The Phase Retrieval Toolbox

Zdeněk Průša

PHASERET is a Matlab/GNU Octave toolbox for phase (signal) reconstruction from the short-time Fourier transform (STFT) magnitude. The toolbox provides an implementation of various, conceptually different algorithms including real-time capable algorithms which are also implemented in real-time audio demos running directly in Matlab/GNU Octave. The toolbox depends on the Large Time-Frequency Analysis Toolbox – LTFTAT (http://ltfat.github.io), it is well-documented, open-source and it is available under the GPL3 license at http://ltfat.github.io.

The Problem

The discrete STFT (also known as the discrete Gabor transform – DGT) is defined as

\[ c(n+m) = \sum_{f=0}^{F-1} c(f) g(f-2\pi f/M) \]

where \( n = 0, \ldots, L-1 \), \( m = 0, \ldots, M - 1 \) and where \( c \in \mathbb{C}^M \) is the DGT coefficient vector, \( f \in \mathbb{R} \) is the input signal, \( g \in \mathbb{R}^M \) is the analysis window, \( a \) is the time step size, \( s \) is the number of frequency channels (FFT size). Alternatively using matrix notation (analysis)

\[ c = F s F \]

where columns of \( F \in \mathbb{C}^{L\times M} \) are in a form of \( \sum_{n} h(n) g(n-a \Delta f) e^{-2\pi i f n} / \Delta f \) for \( l = 0, \ldots, L - 1 \). The signal can be recovered using

\[ f = (F^H F)^{-1} F^H s \]

where \( g = (F F^H)^{-1} g \).

The magnitude and phase components of the coefficients are obtained as

\[ s(p) = |c(p)|, \quad \phi(p) = \arg(c(p)) \]

for \( p = 0, \ldots, MN - 1 \), where \( s \) is also referred to as the consistent STFT magnitude (spectrogram) i.e. there exists some signal \( f \) with such STFT magnitude.

Offline algorithms

Griffin & Lim algorithm (GLA) – gla [1] proceeds by projecting the coefficients iteratively onto two subsets of \( \mathbb{C}^{MN} \) denoted as \( C_1 \) and \( C_2 \). The set \( C_1 \) is identical with the range (column) space of \( F \) i.e.

\[ C_1 = \{ c \in C^L \mid F c \neq 0 \} \quad \text{and its orthogonal complement is} \quad C_2 = \{ c \in C^L \mid F c = 0 \} \]

Any \( c \in \mathbb{C}^{MN} \) can be written as a sum of components in these two spaces as \( c = c_1 + c_2 \) and the unique orthogonal projection onto \( C_1 \) is

\[ P_{C_1} c = F (F^H F)^{-1} F^H c = F c - F F^H c \]

The set \( C_1 \) is a set of coefficients with magnitude equal to the target magnitude \( s \)

\[ C_1 = \{ c \in C^L \mid |c(p)| = s(p) \quad \text{for all} \quad p \in C_1 \} \]

Since it is not a convex set, the projection is substituted simply by forcing the magnitude of the coefficients to the target magnitude

\[ P_{C_1} c (p) = s(p) e^{j \phi(p)} \quad \text{for all} \quad p \in C_1 \]

given \( c(p) = |c(p)| e^{j \phi(p)} \) for all \( p \), where \( s \) is the target magnitude and \( \phi(p) \) is the initial phase. An acceleration technique proposed in

Le Roux's Modifications of GLA (leGLA) – legla [3] Modification 1) exploiting a structure in the projection \( P_{C_1} \) and introducing its approximation using a truncated kernel. Modification 2), since this way the projection \( P_{C_1} \) can be done coefficient-wise and so can be done \( P_{C_2} \), the authors proposed to do the on-the-fly updates i.e. to reuse the just updated coefficient for computing others within the same iteration. Modification 3) Introduction of a modified phase update (forcing coefficient of the coefficient being updated).

Decoarse's Unconstrained Optimization – decoatsfg [4] uses the IFRGS algorithm for minimizing a cost function

\[ \min \{ |c(p)|^2 = |s(p)|^2 \} \]

while exploiting an explicit formula for its gradient.

Phase Gradient Heap Integration (PGHI) – pgghi [5] reconstruction method based on the relationship between the magnitude and the phase gradients. Assuming the Gaussian window is used, the phase gradient can be estimated using the magnitude gradient and the phase is then recovered by employing an adaptive integration procedure. Works well also for non-Gaussian windows.

Online algorithms – Online Implementation

Single Pass Spectrogram Inversion (SPSI) – aspi [6] is a phase-phase-locked vocoder-like technique. It employs peak picking followed by quadratic interpolation of the magnitude in order to precisely identify the instantaneous frequency. The inst.

frequencies give the increments of the peak phase between two frames. The peak frame is also assigned to all coefficients within its region of influence.

Real-Time Phase Gradient Heap Integration (RTPGHI) – rtpghi [7] is a real-time version of pgghi. The core of the algorithm was kept, with the exception that the phase is spread only from the previous to the current frame. It is available in one and zero look-ahead frames versions.

Real-Time Iterative Spectrogram Inversion with Look Ahead (RTISLI-RA) – rtislia [6] employs \( N_{LA} \) look-ahead frames and proceeds by doing GLA-type iterations on individual frames from the most recent one to the current one and repeats this procedure for a pre-determined number of iterations. An inherent problem is an asymmetry introduced by windowing the partially reconstructed signal. In order to compensate for this problem, the authors proposed to use two additional asymmetric analysis windows to be used with the most recent look-ahead frame and to use the regular analysis window for other frames.

Gnann and Spretì's RTISLI-G (GSRTISLI-G) – gartisla [8] introduces an alternative way of dealing with the asymmetry of the partially reconstructed signal. It uses \( N_{LA} + 1 \) additional analysis windows. A combination with RTPGHI proved to give excellent results [10].

Basic usage

Listing 1: Code example of a typical workflow

% Load the glockenspiel test signal
% Sampling rate fs = 44100, 262144 samples [fs, sf] = gpsi;
% a = 256; % Time step
% N = 2048; % Number of frequency channels
% g = ‘blackman’; % Blackman window
% (gb, h, gwin) = tfex_glima_fft(g, a, ‘blackman’);
% Compute the DGT coefficients c, l = dgtreal(f, g, s, ‘timefs’);
% Obtain the magnitude
% s = abs(c);
% Reconstruct the phase
% phi = dgtreal(ch, gui, ‘a’, s, ‘timefs’);
% Reconstruct the signal
% x = dgtreal(ch, gui, ‘a’, s, ‘timefs’);
% Extend the audio block
% p = cropblock(p, ‘n’, ‘Gain’, [−20, 20, 0, 21]);
% Start audio processing loop
% while p.flag
% gain = cropblockgain(p, ‘Gain’);
% f = blockread(p, ‘blockprey’); \[10^3 \times \text{gain} \];
% and
% Cleanup blockdev(p);

Real-time Implementation

Exploits the block processing framework for real-time audio processing from LTFTAT.

Listing 2: Minimal working example

% Create the GUI panel
% p = blockpanel([‘GB’; ‘Gain’; [−20, 20, 0, 21]]);
% Initialize the audio device
% block(‘playout’, ‘loadFed’; p);
% Start audio processing loop
% while p.flag
% gain = blockpanelget(p, ‘Gain’);
% f = blockread(p, ‘blockprey’); \[10^3 \times \text{gain} \];
% and
% Cleanup blockdev(p);

Toolbox Organization and Documentation

% Public GitHub repository
% https://github.com/ltfat/phaseret
% C library back-end with MEX interfaces
% https://github.com/ltphaseret
% Web page, nodes generated from the m-file headers using
% home-brewed system nat2dco
% https://github.com/ltfat/nat2dco

References


This work was supported by the Austrian Science Fund (FWF): Y 501-N13.