t *p)`
Performs the FFT of signal `x`. Returns the phase in `p` and the module `m`. Vectors `p` and `m` must be allocated before calling this function and must be able to contain (at least) `npts`/2 points. Either `p` or `m` can be NULL if phase or module is not to be computed.

`int LogFilterBank(sig_t *x, float alpha, int nfilt, float f_min, float f_max, vector_t *v)`
Applies a triangular filter bank of `nfilt` filters to signal `x` using the spectral resolution defined by `alpha` (`alpha`=0 means equal resolution). The frequency range (in the original domain) can be specified by the `f_min` and `f_max` variable. These frequencies must be given as normalized frequency (*i.e.* $\in [0, 0.5]$). The whole frequency band is considered if both frequencies are equal. The log energy in each filter is returned in vector `v`. `v` is must be allocated before calling the function.

`int LogMelFilterBank(sig_t *x, float freq, int nfilt, float f_min, float f_max, vector_t *v)`
Similar to `LogFilterBank()` except that Mel frequency warping is used instead of alpha-warping. `freq` is the sample rate of the signal.

`int DCTInit(int n, int m)`
Initialize a `n` to `m` DCT kernel. Must be called prior to any invocation of `DCTransform()`.

`int DCTransform(vector_t *v1, vector_t *v2)`
Computes cepstral coefficients returned in vector `v2` from the filter-bank output in vector `v1`. `v2` must be allocated prior to the invocation of the function.

6 History

6.1 Changes from SPro 3.3 to 3.3.1

The main difference from the previous versions is the introduction of waveform streams which enables a sequential reading of waveform samples thus making it possible to deal with large audio streams (several hours of waveforms at 48 kHz) without requiring Gbytes of memory! This change comes along with the support of the WAVE file format and with the ability of reading waveform streams from stdout. Release 3.3.1 is fully compatible with release 3.3.

6.2 Changes from SPro 3.2 to 3.3

SPro 3.3 is *not* compatible with previous releases. This is mainly due to the introduction of a new, GPL-distributable FFT algorithm. The current release is back to the stand-alone philosophy and no more external libraries are needed except for the SPHERE support. Configuration and installation now made easy through `automake` and `autoconf`.

The list of changes is:

- replaced numerical recipes FFT algorithm with a distributable one
- implemented kernel based transforms for FFT and DCT
- removed libgg dependencies (using GNU getopt for option parsing)
- added autoconf/automake stuff
- added windowed CMS
- corrected major bug in band-limited filter-bank analysis

6.3 Changes from SPro 3.1 to 3.2

SPro 3.2 is fully compatible with version 3.1 and only minor corrections and add-ons were made[5]. The list of modifications is:

- long option names were added to all runtimes (now require version 2.2 of libgg),
- tracing has been simplified,
- energy scale factor option added to related tools,
- frequency range limiting in FFT based spectral analysis, and
- the bug in the computation of the $\Delta\Delta$ energy has been corrected ($\Delta\Delta$E was previously computed as ΔE).

[5] Compatibility is only true for the runtimes and for the algorithms but not for the library since two function prototypes were changed in `spro.h`.

6.4 Changes from SPro 3.0 to 3.1

The organization changed from SPro 3.0. The former spro_util.* codes are now part of the libgg 2.1 library and SPro 3.1 requires that library. This library also provides an error reporting function as well as some standard error codes.

The FFT algorithm was changed to make it faster. It was previously based on Nielsen's implementation of the algorithm. The new version was adapted from the real FFT function described in the Numerical Recipes in C[6]. In SPro 3.0, the FFT was done on 1024 points and it is now done on 512 points. Current version of sfbank is about 5 times faster than the previous one.

The computation of the Δ coefficients slightly changed at the boundary (the boundary frames are now repeated instead of being avoided).

Known bugs have been corrected, principally concerning the cepstral mean subtraction and error reporting is now better. Note that for a better error reporting, many function prototypes changed so that some tools developed with the previous version may be adapted before they are compatible with SPro 3.1. The order of the enumeration for the datakind_t also changed so that when using `slist` with old data file, the data kind printed on screen may not be the correct one.

The SPHERE compatibility is also a new feature of SPro 3.1.

References

[1] Christian Chouzenoux. *Analyse spectrale rsolution variable. Application au signal de parole.* PhD thesis, Ecole Nationale Superieure des Tlcommunications, 1982.

[2] G. Gravier, C. Mokbel, and G. Chollet. Model dependent spectral representations for speaker recognition. In *EuroSpeech*, volume 5, pages 2299–2302, 1997.

[3] C. Mokbel, G. Gravier, and G. Chollet. Optimal state dependent spectral representation for HMM modeling: a new theoretical framework. In *EuroSpeech*, volume 3, pages 1351–1354, 1997.

[4] Steve Young. *The HTK book v2.0.*

[5] René Boite and Murat Kunt. *Traitement de la parole.* Presse Polytechniques Romandes, 1982.

[6]See for example the following URL: http://cfatab.harvard.edu/nr/bookc.html