MCGILL UNIVERSITY

MUMT 501: DIGITAL AUDIO SIGNALS PROCESSING

Audio Restoration

Dorian Desblancs, 260722712
James Mesich, 260714836

Professor:
Philippe Depalle

29-04-2020

Abstract

In this project, we implemented two well known audio restoration techniques. The first was taken from *Digital Audio Restoration* by Godsill et al. [1] whilst the second was presented by Laurent Oudre in 2015 and 2018 [2] [3]. The former is primarily focused on wow and flutter removal. Both artefacts are due to unwanted pitch variations in a recording, and are restored using an estimate of the pitch variation curve. This technique relies heavily on clever pre-processing and Bayesian probability. While testing on simple signals produced reliable results, the implementation failed to remove defects in more complex audio such as gramophone recordings. The second algorithm relies on identifying and interpolating bursts of noisy audio. This method was designed to remove defects such as clicks and hiss commonly found in vinyl records. Through proper tuning of parameters, strong restoration results were achieved on complex audio recordings.

Introduction

Audio restoration encompasses the various methods used to remove imperfections from sound recordings. These imperfections range from hiss to clicks to wow. Originally, the field grew with the advent of digital technologies. Past analog recordings needed to be digitized, and artefacts due to dust, scratches, or tape needed to be fixed. Today, the field is expansive and wide-reaching. Since 1979, Grammy awards for the best historical album have been given to some of the best restored music albums and collections. These awards are a reflection of the progress made by the field of audio restoration each year.

Audio restoration techniques are usually implemented by first digitizing a piece of audio. Its digital representation is then transformed to rid it of as many artefacts as possible. Whilst most famous restoration tasks are done by professional audio engineers, fully automated solutions have also been developed. In this project, we implemented and tested two different audio restoration algorithms. The first algorithm we implemented was specifically designed to remove wow and flutter defects from corrupt audio recordings. The second algorithm, on the other hand, focuses on detecting and interpolating noisy samples to fully restore a piece of audio. We measured the performance of these techniques through various visualizations and listening tests. This report covers each technique in detail. For each technique, the algorithm's steps and performance are studied. Potential improvements are also presented in the form of a discussion.

Related work

The field of audio restoration gained traction during the 1990s, when analog musical formats such as vinyl records and cassettes were replaced by digital formats such as the compact disc (CD). In 1994, Ruanaidh and Fitzgerald [4] published one of the first prominent papers on audio restoration. They modelled sections of audio using stationary autoregressive processes, and then interpolated missing samples using Monte-Carlo Markov chains and Gibbs sampling. Their methodology was a pre-cursor of what was to come in the field.

In 1998, Simon Godsill, Peter Rayner, and Olivier Cappé published *Digital Audio Restoration* [1]. This textbook remains the gold standard for audio restoration, and was used in this project. It first covers the fundamentals of signal processing, probability theory, and random processes. From there, click removal, hiss removal, and numerous advanced restoration techniques are explored. Most of these techniques make use of autoregressive processes and advanced statistics for interpolation of noisy signals.

After Godsill et al. published their findings, the field of audio restoration gradually became sparser. Very few significant breakthroughs were published, and most techniques outlined later on built upon the techniques explored in *Digital Audio Restoration*. In 2001, Biscainho et al. [5] further explored the use of ARMA pro-

cesses for audio restoration. They explored a sub-band version of the ARMA model-based technique in order to remove clicks from a damaged audio recording.

More recently, a few very interesting methods for audio restoration have been explored and published. In 2015, Magron et al. published a novel technique for audio restoration. Unlike previous methods, their restoration process relies on the use of the phase response of a signal in the Time-Frequency domain. Phases of successive audio frames are used as a foundation for reconstructing corrupted audio in the frequency domain. This technique was found to outperform classic techniques in many ways, and deserves to be further investigated.

Finally, in 2015 and 2018, Laurent Oudre published two papers [2] [3] that outline a full restoration process. The techniques explored in these papers were used in this project, and involve the detection and interpolation of bursts of noisy audio. Note that this section of our report only skims the surface of the field of audio restoration. The reader is however free to explore the field in more depth if he so desires.

Dataset

In order to test our algorithm implementations, we decided to create and restore a select number of recordings. The data set created was partially built by ourselves.

First, four recordings provided by Laurent Oudre's website[1] were downloaded and used as a baseline for judging our algorithms' performances. These four recordings are extremely noisy, and seem to come from an old gramophone or record player. The short excerpt of Suzanne Vega's *Tom's Diner* acapella was especially useful for judging the performance of our full restoration process. The other three recordings are classical in their nature, although we were unable to identify the songs' names or artists.

We then opted to test our algorithms on six recordings of our own. These recordings are four seconds long, and come from our own vinyl collections. The following songs are included in the data set: *Sex Tonight* by Brian Martin, *Sampling 101* by Dj Ionic, the A1 side from Moodymann's *Don't be Misled* album, *Lost in Lyrics* by the Timewriter, *Welcome* by Gino Latino, and *Mystery Land* by Y Traxx. Note that some of these excerpts are not particularly damaged, whilst others contain noticeable crackle. These were particularly useful for judging whether the full restoration process outlined by Laurent Oudre is effective when dealing with more subtle noise, which is more common when a vinyl record and turntable needle are in good shape.

The original and restored recordings can be found on the GitHub link provided at the end of this report.

Wow and Flutter Removal

The first audio restoration technique we explored focuses on two specific types of audio degradation called wow and flutter. Wow is a smooth pitch variation over a long period of time. Flutter, on the other hand, occurs when a recording's pitch varies extremely rapidly. These defects can often be found in old vinyl and tape recordings. They are usually the result of a variation in a record player's rotational speed. Old tapes recordings can also suffer from these defects if the tape is stretched during the recording process. While one can hear a change in the original pitch, wow and flutter are caused by a time-warping function.

We can represent pitch distortion using the following equation, which describes a time-warped signal x at time t:

$$x_w(t) = x(f_w(t)).$$

[1] https://ipolcore.ipol.im/demo/clientApp/demo.html?id=64

Note that f_w represents the non-uniform warping occurring at time t.

In order to retrieve the desired, un-corrupted, signal, we must recover the signal *x(t)*. This desired output can be represented using

$$x(t) = x(f_w^{-1}(t)).$$

The wow and flutter algorithm is primarily focused on estimating the inverse of the time warping function f_w.

The algorithm we implemented to remove wow and flutter is presented in Chapter 8 of *Digital Audio Restoration* [1]. It can be decomposed into three main parts. First, the frequency components of the degraded audio must be extracted. By examining the frequencies present in the audio, tracks containing common frequency peaks are created. These tracks can then be used to generate a pitch variation vector, through the use of Bayesian probability and partial derivation. Finally, a restored version of the original recording is output. The flow chart in Figure 1 outlines the general steps of the wow and flutter removal algorithm. These will be further explored in the next section.

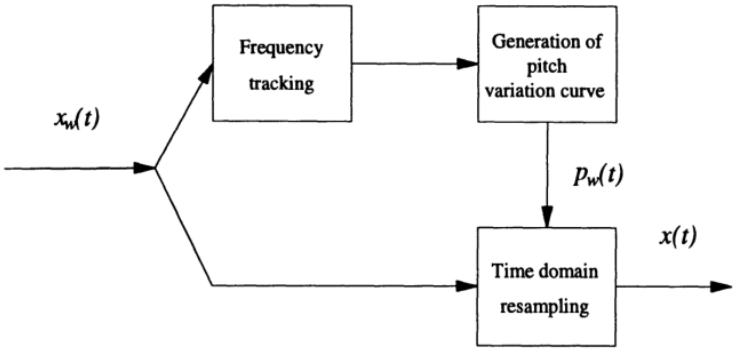


Figure 1: Outline of wow restoration system

1 Wow and Flutter Algorithm Steps

1.1 Frequency Tracking

The goal of this part of the algorithm is to extract frequency peaks from the audio. When dealing with pitch-degraded audio, all frequency peaks are modulated in the same way through time. These variations allow us to detect wow and flutter. More concretely, say three distinct instruments compose a recording, and all their frequencies change in the same way. This change is most likely the result of unwanted pitch variation, and needs to be rectified.

The first step of this section of the algorithm is to split the original signal into overlapping frames. We used frames containing 2048 samples and a 50% overlap. A Hamming window is then applied to each frame. This window better prepares the frame for discrete Fourier analysis over each block. The Scipy [6] library was used in order to implement the steps outlined above.

Once the frequency components of the signal have been extracted, frequency peaks must be analyzed. Note that not all peaks are created equal. Some are of interest, and correspond to un-corrupted audio frequency peaks, whilst others are due to noise. Hence, peaks below a certain amplitude threshold are treated as noise in the algorithm. More precisely, all peaks superior to 25% of the maximum amplitude value were extracted.

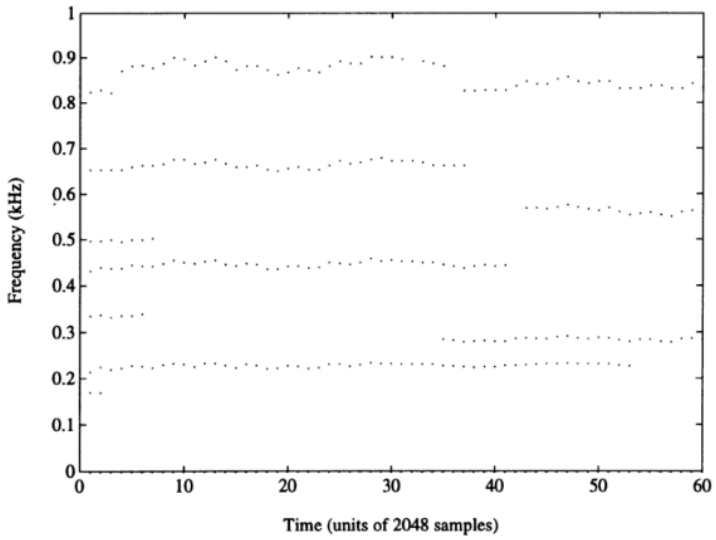


Figure 2: Example graph of frequency tracks

The frequency peaks found were then split into frequency tracks centered around a particular frequency and amplitude. Peaks within +/-10 Hz and of similar amplitude were grouped together to form one track. For example, if a peak had an amplitude of 104 and a frequency of 2353Hz, it was grouped with a frequency track that had a center frequency of 2350Hz and an amplitude of 100. Note that the decision to limit a track to groups of +/-10 Hz may have affected the algorithm's performance greatly. If the pitch variation in a recording is greater than 10 Hz, its detection using frequency tracks becomes extremely hard. Another problem encountered is due to the fact that not all frequencies are present throughout the entirety of the audio recording. As such, the frequency tracks throughout each track may not be the same. This problem was solved by marking all frequency tracks with a value of 0 in the event that they were not present in a certain block.

Algorithm 1: Pseudo-code

```
Result: Frequency track matrix
for each block in audio do
    Extract peak information;
    Distinguish valid peaks;
    for each peak do
        if Similar track exists then
            Add to track;
        else
            Create new track;
        end
    end
end
```

At the end of this stage of the algorithm, a matrix of frequency tracks was created. This matrix indicates the frequencies and amplitudes active in each block of 2048 samples. Note that the textbook [1] did not explicitly explain how this step of the algorithm was to be performed. Our implementation simply used the raw values of the frequency peaks recorded in the matrix. This could however be improved using normalized frequency values within a certain range. Since we tracked all frequencies up to 20000 Hz, the large range of frequencies tracked could be the reason behind some of the unexpected results we got using the algorithm.

1.2 Pitch Variation Estimation

Once we had generated the frequency tracking matrix, the wow and flutter algorithm focuses on detecting pitch variation. Godsill et al. [1] outline two models to do so: an additive one and a multiplicative one. The multiplicative one was used in this project. In order to estimate a pitch variation vector P, the following formulation holds

$$F_n^i = F_0^i P_n V_n^i.$$

where F_0 denotes a track's center frequency and V_n denotes the frequency noise that must be accounted for. The equation above can be further transformed into its logarithmic form

$$f_n = f_0 + p_n 1_R + v_n.$$

This step allows the equation to be in linear form. It can then be solved faster, using positive values only.

Note that the noise is assumed to have a normal distribution with variance σ^2. This simplification has proven to be extremely useful when dealing with audio, although it can be problematic for effects such as vibrato.

The estimation of P and f_0 can be solved using a Bayesian approach. The algorithm's first step was to gather the frequency tracking matrix F. Bayes' Rule allows use to get the following

$$p(P, f_0|F) \infty p(F|P, f_0) p(P, f_0)$$

assuming P and f_0 are independent.

Assuming f_0 has a uniform distribution, only P must be estimated. Wow and flutter usually lead to a smooth pitch variation over time. We therefore do not expect any sharp discontinuities in the time-frequency curve, and can estimate P using a multivariate normal distribution with mean of 0 and a diagonal co-variance matrix made up of random variables sampled from a normal distribution. The estimation of $p(P, f_0|F)$ can then be re-written as

$$p(P, f_0|F) = e^{-\frac{1}{2\sigma_v^2}(\sum_{n=1}^{N}\sum_{i=1}^{R}(v_n^i)^2)+(\sigma_v^2 p^T C_p^{-1} p)}.$$

The Maximum a Posteriori estimate can then be determined using the partial derivatives for each unknown term and setting these to 0. Through basic substitution, the estimate of P used for the remainder of the report was found using the following equation

$$P_{MAP} = (diag(R_{MAX}) - M^T (diag(N)^{-1} M + \sigma_v^2 C_p^{-1})^{-1} (F^T 1_{R_{MAX}} - M^T (diag(N))^{-1} F 1_N))$$

where each variable denotes the following:

- F: frequency track matrix
- diag(R_{max}): each value in its diagonal corresponds the number of active tracks at that time block.
- M: indicator matrix denoting if the corresponding frequency track is active
- diag(N): each value in the diagonal corresponds to the length of each track
- 1_{var}: a vector of *var* ones.
- C_p : P co-variance matrix

The implementation of this portion of the algorithm in python was quite straightforward. The Scipy library [6] allows for easy matrix manipulation. By scanning over the frequency track values over each frame and defining the necessary matrices, the pitch variation vector was successfully estimated. Note that our implementation of the wow and flutter algorithm differs from [1]. A step was added to interpolate the pitch variation values to match the length of the audio. Due to the frequency track matrix being generated from the information of 2048 data points, there would have been a mismatch in length when we re-sample in the next part of the algorithm.

1.3 Sample Time Converter

Once we have estimated the pitch variation curve, the final step of the algorithm is to re-sample the signal. This part essentially implements a sample rate converter in the hopes of restoring the signal. The only difference is that the conversion uses a non-uniform sampling rate. The implementation of this portion was quite tricky, and was not fully outlined in the Godsill textbook [1]. Our implementation was therefore limited to recreating the function outlined in [1]. We however ran into some notable issues.

$$\hat{x}(nT) = \sum_{m=-M}^{M} w(mT' - \tau) sinc(\alpha(mT' - \tau)) x_w((n_w - m)T').$$

First, we came to the conclusion that the M should equal 64 samples through trial and error. We then used a Hamming window w, and were able to use Scipy's [6] *sinc* function effectively. We however ran into some issues when trying to estimate T'. While T' was clearly defined as $\frac{T}{p(t)}$, this did not always result in an integer value. Its value was thus rounded down. This was necessary due to the discrete nature of the windowing function and the degraded signal. The value n_w was also highly problematic. In Godsill's textbook, it defines n_w as the closest sample to the n value in the restructured audio. However without clear guidance, we were unable to transpose the restored audio signal's index values to the original. Instead, in order to simplify the equation, n_w was set to the same value as n.

Next, τ was estimated from $nT - f_w(n_w T)$. To find f_w, the integral of the sequence of $p(t)$ must be calculated. In our implementation, the composite trapezoid method from the Scipy libary [6] was used to do so. Once again this result did not always return an integer value so rounding down was required so the window could be applied. This was typically the most time-consuming part of the algorithm. We are aware that all the changes made to calculate the equation may be the cause of numerous issues with our implementation. These changes may be the reason behind the uneven results we generated using the wow and flutter algorithm.

2 Results

For the baseline test, a time-warped signal was created artificially to measure whether the algorithm did what it was supposed to. The original signal was a simple sine wave. It was then distorted using the time-warping function $1.5(sin(x) + x))$. This resulted in a pitch curve of $1.5(cos(x) + 1$. We assumed that if the algorithm was implemented correctly, it would produce an accurate estimate the pitch curve. From there, re-sampling the signal would create the original, un-corrupted signal. The graph in Figure 3 demonstrates the results of this test.

Our algorithm ended up reproducing the original signal perfectly, hereby proving its validity for a basic pitch variation task. We then tested it on our data set in order to determine whether it could be used for general restoration purposes.

The algorithm was tested on the whole data set. This report, however, will focus on its results on *Sex Tonight* by Brian Martin. Unfortunately, with more complex audio, the algorithm clearly failed. Figure 4 shows the original

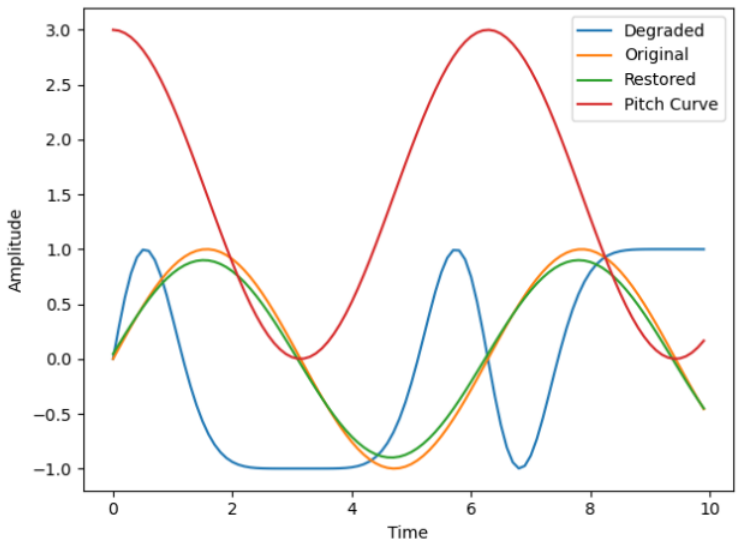


Figure 3: Baseline Test Results

degraded audio whilst Figure 5 displays the wow and flutter technique's output. The output basically consists of noise. This was not the result we expected. Further investigation into our implementation is therefore needed.

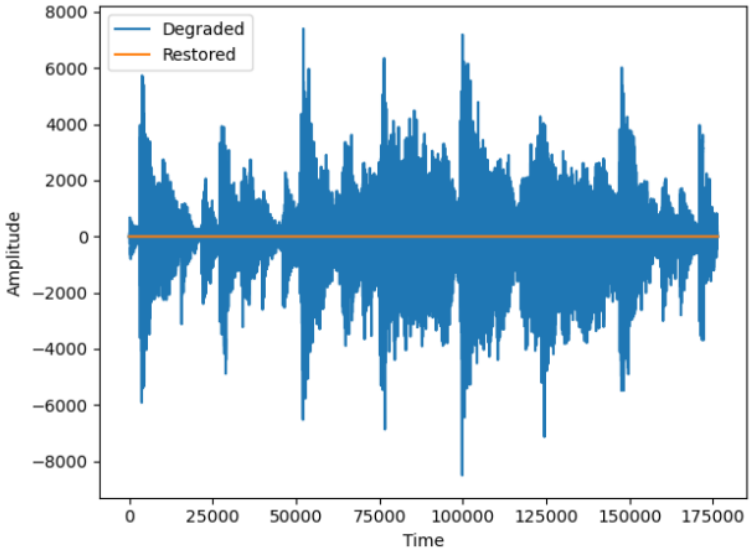


Figure 4: Degraded audio of Sex Tonight by Brian Martin

To begin with, there was clearly a gain issue. We believe that it was caused by our implementation of the peak analysis and matching step. Due to our informal approach to categorizing frequency and amplitude information, we believe that gain information was not properly encoded into our frequency track matrix. As displayed in the two graphs, the amplitude scale in each signal varies greatly. More work is therefore needed in the first step of the algorithm. Gain re-scaling would explain why our baseline example worked fine (amplitudes between 0 and 1) whereas more complex signals failed (very large amplitudes). Next, re-sampling was found to be extremely tricky. The simplifications we used in our implementation are the most likely cause of the noise introduced in the reconstructed signal.

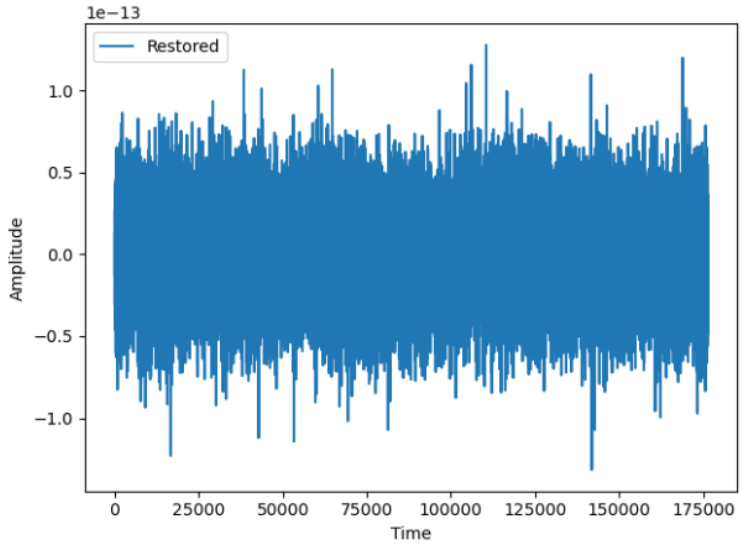


Figure 5: Attempt to restore the audio of Sex Tonight by Brian Martin

3 Discussion

Overall, this algorithm showed us an extremely interesting approach to wow and flutter removal. From the results in the textbook[1], it seems that it is an impressive technique, capable of high-quality restoration. However, the implementation of the technique is not as straightforward as it seems. Godsill et al. omitted most of the algorithm's subtleties, which resulted in our implementation's failure to achieve the high-quality signal reconstructions. While it was clearly able to handle basic time-warped signals, it failed to handle complex audio excerpts.

Burst Restoration

The second audio restoration technique we explored in this project was described by Laurent Oudre [2][3]. Unlike Wow and Flutter removal, this technique attempts to restore damaged audio recordings in a much more comprehensive manner.

The technique relies on identifying and adjusting bursts, or prolonged sections, of noisy audio. These bursts are often the result of defects in an audio recording, and can be due to hiss, crackle, or scratches. Figure 6 shows a sample of damaged audio. Notice the jagged nature of the signal. The defects in the audio are due to the fact that the audio was recorded from a gramophone. The audio is therefore extremely noisy. In this section, we first present the algorithm for burst rectification. We will then explore how well this algorithm performs by presenting its results on the data set. Finally, we will explore potential improvements for the algorithm in the context of a discussion.

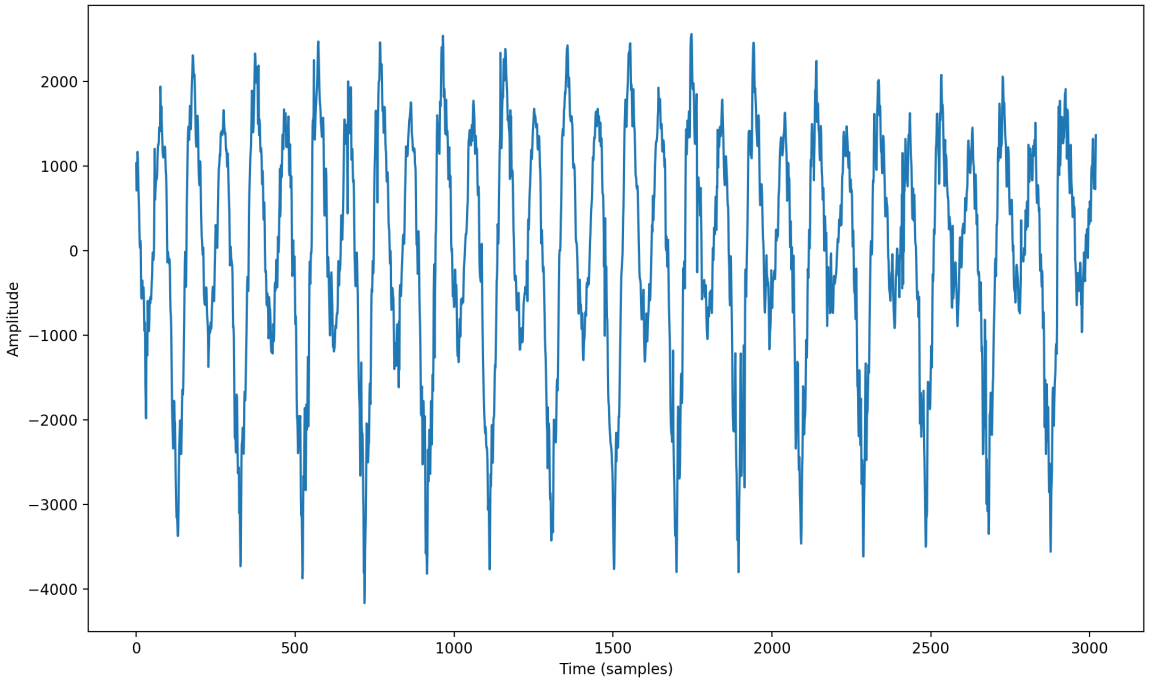


Figure 6: Acapella Damaged Recording

4 Burst Restoration Algorithm

4.1 Algorithm Parameters

Before delving into the algorithm Laurent Oudre outlines in his papers [2][3], one must first introduce the parameters it uses in order to work. These will be explored further in the section outlining the algorithm's steps, but an initial introduction to these is needed for the reader to understand the ensuing sections of the report.

The burst restoration algorithm relies on five parameters that must be chosen by the user. These enable the user to "tune" the restoration process to a specific task, which usually depends on the amount of restoration needed on a particular piece of audio. First, the parameter K works as a burst detection threshold. Its value is usually between 1 and 5, and it controls the extent to which a piece of audio is restored. Lower values such as 1 and 2 allow the restoration algorithm to tackle low amplitude discrepancies in the audio, such as hiss, whereas higher values limit the algorithm's targets to obvious defects such as a scratch on a vinyl record.

Second, the parameter b is used as a burst length threshold. For a value of 20, only noisy sections of audio with a length of at least 20 samples will be restored. Its value is usually between 1 and 100. Third, the parameter p denotes the order of the autoregressive model used for burst detection. More information about this parameter is provided in the following sections. Finally, the parameter N_w denotes the length of the frames used throughout the restoration process.

In his papers [2][3], Laurent Oudre emphasizes that the following conditions must be met for the algorithm to work efficiently. Notably, the parameter p must satisfy $p = 3N_{max} + 2$, where N_{max} denotes the maximum length of a burst. Since burst lengths rarely exceed 100 samples, p works extremely well with a value of 302. Also, N_w must satisfy $N_w > \frac{8}{3}p$ in order for each each sample to be processed at least once throughout the restoration

process. In practice, a value of $N_w = 8p$ optimizes the restoration process. That is why we used a value of 2416 for N_w in most of our experiments.

4.2 Algorithm Steps

Note that the following steps comprise one iteration of the burst restoration algorithm. In practice, one can can run the algorithm for as many iterations as desired, although no more than 5 iterations are usually needed to get a well-restored piece of audio.

4.2.1 Padding and Framing

When dealing with long audio recordings, operating on the entire signal without any pre-processing is usually a bad idea. This often leads to the creation of new audio artefacts. More importantly, when it comes to burst detection, damaged audio can be found much more efficiently when breaking the signal into sections. Hence, the burst rectification algorithm starts with two standard audio processing steps: padding and framing.

Before the signal is operated on, it is padded with zeros. More precisely, N_w zeros are added at the front and rear of the signal. The rear is then padded with additional zeros, the number of which is $(\lceil \frac{N+N_w}{N_h} \rceil N_h - N_w) - N$. Note that N denotes the total signal length and N_h denotes the hop size. The hop size is determined using the overlap percentage between each frame. Once the signal is padded, the signal is divided into overlapping frames of size N_w. For an overlap value of 75% and an N_w value of 2416, the hop size represents the integer value $N_w - 0.75N_w$, or 604. The reason behind the second rear padding of the signal is hence quite simple: it allows the signal to be split into frames of exact size N_w.

Note that we experimented with various overlap values ranging from 50% to 85%. Lower values under 70% were found to create artefacts in the restored signal whereas larger ones above 75% were found to be more computationally demanding and did not offer any gain in performance. We therefore stuck to the 75% overlap value outlined in Laurent Oudre's papers [2][3].

4.2.2 Autoregressive Parameter Estimation

Suppose we have a damaged piece of audio of length N. For $t \in N$, we can write:

$$x_t = s_t + n_t$$

where s_t represents the restored audio we want to retrieve and n_t represents the added noise.

Since the signal data can be represented as a time series, one can therefore use an autoregressive model to depict each frame. More specifically, assuming s can be modelled as a stationary process on each frame, then for all $t \in N$, we can represent s as:

$$s_t = -\sum_{k=1}^{p} a_k s_{t-k} + e_t$$

where $a = [a_1, ..., a_p]^t$ and e represents a white noise of variance σ_e^2. Note that a and σ_e^2 represent the AR parameters to be calculated over each frame. Thus, the AR model fit to each frame has order p.

Furthermore, the two equations above can be used to derive the following:

$$d_t = x_t + \sum_{k=1}^{p} a_k x_{t-k} = e_t + n_t + \sum_{k=1}^{p} a_k n_{t-k}$$

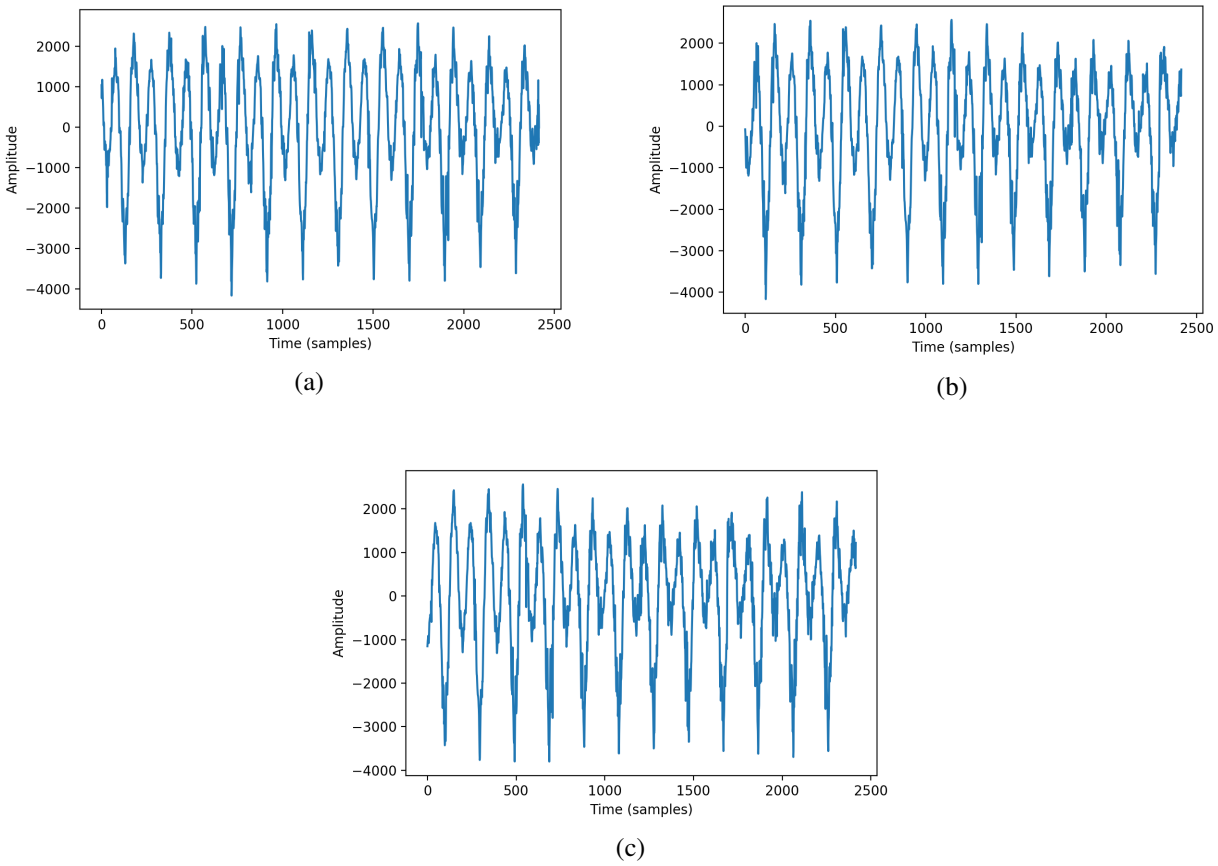


Figure 7: Overlapping Frames of Damaged Acapella Recording

where d_t corresponds to a filtered version of x_t. Note that d_t will be explored more thoroughly in the next section and is referred to as the detection signal.

In order to find the d_t, however, one must first find the AR parameters of the autoregressive models representing each signal frame. This model must have order p. In order to estimate the AR parameters efficiently, Laurent Oudre relies on a method based on the Yule-Walker equations. These equations go as follows. Given an estimate of the autocorrelation signal

$$R(t) = \frac{1}{N} \sum_{k=t+1}^{N} x_k x_{k-t},$$

the AR parameters of the model representing each frame can be estimated by solving the following system of equations:

$$\begin{bmatrix} R(0) & R(1) & \dots & R(p-1) \\ R(1) & R(0) & \dots & R(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ R(p-1) & R(p-2) & \dots & R(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = - \begin{bmatrix} R(1) \\ R(2) \\ \vdots \\ R(p) \end{bmatrix}$$

where the a terms represent the a parameters of the AR model. It is trivial to notice that the first matrix of the system is a Toeplitz matrix of dimension p. As such, the system can be solved in p^2 operations using the Levinson-Durbin algorithm.

Historically, this fast algorithm was introduced by Norman Levinson in 1947 [7], and then improved by James Durbin in 1960 [8]. It was then further improved by Willian F. Trench [9] and other mathematicians during the 1970s and 80s. A fascinating historical account of its development can be found in [10].

Input: $\hat{R}(0), \ldots, \hat{R}(p)$
Output: $a_1, \ldots, a_p$

Initialization : $a_1^{old} = -\frac{\hat{R}(1)}{\hat{R}(0)}$, $\sigma_1^2 = (1 - (\frac{\hat{R}(1)}{\hat{R}(0)})^2)\hat{R}(0)$
for $l = 1$ **to** $p - 1$ **do**
  $k_{l+1} = \frac{1}{\sigma_l^2} \sum_{j=1}^{l} a_j^{old} \hat{R}(l + 1 - j)$
  $a_{l+1} = -k_{l+1}$
  $\sigma_{l+1}^2 = \sigma_l^2(1 - k_{l+1}^2)$
  for $j = 1$ **to** l **do**
    $a_j = a_j^{old} - k_{l+1}\, a_{l+1-j}^{old}$
  end
  $\mathbf{a}^{old} = \mathbf{a}$
end

Figure 8: The Levinson-Durbin Algorithm [3]

Figure 8 outlines the Levinson-Durbin algorithm's steps. Notice that both the a and σ_e^2 terms can be found using the algorithm. These AR parameters are found for each frame of the signal, and are essential for the ensuing steps used to detect and rectify bursts in a damaged input signal.

4.2.3 Burst Identification

Once the AR parameters a and σ_e^2 are estimated for each frame, one can easily calculate a criterion threshold.

$$d_t = x_t + \sum_{k=1}^{p} a_k x_{t-k}.$$

This criterion serves a unique purpose: for each sample index, the criterion value indicates how different the signal is to the rest of the frame. A high criterion value indicates a burst of noisy samples, whereas a lower value indicates that the sample is not corrupt. Notice that the criterion cannot be calculated for samples with indices less than p.

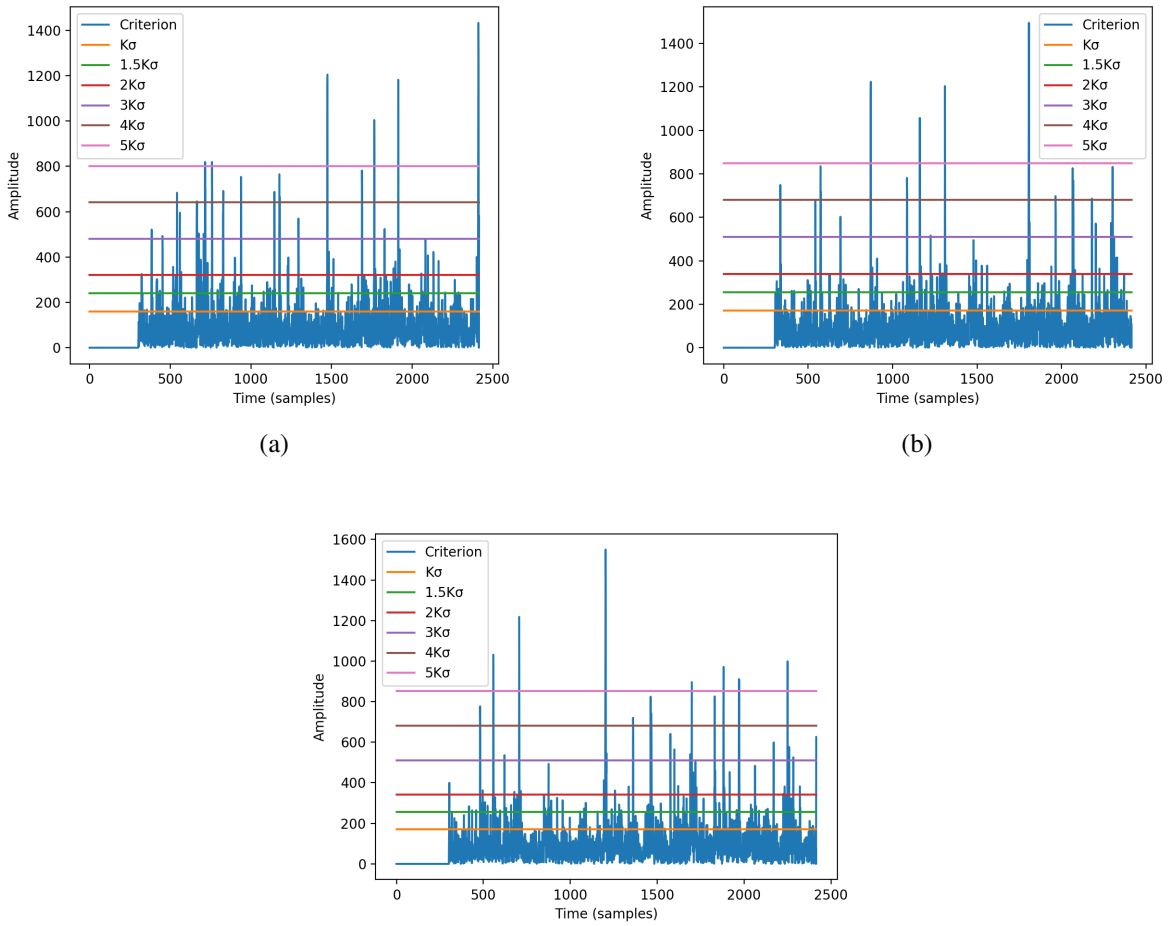
Hence, when identifying bursts, the value $|d_t|$ at each sample is compared to the value $\lambda_k = K\sigma_e$, where K is chosen by the user. Note that σ_e is always positive. If a minimum of b samples in a row have a criterion value superior to λ_k, then the corrupt signal portion of the frame is set aside for interpolation. Figure 9 displays the criterion values for the three frames studied previously. Notice how smaller values of K are much more effective for detecting large noisy portions of the signal. This is due to the fact that, since the original signal is extremely corrupt for its entire duration, most of the signal values that need to be interpolated have a fairly low criterion threshold. For this reason, the Acapella sound recording was restored using a value of $K = 1.5$ and $b = 1$. These values were found to work very well on audio recordings that contain constant hiss and crackle.

4.2.4 Burst Restoration and Interpolation

Let us denote the sequence of damaged samples for each frame $s(T)$. These samples are all more or less noisy, have been determined using the criterion, and must be interpolated for restoration. In his papers, Laurent Oudre [2][3] solves the interpolation step by successively solving a set of systems. Note that the equations and their elements were adapted for clarity.

For a signal with $|T|$ corrupt samples, a vector b is defined such that

$$\forall i \in [0, p], b_i = \sum_{k=i}^{p} a_k a_{k-i}.$$



(a) (b)

(c)

Figure 9: Criterion for Frames in the Acapella Recording

From this vector, a symmetric and positive definite matrix B is defined such that

$$\forall i \in [0,|T|[\,, \forall j \in [i,|T|[\,, \begin{cases} B_{i,j} = B_{j,i} = b_{|t-t'|}, & \text{if } |t-t'| < p+1 \\ 0, & \text{otherwise} \end{cases}.$$

Note that $t, t' \in [0,|T|[$.

A vector d is also defined such that

$$\forall i \in [0,|T|[\,, d_i = \sum_{-p \leq k \leq p, t-k \in |T|} b_{|k|} s_{t-k}.$$

Note that b, B, and d are calculated for $s(T)$ in each frame. The elements B and d are the basis of the interpolation step. In order to interpolate the corrupt signal in each frame, $s(T)$ must be minimized by solving the following equation:

$$Bs(T) = -d$$

This is done using a variant of Cholesky decomposition. The matrix B is decomposed into a set of matrices $B = LDL^t$ where L is a lower triangular matrix with main diagonal values of 1 and D is a diagonal matrix. From this decomposition, the systems $Lx = -d$ and $L^t s(T) = D^{-1}x$ are solved. The resulting $s(T)$ values found correspond to the interpolated signal values and are used to replace each frame's corrupt samples.

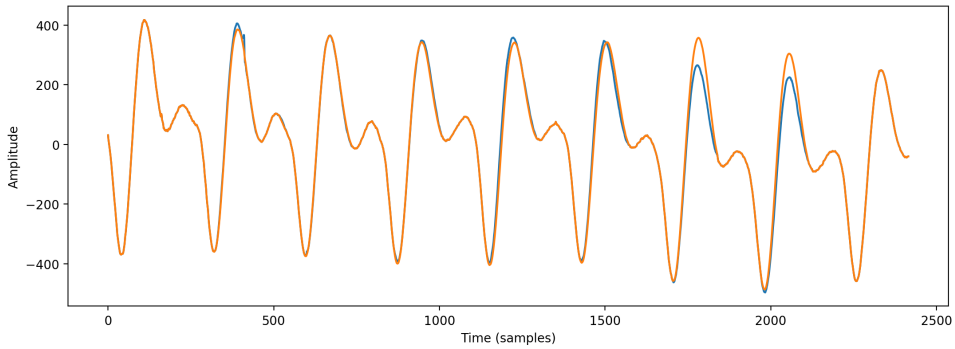


Figure 10: Sample 101 Frame Reconstruction

The orange signal represents the reconstructed frame, whilst the blue one represents the original.

4.2.5 Signal Reconstruction

Once the damaged audio samples have been restored in each frame, it is fairly simple to reconstruct the new, restored audio signal. A Hamming window defined by

$$\forall n \in [0, N_w[\,, w_n = \frac{1}{4 \times 0.54}(0.54 - 0.46 \times cos(2\pi \frac{n}{N_w}))$$

is applied to each frame. From there, all frames are overlapped and added together to reconstruct the full length signal. The zeros at the front and rear are then removed to produce a restored version of the original sound.

Note that, for signals with a left and right channel, the algorithm outlined above was implemented on each channel separately so that the stereo properties of the sound are kept intact.

5 Results

We found that the burst removal algorithm was extremely effective in detecting bursts of noisy samples. For every recording in our data set, the appropriate noisy samples were isolated and interpolated depending on the value of K. The interpolation step was also found the be very effective for reconstructing the damaged samples. As one can notice in Figure 10, the bursts of noisy samples are interpolated smoothly, and do not alter the signal too drastically.

When listening to the reconstruction results, we were extremely pleased to find that the artefacts due to crackle, hiss, and scratches in the original recordings were toned down. The overall musicality of the reconstructed sample was also extremely good, as the algorithm did not introduce new artefacts of its own. Overall, we recommend this algorithm for old recording restoration. Its implementation is quite simple, and it is suitable for most restoration tasks.

We did however find that it could be improved in a few ways. First, when dealing with Moodymann's song, we found that the burst removal algorithm had a tendency to interpolate background noise that is purposefully present in the song. As such, the value of k had to be set to at least 2, a value at which barely any samples were identified and interpolated. Second, although the technique definitely tones down the amount of noise present in the signal, it does not completely eliminate it. As such, its performance compared to professional audio engineers is still lesser.

6 Discussion

Future users of our implementation of the algorithm outlined in [2] and [3] should be aware that multiple trials are needed to find the appropriate parameters for the restoration task at hand. Most notably, the parameter K must be experimented with. Lower values tend to provide a more complete restoration that can lead to the transformation of unwanted features, whereas higher values limit the restoration process to significant bursts.

In the future, we also believe that using other interpolation techniques could lead to better results. Perhaps mixing the algorithm's powerful burst identification steps with other interpolation algorithms, such as Savitsky-Golay smoothing [11], would lead to better interpolation of noisy samples. These ideas are left as future work. We are confident that the burst restoration techniques explored in this report can provide us with a framework for improved automatic audio restoration.

Conclusion

In conclusion, we implemented two different audio restoration techniques which focused on a variety of audio defects. The method outlined by the Godsill textbook [1] attempts to remove wow and flutter artefacts in audio signals. Our implementation was successful when restoring simple signals. It however failed on longer musical excerpts. On the other hand, our implementation of the burst restoration algorithm presented in [2] [3] was able to remove most of the artefacts present in the degraded audio excerpts we tested it with. With proper parameter tuning, it toned down most of the defects present in our test recordings. Overall, we are happy with the knowledge we gained by completing this project. We hope it can serve as a basis for some of our future work in the field of music technology, a field which we were introduced to at McGill University, and that we will cherish for years to come.

Statement of Contributions

Dorian Desblancs: Burst Restoration algorithm implementation and testing, report
James Mesich: Wow and Flutter algorithm implementation and testing, report

GitHub

Link: https://github.com/jmesich/mumt501Final

References

[1] S. Godsill, P. Rayner, and O. Cappé, "Digital audio restoration," in *Applications of digital signal processing to audio and acoustics*, pp. 133–194, Springer, 2002.

[2] L. Oudre, "Automatic detection and removal of impulsive noise in audio signals," *Image Processing On Line*, vol. 5, pp. 267–281, 2015.

[3] L. Oudre, "Interpolation of missing samples in sound signals based on autoregressive modeling," *Image Processing On Line*, vol. 8, pp. 329–344, 2018.

[4] J. Ó. Ruanaidh and W. Fitzgerald, "Interpolation of missing samples for audio restoration," *Electronics Letters*, vol. 30, no. 8, pp. 622–623, 1994.

[5] L. W. Biscainho, P. S. Diniz, and P. A. Esquef, "Arma processes in sub-bands with application to audio restoration," in *ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No. 01CH37196)*, vol. 2, pp. 157–160, IEEE, 2001.

[6] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[7] N. Levinson, "The wiener rms error criterion in filter design and prediction, appendix b of wiener, n.(1949)," *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*, 1949.

[8] J. Durbin, "The fitting of time-series models," *Revue de l'Institut International de Statistique*, pp. 233–244, 1960.

[9] W. F. Trench, "An algorithm for the inversion of finite toeplitz matrices," *Journal of the Society for Industrial and Applied Mathematics*, vol. 12, no. 3, pp. 515–522, 1964.

[10] W. F. Trench, "The relationship between trench's toeplitz inversion algorithm and the gohberg-semencul formula," *J. Soc. Indust. Appl. Math*, vol. 12, pp. 515–522, 1964.

[11] A. Savitzky and M. J. Golay, "Smoothing and differentiation of data by simplified least squares procedures.," *Analytical chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964.