PsySound algorithm notes

Matt Flax <flatmax>

March 27, 2007

1 Introduction

Psysound is a project which is primarily monitored by Densil Cabrera who is currently employed by the University of Sydney.

These notes treat various analysis algorithms from the psychoacoustics sound analysis package available from psysound.org.

Certain paradigms are addressed here which may not be treated in reference algorithm articles. This document is an effort to yield the Psysound software as open as possible.

These algorithms were developed independently of the graphical user interface. Ultimately, the code base is the best reference for the various algorithms.

2 General psysound program operation

PsySound is capable of arbitrary sound file analysis in a block operation paradigm. The sample rate of the file is arbitrary, in most cases, and the sample bit count is converted to double format automatically. The block operation paradigm allow very large files to be analysed without running out of memory.

In general the algorithms operate on approximately thirty two different file types. These file types are converted to 'wav' file format, which Matlab is capable of opening naively. The software which is used to generate such diverse file handling is the 'sox' program. Please read the README file for more information on the location of sox on the Internet. Psysound is released with two versions of sox. One version for win32 and the other version for Mac OSX.

The general algorithm flow is to convert the desired file into 'wav' file format. This new wav file is dumped into a temporary directory and analysis is conducted on this file.

In order to both handle files effectively and reduce re-computation of fundamental data and data-structures, PsySound uses a 'fileHandle' data structure to store vital file information and data. Further, various transforms and modules store filter coefficients, states and other vital information which need only be generated once at the beginning of operation. All resident structure and data storage is accomplished with the 'fileHandle' data structure. This fileHandle is associated with ONLY one file.

All file data is read using the 'readData' utility. This utility is capable of scaling the input signal upon reading in order to calibrate the input data automatically. Further, the readData utility defines the window or block size. ReadData is also capable of block overlap and other features. See the file readData.m for more information.

3 Directory organisation

3rdparty	Contains all third party code used in the project. Most of this code is altered to work with the PsySound project.
bin	Contains the sox binaries
doc	Contains this documentation
modules	Contains the various analysis modules
otherCode	Ported code and other codes which don't fit into other directories
splash	the startup splash screen control.
temp	standard temporary file dump
testscripts	various testing scripts for transforms and algorithm modules
testsounds	some test files which have been used in the development of this software
transforms	various transforms used in analysis
utils	utility scripts

4 Transforms and modules

4.1 Auto-correlation

Implemented using the DFT. To avoid cyclic overlapping, the algorithm zero pads the signal.

4.2 Cepstrum

The standard Matlab implementation of the Cepstrum transform is utilised. Various statistical measures of moments, standard deviation, skewness and kurtosis are also calculated.

4.3 Dynamic Loudness Models

Two forms of the Dynamic Loudness Models are implemented.

4.3.1 Moore-Glasberg Dynamic Loudness

The Moore-Glasberg static loudness model is used implemented using third party and personally implemented code. The dynamic loudness code is implemented directly from the static loudness code. The static and dynamic loudness code is implemented directly from the following references. The static code is altered to the most recent methods implicitly mentioned in the 2002 paper. These are implemented as separate modules.

[1] @article{glasberg2002mla,
title={{A model of loudness applicable to time-varying sounds}},
author={GLASBERG, B.R. and MOORE, B.C.J.},
journal={Journal of the Audio Engineering Society},
volume={50},
number={5},
pages={331–342},
year={2002},
publisher={Audio Engineering Society}}

[2] @article{moore1997mpt,
title={{A model for the prediction of thresholds, loudness, and partial loudness}},
author={MOORE, BCJ and GLASBERG, BR and BAER, T.},
journal={Journal of the Audio Engineering Society},
volume={45},
number={4},
pages={224–240},
year={1997},
publisher={Audio Engineering Society}}

[3] @article{glasberg1990daf,
title={{Derivation of auditory filter shapes from notched-noise data.}},
author={Glasberg, BR and Moore, BC},
journal={Hear Res},
volume={47},
number={1-2},
pages={103–38},
year={1990}}

[4] @article{moore1987fdf,
title={{Formulae describing frequency selectivity as a function of frequency and level, and their use in calculating excitation patterns}},
author={MOORE, BCJ and GLASBERG, BR},
journal={Hearing research},
volume={28},
number={2-3},
pages={209–225},
year={1987},
publisher={Elsevier}

4.3.2 Chalupper dynamic loudness

Taken from code provided by Chalupper. Published in the following two references :

Chalupper, J.,Fastl, H. (2002): Dynamic loudness model (DLM) for normal and hearing-impaired listeners. ACUSTICA/acta acustica, 88: 378-386

Chalupper, J. (2001) - in german - : Perzeptive Folgen von Innenohrschwerh?rigkeit:

Modellierung, Simulation und Rehabilitation. Dissertation at the Technical

University of Munich, Shaker Verlag.

4.4 Discrete Fourier Transform

The DFT is implemented using the built in Matlab FFT algorithm. Post processing revealing moments, skewness and kurtosis are further implemented.

4.5 Hilbert Transform

The Hilbert Transform is implemented using the built in Matlab Hilbert algorithm.

4.6 Parncutt Measures

Parncutt measures are implemented using ported code from the original psysound pascal program. This code operates on the output of Terhardt's virtual pitch algorithm. The Parncutt Measures implemented are pure tonalness, multiplicity and salience.

4.7 Pitch Scale

A novel implementation of a 'pitch scale' is implemented from the following reference. This pitch scale is capable of determining pitches and a reasonable pitch salience for signals in noise. The actual pitch scale is a non-linear abstraction which has a one to one mapping onto the Hertz frequency scale.

[1] Afferent/Efferent Pitch Processing

Proceedings of the 9'th Australian International Conference on Speech

Science & Technology 2004

Available from : http://mffmpitch.sourceforge.net/

4.8 Roughness

This roughness algorithm is implemented from a recent advancement of the Danie Weber roughness algorithm. It is available upon consultation from here : http://home.tm.tue.nl/dhermes/

The original algorithm is ported into the code base for psysound.

4.9 Sound Level Meter

A combination of one weighting filter and a leaky integrator.

4.10 Sharpness

An implementation of the bark band definition for sharpness.

[1] http://www.acoustics.salford.ac.uk/research/arc/cox/sound_quality/sharpness.htm

[2] Zwicker and Fastl

4.11 Specific Loudness

See dynamic loudness.

4.12 Virtual Pitch

A Matlab port of Terhardt's own Virtual Pitch code. The original C implementation is in the third party code directory. This implementation first extracts spectral peaks and magnitudes. These are then passed onto Terhardt's algorithm.

The following options, from the original code, are disabled in this version of the port :
shiftflag : option '-t'
vplowpass : option '-s'
monweight : option '-w'
noteflag : option '-n'
nofpitches: option '-p'

4.13 Third Octave Band Filters

Incorporates the Salford (http://www.acoustics.salford.ac.uk/research/) third octave band filters. These filters are implemented as decimated filter sets. For this reason, re-interpolation is conducted to preserve original vector dimension. It is known that the low frequency filters suffer scaling issues. Whilst a fix is Incorporated in this code for the decimated filters, the interpolation of these filters destroys the scaling. This destruction of scaling is due to limited sample count for the low frequency filters.

4.14 Weighting filters

Weighting filters are implemented from transfer functions in the Laplacian domain, the S-plane.

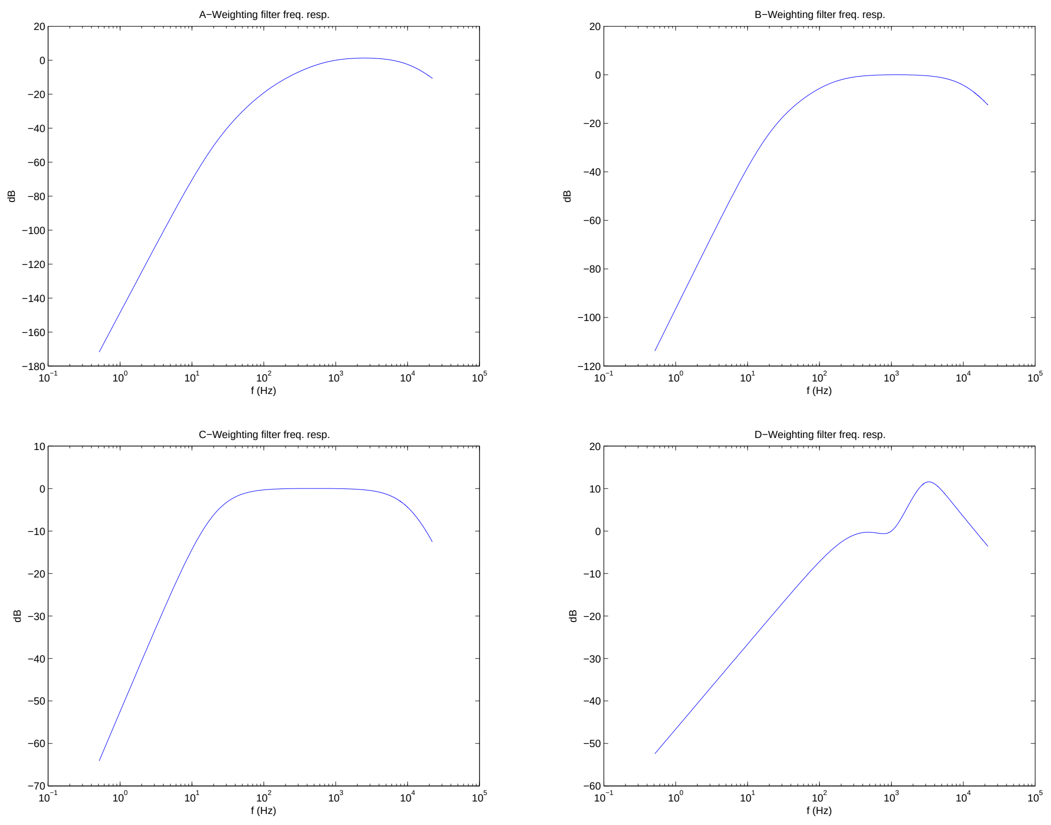

Figure 1: A, B, C and D weighting filter transfer functions.

Signal weighting is a two step process. The first step generates the weighting transfer function. The second step leakily integrates the rectified output of the weighting filters.

The transfer functions used are;

A-Weighting : $H_W(s) = \frac{k_A s^4}{(s+129.4)^2(s+676.7)(s+4636)(s+7665)^2}$ where $k_A = 7.39705e9$.

B-Weighting : $H_W(s) = \frac{k_B s^3}{(s+129.4)^2(s+995.9)(s+7665)^2}$ where $k_B = 5.99185e9$.

C-Weighting : $H_W(s) = \frac{k_C s^2}{(s+129.4)^2(s+7665)^2}$ where $k_C = 5.91797e9$.

D-Weighting : $H_W(s) = \frac{k_D s(s^2+6532s+4.0975e7)}{(s+1776.3)(s+7288.5)(s^2+21514s+3.8836e8)}$ where $k_D = 91104.32$.

These transfer functions are shown in Figure 1

4.15 Leaky integrator

Temporal information is maintained by a leaky integrator. In the time domain, a perfect integrator is the convolution of the signal with the unit step. A leaky integrator is treated as the convolution of the decaying exponential with the input signal. The filter coefficients used by the leaky integrator are derived directly from the Z-Transform of the time domain impulse response. Namely for discrete time $t = nT$ where T is the time period, the Z-Transform is given to be

$$\mathcal{Z}\{e^{-\frac{nT}{RC}}\} = \frac{1}{1 - e^{-\frac{T}{RC}} z^{-1}}$$

The implementation of this transfer function uses coefficients $a = 1$ and $b = \{1,\ -e^{-\frac{T}{RC}}\}$. Which yields the following discrete sample equation

$$b_1 y(n) + b_2 y(n-1) = a_1 x(n)$$

which is explicitly written

$$y(n) - e^{-\frac{T}{RC}} y(n-1) = x(n)$$

and is simplified to

$$y(n)\left(1 - z^{-1} e^{-\frac{T}{RC}}\right) = x(n)$$

which has the previously stated transfer function

$$\frac{y(n)}{x(n)} = \frac{1}{1 - z^{-1} e^{-\frac{T}{RC}}}$$

Finally, it is necessary to scale the output according to the magnitude of the transfer function over all causal time. In order to do this, the following is calculated

$$\int_0^\infty e^{-\frac{nT}{RC}} dn = -\left.\frac{e^{-\frac{nT}{RC}}}{\frac{T}{RC}}\right]_0^\infty = \frac{RC}{T}$$

which is the scaling of the output due to the impulse response. Consequently, the following scaling factor is chosen to yield an output which roughly matches the rectified audio waveform

$$a_1 = \frac{T}{RC}$$

. The filter coefficients are set to

$$\begin{aligned} a &= \frac{T}{RC} \\ b &= \{1,\ -e^{-\frac{T}{RC}}\} \end{aligned}$$

The implementation of the total system transfer function is non-linear. The non-linear step is the signal rectification, which occurs before the leaky integration. For this reason, the weighting filters and non-linear leaky integrator are implemented in separate scripts.