REAL-TIME SPECTROGRAM INVERSION USING PHASE GRADIENT HEAP INTEGRATION

Zdeněk Průša *

Acoustics Research Institute,
Austrian Academy of Sciences
Vienna, Austria
zdenek.prusa@oeaw.ac.at

Peter L. Søndergaard

Oticon A/S
Smørum, Denmark
peter@sonderport.dk

ABSTRACT

The knowledge of the phase of STFT is a prerequisite for a successful signal reconstruction. However, the phase might be lost or no longer applicable depending on the kind of processing involved.

We propose a real-time spectrogram inversion algorithm based on the relationship of the gradients of the phase and the logarithm of the magnitude and on the gradient integration theorem. We present a detailed comparison with the state-of-the-art phase reconstruction algorithms.

1. INTRODUCTION

The first algorithm for the signal reconstruction from the (modified) STFT magnitude was introduced by Griffin and Lim in [1] (Griffin-Lim Algorithm – GLA) more than 30 years ago. Since then, several other algorithms have been developed, but the fact the algorithms typically require many iterations acting on the whole signal prohibited their widespread use e.g. as an alternative reconstruction procedure for the phase vocoder [2]. Other possible applications include e.g. compression, source separation, channel mixing, adaptive filtering and denoising. See e.g. [3] for a detailed overview of the reconstruction algorithms and applications.

A real-time version of GLA was introduced in [4] (Real Time Spectrogram Inversion Algorithm with Look Ahead – RTISI-LA). The algorithm is still iterative, but the signal is reconstructed frame-by-frame using a clever phase initialization such that only few iterations are necessary in order to get a good result. The downside of this algorithm is that it requires several "look-ahead" frames which increases the processing delay. Modifications of RTISI-LA were proposed in [5, 6, 7].

Recently, another real-time capable algorithm was introduced in [8] (Single Pass Spectrogram Inversion – SPSI). The algorithm is based on the notion of *phase consistency* introduced in connection with the phase-locked vocoder. Assuming the signal consists of sum of sinusoidal components, their phase grows in time at the rate of their instantaneous frequencies. In the algorithm, the instantaneous frequency is estimated in each frame by peak picking and quadratic interpolation. Thanks to this, the algorithm does not introduce any additional delay, but, on the other hand, it cannot properly handle any deviation from the model assumptions e.g. the quality of reconstructed transients and impulse-like components is poor.

The proposed algorithm (Real-Time Phase Gradient Heap Integration – RTPGHI) is based on the STFT phase-magnitude relationship first introduced in [9]. The offline version of the present algorithm (PGHI) along with the theoretical background has already been presented in [10]. This paper focuses on adapting the algorithm to the real-time setting. In the basic form it requires one look-ahead frame, but even zero delay can be achieved at a cost of a performance degradation.

The paper is organized as follows: section 2 contains a short theoretical introduction while section 3 presents the phase reconstruction algorithm itself. The paper is concluded with section 4 where the performance evaluation can be found.

* This work was supported by the Austrian Science Fund (FWF) START-project FLAME ("Frames and Linear Operators for Acoustical Modeling and Parameter Estimation"; Y 551-N13).

2. THEORY SUMMARY

The STFT of f with respect to a real valued window g is usually defined as

$$(\mathcal{V}_g f)(\omega,t) = \int_{\mathbb{R}} f(\tau+t)g(\tau)\mathrm{e}^{-\mathrm{i}2\pi\omega\tau}\,\mathrm{d}\tau,\ \ \omega,t\in\mathbb{R} \tag{1}$$

$$= M_g^f(\omega,t)\mathrm{e}^{\mathrm{i}\Phi_g^f(\omega,t)}, \tag{2}$$

and the *spectrogram* as the modulus squared. The (complex) logarithm separates the modulus and the phase such as

$$\log(\mathcal{V}_g f)(\omega,t) = \log M_g^f(\omega,t) + \mathrm{i}\Phi_g^f(\omega,t). \tag{3}$$

The Gaussian window with time-frequency support ratio γ

$$\varphi_\gamma(t) = \left(\frac{\gamma}{2}\right)^{-\frac{1}{4}} \mathrm{e}^{-\pi\frac{t^2}{\gamma}} \tag{4}$$

is known to posses optimal properties and, moreover, to allow algebraic treatment of the formulas. In particular, it has been shown that the phase gradient

$$\nabla\Phi_{\varphi_\gamma}^f(\omega,t) = \left(\tfrac{\partial\Phi_{\varphi_\gamma}^f}{\partial\omega}(\omega,t), \tfrac{\partial\Phi_{\varphi_\gamma}^f}{\partial t}(\omega,t)\right) \tag{5}$$

and the gradient of the log-magnitude relate to each other in the following way [9, 11, 10]

$$\frac{\partial\Phi_{\varphi_\gamma}^f}{\partial\omega}(\omega,t) = -\gamma\frac{\partial}{\partial t}\log(M_{\varphi_\gamma}^f(\omega,t)) \tag{6}$$

$$\frac{\partial\Phi_{\varphi_\gamma}^f}{\partial t}(\omega,t) = \frac{1}{\gamma}\frac{\partial}{\partial\omega}\log(M_{\varphi_\gamma}^f(\omega,t)) + 2\pi\omega. \tag{7}$$

In theory, the knowledge of the original phase at a single point $\Phi_{\varphi_\gamma}^f(\omega_0,t_0)$ and the *gradient theorem* would be sufficient to recover the original phase using

$$\Phi_{\varphi_\gamma}^f(\omega,t) = \int_0^1 \nabla\Phi_{\varphi_\gamma}^f\left(L(\tau)\right)\cdot\frac{\mathrm{d}L}{\mathrm{d}\tau}(\tau)\,\mathrm{d}\tau + \Phi_{\varphi_\gamma}^f(\omega_0,t_0), \tag{8}$$

where $L(\tau) = [L_\omega(\tau), L_t(\tau)]$ is any line $(\omega_0, t_0) \to (\omega, t)$. When the phase is unknown completely, one obtains a global constant phase shift. In case of real signals, the global phase shift turns into the reconstructed signal sign ambiguity.

In practice however, and in the real-time setting in particular, several factors make the direct application of such result difficult. First and foremost, the discretization of STFT inevitably introduces aliasing in some form. This causes the relations (6) and (7) not to hold exactly everywhere. Second, one can only work with truncated Gaussian window or with other finitely supported windows, for which the relations hold only approximately. Third, the numerical line integration is prone to error propagation. As a result, the reconstruction algorithm typically produces a phase error with "patches" of constant phase shifts, which is common for all algorithms available. Fig. 1 shows an example of such an error using an excerpt from the *glockenspiel* test signal. It depicts the absolute difference between the original and the reconstructed phase (modulo 2π) taking the circular nature of the phase into account i.e. taking the shorter distance on the circle. The phase error is in rad$/\pi$ and it was set to zeros for very small coefficients.

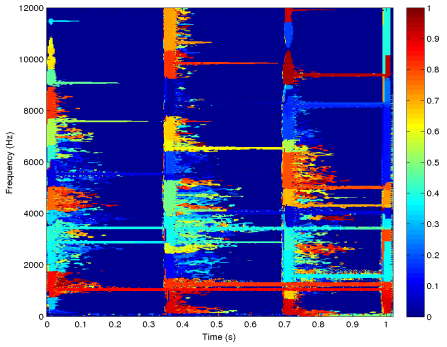

Figure 1: Typical phase difference pattern for the proposed algorithm.

Luckily, it seems that such relative phase shifts of time-frequency components do not degrade the perceived quality substantially.

Even more ambiguity enters in case of modified or completely synthetic spectrograms, for which it is not clear whether (6) and (7) hold at all.

3. THE ALGORITHM

The goal of the present algorithm is to exploit (6),(7) and (8) to recover phase from the magnitude and, ultimately, to reconstruct the original signal.

The first step is obtaining the phase gradient via numerical differentiation and the second one is the numerical line integration on the rectangular grid.

3.1. Phase Gradient Approximation

The discrete version of STFT is defined as

$$(V_g f)(m,n) = \sum_{l\in\mathcal{I}} f(l+na)g(l)\mathrm{e}^{-\mathrm{i}2\pi ml/M} \tag{9}$$

$$= s(m,n)\mathrm{e}^{\mathrm{i}\phi(m,n)}, \tag{10}$$

where $f \in \ell^2(\mathbb{Z})$ and $g \in \ell^2(\mathbb{Z})$ are both real sequences, $m = 0, \ldots, \lfloor M/2 \rfloor$ where M is the number of frequency channels, $n \in \mathbb{Z}$ and the parameter a is the time step in samples. The finite support of g will be bounded such that it is zero outside of the sum index set $\mathcal{I} = \{-\lceil M/2 \rceil + 1, \ldots, \lfloor M/2 \rfloor\}$. The windows will be further restricted to be whole point symmetric such that their discrete-time Fourier transform is real.

The time-frequency ratio γ of the Gaussian window which is closest to the given window g can be obtained as

$$\gamma = C_g \cdot \mathrm{len}(g)^2, \tag{11}$$

where $\mathrm{len}(g)$ determines the length of the window support in samples and the constant C_g is window specific. The values for windows used in this paper can be found in Table 1. They were obtained by a simple heuristic search.

Table 1: C_g constants for (12) for common windows

Hann	0.25645
Hamming	0.29794
Blackman	0.17954

It is also possible to use a truncated discrete Gaussian window $\varphi_\gamma^{\mathrm{T}}$. Assume the window is truncated at the relative height h and it is required to be w samples long; then the γ parameter is obtained as

$$\gamma = -\frac{\pi}{4}\frac{w^2}{\log(h)}. \tag{12}$$

The (scaled) discrete STFT phase gradient $\nabla\phi = (\phi_\omega, \phi_t)$ can be approximated using centered difference scheme

$$\phi_\omega(m,n) = -\frac{\gamma}{2aM}\left(s_{\log}(m,n+1) - s_{\log}(m,n-1)\right), \tag{13}$$

$$\phi_t(m,n) = \frac{aM}{2\gamma}\left(s_{\log}(m+1,n) - s_{\log}(m-1,n)\right) + 2\pi am/M, \tag{14}$$

assuming $s_{\log}(m,n) = \log\left(s(m,n)\right)$ and setting $\phi_t(0,n)$ and $\phi_t(\lfloor M/2 \rfloor, n)$ to zeros. The use of the centered difference scheme ensures the correct alignment of the sampling grids of both the gradient components with the sampling grid of the coefficients. The gradient is scaled such that the lengths of steps in the following numerical integration scheme are equal to 1 in both directions. Alternatively, a causal finite difference scheme can be used for computing ϕ_ω

$$\phi_\omega^{\mathrm{causal}}(m,n) = -\frac{\gamma}{2aM}\left(3s_{\log}(m,n) - 4s_{\log}(m,n-1) + s_{\log}(m,n-2)\right). \tag{15}$$

In general, the misalignment of the sampling grids however introduces a performance degradation.

3.2. Real-Time Heap Integration

The gradient integration is done using the cumulative sum and the trapezoidal rule. The integration paths are chosen adaptively according to the coefficient magnitude following the direction of the ridges fist. The algorithm actually does not use integration paths

directly because only a single coefficient and its immediate neighbors are needed at a time.

Another simplification comes from the fact that the integration is only done in the horizontal or in the vertical directions which makes the (vector) derivative of the line segment ($\frac{\mathrm{d}L}{\mathrm{d}\tau}$ from (8)) to have only one nonzero element. Therefore, only one element of the gradient is active at a time during the integration.

For the purpose of keeping track of the coefficients with already computed phase, the algorithm employs a *heap* data structure which always keeps the coefficient with the largest magnitude at the top. The heap data structure is equipped with efficient insertion and deletion operations. The algorithm further accepts a relative magnitude threshold tol which controls which coefficients are included in the integration. The coefficients below tol are assigned a random phase value. The reasoning for this choice is that when tol is high, the random phase causes less disturbing artifacts when compared to zero phase. The algorithm is summarized as Alg. 1.

In words (assuming $tol = 0$ for simplicity), the algorithm starts of by marking coefficients from the n frame as unknown (line 2) and continues by inserting coefficients from the $n-1$ frame into the *heap* (line 5) making them all potential initial points. The integration itself starts by removing the biggest coefficient from the heap (line 8) and it is used to spread the phase to the only neighbor in the n frame (line 11). The just computed coefficient is marked as known (line 12) and it is then inserted into the heap (line 13) to serve as a potential phase source. The algorithm then continues by selecting and removing the biggest coefficient from the heap and this time, the coefficient from frame n might have been selected (line 16) and in that case, the phase is spread to the neighbors above (line 18) and below (line 23) and both are marked as known and they are inserted into the heap. The algorithm continues until no coefficients with unknown phase are left.

The reconstructed phase $\widehat{\phi}(m,n)$ is combined with the magnitude such that $\widehat{c}(m,n) = s(m,n)\mathrm{e}^{\mathrm{i}\widehat{\phi}(m,n)}$ and the signal $\widehat{f}$ is reconstructed using a dual window and the overlap-add procedure.

In some cases, the phase is partially known. Such information can be easily exploited by altering the algorithm such that the line 5 in Alg. 1 is repeated with indices (m,n) of the coefficients with the known phase of the current frame n.

Algorithm 1: Phase Gradient Heap Integration for n-th frame

Input: Phase time derivative $\phi_t(m,n)$ and magnitude $s(m,n)$ of frames n and $n-1$, phase frequency derivative $\phi_\omega(m,n)$ for frame n, estimated phase $\widehat{\phi}(m,n)$ for frame $n-1$ and relative tolerance tol.

Output: Phase estimate $\widehat{\phi}(m,n)$ for frame n.

1 $abstol \leftarrow tol \cdot \max\big(s(m,n) \cup s(m,n-1)\big)$;
2 Create set $\mathcal{I} = \big\{(m,n) : s(m,n) > abstol\big\}$;
3 Assign random values to $\widehat{\phi}(m,n)_{(m,n)\notin\mathcal{I}}$;
4 Construct a self-sorting *heap* for (m,n) tuples;
5 Insert $(m,n-1)$ for $m = \big(m : s(m,n-1) > abstol\big)$ into the *heap*;
6 **while** $\mathcal{I}$ *is not* $\emptyset$ **do**
7 **while** heap *is not empty* **do**
8 $(m_{heap}, n_{heap}) \leftarrow$ remove the top of the *heap*;
9 **if** $n_{heap} == n-1$ **then**
10 **if** $(m_{heap}, n) \in \mathcal{I}$ **then**
11 $\widehat{\phi}(m_{heap},n) \leftarrow \widehat{\phi}(m_{heap},n-1) + \frac{1}{2}\big(\phi_t(m_{heap},n-1) + \phi_t(m_{heap},n)\big)$;
12 Remove (m_{heap},n) from $\mathcal{I}$;
13 Insert (m_{heap},n) into the *heap*;
14 **end**
15 **end**
16 **if** $n_{heap} == n$ **then**
17 **if** $(m_{heap}+1,n) \in \mathcal{I}$ **then**
18 $\widehat{\phi}(m_{heap}+1,n) \leftarrow \widehat{\phi}(m_{heap},n) + \frac{1}{2}\big(\phi_\omega(m_{heap},n) + \phi_\omega(m_{heap}+1,n)\big)$;
19 Remove $(m_{heap}+1,n)$ from $\mathcal{I}$;
20 Insert $(m_{heap}+1,n)$ into the *heap*;
21 **end**
22 **if** $(m_{heap}-1,n) \in \mathcal{I}$ **then**
23 $\widehat{\phi}(m_{heap}-1,n) \leftarrow \widehat{\phi}(m_{heap},n) - \frac{1}{2}\big(\phi_\omega(m_{heap},n) + \phi_\omega(m_{heap}-1,n)\big)$;
24 Remove $(m_{heap}-1,n)$ from $\mathcal{I}$;
25 Insert $(m_{heap}-1,n)$ into the *heap*;
26 **end**
27 **end**
28 **end**
29 **end**

4. EVALUATION

In the experiments we used the following error measure introduced in [1]

$$E = \frac{\left\| s - \left| V_g \widehat{f} \right| \right\|_{\text{fro}}}{\|s\|_{\text{fro}}}, \tag{16}$$

where s is the target magnitude spectrogram, $\widehat{f}$ is the reconstructed signal and $\|\cdot\|_{\text{fro}}$ denotes the Frobenius norm. The transform V_g uses the same parameters (g, a and M) as the one used to obtain s. Values in decibels are obtained by $20\log_{10}(E)$. It has often been pointed out that such error measure does not reflect error of the reconstructed signal (see e.g. [3]) nor the actual perceived quality degradation. Therefore it should only be considered as a rough indicator. Listening tests would have been conducted in order to get a fair comparison.

For the tests, we used the SQAM database [12] which consists of 70 recordings sampled at 44.1 kHz. Only the first 10 seconds from the first channel of each sound sample was used in the evaluation.

The code (runnable in Matlab or GNU Octave[13]) for reproducing tables presented in this manuscript are freely available[1]. Note that LTFAT – Large Time-Frequency Analysis Toolbox[2] [14] (version 2.1.2 and above) and PHASERET – Phase Retrieval Toolbox[3] (version 0.1.0 and above) are required in order to run the scripts. Both toolboxes are freely available.

In the tests, 4 different compactly supported windows of full FFT length $M = 2048$ are used. The Gaussian window was truncated at relative height $h = 0.01$. The hop factor a and therefore the redundancy M/a was varied.

[1] `http://ltfat.github.io/notes/043`
[2] `http://ltfat.github.io`
[3] `http://ltfat.github.io/phaseret`

4.1. Performance Comparison

We evaluate the performance of variants of the algorithms for zero (0) or one (1) lookahead frames. The SPSI algorithm naturally falls just into the former category, while RTISI-LA can benefit from more than one lookahead frame. The relative tolerance of the proposed algorithm was set to 10^{-6}. For the RTISI-LA algorithm an asymmetric window and 16 per-frame iterations were used. All tested algorithms are able to run in real-time in the tested setting.

Tables 2, 3 and 4 show average E in dB for the test database for $a = 512$, $a = 256$ and $a = 128$ respectively. Scores for individual files and Matlab/GNU Octave scripts reproducing the results can be found at the accompanying web page.

Table 2: Algorithm comparison, $a = 512$.

	Gauss	Hann	Hamming	Blackman
SPSI	-17.82	-16.72	-16.53	-17.62
RTPGHI (0)	-17.76	-17.50	-17.58	-17.82
RTISI (0)	**-22.80**	**-21.75**	**-21.08**	**-22.82**
RTPGHI (1)	-20.91	-20.25	-20.07	-20.76
RTISI-LA (1)	**-26.08**	**-25.14**	**-24.01**	**-26.63**

Table 3: Algorithm comparison, $a = 256$.

	Gauss	Hann	Hamming	Blackman
SPSI	-17.88	-17.09	-16.79	-17.79
RTPGHI (0)	**-21.79**	**-21.10**	**-21.01**	**-21.78**
RTISI (0)	-21.15	-20.20	-19.53	-21.07
RTPGHI (1)	**-24.87**	**-22.74**	**-21.98**	**-24.59**
RTISI-LA (1)	-22.11	-19.90	-19.14	-21.90

Table 4: Algorithm comparison, $a = 128$.

	Gauss	Hann	Hamming	Blackman
SPSI	-17.99	-17.16	-16.82	-17.92
RTPGHI (0)	**-26.13**	**-23.48**	**-22.33**	**-25.93**
RTISI (0)	-20.18	-19.70	-19.01	-20.35
RTPGHI (1)	**-26.83**	**-23.50**	**-22.47**	**-26.21**
RTISI-LA (1)	-17.85	-16.66	-16.12	-17.71

Table 2 clearly shows that the RTISI-LA algorithm is superior when a bigger hop factor ($a = 512$) is used.

Table 3 already shows improvement for the proposed algorithm and in table 4, the proposed algorithm performs the best by a large margin. Another observation is that the performance gap between both variants of the proposed algorithm is virtually nonexistent when high enough overlap is used.

While the proposed algorithm clearly benefits from a higher window overlap due to the reduction of the aliasing, it is not true for the other two algorithms. The error achieved by SPSI seems to be unaffected and for RTISI it actually grows. This is a known property of RTISI and it is explained in [4]. Moreover, since higher overlap STFTs are more robust to (unwanted) coefficient perturbations the reconstructed signals tend to sound better than lower window overlap STFT reconstruction with the same error E.

The performance of the RTISI-LA algorithm can be obviously further improved by employing more lookahead frames and more iterations. Each lookahead frame however introduces additional a samples to the overall processing delay and only a limited number of per-frame iterations can be done within the real-time deadline restriction.

A real-time demo script comparing all three algorithms can be found in the PHASERET toolbox [4].

4.2. Modified Spectrogram

In order to compare performance of the algorithms on modified spectrograms, we implemented comb-filter free audio mixing of two signals from [5]. The phaseless reconstruction is done with the sum of STFT modulus of a signal with its slightly delayed version. Since usage of any objective measure does not seem to be relevant for a systematic comparison, we only provide the real-time implementation and leave the evaluation to interested readers. The demo is available in the PHASERET toolbox [5].

4.3. Timing

All three algorithms were implemented in C and per-frame execution times were compared in Tab. 5 for the same setting as in Tab. 3. Because the execution time of the present algorithm is highly signal dependent, we are mainly interested in the worst-case performance, which is also the crucial indicator for the real-time setting. The implementations were done in such a way that no memory allocation occurs during the execution.

Table 5: Worst per-frame exetution time in ms, from $\sim 10^5$ frames

SPSI	RTPGHI	RTISI-LA			
		1 it.	4 it.	8 it.	16 .it
0.23	0.56	0.31	0.58	1.16	2.18

The benchmark was run with the highest priority on an idle PC equipped with Intel Core i5-4570@3.20GHz and 8 GB RAM running Xubuntu 14.04.3 LTS with generic kernel 3.13.0-65. GCC 5.3.0 with the `-O3` optimization switch was used to produce the binary. The FFTW3 library [15] was used for computing (shortened) FFT for real signals with the `FFTW_MEASURE` plan option. Though we cannot rule out the possibility that the implementations are suboptimal (no explitic attempt was made to exploit SSE/AVX instruction sets or paralelization of the code), we tried to use the best practices for all of them. Moreover, the source code of the benchmark is available at the accompanying webpage.

5. CONCLUSION

A novel real-time algorithm for signal reconstruction from the STFT magnitude was presented. The tests showed that for fixed delay it is able to outperform the state-of-the-art algorithms when high enough window overlap is used. Moreover, the benchmark shows that the worst per-frame execution time of the proposed algorithm is about twice as much as of the SPSI algorithm and rouhly equal to execution time of the RTISI-LA algorithm with 4 iterations.

[4] `demo_blockproc_phaseret.m`

[5] `demo_blockproc_phaseretmix.m`

6. REFERENCES

[1] D. Griffin and J.S. Lim, "Signal estimation from modified short-time Fourier transform," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 32, no. 2, pp. 236–243, Apr 1984.

[2] J. Laroche and M. Dolson, "Improved phase vocoder time-scale modification of audio," *Speech and Audio Processing, IEEE Transactions on*, vol. 7, no. 3, pp. 323–332, May 1999.

[3] Nicolas Sturmel and Laurent Daudet, "Signal reconstruction from STFT magnitude: A state of the art," *Proc. Int. Conf. Digital Audio Effects DAFx*, vol. 2012, pp. 375–386, 2011.

[4] Xinglei Zhu, Gerald T. Beauregard, and Lonce Wyse, "Real-time signal estimation from modified short-time Fourier transform magnitude spectra," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 15, no. 5, pp. 1645–1653, July 2007.

[5] Volker Gnann and Martin Spiertz, "Comb-filter free audio mixing using STFT magnitude spectra and phase estimation," in *Proc. 11th International Conference on Digital Audio Effects (DAFx-08)*, Sept. 2008.

[6] Volker Gnann and Martin Spiertz, "Inversion of STFT magnitude spectrograms with adaptive window lengths," in *Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing ICASSP '09*, Apr. 2009, pp. 325–328.

[7] Volker Gnann and Martin Spiertz, "Improving RTISI phase estimation with energy order and phase unwrapping," in *Proc. 13th International Conference on Digital Audio Effects (DAFx-10)*, Sept. 2010.

[8] Gerald T. Beauregard, Mithila Harish, and Lonce Wyse, "Single pass spectrogram inversion," in *Digital Signal Processing (DSP), 2015 IEEE International Conference on*, July 2015, pp. 427–431.

[9] Michael R. Portnoff, "Magnitude-phase relationships for short-time Fourier transforms based on Gaussian analysis windows," in *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '79.*, Apr 1979, vol. 4, pp. 186–189.

[10] Zdeněk Průša, Peter Balazs, and Peter L. Søndergaard, "A Non-iterative Method for (Re)Construction of Phase from STFT magnitude," 2016, Preprint available from http://ltfat.github.io/notes/ltfatnote040.pdf.

[11] F. Auger, E. Chassande-Mottin, and P. Flandrin, "On phase-magnitude relationships in the short-time Fourier transform," *Signal Processing Letters, IEEE*, vol. 19, no. 5, pp. 267–270, May 2012.

[12] "Tech 3253: Sound Quality Assessment Material recordings for subjective tests," Tech. Rep., The European Broadcasting Union, Geneva, Sept. 2008.

[13] John W. Eaton, David Bateman, Søren Hauberg, and Rik Wehbring, *GNU Octave version 4.0.0 manual: a high-level interactive language for numerical computations*, 2015.

[14] Zdeněk Průša, Peter L. Søndergaard, Nicki Holighaus, Christoph Wiesmeyr, and Peter Balazs, "The Large Time-Frequency Analysis Toolbox 2.0," in *Sound, Music, and Motion*, Lecture Notes in Computer Science, pp. 419–442. Springer International Publishing, 2014.

[15] Matteo Frigo and Steven G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, Special issue on "Program Generation, Optimization, and Platform Adaptation".