STOCHASTIC RESONANCE: MOLDING SOUNDS FROM NOISE

Esteban GUTIÉRREZ[1] and **Rodrigo F. CÁDIZ**[2,3]

[1] *Department of Information and Communications Technologies*, **Universitat Pompeu Fabra**, Barcelona Spain
[2] *Music Institute*, **Pontificia Universidad Católica de Chile**, Santiago Chile
[3] *Department of Electrical Engineering*, **Pontificia Universidad Católica de Chile**, Santiago Chile

ABSTRACT

This paper explores the application of stochastic resonance, a phenomenon known for enhancing signal detection through the introduction of random noise or disorder, to sound synthesis and processing. We detail the various models of stochastic resonance described in the literature and introduce a software implementation as an external for Max/MSP. Through a series of examples, including sound synthesis and processing, reducing the signal-to-noise ratio of noisy sounds, and retrieving the missing fundamental, we demonstrate the efficacy and versatility of this approach in molding sounds from noise.

1. INTRODUCTION

Stochastic resonance is a phenomenon observed in nonlinear systems where the introduction of random noise or a moderate level of disorder enhances the detection or transmission of weak or small signals. This implies that, although it may seem counter-intuitive, the addition of noise can improve the performance of a system in responding to a weak input signal.

The phenomenon of Stochastic Resonance was first introduced by Benzi, Parisi, Sutera and Vulpiani [1] in the context of a particular stochastic dynamical system, and was later formulated in many different contexts (e.g. [2], [3], [4], [5] and [6]). This phenomenon should not be understood as a single model, but rather as a technique for creating models that allows the study of different stochastic dynamical systems.

In a typical stochastic resonance scenario, a system is exposed to a weak input signal that may not trigger a response on its own. However, when a certain level of random noise is introduced into the system, it can help amplify and enhance the detection or transmission of the weak signal. This occurs because the noise can help push the system over its threshold, making it more likely to respond to the weak input signal.

Stochastic resonance is not just a theoretical phenomenon, as it has been observed in various fields, including theoretical physics [6], signal processing [7], metrology [8] and sensory biology [9]. Despite its clear potential as a signal processing tool, there are very few documented uses of these ideas in sound synthesis and processing.

In the past, efforts have been made to study and apply the Stochastic Resonance phenomenon in sound synthesis [10], [11]. However, the models proposed in these works lacked an easy-to-use interface or were impractical for real-time use due to implementation limitations.

In this paper we present the package `Stochastic Resonance`, a package for Max/MSP that implements real-time algorithms based on Stochastic Resonance techniques that we have found can be used in sound and music applications. This package was written using the Min devkit, a software development kit to develop Max/MSP externals using modern C++ code. Due to this, we were able to take advantage of the raw power of C++ and ensured that the externals in the package run as efficiently and fast as possible.

This paper also contains a theoretical discussion for all the algorithms in the package, a list of detailed sound and music potential applications and several sound examples. Thus, we believe this paper summarizes all known models and applications of the Stochastic Resonance phenomenon in the context of sound synthesis and processing.

In Section 2, we discuss all the algorithms included in the package. In section 3, we introduce the package itself together with all the implementation details. In section 4, we discuss a series of potential uses for this package in the context of sound synthesis and processing. Finally, in Section 5, we present the details of a series of sound examples that accompany this paper to demonstrate the potential of these tools.

2. MODELS

This section discusses the algorithms implemented in the `Stochastic Resonance` package. Some of them are based on our previous works [10], [11], but they underwent significant changes. Therefore, we will discuss these changes in detail. Each model is built with very different applications in mind. For a discussion on the different applications of each model and our use suggestions, see Section 4.

2.1 Threshold Stochastic Resonance

The Threshold Stochastic Resonance (TSR) algorithm, first proposed in [10], is the simplest implementation of the Stochastic Resonance phenomenon in signal processing. It

involves adding noise to a signal and then successively applying a constant threshold function over time. The algorithm is presented in Algorithm 1.

Algorithm 1 TSR

Input: Let x be an audio signal, $\alpha, \Theta, \sigma \in \mathbb{R}$ the attenuation, threshold and standard deviance constants, and η a noise signal with $\mathbb{E}(\eta(n)) = 0$ and $\mathbb{V}(\eta(n)) = \sigma^2$ for all time indexes n.

Output: Processed signal y.

1: Attenuate the audio signal $x_\alpha = \alpha x$.
2: Add the noise to the attenuated signal $x_\eta = x_\alpha + \eta$.
3: Apply the threshold Θ

$$y = \mathbb{1}_{|x_\eta| > \Theta} x_\eta.$$

For this algorithm to make sense in the context of stochastic resonance phenomena, it is important to choose α such that at least most of the signal x_α falls below the threshold Θ (see Figure 2), that is,

$$P(|x_\eta| > \Theta) \approx 0.$$

2.2 Multi-unit Threshold Stochastic Resonance

Due to the stochastic nature of the algorithm, applying it to a given signal multiple times will result in different outputs each time. Therefore, averaging all these outputs can provide a less noisy representation of the expected result. In the field of sensory biology, this is known as an N-unit model (see [9]). Inspired by this concept, taking the average of multiple TSR algorithm outputs will be referred to as the Multi-unit Threshold Stochastic Resonance (MTSR) algorithm. A version of this algorithm with a different name was also proposed in [10]. The algorithm is presented in Algorithm 2.

Algorithm 2 MTSR

Input: Let x be an audio signal and $N \in \mathbb{N}$.

Output: Processed signal y.

1: Make N copies of the original signal, that is, $x_k = x$, for $k = 0, \ldots, N-1$.
2: Apply the TSR algorithm to each copy using the same parameters $y_k = \text{TSR}(x_k)$.
3: Compute the average of all the new signals

$$y = \frac{1}{N} \sum_{k=0}^{N-1} y_i.$$

2.3 Spectral Threshold Stochastic Resonance

Applying noise and a threshold function to the spectrum of a signal requires careful consideration. On one hand, the perception of loudness is strongly dependent on frequency (see e.g., [12], [13]), so the addition of noise must be done with attention to this fact. On the other hand, applying a threshold function requires an ordering notion that is not inherent in the complex numbers, which is the natural space for the spectrum. An easy solution for this problem would be applying the threshold to the real or imaginary part of the spectrum, but this lacks perceptual interpretation. Another way of solving this would be applying the threshold function to either the magnitude spectrum or the phase spectrum. Since our goal is to implement these algorithms in real-time, we encounter the challenge of dealing with spectrum frames. Manipulating the phase while noise is involved can lead to crackling sounds due to the inconsistency of the phase across frames. Because of the latter, the only candidate left is the magnitude spectrum, however there is still a problem: it is built-in in the nature of the Fourier Transform that the magnitude spectrum of a periodic signal will show at least some decay. This outcome is a consequence of the Riemann-Lebesgue Theorem, suggesting that a constant threshold function may exhibit a preference for the lower part of the spectrum in certain sound contexts. Additionally, the threshold function must depend on the input signal's nature. In order to solve these issues, the Spectral Threshold Stochastic Resonance (spectral TSR) algorithm adds noise balanced according to an equal-loudness contour, and replaces the threshold constant for a function that can be built by the user (see subsection 3.2 for implementation details). The algorithm itself is presented in Algorithm 3.

It is important to note that this algorithm is very different to the one presented in [11]. In such work, the Spectral Stochastic Resonance algorithm presented generates completely random resonances. More precisely, given a set of parameters, the SSR algorithm applies plain noise to the magnitude spectrum and then independently applies a random threshold to each bin. As a result, it does not account for the equal-loudness effect of the added noise or the characteristics of the sound when determining the threshold.

2.4 Spectral Multi-unit Threshold Stochastic Resonance

Similar to the time domain, the Spectral TSR algorithm can be applied multiple times and the average can be computed to reduce the noisy nature of the expected output. The algorithm itself is presented in Algorithm 4.

3. IMPLEMENTATION IN MAX

To implement our algorithm efficiently within a friendly environment, we opted to utilize the Min dev-kit, a software development kit containing an example package with the current best practices for package creation using modern C++ code. Specifically, with the Min dev-kit, we successfully built a comprehensive package for Max/MSP that is very user-friendly and can be freely downloaded from our GitHub repository [1] .

The resulting package was named `Stochastic Resonance`, and it encompasses all the algorithms presented in Section 2, implemented as independent externals.

[1] The GitHub repository can be accessed here: `https://github.com/cordutie/Stochastic-Resonance.git`

Algorithm 3 Spectral TSR

Input: Let x be an audio signal, $\alpha, \sigma \in \mathbb{R}$ the attenuation and standard deviance constants, Θ the threshold function, S the sampling rate, $\{\eta_{j,k}\}_{j,k\in\mathbb{N}}$ a sequence of random variables with $\mathbb{E}(\eta_{j,k}(n)) = 0$ and $\mathbb{V}(\eta_{j,k}(n)) = \sigma^2$ for all j, k, n, ω a window function and m, h the frame and hop sizes respectively.

Output: Processed signal y.

1: Attenuate the audio signal $x_\alpha = \alpha x$.

2: Define each frame of the signal by the sequence x_k given by

$$x_k(j) = x_\alpha(j + kh)\omega(j), \quad j = 0, \dots, m-1.$$

3: Compute the magnitude and phase spectrum of the frame using the DFT

$$\mathcal{F}(x_k)(j) = M_k(j)\exp(iP_k(j)).$$

4: **for** each frame k and $j \in \{0, \dots, m-1\}$ **do**

5: Compute the frequency of the bin $f_j = j \cdot \dfrac{S}{m}$.

6: Correct the loudness of the noise according to the frequency of the bin using a contour curve C

$$\tilde{\eta}_{j,k} = \eta_{j,k} C(f_j).$$

7: Add the corrected noise to the magnitude spectrum

$$M_{k,\eta}(j) = M_k(j) + \tilde{\eta}_{j,k}.$$

8: Apply the threshold function

$$\tilde{M}_k(n) = \mathbb{1}_{M_{k,\eta}(n) > \Theta(f_n)} M_{k,\eta}(n).$$

9: **end for**

10: Build new frames using the IDFT and the new magnitude spectra and the same phase spectra

$$y_k = \mathcal{F}^{-1}(\hat{M}_k \exp(iP_k).$$

11: Overlap and add the new frames y_k to obtain a new signal y.

Algorithm 4 Spectral MTSR

Input: Let x be an audio signal and $N \in \mathbb{N}$.

Output: Processed signal y.

1: Make N copies of the original signal, that is, $x_k = x$, for $k = 0, \dots, N-1$.

2: Apply the Spectral TSR algorithm to each copy using the same parameters $y_k = \text{Spectral TSR}(x_k)$.

3: Compute the average of all the new signals

$$y = \frac{1}{N} \sum_{k=0}^{N-1} y_i.$$

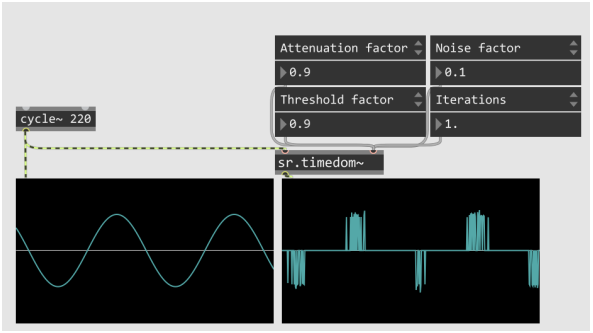

Figure 1. `sr.timedom` object running on Max. The parameters of the algorithm are passed to the object using the `attrui` object for clarity.

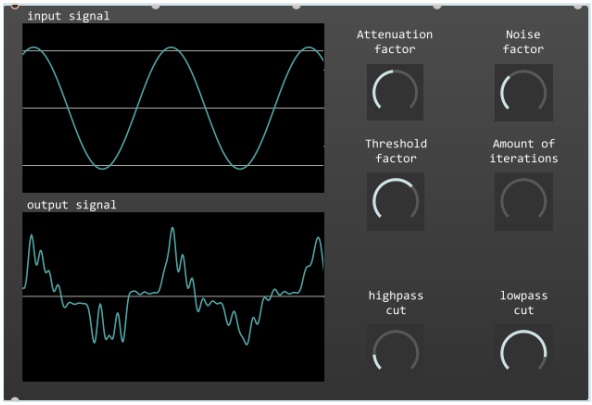

Figure 2. Example patch using the `sr.timedom~` object loaded with the `bpatcher` object. All the parameters are controllable using knobs. The threshold and attenuation are visualized in the input signal and a pair of knobs that control the filters were placed in the bottom right of the patch.

A concise explanation of the usage for each object within the package is provided here.

3.1 `sr.timedom`: Stochastic Resonance in the time domain

Since the TSR algorithm can be thought of as applying the MTSR algorithm with $N = 1$, we decided to implement both in only one object called `sr.timedom`. The left inlet is designated for the input signal, while the right one receives parameters as a list of numbers (see Figure 1). Additionally, the object has a built-in method to print the parameters used to the console when it receives a bang.

For an easy-to-use experience, we highly recommend opening the help of the object. Inside, we've included a subpatch read through `bpatcher` to provide a simple but intuitive GUI that implements a visualization of the attenuation factor, threshold function and both a low pass and a high pass filter to reduce noise (see Figure 2).

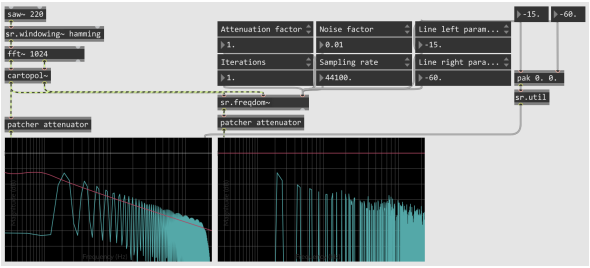

Figure 3. `sr.freqdom~` object running on Max. The parameters of the algorithm are passed to the object using the `attrui` object for clarity. The auxiliary objects `sr.windowing~` and `sr.util` are used to apply windows and compute the threshold function for plotting.

3.2 `sr.freqdom`: Stochastic Resonance in the frequency domain

Again, both algorithms Spectral TSR and Spectral MTSR are implemented in the same object named `sr.freqdom`. Since any implementation of the overlap-and-add algorithm requires precise use of a buffer and we wanted to make this package as easy to use as possible, we decided to utilize the built-in Max/MSP environment created by the `pfft~` object. This object is capable of applying a given window, computing the both the FFT and IFFT, and applying the overlap and add algorithm. Thus, the only part left to correctly implement our algorithms is to add the corrected noise and apply the threshold function.

To correct the noise, we applied a contour curve using the C-weights [14], known for its effectiveness in measuring loudness for noisy sounds. For the threshold function, we decided to implement a method in the `sr.freqdom~` object that builds a threshold function that behaves as a line in the range from 100Hz to the Nyquist frequency when viewed as a function from the log-frequency domain to amplitude measured in decibels, and that behaves as a constant for frequencies under 100Hz (see Figure 4). This convention allows the user to rapidly create threshold functions that suit the input signal and also add a component that can be used creatively.

The object itself has two signal inlets and outlets for magnitude and phase spectra, and a third inlet for receiving parameters (see Figure 3). Additionally, it has a built-in method to print the parameters used to the console when it receives a bang.

Like its time-domain counterpart, we included a subpatch in the repository. When read through `bpatcher`, it provides a simple yet intuitive GUI that implements a visualization of the attenuation factor and threshold function together with the magnitude spectrum of the input and output (see Figure 4).

4. SOUND AND MUSIC APPLICATIONS

The `sr.timedom~` and `sr.freqdom~` objects are versatile tools capable of processing various signal types, making them suitable for both creative and analytic endeavors. In this section, we will briefly discuss some in-

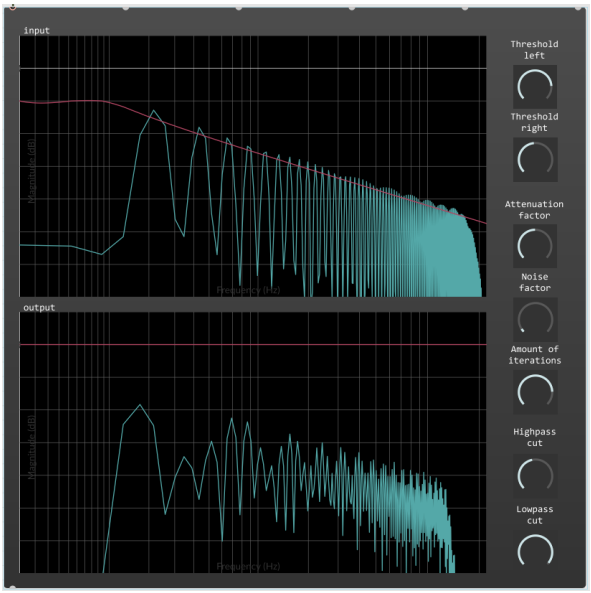

Figure 4. Example patch using the `sr.freqdom~` object loaded with the `bpatcher` object. All the parameters are controllable using knobs. The threshold and attenuation are visualized in the magnitude spectrum of the input signal and a couple of filters are implemented in the bottom right of the patch.

triguing ways of employing these objects. For specific examples, refer to Section 5.

4.1 Sound Synthesis

While the `sr.timedom~` and `sr.freqdom~` objects function as signal processors, when a simple signal such as a sine, saw, square, triangle, pulse wave, or white noise is used as input, we conceptualize the result as a novel form of sound synthesis. In such case, one must consider the type of input signal as an extra parameter together with all the parameters of the algorithms implemented. A vast array of sound types can be generated using this method. For some examples, see Section 5.1

4.2 Recovering Weak Signals

Recovering weak signals represents a prominent application of the Stochastic Resonance phenomenon in signal processing. To be more specific, empirical evidence has demonstrated that introducing noise alongside the application of a detection function can result in the identification of signals that are typically undetectable (see [8]). However, this phenomenon represents more of an analog application of Stochastic Resonance, given that issues related to the detection and measurement of signals are not a problem in the digital domain. Consequently, this application falls outside the scope of our implementation. Nevertheless, empirical validation has shown that this procedure can enhance the signal-to-noise ratio under certain circumstances (see [7]). This capability finds particular relevance in music applications, making our package suitable for such purposes. It is important to note, however, that the results

achievable through these techniques in this context significantly lag behind the outcomes attainable with contemporary denoising and source separation methods. For specific examples, refer to sub-section 5.2.

4.3 Missing Fundamental Retrieval/Ghost Stochastic Resonance

The missing fundamental is a well-known psychoacoustic phenomenon where the listener can perceive the pitch of a harmonic signal with an absent first harmonic. Given its psychoacoustic nature, grasping this phenomenon involves understanding how our brain processes information transmitted by our auditory system. In this context, Stochastic Resonance models have been proposed to understand our pitch perception in certain scenarios. Specifically, empirical evidence has shown that for a signal made of a harmonic complex of pure tones with a missing fundamental, the application of Stochastic Resonance techniques can lead to the sudden emergence of the fundamental [15]. Given that the Stochastic Resonance phenomenon is used to model the emergence of a pitch that was not originally in the signal, this effect has also been termed Ghost Stochastic Resonance.

The Ghost Stochastic Resonance effect can be easily utilized to create new pitches that are consonant with the input signal, as in the case of the missing fundamental described before. Due to this, we believe it has immense creative potential, and our package can be successfully used to implement it. For some examples, see sub-section 5.3.

5. EXAMPLES

Alongside this article, we have created some videos [2], demonstrating a series of sound examples built using the `Stochastic Resonance` package. A brief explanation for each of the sound examples can be found in this section.

5.1 Sound Synthesis and Sound Processing

For the examples in this subsection, we created two Max for Live devices. One of them is a MIDI synthesizer that wraps a sound wave generator processed through the `sr.timedom~` object and an amplitude envelope module (see Figure 5). The other one is simply a wrapper for the `sr.freqdom~` object (see Figure 6). Both Max for Live devices can be found in our repository.

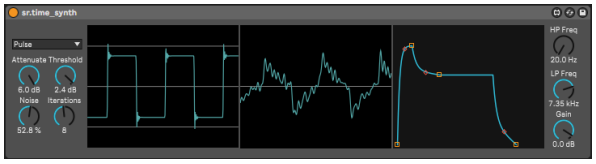

Figure 5. Max for Live device using the `sr.timedom~` object as a MIDI synthesizer.

[2] All videos are available in the GitHub repository `https://github.com/cordutie/Stochastic-Resonance`

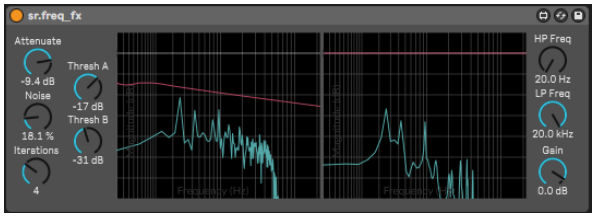

Figure 6. Max for Live device using the `sr.freqdom~` object as a sound effect.

5.1.1 Stochastic Resonance-Based Bass Synthesizer

In this example, a bass sound is synthesized using a sawtooth sound wave processed with the `sr.timedom~` object. Initially, a simple sawtooth can be heard, followed by aggressive processing with our object. Then, the `sr.freqdom~` is used to add distortion. Finally, the synthesized sound is contextualized with a drum set and an electric piano.

5.1.2 Pushing Dynamics and Resonances of an Electric Piano

A sound effect customary for electric pianos since the 1970s is the tremolo. In this example, an electric piano sample with a tremolo is processed using the Max for Live wrapper for the `sr.freqdom~` object. The sample is first played, then processed with our object using a threshold function with decay. This process pushes the dynamics of the tremolo effect with a gate-type model in the frequency domain. Noise is then added in the frequency domain, introducing stochastic resonances in the higher harmonics of the sound. Finally, all elements are combined with a drum set and a electric bass guitar.

5.2 Recovering Weak Signals

5.2.1 Sawtooth + Noise

In this example, a signal created by summing a sawtooth wave with a frequency of 220Hz and amplitude -12dB, and white noise with an amplitude of 0dB, is processed using the `sr.timedom~` object and a specific set of parameters to increase the signal-to-noise ratio. Additionally, a low-pass filter is applied to both the original and processed signals for a fair comparison. The result shows that the processed signal has a noticeably lower noise floor.

5.2.2 Street Recording

In this example, a reasonably noisy street recording is processed using a compressor together with the `sr.timedom~` object and a specific set of parameters to increase the signal-to-noise ratio. Additionally, a low-pass filter is applied to both the original and processed signals for a fair comparison. The results show that the processed signal has a distinct noise floor. Specifically, the gain of the noise floor seems to have decreased, but several crackling noises have appeared as a trade-off. As mentioned in Section 4, the results obtained using this method significantly lag behind the outcomes attainable with contemporary denoising and source separation methods.

5.3 Missing Fundamental Retrieval/Ghost Stochastic Resonance

5.3.1 Recovering missing fundamental for a complex of sinusoids

In this example, a complex of 8 sinusoids is processed in real-time using the `sr.timedom~` object. The frequencies of the complex of sinusoids follow a harmonic distribution with a missing fundamental, i.e., $2f, 3f, \ldots, 9f$ where f is the fundamental frequency. Additionally, the amplitudes and phases of each sinusoid in the complex are manipulated in real-time. The results show that the missing fundamental can be retrieved with ease, which is consistent with the simulations run in [15].

5.3.2 Building missing fundamental for a complex of voices

For this example, we used a set of previously recorded samples containing a single note sang by a female singer [3]. These samples are played at different rates so that their fundamentals follow a harmonic distribution with a missing fundamental, as in 5.3.1. The results show that applying the `sr.timedom~` object introduces a fundamental with ease in every scenario. Moreover, since the original signal consists of a sum of signals that resemble the harmonics of a sound but already have harmonics, some other resonances appear in the spectrum corresponding to what seems to be the fundamentals of the harmonics of these complex sounds. When a low-pass filter is applied to the resulting signal, a new synthesized sound that resembles some characteristics of the original voice appears.

6. CONCLUSIONS

In this paper, we propose a series of algorithms based on the Stochastic Resonance phenomenon for use in sound, accompanied by a theoretical discussion and exploration of their potential applications in sound synthesis and processing.

Our contribution goes beyond mere theory, providing a user-friendly and efficient environment for musicians, sound designers, and researchers. The `Stochastic Resonance` package, implemented in real-time in both Max/MSP and Live as a Max for Live device, streamlines the utilization of our proposed algorithms, seamlessly integrating them into any workflow. The package can be freely downloaded from our GitHub repository. Additionally, we present a series of examples to showcase the practical applications of the Stochastic Resonance package in sound synthesis and processing.

In summary, by amalgamating theoretical insights with practical examples, this paper serves as a comprehensive resource for those keen on unlocking the creative potential of the Stochastic Resonance phenomenon in the realm of sound and music.

Acknowledgments

This research was partially funded by ANID Fondecyt Grant #1230926, Government of Chile.

[3] The samples were provided personally by their author, Jessica Torrey.

7. REFERENCES

[1] R. Benzi, A. Sutera, and A. Vulpiani, "The mechanism of stochastic resonance," *Journal of Physics A: Mathematical and General*, vol. 14, p. L453, 01 1981.

[2] L. Gammaitoni, "Stochastic resonance and the dithering effect in threshold physical systems," *Phys. Rev. E*, vol. 52, pp. 4691–4698, Nov 1995. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.52.4691

[3] ——, "Stochastic resonance in multi-threshold systems," *Physics Letters A*, vol. 208, no. 4, pp. 315–322, 1995. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0375960195007316

[4] L. Gammaitoni, F. Marchesoni, E. Menichella-Saetta, and S. Santucci, "Stochastic resonance in the strong-forcing limit," *Phys. Rev. E*, vol. 51, pp. R3799–R3802, May 1995. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.51.R3799

[5] L. Gammaitoni, F. Marchesoni, and S. Santucci, "Stochastic resonance as a bona fide resonance," *Phys. Rev. Lett.*, vol. 74, pp. 1052–1055, Feb 1995. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.74.1052

[6] L. Gammaitoni, P. Hänggi, P. Jung, and F. Marchesoni, "Stochastic resonance," *Rev. Mod. Phys.*, vol. 70, pp. 223–287, Jan 1998. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.70.223

[7] M. Ueda, "Improvement of signal-to-noise ratio by stochastic resonance in sigmoid function threshold systems, demonstrated using a cmos inverter," *Physica A: Statistical Mechanics and its Applications*, vol. 389, no. 10, pp. 1978–1985, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378437110000890

[8] A. Palonpon, J. Amistoso, J. Holdsworth, W. Garcia, and C. Saloma, "Measurement of weak transmittances by stochastic resonance," *Opt. Lett.*, vol. 23, no. 18, pp. 1480–1482, Sep 1998. [Online]. Available: https://opg.optica.org/ol/abstract.cfm?URI=ol-23-18-1480

[9] J. J. Collins, C. C. Chow, and T. T. Imhoff, "Stochastic resonance without tuning," *Nature*, vol. 376, no. 6537, pp. 236–238, Jul 1995. [Online]. Available: https://doi.org/10.1038/376236a0

[10] R. Cadiz and P. De la Cuadra, "Stochastic resonance sound synthesis," *International Computer Music Conference, ICMC 2008*, 08 2008.

[11] R. Cadiz and P. de la Cuadra, "Spectral stochastic resonance sound synthesis," *International Computer Music Conference, ICMC 2010*, pp. 353–356, 01 2010.

[12] D. W. Robinson and R. S. Dadson, “A re-determination of the equal-loudness relations for pure tones,” *British Journal of Applied Physics*, vol. 7, no. 5, p. 166, 1956.

[13] Y. Suzuki and H. Takeshima, “Equal-loudness-level contours for pure tones,” *The Journal of the Acoustical Society of America*, vol. 116, no. 2, pp. 918–933, 08 2004. [Online]. Available: https://doi.org/10.1121/1.1763601

[14] International Electrotechnical Commission (IEC), “Electroacoustics - Sound level meters - Part 1: Specifications,” IEC, Geneva, Switzerland, Tech. Rep. IEC 61672-1, 2002.

[15] P. Balenzuela, H. Braun, and D. Chialvo, “The ghost of stochastic resonance: An introductory review,” *Contemporary Physics - CONTEMP PHYS*, vol. 53, 10 2011.