SPECTUTILS, AN AUDIO SIGNAL ANALYSIS AND VISUALIZATION TOOLKIT FOR GNU OCTAVE

Kai Lassfolk
Department of Musicology
University of Helsinki
Helsinki, Finland
Kai.Lassfolk@helsinki.fi

Jaska Uimonen
Nokia Devices
Helsinki, Finland
Jaska.Uimonen@nokia.com

ABSTRACT

Spectutils is a GNU Octave toolkit for analyzing and visualizing audio signals. Spectutils allows to display oscillograms, FFT spectrograms as well as pitch detection graphs. Spectutils can best be characterized as a user interface for GNU Octave, which integrates signal analysis and visualization functionality into dedicated function calls. Therefore, signal analysis with Spectutils requires little or no prior knowledge of Octave or MATLAB programming.

1. INTRODUCTION

Spectutils contains a set of GNU Octave [1] functions and Unix-style command line utility programs. The main Octave functions are intended for displaying signal analysis plots from sound files. Dedicated Octave functions are provided for oscillograms, 2D spectrum plots, 3D spectrograms, sonograms, and 2D pitch detection plots. The command line programs are intended for pre-processing sound files for use with the Octave functions. Several versions of Spectutils have been already released for public distribution. This paper describes the new Spectutils version 1.0, which is written specifically for Octave 3.0.

Spectutils is being developed with emphasis on musicological applications. There is a growing interest in applying spectrum analysis for studying musical performances. Therefore, Spectutils is intended also for users with little or no programming experience.

Being based on GNU Octave, Spectutils has a text-based user interface. The initial learning phase is therefore somewhat more difficult compared to graphical user interface (GUI) based tools. On the other hand, repeated analysis and handling of large amounts of input data is considerably easier than with a typical GUI-based program. Programmability provided by Octave is another advantage and enables automation of routine tasks.

Spectutils originally started as an attempt to provide convenient FFT spectrogram display capability for Octave version 2.0. Octave's 3D capabilities were at that stage considerably less convenient than those of MATLAB [2]. In particular, earlier Octave versions were heavily dependent on Gnuplot [3]. One of the main purposes of Spectutils was to hide the Gnuplot-specific details from the user and to provide a simple user interface for creating spectrograms. Similarity with MATLAB's spectrum analysis functions or optimization of computational efficiency was not considered crucial. Over time, Spectutils grew into a set of signal analysis tools.

With version 2.9 and, finally, 3.0, Octave's 3D plotting was changed for closer syntactical compatibility with MATLAB. This also required a major rewrite of Spectutils, released as version 0.9. Octave's latest versions also provided many helpful additions regarding sound file support and graphics that were also included in Spectutils. Despite Octave's closer MATLAB compatibility, Spectutils is not currently compatible with MATLAB.

Spectutils is being developed at the Department of Musicology, University of Helsinki. It is distributed under the GNU General Public License (GPL). The software package can be downloaded from http://www.music.helsinki.fi/research/spectutils/. The package includes installation instructions for Linux, Windows, and Mac OS X.

2. OCTAVE FUNCTIONS

Spectutils contains five Octave functions for producing graphical output: oscgram() for oscillograms, spec2dw() for 2D spectrum plots, spec3dw() for 3D spectrograms, sonogw() for sonograms, and hps2dt() for pitch detection. Each of the functions can be used independently and do not require additional Octave programming, unless automation or further customization of the graphical output is desired.

Most of the Spectutils' Octave functions use standard signal processing algorithms included in Octave such as Fast Fourier Transform (FFT) and Short-Time Fourier Transform (STFT). However, Spectutils contains an implementation of the Harmonic Product Spectrum (HPS). The implementation is described in more detail below.

The analysis functions read the input audio signal directly from a sound file. The octave functions support multichannel WAV files including 16-bit and 24-bit integer as well as 32-bit floating point. Sampling rate, sample encoding, and channel count are read from the WAV file header. Raw format sound files are also accepted but restricted to monophonic 16-bit integer, or either 32-bit or 64-bit floating point data. There, any floating point sampling rate value is allowed. With raw files, both sampling rate and sampling encoding can be specified as parameters for the Octave functions, but may be omitted for 44,1 kHz, 16-bit integer files.

The Spectutils Octave functions are designed to be self-documenting: By default, each function outputs its parameters either on the plot axes or in the title text above the graphical plot. Also date and time information is included in the title text. The title text may also be suppressed, if desired. Octave's standard graphics functions, such as view(), colormap(), and replot(), can be used for adjusting Spectutils plots as well as for avoiding repeated STFT analyses of the same signal.

2.1. Oscillograms

Oscillogram display is provided by the oscgram() function. Its parameters include a sound file specifier, signal offset and duration (in seconds), as well as an optional comment text string. The sound file specifier is either a sound file name (as a text string) or a structure (enclosed in braces) containing the sound file name and additional parameters. These enable either to select a channel from a multichannel WAV file (by default, channel 1 is analyzed) or to specify the sampling rate and sample encoding format of a raw format sound file (default is 44.1 kHz, 16-bit). While the syntax for specifying these additional parameters is a bit cumbersome, in most cases only the sound file name is sufficient and simple function call syntax is maintained.

A sample function call is as follows:

```
oscgram('flute.wav', 10.4, 3, 'Solo flute.');
```

It plots an oscillogram of 3 seconds starting from an offset of 10.4 seconds. The input soundfile name is flute.wav and the comment text 'Solo flute.' is added to the plot title.

As a special feature, oscgram() allows to use spline interpolation for smoothing a closely zoomed signal, where a conventional waveform display would appear as a jagged line. Individual sample values may be optionally displayed with '+' signs on top of the interpolated "continuous" signal. Spline interpolation is specified with an additional parameter, supplied after the comment text string.

2.2. 2D spectrum display

Spec2dw(), the 2D FFT magnitude spectrum function, follows a similar calling convention to oscgram(). The file specifier is treated similarly. Also, a signal offset, comment text, and use of spline interpolation are specified similarly to oscgram(). As a distinction, the signal duration is replaced by three FFT specific parameters: number of FFT points, window length (in samples), and window type (either Hanning, Hamming, or rectangular). Furthermore, a frequency range for the plot may be specified. Spec2dw supports both linear and logarithmic display of both frequency and magnitude axes.

As a special feature, a high frequency weight parameter is provided for emphasizing high frequencies in the graphical output. The weighting behaves linearly with respect to frequency. For example, a weight of 100 multiplies the magnitude value of the spectrum by 100 at $f_s/2$ (i.e. Nyquist frequency), by 50 at $f_s/4$, 25 at $f_s/8$, etc. Weighting is useful in examining the high frequency content of a signal with a linear magnitude axis display and when a harmonic overtone structure is difficult to examine with a logarithmic magnitude display due to high frequency noise components.

Spline interpolation, as in oscgram(), makes it easier to estimate the peaks in the magnitude spectrum. A sample 2D spectrum plot using the spline feature is shown in Figure 1. There, the individual samples are displayed with +-signs along the spline-interpolated magnitude spectrum graph.

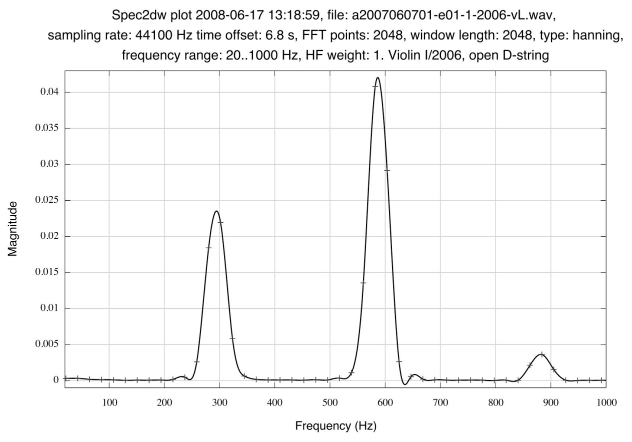

Figure 1: *A sample magnitude spectrum plot from spec2dw().*

2.3. 3D spectrograms

The spec3dw() function produces 3D magnitude spectrograms. Parameters are almost the same as in spec2dw(), additional parameters being the duration of the spectrogram (in seconds) and a window increment parameter (in samples) for controlling the interleaving factor of consecutive FFT windows. Spline interpolation is currently not implemented. Spec3dw also allows 2D grayscale sonograms to be plotted, although a dedicated function, sonogw() is provided especially for that purpose.

Spec3dw() uses Octave's default settings for the viewing angle and color map for 3D plots. These settings may be adjusted with Octave's view() and colormap() functions, respectively. A sample spectrogram (using a black-only color map and a "-80°, 15°" viewing angle) is shown in Figure 2. The respective function call is as follows:

```
spec3dw('a2007060701-e01-1-2006-vL.wav',
    6.3, 0.5, 2048, 1024, 'hanning', 512,
    20, 5000, 1,
    'Violin II/2006, open D-string');
```

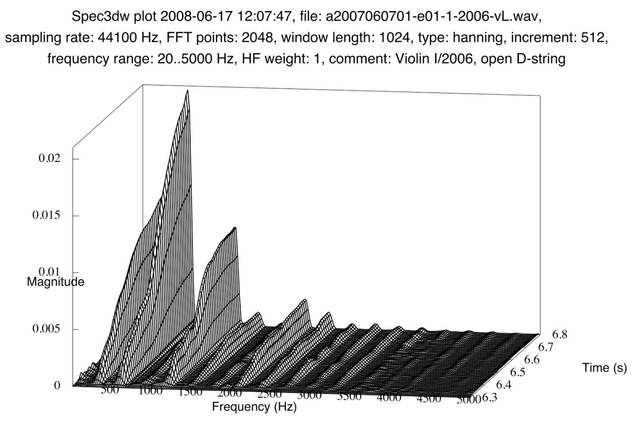

Figure 2: *A sample spec3dw() plot of a bowed violin open D string.*

The function call parameters are (listed from left to right) sound file name, start offset, duration, number of FFT points, window length, type and increment, low and high frequency limits, high frequency weight factor and comment text.

With an optional parameter, spec3dw() allows to flip the frequency axis for viewing the spectrogram from a reverse angle. Also, logarithmic display of both magnitude and frequency is supported. The syntax of these additional parameters is demonstrated in the spec3dw() function call below:

```
spec3dw('a2007060701-e01-2-2006-vL.wav',
    6.3, 0.5, 2048, 1024, 'hanning', 512,
    20, 5000, 1,
    'Violin I/2006, open D-string',
    'revfreq,magdb,logfreq');
```

2.4. Sonograms

The sonogw() behaves almost similarly to spec3dw(). The main difference is that the spectrogram is displayed as a 2D gray scale image view instead of a waterfall plot. The parameters are also basically the same as in spec3dw(). For example, linear and logarithmic display of both frequency and magnitude are supported. A sample sonogw() plot is shown in Figure 3. The respective function call is as follows:

```
sonogw('flute.wav', 0, 15, 2048, 2048,
    'hanning', 512, 20, 12000, 1,
    'Solo flute.', 'magdb');
```

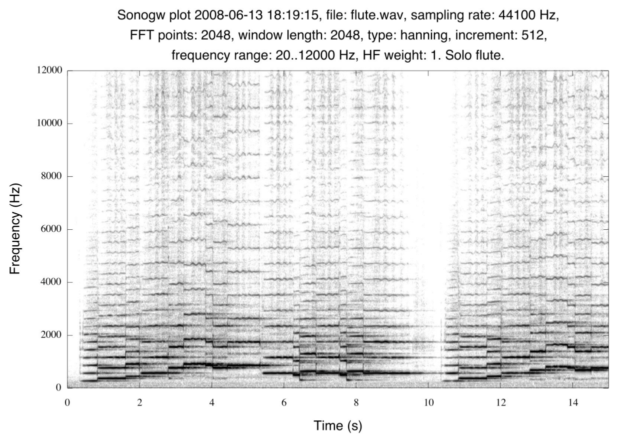

Figure 3: *A sample sonogram from sonogw().*

By default, sonogw() uses a preset image contrast setting, but manual control is also provided through additional function call parameters. Also the high frequency weight parameter, similar to spec2dw() and spec3dw(), can be used for adjusting the contrast balance between low and high frequencies.

2.5. 2D harmonic product spectrum

There are a number of methods used to discover the fundamental frequency of a pitched sound. For a list and description of different methods the reader is recommended to refer to the paper by de la Cuadra et al. [4]. Spectutils includes an implementation of one of these methods called *Harmonic Product Spectrum* (hereafter HPS). The HPS method was discovered independently by both Schroeder and Noll [5] and in recent studies HPS is used for example to recognize musical instruments [6] and speech [7].

The implementation of HPS uses Fast Fourier Transform (FFT) to compute the instantaneous frequency content of a sound. First a segment of digitized sound samples is multiplied with a window function to smooth the errors in the finite transform. After this the FFT of the frame is computed, absolute values are taken and the negative frequencies are removed. The frequency bins are then scaled down by integers from n to N (N ranging usually from 3 to 5) and the scaled down spectra are multiplied together (see equation 1).

$$Y(f) = \prod_{n=1}^{N} S(\frac{f}{n}) \qquad (1)$$

In this process the possible harmonic components align on top of the fundamental and the frequency content in the other regions is reduced. It must be noted, that the method in equation 1 is presented in [4] and [6], but it differs from the HPS algorithm introduced originally by Schroeder in [5]. Instead of multiplication Schroeder sums the scaled down spectrums and weights the end result with a logarithm. Schroeder also states that Noll has used multiplication, but on the cepstrum domain.

Finally the component with the maximum value is searched and kept as the estimate of the fundamental pitch (see equation 2).

$$\hat{Y} = \max_{f_i}\{Y(f_i)\} \qquad (2)$$

It is also possible to calculate an error value for the fundamental frequency estimate in each frame. This is usually the relation of the maximum valued component to the other components in the frame. Nielsen et al. [6] calculate the relation of the maximum value to the total energy in the frame and this method is also used in Spectutils. Zolnay et al. [7] for example calculate a speech "voicing" value, which is the relation of the maximum valued FFT bin to the geometric mean of the neighbouring bins.

The HPS algorithm is used in function hps2dt(), which calculates the HPS of every consecutive Short Time Fourier Transform frame of a sound. This way the complete melody line and its reliability can be plotted as a function of time. A sample function call is made as follows:

```
hps2dt('test.wav', 1.5, 2, 1024, 1024,
    'hanning', 512, 50, 3000, 3
    'test figure', 'logfreq');
```

The function reads from a file "test.wav" from offset 1.5 seconds a 2 second window. It then computes 1024-point FFT with 1024-point hanning window (also 'hamming' and 'rectangle' can be used) with 512-sample overlap. The frequency scale is limited into the range 50-3000 Hz and the HPS algorithm is iterated 3 times. The comment "test figure" is printed on the plot and frequency is plotted on a logarithmic scale (if not specified a linear scale is used). The output can be seen in Figure 4.

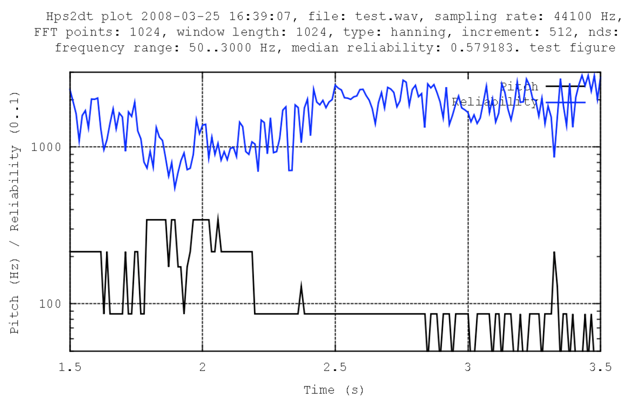

Figure 4: *Fundamental frequency plotted from a 2 second clip of a solo viola performance. Upper curve represents the reliability of the pitch estimate.*

3. UTILITY PROGRAMS

Spectutils versions prior to 0.9 were restricted to raw format sound files. Therefore, a set of command line programs was developed mainly for converting WAV files to monophonic raw format. Even with the current WAV file support, the utility programs are sometimes useful and are thus still included in Spectutils. The file conversion programs are named stripwhdr, st2m, and cdda2mono. The stripwhdr simply skips the header sections of an input WAV file and outputs only the contents of audio data as a raw signal stream. The st2m program converts a 2-channel raw stereo stream into mono by selecting either its left or right channel or by calculating either the sum or difference signal of the two stereo channels.

Cdda2mono is an “audio CD to raw mono sound file” conversion program. It is implemented as a front end to st2m and the cdda2wav CD extraction program included in many Linux distributions. Therefore, cdda2mono requires cdda2wav, which restricts its portability.

The program pronsets is a tool that attempts to find and print the onsets of individual sound events (separated by a pause) from a raw format sound file. Pronsets was written specifically for analyzing string instrument tones.

4. CONCLUSION

Spectutils provides a set of tools for analyzing and visualizing audio files with GNU Octave. Although installation and basic use of Octave and its support programs (e.g. Gnuplot) requires some expertise (or help from an experienced user), Spectutils is reasonably easy to learn, especially compared to achieving the same graphical output by using standard Octave 3.0.

The intension in designing the function call syntax was to keep the amount of required parameters small. Nevertheless, the functions allow a high level of control over the analysis process and the graphical output. The aim was also to allow experienced users access to as many parameters as possible. The payoff is that in practice the function calls typically contain extensive lists of parameters, where the syntax is not particularly intuitive. Fortunately, Octave’s command line memory relieves the user from repeated typing of these long parameter lists and helps in analyzing large sets of sound files.

Standard MATLAB-style online help texts are included in the Octave functions and Unix-style “man” pages are included for the command line programs. Also included is a Finnish language HTML-tutorial; an English language translation is being prepared.

An advantage in using Octave for performing spectral analyses is its flexibility and programmability. Moreover, processing of large amounts of sound files is often more convenient than with graphical interactive user interfaces. A downside is that Octave is not computationally efficient in handling large data structures. Especially 3D spectrograms and HPS plots are not instantaneous even with modern microcomputers.

The graphical output of some of the Octave functions looks still somewhat unpolished, partly due to Octave itself. Moreover, support for soundfile formats other than WAV and raw and more flexible handling of multichannel files would enhance usability. Future development of both Spectutils and Octave will address these issues.

5. REFERENCES

[1] http://www.octave.org/, Accessed June 10, 2008.

[2] http://www.mathworks.com/products/matlab/, Accessed June 10, 2008.

[3] http://www.gnuplot.info/, Accessed June 10, 2008.

[4] P. De la Cuadra, A. Master and C. Sapp, “Efficient Pitch Detection Techniques for Interactive Music,” in *Proceedings of the International Computer Music Conference,* Havana 2001, Sept. 18-22, pp. 403-406.

[5] M. R. Schroeder, “Period Histogram and Product Spectrum: New Methods for Fundamental-Frequency Measurement,” in *The Journal of the Acoustical Society of America,* Volume 43, Number 4, pp. 829-834, April 1968.

[6] A. B. Nielsen, L. K. Hansen and U. Kjems, “Pitch Based Sound Classification,” in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, Toulouse, France, May 14-19, 2006, pp. 788-791.

[7] A. Zolnay, R. Schlüter and H. Ney, “Extraction Methods of Voicing Feature for Robust Speech Recognition,” in *Proc. European Conference on Speech Communication and Technology*, Vol. 1, Geneva, Switzerland, September 2003, pp. 497-500.