Techniques for the rapid calculation of the excitation pattern in the time varying extensions to ANSI S3.4-2007

S. Hales Swift, and Kent L. Gee

Citation: Proc. Mtgs. Acoust. **36**, 040002 (2019); doi: 10.1121/2.0001206
View online: https://doi.org/10.1121/2.0001206
View Table of Contents: https://asa.scitation.org/toc/pma/36/1
Published by the Acoustical Society of America

ASA
POMA
Proceedings of Meetings on Acoustics
Turn Your ASA Presentations and Posters into Published Papers!

Volume 36 http://acousticalsociety.org/

177th Meeting of the Acoustical Society of America

Louisville, Kentucky

13-17 May 2019

Noise: Paper 4pNS2

Techniques for the rapid calculation of the excitation pattern in the time varying extensions to ANSI S3.4-2007

S. Hales Swift and Kent L. Gee
Department of Physics and Astronomy, Brigham Young University, Provo, UT, 84602; hales.swift@gmail.com; kentgee@byu.edu

The ANSI S3.4-2007 standard gives a method for calculating the predicted loudness of stationary sounds for an average listener. Glasberg and Moore (2002) provide an extension of the method to time-varying sounds. The mathematical structure of the excitation in the loudness calculation is amenable to significant acceleration in MATLAB by expressing the entirety of the calculation representing the cochlear filtering process in terms of matrices and array operations. Thus, procedures to achieve rapid processing of loudness are set forth. The procedures explained here are also applicable to other metrics in this family including those in ISO 532-2 and the upcoming ISO 532-3. Further steps for obtaining faster than real-time processing of loudness employing approximation are explained.

Published by the Acoustical Society of America

1. INTRODUCTION

Sounds quality metrics serve an important role by providing inexpensive estimates of human perceptual experience of sound. By providing estimates of perceived sound attributes at minimal cost relative to jury studies, metrics make consideration of sound quality a viable point of comparison and competitive advantage among products, and enable deliberate improvement. Of the documented dimensions of sound quality, loudness has consistently ranked as the most salient. Consider, for example, its prominent role in models of psychoacoustic annoyance (Widmann, 1992) and sensory pleasantness (Fastl and Zwicker, 1999) or, on a more humorous note, to the number of loudness complaints directed against young children despite the fact that such terms as sharpness may, at times, be equally valid.

Multiple methods for predicting loudness currently exist. Among internationally standardized metrics, ISO 532-1 embodies Zwicker's method for calculating loudness and includes an extension to time varying sounds. ISO 532-2 embodies the method of Moore and Glasberg and coworkers for calculating stationary loudness based on (Moore, Glasberg, Baer, 1997) and additionally includes binaural inhibition (Moore and Glasberg, 2007), while ANSI S3.4-2007 also is based on Moore et al.'s metric, but without the binaural inhibition mechanics. Currently, ISO 532-3 is under development and will include the time varying extensions to the version of Moore et al.'s metric with binaural inhibition (Moore, Glasberg, Varathanathan, Schlittenlacher, 2016).

Between the two main branches of loudness metrics treated in the ISO standards—Moore's and Zwicker's—some advantages have accrued to each side. Moore's model embodies a more recent model, the $\mathrm{ERB_N}$ scale, of the spread of masking and excitation in the cochlea as well as much more fine-grained resolution of excitation in the cochlea through a closely spaced set of filters to represent the middle ear (Moore, Glasberg, Baer, 1997). This proves to be both one of its more notable advantages and disadvantages, because the close spacing provides both fine resolution and longer-than-desired computation times.

In consequence of this, methods to speed up the calculation of loudness are highly desirable inasmuch as they can allow enjoyment of the advantages of Moore's metric while ameliorating a chief drawback. This is especially of interest in areas such as edge computing, where the ability to perform metric calculations with modest hardware could enable improved representation of the soundscape of the human environment. This paper will focus specifically on the construction of a very fast implementation of the ANSI S3.4-2007 standard with its published but not standardized extension to time varying sounds (Glasberg and Moore, 2002) in MATLAB®. The chief bottleneck in the calculation is the determination of the excitation pattern in the cochlea. Because this is the most time consuming feature in both this and the later metrics and because the calculation of the excitation pattern is performed similarly in the more recent metrics of this family (Moore and Glasberg, 2007; ISO 532-2; Moore, Glasberg, Varathanathan, Schlittenlacher, 2016; and thus ISO 532-3), the technique shown here for accelerating the code will apply with little to no modification to the later generations of metrics as well as to the simpler example at hand. Particularly, in the calculation of the excitation pattern dealt with in this paper, no modification is needed except that it will need to be applied to two ears rather than one.

Occasionally, a certain amount of skepticism is elicited on the subject of using MATLAB® to write fast codes. It is true that MATLAB®does not always perform certain operations—notably some types of loops—as quickly as might be desired. However, it is extremely fast at linear algebra, having been designed to this purpose. William Stein said, "*Mathematics is the art of* *reducing any problem to linear algebra*," (Humpherys, Jarvis, Evans, 2017) and the approach taken in this paper will thus be to largely determine ways in which to recast the calculation of the excitation pattern as a linear algebra problem. One caveat about the techniques used here is that they trade higher memory usage for higher speed, and thus may not be the best approach for all systems in their simplest form. However, with appropriate division of the signal or its running spectra into blocks, this could likely be overcome for most systems. Also worth noting, in order to get optimum performance, it may be necessary to close browsers or other applications with high memory consumption.

First, however, when speeding up MATLAB®codes, several best practices and pieces of information are worth keeping in mind:

- Suppress all unnecessary output by using semicolons as writing output can have a substantial computational cost
- Preallocate all arrays that would otherwise grow in size so that arrays do not have to be dynamically resized
- MATLAB®is column major like Fortran (unlike most of the C-language family) so there is often a best way to choose the shape of a matrix
- Minimize unnecessary calculations inside of loops, which are not one of MATLAB®'s greater strengths
- Recast the metric model matrix or array operations in order to take advantage of MATLAB®'s fast algorithms for array handling
- Take advantage of sparse matrix routines where possible

A number of these will receive direct comment, and those that do not also played a modest role, but would tend to complicate the discussion more than enlighten.

2. MATHEMATICS AND METHODS

As was stated above, the function that calculates the cochlear excitation pattern is easily the most demanding part of the algorithm, routinely involving well over half of the computational effort in the implementations we have seen. Thus, optimizing this portion of the implementation is critical to a rapid calculation. The excitation pattern calculation includes several steps:

1. Calculate the input power per critical band using a static filter centered at each input spectrum frequency at each time step
2. Use the input power from the previous step to determine the shape of the level-dependent filters at each time step for each incoming component
3. Multiply the level-dependent filters (this time centered at increments of the ERB or "Cam" scale) with the corresponding spectrum for each time step to determine the output power in each portion of the cochlea

The first step is often executed in a loop, but MATLAB® is not ideally suited for loops and it can just as easily be expressed as a matrix. Because the filters used for calculating the power within a critical band of each input component do not change shape with level, and because there is one input and one output per input frequency, they can be easily expressed as a square matrix F^1_{static} of dimension $N_{\mathrm{spectra}} \times N_{\mathrm{spectra}}$ where N_{spectra} is the number of frequency lines in the incoming spectrum as shown in Fig. 1, where values range between nearly zero away from the diagonal to one at the diagonal. The input spectrum can thus be expressed as a $N_{\mathrm{spectra}} \times N_{\mathrm{time\ steps}}$ matrix P_{input}, where $N_{\mathrm{time\ steps}}$ is the number of time steps being calculated, and the product $F^1_{\mathrm{static}} P_{input} = X$, which also has dimensions of $N_{\mathrm{spectra}} \times N_{\mathrm{time\ steps}}$, and is used later in the calculation of the level-dependent filters.

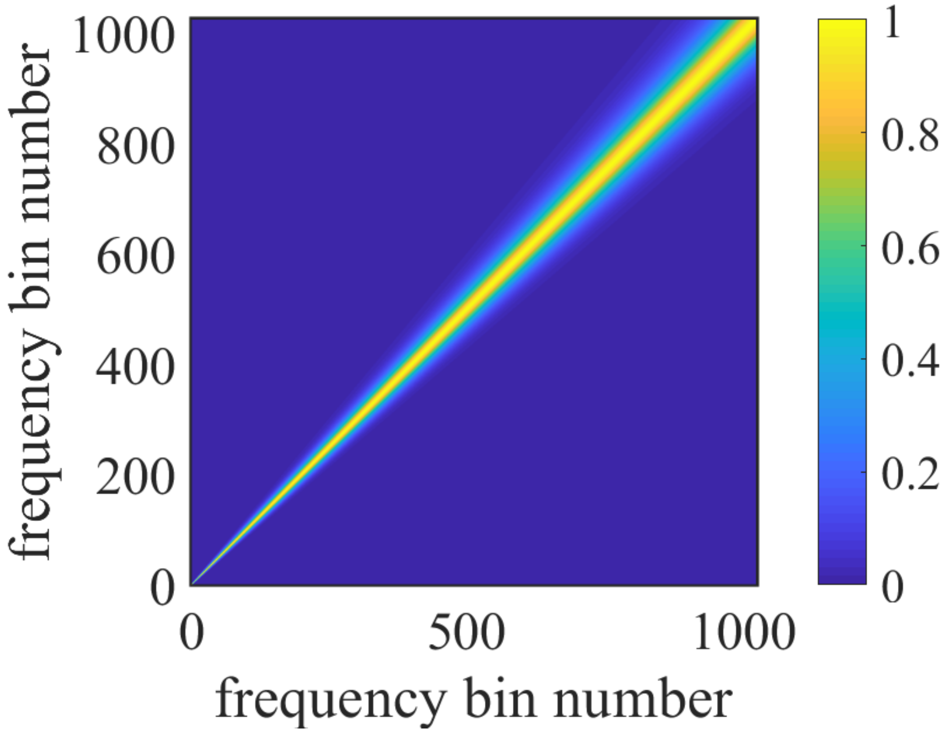

Figure 1: Filter used for determining the input power in a critical band centered at each of the input components.

While the input power at each critical band can be straightforwardly expressed as a matrix product (and doing so tends to yield some easy savings), the level-dependent filter calculations in steps two and three require something a bit more creative. The first instance where saving can be achieved in the level-dependent filter calculation is upon realizing that only part of the filters are actually level-dependent. We will call the filter that is determined in step 2 and used in step 3 the dynamic filter for convenience, though this term is not used in the standard. The dynamic filter, F_{dynamic}, is the sum of static and level-dependent portions, $F_{\mathrm{dynamic}} = F_{\mathrm{static}} + F_{\mathrm{level\ dependent}}$ as seen in Fig. 2, with most of the matrix actually static (blue).

part inside the loop each time step

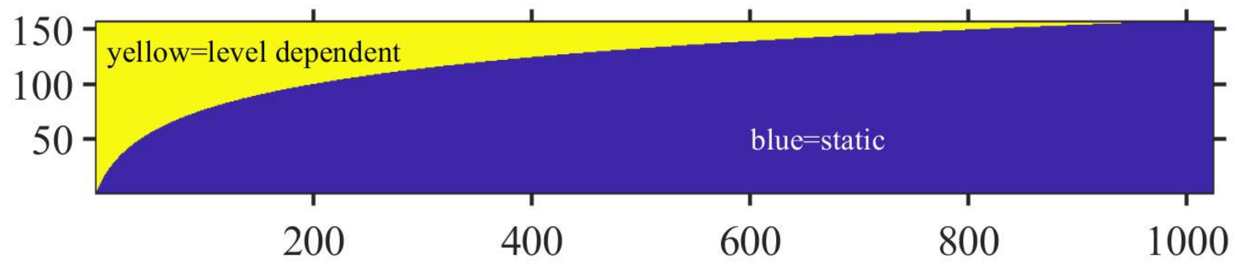

Figure 2: Static and dynamic parts of the level-dependent filters used to calculate cochlear excitation. The input spectral band elements (with 10 Hz per entry as shown) are the horizontal coordinate and the vertical axis corresponds to the output elements (filters centered at 1/10th ERB$_N$ intervals).

Because only part—in fact, the lesser part—of the filter elements are level-dependent, the static part can be separated and use the same matrix multiplication approach that was used with the input power calculation, so that $P_{\text{static output}} = F_{\text{static}} \times P_{\text{input}}$. This part can then be set aside until recombining with the level-dependent output in the final step.

For the level-dependent part of the operation, things get a bit more complicated still. The main complication is that at each time step it is necessary to essentially compose a matrix, different in general from time step to time step, and then multiply the input spectrum for that time step by the assembled matrix. This is typically performed in a loop, but MATLAB® is not overly good at loops, so it is best to avoid using them if practical. One way to avoid using a loop in this case, is to realize that the level-dependent part of the matrix can be reshaped into an array. The individual elements of the matrix are thus stacked into a single vector—one vector per time step which includes all the level-dependent parts of the filter. The input spectrum can be similarly broadcasted into an array of the same size with input power elements corresponding to each of the filter elements it would ordinarily multiply. They can then be multiplied together using array operations and a matrix operation can send the element by element product into a matrix that gives the excitation in each ERB at 0.1-ERB intervals in one dimension and time in the other without ever using a loop. The steps for doing this are shown in detail below using examples from a MATLAB® code included in the appendix.

1. Determine which indices are level-dependent and which are not; this involves finding the indices of the level-dependent set by comparing `fvector`, a list of frequencies of the incoming line spectrum, and `cf`, a list of the center frequencies of the auditory filters to determine the elements for which the input frequency is below the center frequency:

   ```
   idx=repmat(cf',1,N)>repmat(fvector,length(cf),1);
   ```

   only the lower side of the filter is level-dependent, so only if `fvector` is the smaller is it level-dependent.

2. Make a matrix which will broadcast the power spectra to each of the elements of the reshaped filter that interact with it and a matrix that will sum the product of the filter array and the power array and project them to the desired ERB_N output scale:

```
test1=1:N;
test2=1:length(cams);
A=repmat(test1,length(cf),1);
B=repmat(test2',1,N);
a=A(idx);
b=B(idx);
broadcast=sparse(1:length(a),a,ones(1,length(a)),length(a),N);
condense=sparse(b,1:length(b),ones(1,length(b)),length(cams),...
    length(b));
```

Nota bene: the Einstein summation convention does not apply here. A single line, (e.g., the first line) of the input spectrum, $P^{\text{in}}_{k,1}$, multiplied by the level-dependent filters, $F_{i,k}$, should work like

$$P^{\text{out}}_{i,1} = \sum_{k=1}^{n} F_{i,k} P^{\text{in}}_{k,1}, \tag{1}$$

following the conventions for ordinary matrix multiplication. In reality, however, the level-dependent filter matrix changes with each time level, so it has a third index t, thus $F_{i,k}$ is really $F_{i,k,t}$, and the input spectrum also has a dimension of time index, $P^{\text{in}}_{k,t}$. This means that one could think of the level-dependent filter operation as involving a sort of product of a 3-dimensional array and a 2-dimensional array to yield a 2-dimensional array or as an array (indexed by time) of matrix vector products

$$P^{\text{out}}_{i,t} = \sum_{k=1}^{n} F_{i,k,t} P^{\text{in}}_{k,t}. \tag{2}$$

Now part of the matrix does not actually change with level, and this is, in fact, most of the matrix as shown in Fig. 2, so we can split the matrix product into level-dependent and non-level-dependent parts

$$P^{\text{out}}_{i,t} = \sum_{k \in \text{idx(i)}} F_{i,k,t} P^{\text{in}}_{k,t} + \sum_{k \notin \text{idx(i)}} F_{i,k,t} P^{\text{in}}_{k,t} = \sum_{k \in \text{idx(i)}} F_{i,k,t} P^{\text{in}}_{k,t} + \sum_{k \notin \text{idx(i)}} F_{i,k} P^{\text{in}}_{k,t}, \tag{3}$$

where it should be noted that the last summation is the ordinary matrix product of the static part of F with P^{in}, resulting in

$$P^{\text{out}}_{i,t} = \left(\sum_{k \in \text{idx(i)}} F_{i,k,t} P^{\text{in}}_{k,t} \right) + F_{static} P^{\text{in}}. \tag{4}$$

The static matrix multiplication, the second term on right of the equal sign, has a substantial zero-valued component and so can be accelerated with sparse matrix handling. The first term on the right hand side of the equation continues to have a time dependency, which at first glance would seem to require it be dealt with in a loop, however, another option exists. Define a pair of addressing or indexing functions $a(e)$ and $b(e)$ that sweep over the values of k and i, respectively, that are level-dependent. e is an independent parameter with values between one and the total number of level-dependent elements (inclusive) that indexes the level-dependent elements, so that each element has a unique number. We can then rewrite the level-dependent part of the operation as

$$\sum_{k \in \mathrm{idx(i)}} F_{i,k,t} P^{\mathrm{in}}_{k,t} = \sum_{e=1}^{N_{\mathrm{L.D.}}} \delta_{i,b(e)} F_{b(e),a(e),t} P^{\mathrm{in}}_{a(e),t} = \sum_{e=1}^{N_{\mathrm{L.D.}}} \delta_{i,b(e)} f(e,t) p^{\mathrm{in}}_{e,t}, \tag{5}$$

where, for every t, we have effectively reshaped $F_{b(e),a(e),t}$ into the vector $f(e,t)^T$ and where we have broadcast P^{in} into p^{in}, making it possible to compute the output power as the product of the matrix $\delta_{i,j}$, containing only ones and zeros, with the element by element array product $f(e,t)p^{\mathrm{in}}_{e,t}$ without making use of a for loop. Recasting the process as matrix and array operations takes advantage of well-designed array operations to boost efficiency.

At this point, it makes sense to briefly review the equations that are used to construct the filter. These come from ANSI S3.4-2007.[7] The equivalent rectangular bandwidth for a normally hearing person, $\mathrm{ERB_N}$, is

$$\mathrm{ERB_N} = 24.673(0.004368 f_c + 1), \tag{6}$$

with f_c the center frequency of the band in question. The filter shape is given by

$$W(g) = (1 + pg)\exp(-pg), \tag{7}$$

where p is a filter parameter which can be different for the upper and lower slopes of the filter, and $g(f, f_c) = \frac{|f-f_c|}{f_c} = \left|\frac{f}{f_c} - 1\right|$, where f_c is the filter center frequency and f is the frequency at which the filter's response is being evaluated, but g is restricted to a maximum value of 4 on the high frequency side of the filter. $p = \frac{4f_c}{\mathrm{ERB_N}}$ for the initial (not level-dependent) filter calculation shown in Fig. 1. Finally, for the level-dependent filter,

$$p_l(X) = p_l(51) - 0.35(p_l(51)/p_l(51, 1k))(X - 51) \tag{8}$$

gives the level-dependent filter shape parameter, which controls the lower filter slope only in the level-dependent stage (the upper filter slope is static).

The equations are next algebraically manipulated in order to reduce the number of machine operations required for their calculation. Beginning with the filter shape equation for the static input power calculation, which is also used for the stationary upper slopes of the filters in the output power filters,

$$p = \frac{4f_c}{\mathrm{ERB_N}} = \frac{4f_c}{24.673(0.004368 f_c + 1)} = \frac{0.162120536618976}{0.004368 + 1/f_c}. \tag{9}$$

The explicitly level-dependent filter shape parameter function given in Eq. 8 can likewise be numerically simplified to

$$p_l(X) = p\left(1 + \frac{51 * 0.35}{p(1k)}\right) - 0.35\frac{p}{p(1k)}X = \frac{0.257939336618976}{0.004368 + 1/f_c} - \frac{0.0018788}{0.004368 + 1/f_c}X. \tag{10}$$

3. RESULTS, AND DISCUSSION OF FUTURE POSSIBILITIES

Using a calculation of the excitation pattern as described above, a test program (included at MATLAB File Exchange® https://www.mathworks.com/matlabcentral/fileexchange/73808-codes-for-the-rapid-calculation-of-loudness-and-sharpness gives results consistent with the results published in the standard at the listed level of precision. We have shown one additional significant figure to give an idea of the size of the roundoff error in Tables 1 and 2. All stationary loudness values are consistent with the standard to within the precision with which they are given in the standard. It is important to realize that values using the time-varying features may not accord precisely with those in the stationary standard. The auditory filter spacing recommended in the Glasberg and Moore extension of the method to time-varying sounds is somewhat greater, potentially diminishing accuracy (though increasing efficiency). Additionally, the signal processing involved in the moving windows of the time-varying calculation leads to some spectral spreading, which would tend increase estimates for tones, while likely having less impact for noise. The time-varying results for the tones tabulated in the standard tend to be within 1-2 dB of the stationary results listed in the standard.

Table 1: Loudness values for 1000 Hz sinusoids from the standard and test

L (dB)	N_{ANSI} (sones)	N_{ANSI} (phon)	N_{test} (sones)	N_{test} (phon)	$N_{test\text{-}TV}$ (phon)
10	0.03	10	0.029	9.9	10.1
20	0.14	20	0.142	19.9	20.6
30	0.42	30	0.423	30.0	30.8
40	1.0	40	1.00	40.0	40.9
50	2.1	50	2.10	50.0	50.8
60	4.2	60	4.17	60.0	60.6
70	8.1	70	8.10	70.0	70.4
80	16.0	80	15.96	80.0	80.2

Table 2: Loudness values for 3000 Hz sinusoids from the standard and test

L (dB)	N_{ANSI} (sones)	N_{ANSI} (phon)	N_{test} (sones)	N_{test} (phon)	N_{TV} (phon)
20	0.35	28	0.348	28.0	29.5
40	1.8	48	1.82	48.0	49.7
60	7.1	68	7.09	68.0	69.0
80	27.5	87.5	27.48	87.45	87.7

With this approach, it is possible to get a total execution time around 1.1 seconds of calculation per second of signal on the laptop computer where the method was tested (Microsoft Surface Book 2, Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz, 4 Cores, 16.0GB RAM), which is within 10% of real-time performance. However, it may be possible to do better. If GPUs could be used to accelerate matrix products, then this might once again boost the speed provided communication times were not overly high. No effort at parallelization has been undertaken here, which might also be possible, but might also be excessive. Suffice to say that, in principle, one could pass chunks of the input power spectrum for a range of time indices to many processors. Provided that a balance were maintained between efficiency of computation and efficiency of communication, this approach could be expected to speed up calculations considerably, but there does not currently seem to be much demand for this intensive of an approach.

Perhaps a more interesting approach has to do with limiting filters that do not contribute much energy. If filter skirts are truncated and set to zero, say, 50 dB down from their maximum response, a good deal more of the level-dependent filters can be cut out of the calculation and one can accelerate the code by another factor of two or so without too much loss in prediction as verified in the course of this study. Though this is not discussed in the derivation above, which is focused on implementing the standard exactly as written, the accompanying code contains a commented out section which, if enabled, enacts this possibility. If one were ambitious, it might be interesting to make an asymptotic analysis of the filter skirts and come up with an optimal adjustment to the last included element of the filter to account for what was being cut off, or include more widely spaced filter elements at lower levels in order to keep a roughly uniform level of sensitivity, i.e., the rough product of the filter magnitude and the spacing of filter elements were kept equal to some constant. The determination of whether any of these approaches is viable in a given context comes down to the question of whether a given decrease in precision or accuracy is acceptable in light of a given increase in computational efficiency.

To reiterate a point made above, the calculation of the excitation pattern is, to the best of our knowledge, unchanged between the ANSI S3.4-2007 and ISO 532-2 and the upcoming ISO 532-3. Thus the approach taken here can be applied to calculations using these later algorithms. Constants used in later stages of the calculation, particularly the calculation of loudness from the excitation pattern, however, do change in connection with the interaural inhibition modeling of the later ISO 532-2 and 532-3, and this must be taken into account to produce accurate values for these methods.

4. CONCLUSION

A method for rapid calculation of the excitation pattern in ANSI S3.4-2007 in MATLAB® has been demonstrated. This acceleration was accomplished by algebraicly pre-evaluating the equations to reduce the amount of computational overhead for key variables, and also recasting the interactions between the input spectrum and the filters as a set of matrix and array products. Loops were avoided by reshaping the 3-dimensional array containing the level-dependent filters through time into a two dimensional array which could then be calculated using array operations and multiplied by a broadcast version of the input spectrum using element by element array multiplication. This made it possible to both avoid MATLAB®'s weaknesses and play to its strengths. Further developments could reasonably bridge the gap between this family of metrics and real-time performance on available hardware.

An implementation of the methods described in this paper can be found on the MATLAB® File Exchange website at https://www.mathworks.com/matlabcentral/fileexchange/73808-codes-for-the-rapid-calculation-of-loudness-and-sharpness (with the usual lack of any guarantee of fitness) with the other functions necessary for its use. The function for calculating excitation by the method of this paper is additionally included in an appendix.

ACKNOWLEDGMENTS

Parts of this code were developed while the first author was at Brigham Young University and ERDC-CERL, though key optimization steps highlighted here were developed thereafter.

REFERENCES

[1] U. Widmann, *Ein Modell der Psychoakustischen Lästigkeit von Schallen und seine Anwendung in der Praxis der Lärmbeurteilung*. Dissertation TU München (1992a).

[2] H. Fastl and E. Zwicker. *Psychoacoustics: Facts and models* (2007).

[3] ISO (2017). ISO 532-1-2017. *Acoustics—Methods for Calculating LoudnessPart 1: Zwicker Method* (International Organization for Standardization, Geneva).

[4] ISO (2017). ISO 532-2-2017. *AcousticsMethods for Calculating LoudnessPart 2: Moore-Glasberg Method* (International Organization for Standardization, Geneva).

[5] B. C. J. Moore, B. R. Glasberg, T. Baer, "A model for the prediction of thresholds, loudness and partial loudness," *J. Audio Eng. Soc.*, **45**, 224-240 (1997).

[6] B. C. J. Moore and B. R. Glasberg, "Modeling binaural loudness," *J. Acoust. Soc. Am.*, **121**, 1604-1612 (2007).

[7] ANSI (2007). ANSI S3.4-2007. *Procedure for the Computation of Loudness of Steady Sounds* (American National Standards Institute, Washington, DC).

[8] B. C. J. Moore, B. R. Glasberg, A. Varathanathan, and J. Schlittenlacher, "A loudness model for time-varying sounds incorporating binaural inhibition," *Trends Hear.*, **20**, 1-16 (2016).

[9] B. R. Glasberg and B. C. J. Moore, "Prediction of absolute thresholds and equal-loudness contours using a modified loudness model," *J. Acoust. Soc. Am.* **120**, 585-588 (2006).

[10] J. Humpherys, T. J. Jarvis, and E. J. Evans. *Foundations of Applied Mathematics, Volume I: Mathematical Analysis* Vol. 152. SIAM, (2017).

APPENDIX A

```
function [excitation] = auditoryfilters_h(matrox,df,dCam)
%% auditoryfilters
% The auditory filters function and calculates the shape and
% response of the auditory filters to the power spectrum arriving at the
% cochlea spaced in frequency increments of df. According to ANSI S3.4-2007
% df should be 10 Hz.
% dCam is also adjustable, but .1 is used in ISO 532-2/ANSI S3.4-2007 and .25 in ISO ↙
532-3
%% Set up various numerical arrays needed for the computation
[M,N]=size(matrox); % Determine number of time steps M and frequency bins N
ind=1:N;
fvector=ind*df; % gives the frequencies of the incoming spectral components
cams=dCam:dCam:39.0;
if dCam==.1
    activeIdxs=find(cams>=1.8&cams<=38.9);
    cams=dCam:dCam:38.9;
elseif dCam==.25
    activeIdxs=find(cams>=1.75&cams<=39.0);
else
    activeIdxs=find(cams>=1.75&cams<=39.0);
end
cams=cams(activeIdxs);
cf=(10.^(cams/21.366)-1)/.004368; % list of auditory filter center frequencies
invcf=1./cf; % gives the multiplicative inverse of the filter center frequencies

%% Determine level dependent filter elements to speed up processing
idx=repmat(cf',1,N)>repmat(fvector,length(cf),1);

%% Calculate static filter centered at each input component and associated power
g=abs(1./fvector'*fvector-1); % calculate normalized deviation from center frequency
p0=0.162120536618976./(0.004368+1./fvector); % find p value for initial calculation ↙
(ANSI S3.4-2007 equation 1)
p0g=repmat(p0',1,N).*g; % calculate product pg from ANSI equation 2
W00=(1+p0g).*exp(-p0g); % calculate static filter shape (ANSI equation 2)
W00(g>4)=0;
epsilon=1e-20; % avoids problems with logarithm function in power calculation
X=10*log10(epsilon+matrox*W00'); % calculate power through each filter at each time

%% Calculate output of static part of dynamic filter
g2=abs(invcf'*fvector-1); % calculate normalized deviation from center frequency
g2(g2>4)=4;
p02=0.162120536618976./(0.004368+invcf);% find static p value for initial calculation ↙
(ANSI S3.4-2007 equation 1) (4*cf)/ERBN=(4*cf)/24.673/(0.004368*cf+1)
p0g2=repmat(p02',1,N).*g2; % calculate product pg
W=(1+p0g2).*exp(-p0g2); % calculate static filter shape (ANSI equation 2);
W0=zeros(size(W)); % allocate space for static filter
W0(~idx&g2<4)=W(~idx&g2<4); % load in non-zero non-level-dependent part of filters
W0=sparse(W0); % use sparse matrix operations
o2ea=W0*matrox'; % calculate excitation from level-independent partition
```

```
%% Prepare for level-dependent calculation
c1=0.257939336618976;c2=0.0018788;c3=0.004368;  % constants
beta1=c1*g2./(c3+repmat(invcf',1,N));  % additive constant matrix term used in pg
beta2=c2*g2./(c3+repmat(invcf',1,N));  % multiplicative constant matrix term used in pg

%% Determine limiting filter shapes in order to avoid unnecessary calculations
% maxX=max(X);
% pg=beta1-beta2.*repmat(maxX,length(cams),1);
% Wlim=zeros(size(W));
% Wlim(idx)=(1+pg(idx)).*exp(-pg(idx));
% dBdown=-50;
% idx2=10*log10(Wlim(idx))>dBdown;
% idx=idx(idx2); % Trim all filter elements that are always dBdown dB below or more

% Prepare sparse matrices to broadcast and condense the arrays
test1=1:N;
test2=1:length(cams);
A=repmat(test1,length(cf),1);
B=repmat(test2',1,N);
a=A(idx);
b=B(idx);
broadcast=sparse(1:length(a),a,ones(1,length(a)),length(a),N);
condense=sparse(b,1:length(b),ones(1,length(b)),length(cams),length(b));

%% Calculate dynamic filter output as a sum of both level-dependent and level- ↙
independent parts
beta1idx=beta1(idx);
beta2idx=beta2(idx);
PG=repmat(beta1idx,1,M)-repmat(beta2idx,1,M).*(broadcast*X');  % calculate PG projected
% condense product and add contribution from level-independent filter
excitation=(condense*((broadcast*matrox').*(1+PG).*exp(-PG))+o2ea)';
```