Practical Work Report ES101

Gabriel Dupuis - Jules Bernard

January 30, 2025

Abstract

This practical work report considers various methods used to address signal processing problems. Its objective is to utilize different tools presented during the ES101 course, such as the Z-transform, Fourier transform, and cross-correlation. This study will address both practical and theoretical questions, explaining each program step-by-step and supporting the results with calculations and justifications to fully apply the fundamental principles of the ES101 course.

Contents

1 Introduction

This report explains the methods used to address signal processing problems in the ES101 course. Each of these problems consists of 2 or 3 exercises to be solved using MATLAB with detailed signal processing functions introduced at the beginning of each exercise.

2 Project No. 1

2.1 Exercise 1: Filtering Sinusoids

The objective of this exercise is to reconstruct a distorted audio signal, with only a .wav file available. The steps to follow are:

1. Discretize, digitize, and display the frequency spectrum of the signal to determine the noise frequency.

2. Remove noisy frequencies from the signal using two methods: the first being a brute-force approach to zero out certain frequencies and the second, the intended method, involves synthesizing a reject filter of the IIR type.

The functions used are `audioread('...')` to convert the audio signal to an array, `fft(y)` to apply the Fourier transform, `ifft(x)` for the inverse transform, and `filter(A,B,x)` to apply a quotient-type filter with numerator coefficients `A`, denominator coefficients `B`, and `x` the time-domain signal.

```matlab
% MATLAB code for signal filtering
[y, Fs] = audioread('Mo11.wav');
N = length(y);
x = fft(y);
disp(Fs);
disp(N);

plot(abs(x));

for i = 1:2 % Repeat the operation twice
    [m, i_m] = max(abs(x)); % Get the index of the maximum
    f = (i_m-1)/N*Fs; % Get the associated frequency
    A = [1 -2*cos(2*pi*f/Fs) 1]; % Filter numerator (notch filter)
    B = [1 -2*0.9*cos(2*pi*f/Fs) 0.9^2]; % Filter denominator
    x2 = filter(A,B,ifft(x)); % Filtered signal
    x = fft(x2);
end

audio = ifft(x);
sound(audio, Fs);
```

Listing 1: Exercise 1 Code

We start by digitizing and analyzing the signal:

```
[y, Fs] = audioread('Mo11.wav');
N = length(y);
x = fft(y);
plot(abs(x));
```

The program outputs the following graph:

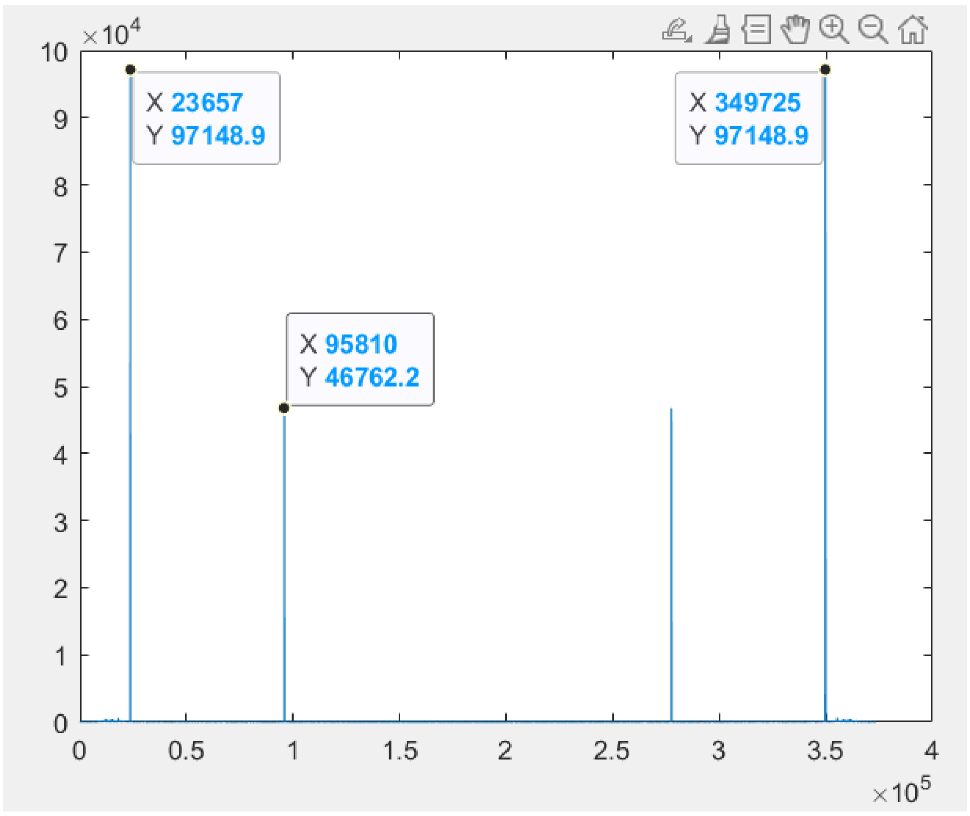

Figure 1: FFT of the obtained signal

2.1.1 Brute-Force Method

One initial approach would be to simply cancel out these two frequencies, or more precisely, a range of frequencies around them, since these frequency peaks are not unique but rather intervals. We obtain the following code to achieve this:

```
for i = 23580:23740
    x2(i) = 0;
end

for i = 95710:95910
    x2(i) = 0;
end

plot(abs(x2))
sound(ifft(x2), Fs);
```

This method is unreliable as it only works for this specific case and requires adjustments for different signals. Additionally, it is not very realistic, hence a more reasonable choice is to simulate a filter using the `filter` function.

2.1.2 IIR Filter Method

We aim to construct an IIR filter that cuts a specific frequency f_0. The filter has the form:

$$H(Z) = \frac{(Z - Z_0)(Z - Z_0^*)}{(Z - P)(Z - P^*)}$$

Here, $Z_0 = e^{2\pi j f_0 T_e}$.

We then place zeros and poles on the unit circle, noting that $H(Z)$ is stable only if the poles are inside the unit disk:

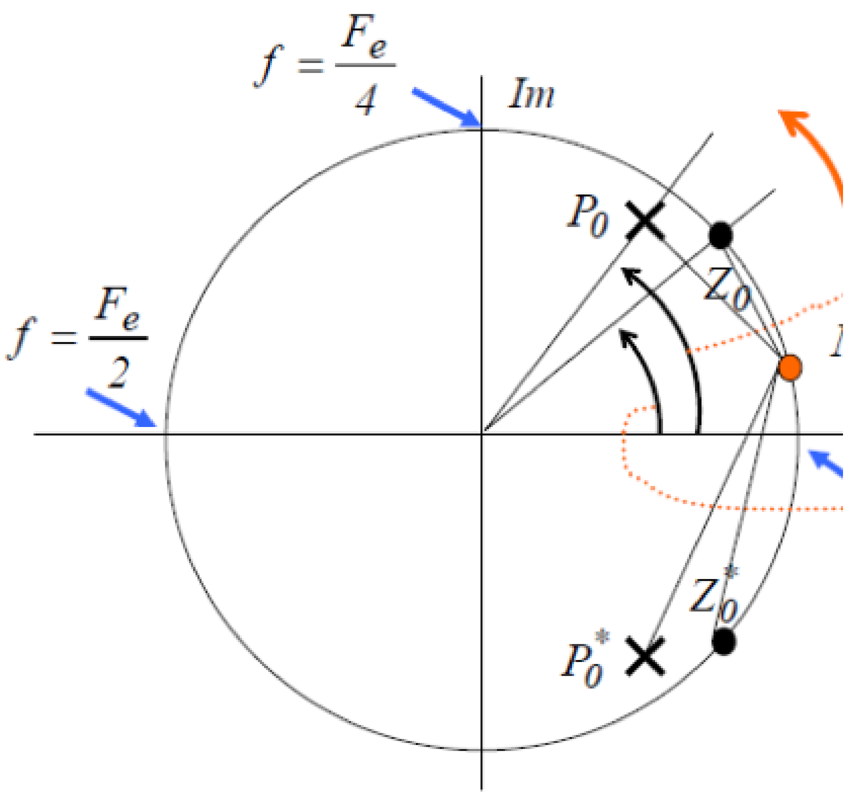

Figure 2: Placement of poles and zeros on the unit circle

The poles and zeros will have the same argument with $|P_0| < |Z_0|$. We choose $|P_0| = 0.9$ in our program. Under these assumptions, the filter will reduce frequencies near f_0 without affecting others.

```
Z0 = exp(2*pi*1i*(I-1)/length(x2));
b = [1, -Z0-conj(Z0), Z0*conj(Z0)]; % Numerator
```

a = [1, -Z0*0.5-conj(Z0)*0.5, 0.25*Z0*conj(Z0)]; x2 = filter(b, a, ifft(x2));

The coefficients of the numerator are labeled `b`, while those of the denominator are `a`. Additionally, the index `I-1` is used to account for MATLAB's 1-based indexing compared to the 0-based indexing typically used in signal processing. Finally, the filter function must be applied to the time-domain signal.

2.1.3 Obtaining the Maximum and Results

We now extract the value of f_0 using the `max` function, which returns the index of the maximum value in the list. The loop is repeated as many times as there are noisy frequencies, which, based on Figure 13, is twice.

```matlab
for i = 1:2
    [m, i_m] = max(abs(x)); % Retrieve the index of the maximum
    ...
end
```

After filtering, we listen to the signal and recognize the "Queen of the Night" aria from Mozart's *The Magic Flute*.

2.2 Exercise 2: Simulated Rotating Sound

The objective of this exercise is to generate a rotating sound from a given audio file (`Erlk.wav`). The functions used are the same as in Exercise 1, and all calculations are performed on the time-domain signal. Each sample $x[k]$ is shifted by a calculated value.

```matlab
% MATLAB code to generate a rotating sound
v = 340; % Speed of sound in air (m/s)
d = 0.30; % Distance between the ears (m)
[x1, Fe] = audioread("Erlk.wav"); % Read the audio file
theta = zeros(length(x1), 1); % Initialize angles

x2 = zeros(length(x1), 1); % Initialize the rotated sound
x = 2; % Number of rotations (x/2)
for i = 1:size(x1)-1
    theta(i) = x * pi * i / length(x1);
    tau = d * sin(theta(i)) / v;
    q = floor(tau * Fe); % Calculate integer shift
    x2(i) = x1(i - q); % Apply the time shift
end

y = [x1, x2];
sound(y, Fe);
```

Listing 2: Exercise 2 Code

The justification for these formulas is based on the diagram and reasoning below:

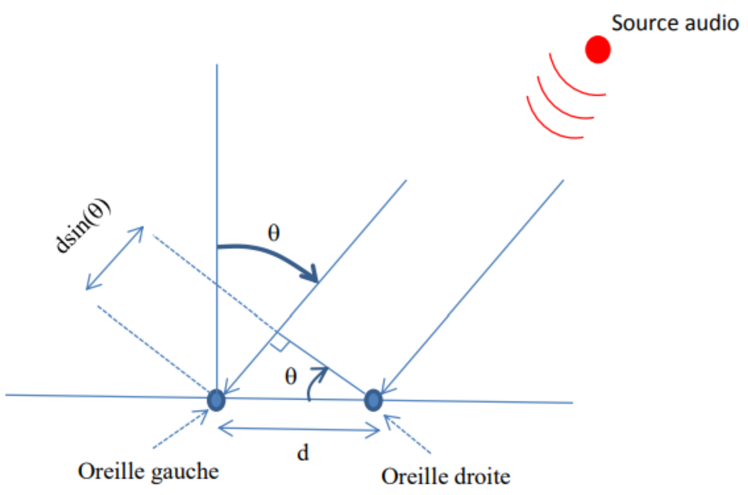

Figure 3: Diagram of the time shift between ears for an incident sound wave

We define $\tau = \frac{d\sin(\theta)}{v}$ with θ as shown in the figure.

$$x_2(nT_e) \approx x_1\left((n - \lceil\frac{\tau}{T_e}\rceil)T_e\right)$$

The variable x in our program modifies the rotation speed. Listening to the sound confirms the program's correctness.

3 Project No. 2

3.1 Exercise 1: Correcting a Visual Offset

The objective of this exercise is to reconstruct an image with randomly shifted rows using signal analysis tools such as correlation.
We start by reading the input image:

```
% MATLAB code for filtering the signal
clear B;
B = imread('fichier2.bmp', 'bmp');
B = 255 * B;

% Image size
len = size(B);
```

Listing 3: Reading the image.

Next, the steps are:

1. Determine the offset corresponding to the "maximum" correlation between successive rows. This is done using the `xcorr` function, which returns correlation values at each offset index. Rows close to their original position will have a higher correlation. We extract the maximum from the vector returned by `xcorr`.

2. Shift the second row by the offset found in step 1 using the `circshift` function, which circularly shifts a vector by the specified amount.

```
for i = 2:len(1)
    g = xcorr(B(i-1, :), B(i, :)); % Step 1
    [m, i_m] = max(g);
    B(i, :) = circshift(B(i, :), i_m); % Step 2
end
```

Listing 4: Performing offset correction using `xcorr` and `circshift`.

Here, we notice that the first row has not been shifted and has instead forced all subsequent rows into place, creating a global offset in the image (see Figure 1).

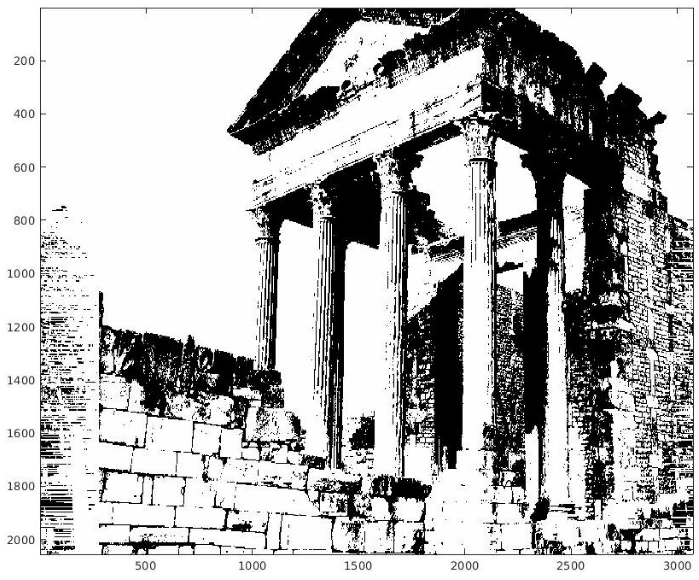

Figure 4: Reconstructed image with a global offset

The image now needs a global shift correction. Two approaches are described below.

3.1.1 Brute-Force Method

In this approach, we manually determine the index corresponding to the break in the image on the left.
We find that the index is 277.
Next, we apply a global shift to all rows using the `circshift` function.

```
% Shift the image by 277
for i = 2:len(1)
    B(i, :) = circshift(B(i, :), -277);
end

image(B);
colormap("GRAY");
```

Listing 5: Finding the global offset

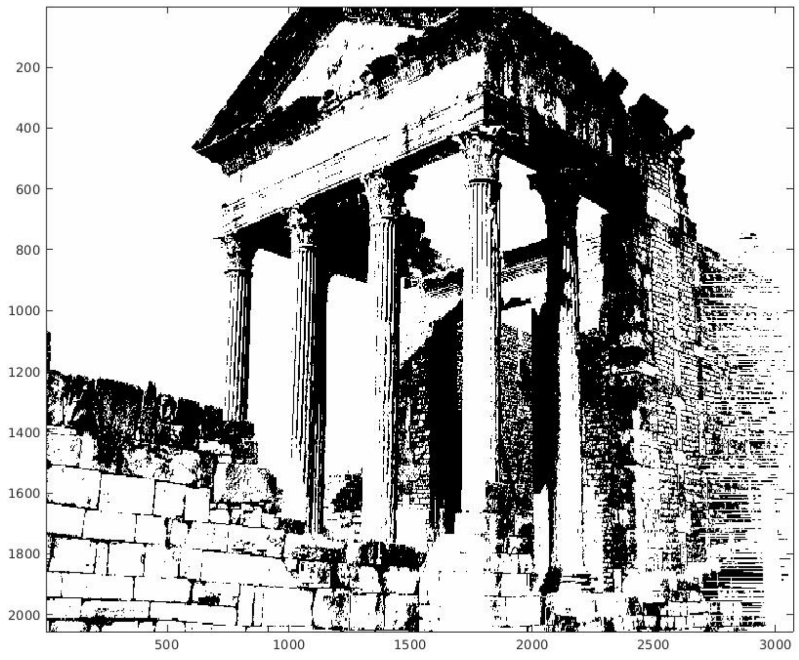

Figure 5: Manually corrected image after global shift

3.1.2 Correlation Method

In this approach, we search for a break between columns by analyzing the correlation between successive columns. The minimum correlation between columns indicates the most likely point of a break.
The index of this break will provide the offset needed to move the break to index 0.
We ultimately find that the index is 303.

```
X_max = [];
for j = 2:len(2)
    g = xcorr(B(:, j-1), B(:, j));
    X_max = [X_max g(1)];
end
[min, j_min] = max(-X_max);

% Shift the image by j_min = 303
for i = 2:len(1)
    B(i, :) = circshift(B(i, :), j_min);
end

image(B);
colormap("GRAY");
```

Listing 6: Finding the global offset using correlation

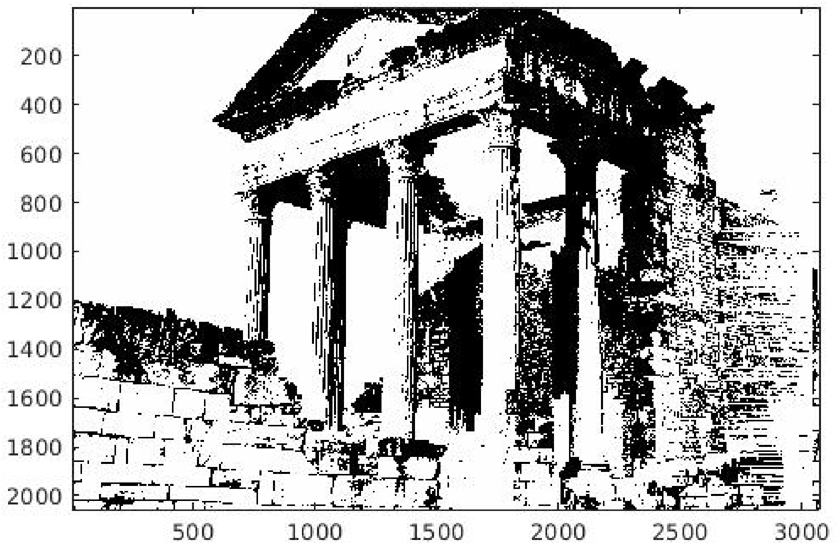

Figure 6: Image corrected using the correlation method

3.1.3 Complete Code

Complete code:

```
% Reading the image
clear B;
B = imread('fichier2.bmp', 'bmp');
B = 255 * B;

% Image size
len = size(B);

% Steps:
%   1. Determine the offset corresponding to the "maximum" correlation
    between
%       each pair of suçessive rows.
%   2. Shift the second row by the offset found in step 1.
for i = 2:len(1)
    g = xcorr(B(i-1, :), B(i, :)); % Step 1
    [m, i_m] = max(g);
    B(i, :) = circshift(B(i, :), i_m); % Step 2
end

% Search for columns with the lowest correlation and extract the index
% corresponding to the break
X_max = [];

for j = 2:len(2)
    g = xcorr(B(:, j-1), B(:, j));
    m = max(g);
    X_max = [X_max g(2055)];
end
[min, j_min] = max(-X_max);

% j_min gives the correct global offset index
for i = 2:len(1)
    B(i, :) = circshift(B(i, :), -j_min);
end

image(B);
colormap("GRAY");
```

Listing 7: Complete code for Exercise 1

3.2 Exercise 2: Inverse Permutation of a Signal

The objective of this exercise is to exploit the symmetry of a permutation applied to a signal in order to reconstruct it.

We analyze the signal in the frequency domain using the Fourier transform:

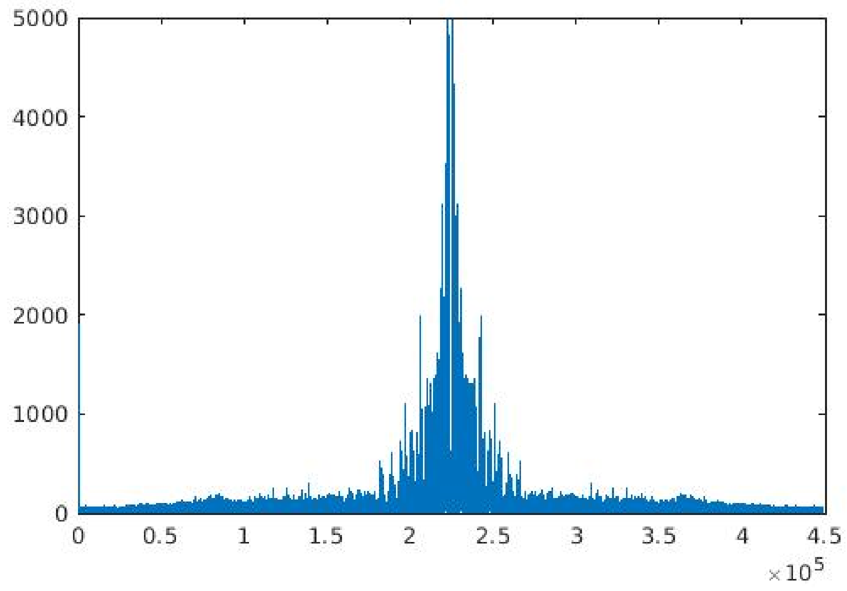

Figure 7: Initial Fourier spectrum

Assumptions: We assume that the sampling frequency is such that N is divisible by 2 and 4.

Method: All frequencies in the range $]0, F_e/4[$ are symmetrically permuted with those in $]F_e/4, F_e/2[$ around $F_e/4$. Similarly, frequencies in $]F_e/2, 3F_e/4[$ and $]3F_e/4, F_e[$ are permuted around $3F_e/4$.
The `fft` function returns the discrete Fourier transform of the signal, which contains N points, $n \in [0, N-1]$, with each normalized frequency given by $\nu = n/N$.
Thus, the point associated with F_e/k is N/k.
We permute each point $k \in [1, ..., N/4 - 1]$ with its symmetric point $k' = N/4 + k \in [N/4 + 1, N/2 - 1]$, and similarly for points in $[N/2 + 1, 3N/4 - 1]$.

Code: We implement the above description by adding +1 to each index to account for MATLAB's indexing convention. We iterate over the indices from $1(+1)$ to $N/4 - 1(+1)$, performing the described permutations. The modified Fourier spectrum confirms that the operation was successful (see Figure 4).

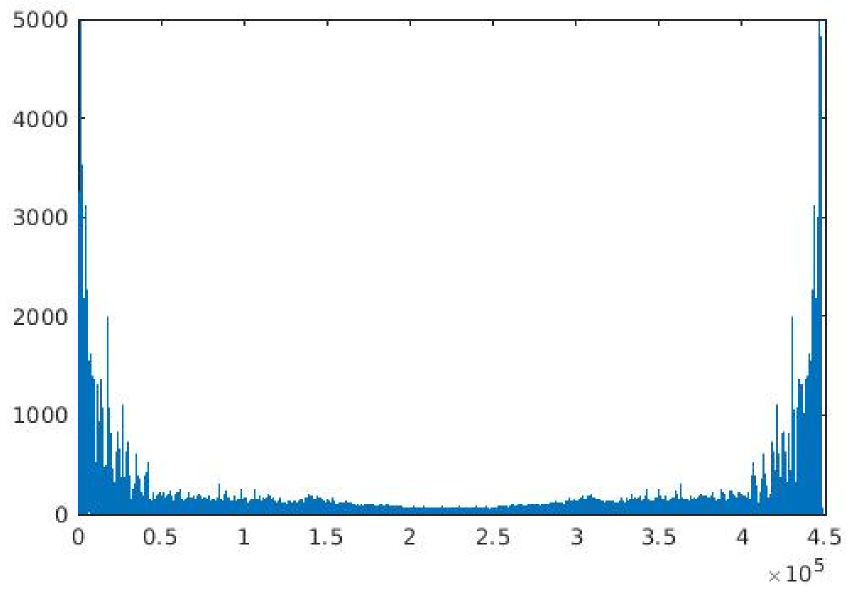

Figure 8: Final Fourier spectrum

3.2.1 Complete Code

Complete code:

```
[y, Fs] = audioread('sons_projet_2/canal.wav');
N = length(y);
x = fft(y);

for i = 1:floor((N/4+1)-1) % Permute frequencies symmetrically around
    Fe/4
     % Index convention: X[i] <-> X[n-1].
     % There are N frequency points: 0, 1/N, ..., (N-1)/N
     % The index for Fe/2 is N/2 + 1
     a = x(i+1);
     b = x((N/2+1) - i);
     x(i+1) = b;
     x((N/2)+1 - i) = a;

     a = x((N/2+1) + i);
     b = x(N+1 - i);
     x((N/2+1) + i) = b;
     x(N+1 - i) = a;
end

y = ifft(x);
sound(y, Fs);
```

3.3 Exercise 3:

The objective of this exercise is to reconstruct the two interlaced signals. The method is very simple and can be summarized in a few lines of code:

```
[y, Fs] = audioread('sons_projet_2/Encode.wav');
N = length(y);

y1 = y(1:2:N);
y2 = y(2:2:N);

sound(y2, Fs);
```

Listing 8: Complete code.

3.3.1 Signal Analysis

When listening to the signal, one can hear a dissonant superposition of two sounds that need to be decomposed. The spectrum shows a form of superposition of two signals.

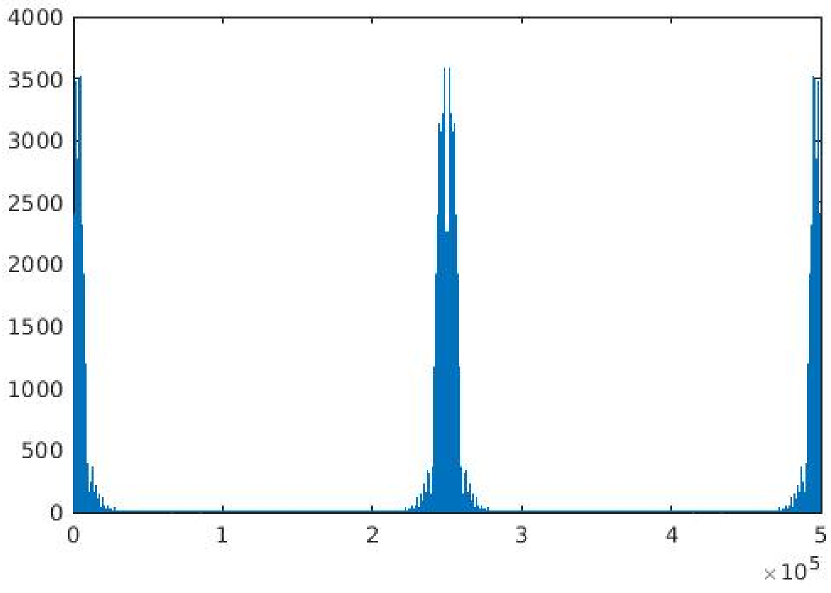

Figure 9: Combined Fourier Spectrum

3.3.2 Computation of the Interlaced Fourier Transform

Let $x[k]$ be an arbitrary sampled signal.
We define the interlaced signal with zeros as:

$$x'[k] = \begin{cases} x[k/3] & \text{if } k \equiv 0 \mod 3 \\ 0 & \text{otherwise} \end{cases}$$

By calculating the Fourier transform:

$$X(n) = \sum_{k=0}^{N-1} x[k] e^{-2\pi i k n / N}$$

With a change of indices:

$$\begin{aligned}
X'[n] &= \sum_{k=0}^{3N-1} x'[k]e^{-2\pi ikn/3N} \\
&= \sum_{k=0}^{N-1} x'[3k]e^{-2\pi i3kn/3N} + 0 \\
&= \sum_{k=0}^{N-1} x[k]e^{-2\pi ikn/N} + 0 \\
&= X[n \mod N]
\end{aligned}$$

Finally, $X'[n] = X[n \mod N]$, meaning that interlacing with zeros leaves the Fourier spectrum almost unchanged, repeating it three times. Similarly, interlacing two signals x_1 and x_2 into x' results in:

$$X'[n] = X_1[n \mod N] + X_2[n \mod N]$$

This duplicates the signal twice, increasing the sampling frequency.

3.3.3 Fourier Spectrum Analysis

According to the spectrum shown in Figure 6, we observe a duplication of the spectrum, suggesting that the two signals are interlaced. By extracting every other sample using vector operations in Matlab, we obtain two signals, $y1$ and $y2$, whose Fourier spectra are shown in Figures 7 and 8, respectively. This yields the desired result.

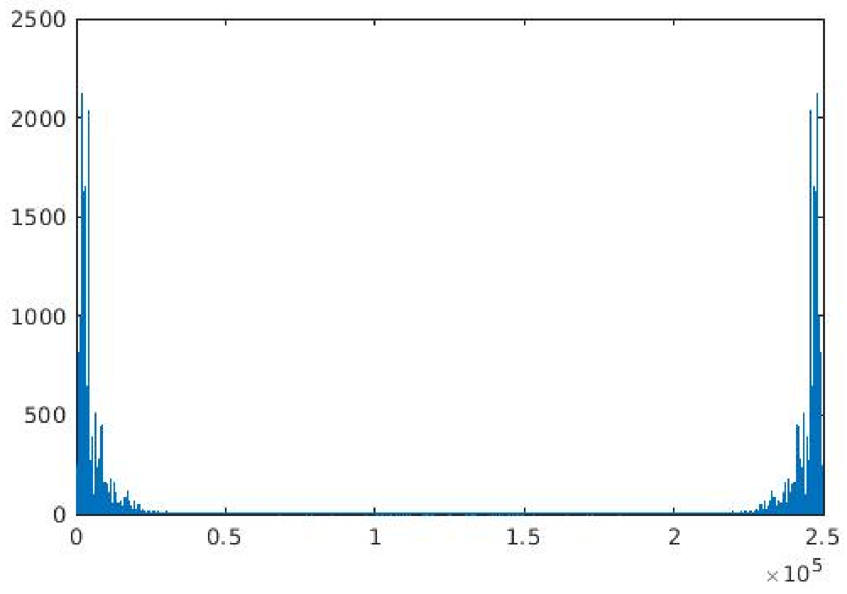

Figure 10: Fourier Spectrum of Signal 1

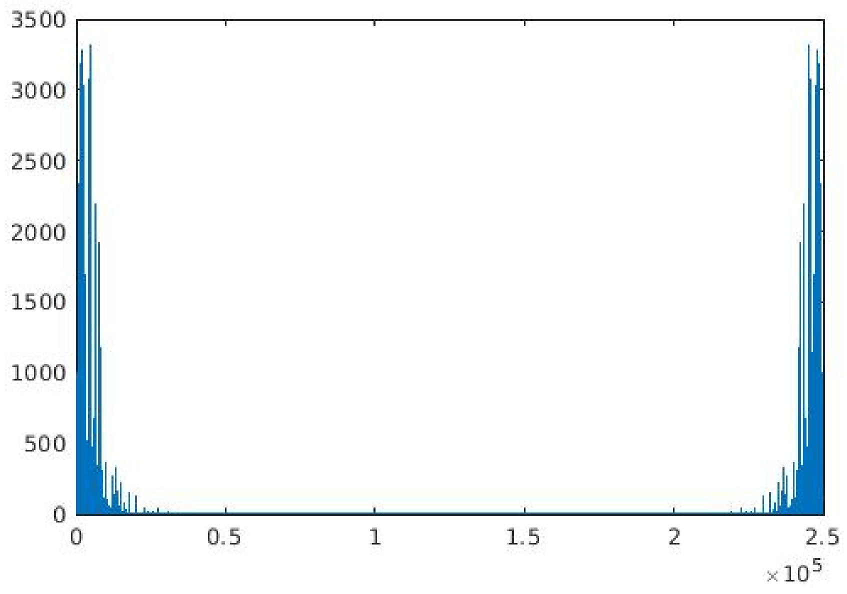

Figure 11: Fourier Spectrum of Signal 2

```
[y, Fs] = audioread('sons_projet_2/Encode.wav');
N = length(y);

y1 = y(1:2:N);
y2 = y(2:2:N);

sound(y2, Fs);
```

Listing 9: Complete code.

3.3.4 Results

We can hear that the two signals have been successfully separated. One of the signals is a piano excerpt from "Les copains d'abord" by the illustrious Georges Brassens.

4 Project No. 3: Radio Communications

4.1 Exercise 1: Filtering a Signal with an Echo

4.1.1 Hypotheses

1. The echo can be simply modeled here as passing through a filter:

$$H(Z) = 1 + \alpha Z^{-p_0}$$

   This implies that the original signal is summed with a delayed version by a time $\tau = p_0 T_e$ and attenuated by a real coefficient α less than 1 (these are the characteristics of this project).
   The provided signal can therefore be expressed as:

$$x(n) = s(n) + \alpha s(n - p_0)$$

2. The signal is white noise, thus it is WSS (Wide-Sense Stationary), centered, and uncorrelated.

4.1.2 Steps

We aim to find p_0 and α. The steps are as follows:

1. Compute the autocorrelation using the function `xcorr`.

2. Extract the first maximum of the autocorrelation, obtained for $\tau = 0$.

3. Extract the second maximum of the autocorrelation, obtained for $\tau = p_0$.

4. Calculate α using p_0, the values of $x(n)$, and the known autocorrelation $\gamma_x[k]$.

$$\begin{cases} \gamma_x[0] = \gamma_s[0] + \alpha^2 \gamma_s[0] \\ \gamma_x[p_0] = \alpha \gamma_s[0] \end{cases}$$

   Solving this system yields the equation:

$$\gamma_x[p_0]\alpha^2 - \alpha\gamma_x[0] + \gamma_x[p_0] = 0$$

   From which:

$$\alpha_1 = \frac{\gamma_x[0] + \sqrt{\gamma_x[0]^2 - 4\gamma_x[p_0]^2}}{2\gamma_x[p_0]} = 2$$

$$\alpha_2 = \frac{\gamma_x[0] - \sqrt{\gamma_x[0]^2 - 4\gamma_x[p_0]^2}}{2\gamma_x[p_0]} = 0.6$$

5. Filter the signal using the obtained p_0 and α with the filter $H(Z) = \frac{1}{1+Z^{-\alpha}}$.

4.1.3 Signal Analysis

In the figure of the correlation given by `xcorr`, we observe peaks corresponding to the previously described maxima. The maximum correlation occurs at zero delay, as described in the course.

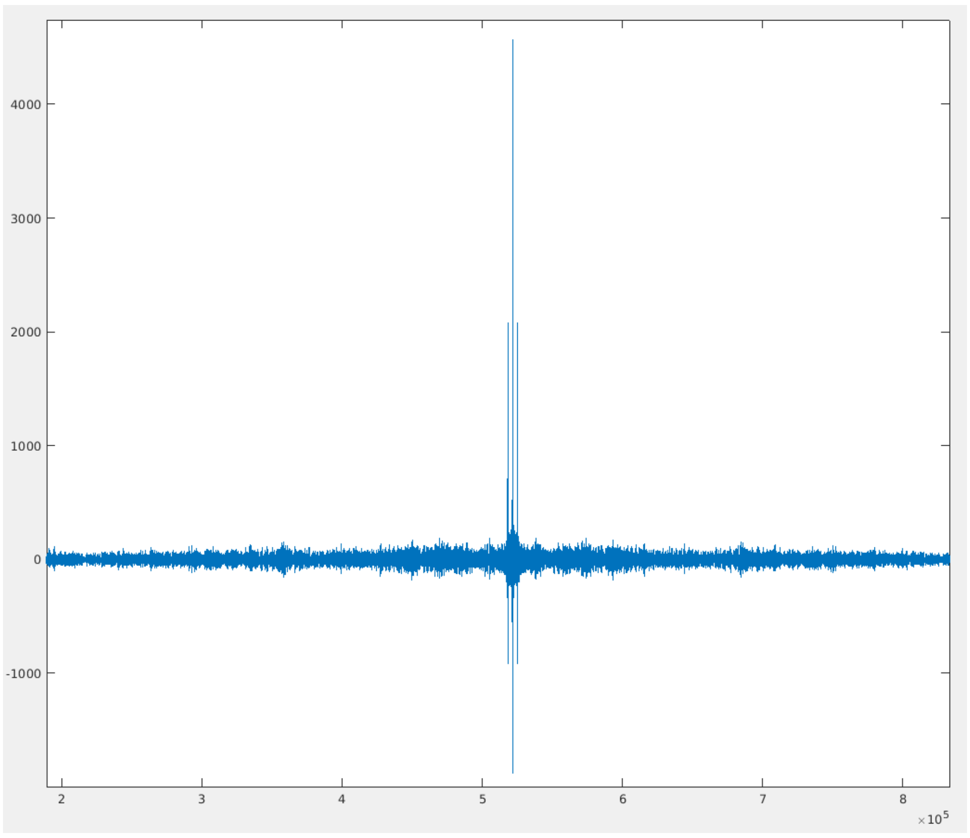

Figure 12: Graphical representation of the initial correlation from -N to N

Next, we filter out the central maximum using these operations:

```
[x,Fe]=audioread("Pa11.wav"); % Reading the signal
u=xcorr(x); % xcorr function (read the help and test it on a small
    vector)

N = length(x);

% Read from the graph:
xmid = (525091 - 521673) / 2; % Approximate distance p_0

[M1, I1] = max(u);
for (i = 521673 - xmid : 521673 + xmid) % Set the values of tau to zero
     in [p_0/2, -p_0/2]
     u(i) = 0;
end
```

We then obtain these correlations:

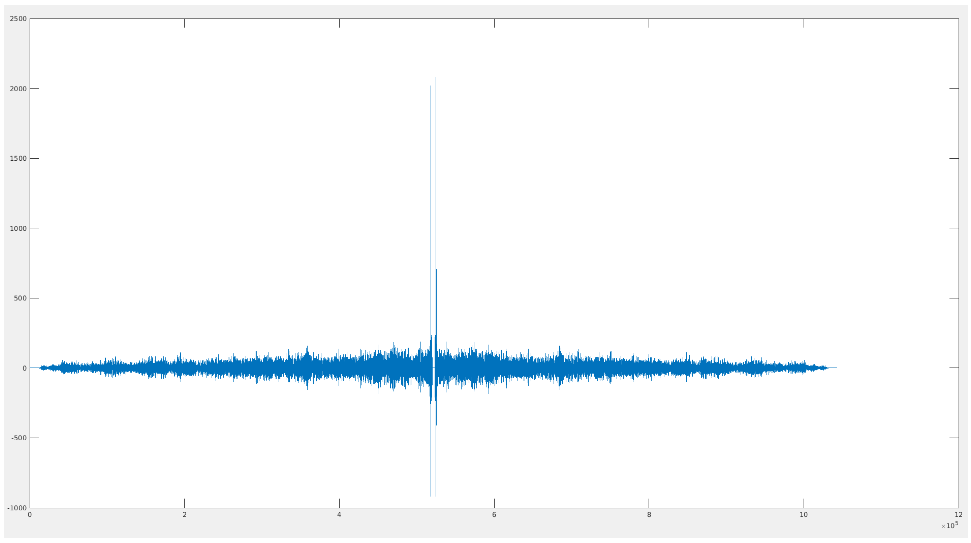

Figure 13: Graphical representation of the manually filtered correlation from -N to N

From this, we can deduce the maxima and thus the index p_0.

```
[M2, I2] = max(u); % Retrieve the index of the second maximum, p_0 + I1
p_0 = I1 - I2; % p_0
```

Finally, we filter the signal using the calculated values of α and γ.

```
gamma = M2 / M1;
alpha = (1 - sqrt(1 - 4 * gamma^2)) / (2 * gamma); % Alpha obtained
    from the calculation
x = filter([1], [1 zeros(1, p_0 - 1) alpha], x);
sound(x, Fe); % Echo output
```

4.1.4 Code

```
[x, Fe] = audioread("Pa11.wav"); % Reading the signal
u = xcorr(x); % xcorr function (read the help and test it on a small
    vector)

N = length(x);

% Read from the graph:
xmid = (525091 - 521673) / 2; % Approximate distance p_0

[M1, I1] = max(u);
for (i = 521673 - xmid : 521673 + xmid) % Set the values of tau to zero
     in [p_0/2, -p_0/2]
    u(i) = 0;
end

[M2, I2] = max(u); % Retrieve the index of the second maximum, p_0 + I1
p_0 = I1 - I2; % p_0

gamma = M2 / M1;
alpha = (1 - sqrt(1 - 4 * gamma^2)) / (2 * gamma); % Alpha obtained
    from the calculation
x = filter([1], [1 zeros(1, p_0 - 1) alpha], x);
sound(x, Fe); % Echo output
```

“‘latex

4.2 Exercise 2: Separation of Combined Signals

The objective of this second exercise is to separate two signals, s and w, where s is the desired signal and w is white noise. These two signals form x_1 and x_2 as follows:

$$x_1(n) = a_1 s(n) + b_1 w(n)$$

$$x_2(n) = a_2 s(n) + b_2 w(n)$$

Our coefficients are such that:

$$b_1 \gg a_1$$

$$b_2 \gg a_2$$

Although this task seems difficult, the fact that b_1 and b_2 are very large is advantageous and helps extract the desired signal. The main part of this exercise is theoretical since the program is very short, as shown below:

```
[x1, Fs1] = audioread('sounds/x1.wav');
[x2, Fs2] = audioread('sounds/x2.wav');

N = length(x1);
rho = sum(x1.*x2) / sum(x2.*x2);

eps = x1 - rho * x2; % Corrected signal (see demonstrations below)

corr = sum(eps .* x2) / N; % Verify that the signals are uncorrelated
disp(corr); % Correlation close to 0

sound(eps, Fs1 * 2);
```

All expressions used in this exercise are detailed in the calculations below. We use a power inversion technique to calculate a minimal coefficient ρ such that $\epsilon(n) = x_1(n) - \rho x_2(n)$. Let us begin by determining the value of ρ that minimizes the power of $\epsilon(n)$.

$$P(x(n)) = E(|x(n)|^2) = \frac{1}{n}\sum_{i=0}^{n-1} x(i)$$

We aim to find $\min_\rho E(|\epsilon(n)|^2)$.

$$|\epsilon(n)|^2 = |x_1(n)|^2 - 2\rho|x_1(n)x_2(n)| + \rho^2|x_2(n)|^2$$

$$E(|\epsilon(n)|^2) = E(|x_1(n)|^2) - 2\rho E(|x_1(n)x_2(n)|) + \rho^2 E(|x_2(n)|^2)$$

$$\frac{dP(\epsilon(n))}{d\rho} = -2E(|x_1(n)x_2(n)|) + 2\rho E(|x_2(n)|^2) = 0$$

$$\rho = \frac{E(|x_1(n)x_2(n)|)}{E(|x_2(n)|^2)} = \frac{a_1a_2 + b_1b_2}{a_2^2 + b_2^2}$$

Next, we need to demonstrate that ϵ and x_2 are uncorrelated:

$$\epsilon(n) = a_3 s(n) + b_3 w(n) = (a_1 - \rho a_2)s(n) + (b_1 - \rho b_2)w(n)$$

$$E(\epsilon(n)x_2(n)) = E((a_3 s(n) + b_3 w(n))(a_2 s(n) + b_2 w(n)))$$

$$E(\epsilon(n)x_2(n)) = a_3a_2E(s(n)^2) + b_2b_3E(w(n)^2)$$

Since s and w are uncorrelated:

$$E(\epsilon(n)x_2(n)) = a_3a_2 + b_3b_2$$

Substituting the expression for ρ:

$$E(\epsilon(n)x_2(n)) = \rho(a_1^2 + a_2^2) - \rho(a_1^2 + a_2^2) = 0$$

Finally, from the previous calculations:

$$a_2a_3 = -b_2b_3, \quad \gamma_e(x_2) = \frac{a_2^2}{b_2^2} = \frac{b_3^2}{a_3^2} = \gamma_s$$

With this equality, we can insert the expression into the code to calculate ρ, and the correction is performed. We also calculate ϵ to verify experimentally that the two vectors are uncorrelated.

4.3 Filtering Two Sinusoids Using the Yule-Walker Method

The goal of this exercise is to filter the signal from Exercise 1.1 to remove the sinusoids using a notch filter. The steps are as follows:

1. Determine the filter's structure and order.
2. Find the filter coefficients using the Yule-Walker equation with the `aryule` function.
3. Filter the signal using the obtained coefficients.

4.3.1 Mathematical Principle

A sinusoidal signal is an autoregressive signal of order 2, expressed as:

$$\sin(n\omega) = 2\cos(\omega)\sin((n-1)\omega) - \sin((n-2)\omega)$$

By applying the Z-transform, the sinusoidal signal can be expressed as:

$$S(Z) = 2\cos(\omega)S(Z)Z^{-1} - S(Z)Z^{-2}$$

Thus, a notch filter of order 2 that cancels the sinusoidal signal can be written as:

$$H_\omega(Z) = 1 - 2\cos(\omega)Z^{-1} + Z^{-2}$$

To cancel two sinusoids of frequencies ω_1 and ω_2, the filter must have the form:

$$H(Z) = H_{\omega_1}(Z)H_{\omega_2}(Z)$$

This results in a filter of order 4.

4.3.2 Algorithmic Principle

Since the signal consists of two superimposed sinusoids and an excerpt from "The Magic Flute," the most primitive part of the signal is the sinusoidal component. The 4th-order prediction using the Yule-Walker method will provide the coefficients for the notch filter described above. We then apply the filter and play the signal.

```matlab
[x, Fs] = audioread('sounds/Mo11.wav');

yw = aryule(x, 4); % Retrieve filter coefficients
y = filter(yw, [1 0 0 0], x); % Filter the signal
sound(y, Fs); % Play the filtered signal
```

Listing 10: Complete Code

5 Conclusion

In the first lab, we worked on audio signals, filtering to recover a scrambled signal, and created a rotating sound effect.

The second lab focused on processing both audio and image signals. We performed autocorrelation and decoding operations, including signal permutation.

Finally, the third lab applied more advanced mathematical concepts to separate signals or remove echoes.

Throughout these signal processing labs, we explored various techniques and methods, demonstrating the practical applications of filters, correlation, and signal analysis with different types of signals, including images and sound waves. This project helped clarify several concepts from the course. “‘