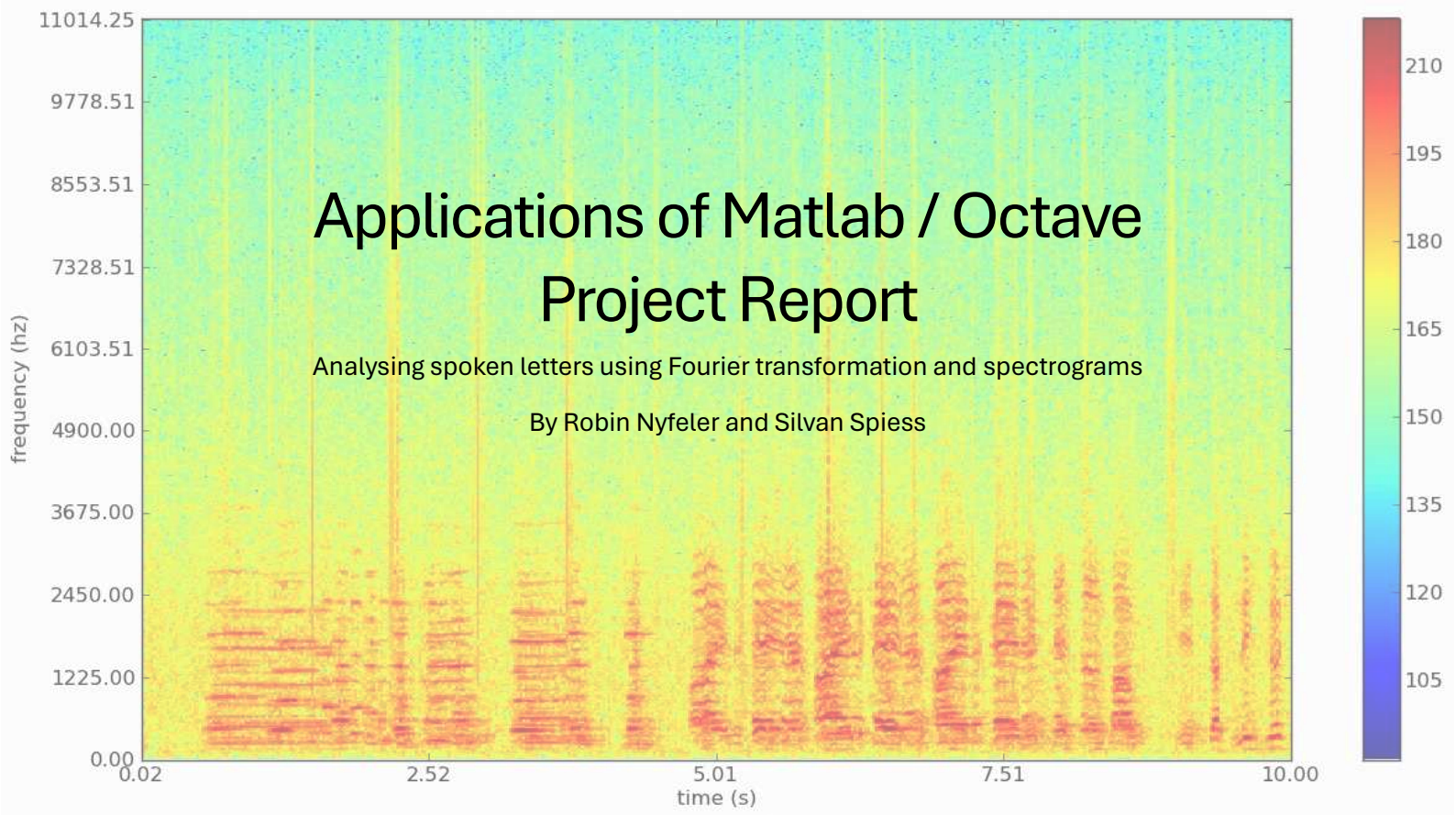
Applications of Matlab / Octave Project Report

Analysing spoken letters using Fourier transformation and spectrograms

By Robin Nyfeler and Silvan Spiess



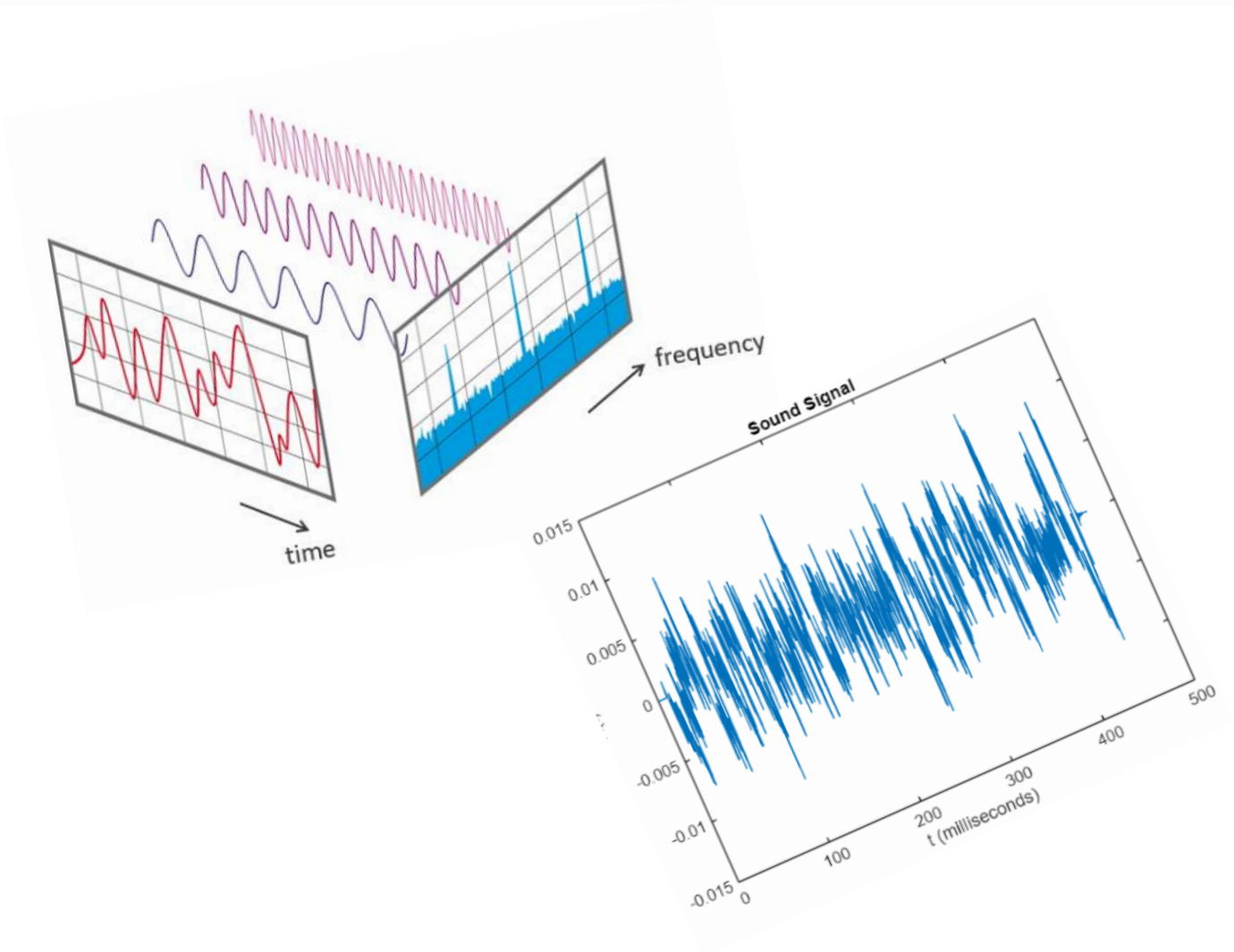


Table of contents

Introduction

In the Course Applications of Matlab / Octave, we have been tasked to work on a project out our own choice. We are students in the second year of the Micro- and medical technology bachelor program. In the current semester, we have introduced to the Fourier transformation in our calculus course. This gave us the motivation to use this application in our Project for this course.

Following up on this, our goal was to analyse audio using the Fourier transformation (and by comparing its coefficients). This led to unsatisfying results, thus we wanted to try the Spectrogram approach to analyse the Fourier transformed audio signals.

Our final goal was to determine, which letter was spoken by a person, by comparing the spectrogram of that audio signal to a pre-built spectrogram library of certain letter, spoken by that person. In the following chapter we will discuss how we achieved that.

Program structure

This project consists of two Matlab life scripts as well as few directories to store the gathered data.

- *audiofiles_lib:* This directory stores all audio files, which are used to fill the library of spectrograms. This directory's purpose is to provide audio files, for verification of the recordings by the user.
- *spectrogramfiles_lib:* This directory stores all the generated spectrogram files, which will be used to be compared with the spectrogram file where the spoken letter must be determined. Furthermore, it allows the user to visually inspect the spectrogram files manually.
- *result_dir:* In this directory, the recorded and generated audio file resp. spectrogram file will be stored, in order to determine the letter, which they represent.

fill_database_audio_reader.mlx

Library directory handling

This section introduces the library names and in case they don't exist, creates them.

```
% Directories used in this project
audiofiles_directory_name = "audiofiles_lib\";
spectrogramfiles_directory_name = "spectrogramfiles_lib\";

% Check if the audiofiles directory exists
if ~isfolder(audiofiles_directory_name)
    mkdir(audiofiles_directory_name);
    fprintf('Directory "%s" created.\n', audiofiles_directory_name);
else
    fprintf('Directory "%s" already exists.\n', audiofiles_directory_name);
end

% Check if the spectrogramfiles directory exists
if ~isfolder(spectrogramfiles_directory_name)
    mkdir(spectrogramfiles_directory_name);
    fprintf('Directory "%s" created.\n', spectrogramfiles_directory_name);
else
    fprintf('Directory "%s" already exists.\n', spectrogramfiles_directory_name);
end
```

Recording and storing the audio

After creating the recObj, the program asks the user for their name, and which letter they are going to say. Then it starts a procedure of a countdown as well as a recording timeslot of one second. After the sound has been recorded, the audio gets stored in the "*audiofiles_lib*" directory following the pattern of "*Name_Letter.m4a*". Obsolete variables get cleared.

Connect the Microphone.

```
Fs = 48000; % Sampling Frequency
recObj = audiorecorder(Fs,16,1,1);
```

Setup file names.

```
person_name = input('Enter your first name: ', 's');
spoken_letter = input('Enter which vowel you are going to say: ', 's');
audiofile_name = audiofiles_directory_name + person_name + "_" + spoken_letter + ".m4a";
spectrogramfile_name = spectrogramfiles_directory_name + person_name + "_" + spoken_letter + ".png";
clear person_name; clear spoken_letter;
for i = 3:-1:1
    disp(i)
    pause(1)
end
clear i;
```

Record voice.

```
recDuration = 1;
disp("Begin speaking.")
recordblocking(recObj,recDuration)
disp("End of recording.")
y = getaudiodata(recObj);
clear recDuration; clear recObj;
```

Cut and save the recording.

```
x = y(2.3e4:4.5e4, 1);
audiowrite(audiofile_name, x, Fs);
disp("audiofile saved successfully!")
clear y; clear x;
```

Generating the spectrogram

To be able to better analyse a signal with changing frequency over time, a spectrogram is used. For this the audio signal is broken down into steps of 500 readings, approximately 0.01s of audio. Next, the frequencies in this smaller piece are calculated with the fft() function.
The function FFT_Matrix() does this over the complete signal and gives back alle frequencies at any time in the audio signal. For better visualization, the amplitudes of the frequency is represented with a colour using FFT_Matrix_Colour(), this image is what is called a spectrogram. The absolute value of the amplitude does not have a meaning on its own.

```
x = audioread(audiofile_name);          % audiofile
step_size = 500;                        % Menge an Messwerten der Tondatei pro Spalte
sample_size = 1000;                     % Amount of FFT samples (erhöt auflösung NICHT!) >= step_size

M = FFT_Matrix(x,sample_size,step_size);
MC = FFT_Matrix_Colour(M(1:50,:),40,10,5);
img = flip(MC,1);                       % Origin (0,0) is bottom left corner
imshow(img)
imwrite(img,spectrogramfile_name)
disp("spectrogramfile saved successfully!")
clear x; clear M; clear freq; clear MC; clear img; clear step_size; clear sample_size;
```

(optional) Display of the recorded sound

To see, what got recorded, the user can look at the audio readings of the last recording. Mainly used for debugging.

```
T = 1/Fs;             % Sampling period
x = audioread(audiofile_name);
L = size(x,1);        % Length of signal
t = (0:L-1)*T;        % Time vector

plot(1000*t,x)
title("Sound Signal");
xlabel("t (milliseconds)");
ylabel("X(t)");
clear x; clear t; clear T; clear L;
```

(optional) Color gradient

Used to see the complete colour gradient and the level each colour represents. Only shows the relation to the other colours.

```
clear img;
img(1:1000,1:10,1:3) = 1;

for n = 1:500
    pic = double2rgb(n,500);
    t = n*10;
    img(1:1000,(t+1):(t+10),1) = pic(1);
    img(1:1000,(t+1):(t+10),2) = pic(2);
    img(1:1000,(t+1):(t+10),3) = pic(3);
end
imshow(img)
clear n; clear t; clear pic;
```

Own functions

- `linear_gradient(val,period,type_size)`:
  This function calculates from a single value the values of red, green and blue.
  val: the value to represent as colour
  period: size of one colour change, ex. blue to violet. From blue back to blue takes 6 periods.
  type_size: double(1.0) or int(255)
- `double2rgb(amp,max)`:
  Calculates the colour for amp in ratio to max. Max is the colour cyan and zero(min) is blue.
- `FFT_Matrix(Signal,sample_size,step_size)`:
  Calculates the Fourier spectrum in steps for a signal.
  Signal: the audio file in vector format.
  sample_size: Number of frequencies calculated with fft(). Does NOT increase the resolution.
  step_size: Number of readings used for one FFT.
- `FFT_Spectogram(FFT_Matrix,Max_Amp,sizeX,sizeY)`:
  Transform the FFT_Matrix to a spectrogram.
  FFT_Matrix: The matrix calculated with FFT_Matrix().
  Max_Amp: Highest absolute value in the FFT_Matrix. Will be cyan in the spectrogram.
  sizeX: Number of pixels in x-axis.
  sizeY: Number of pixels in y-axis.

analyze_audio_program.mlx

Result directory handling

This section introduces the result directory name and in case it doesn't exist, creates it.

```
% Directories used in this project
result_directory_name = "result_dir\";

% Check if the audiofiles directory exists
if ~isfolder(result_directory_name)
    mkdir(result_directory_name);
    fprintf('Directory "%s" created.\n', result_directory_name);
else
    fprintf('Directory "%s" already exists.\n', result_directory_name);
end
```

Record data and store it in a .m4a file

This is a similar procedure to the one in the "*fill_database_audio_reader.mlx*" file. After creating the recObj, it asks the user for their name. Since this code wants to figure out which letter was spoken, it does not ask for the spoken letter of the person. The recorded audio is stored in a m4a file in the "*result_dir*". Afterwards obsolete variables get cleared.

Connect the Microphone.

```
Fs = 48000; % Sampling Frequency
recObj = audiorecorder(Fs,16,1,1);
```

Setup file names

```
person_name = input('Enter your name: ', 's');
audiofile_name = result_directory_name + person_name + ".m4a";
spectrogramfile_name = result_directory_name + person_name + ".png";
for i = 3:-1:1
    disp(i)
    pause(1)
end
clear i; clear result_directory_name;
```

Record voice.

```
recDuration = 1;
disp("Begin speaking.")
recordblocking(recObj,recDuration)
disp("End of recording.")
y = getaudiodata(recObj);
clear recDuration; clear recObj;
```

Cut and save the recording.

```
x = y(2.3e4:4.5e4, 1);
audiowrite(audiofile_name, x, Fs);
disp("audiofile saved successfully!")
clear y; clear x; clear Fs;
```

Generating the spectrogram

Works the same way as when filling the library.

```
x = audioread(audiofile_name);          % audiofile
step_size = 500;                        % Menge an Messwerten der Tondatei pro Spalte
sample_size = 1000;                     % Amount of FFT samples (erhöt auflösung NICHT!) >= step_size
M = FFT_Matrix(x,sample_size,step_size);
MC = FFT_Matrix_Colour(M(1:50,:),40,10,5);
img = flip(MC,1);                       % Origin (0,0) is bottom left corner
imshow(img);
imwrite(img,spectrogramfile_name)
disp("spectrogramfile saved successfully!")
clear x; clear M; clear freq; clear MC; clear img; clear step_size; clear sample_size;
clear audiofile_name;
```

Comparing the generated spectrogram with the spectrograms in the library

This code compares the generated spectrogram with all spectrograms for the specific user in the library. It then adds a score for each comparison and the one with the highest score will represent the letter, which was spoken. The list is dynamically expanded for each compared spectrogram, but since we have only a handful of files in our libraries, this dynamic memory allocation can be neglected. The 3 values which are computed will be explained in a section below.

```
% https://ch.mathworks.com/matlabcentral/answers/1915530-how-to-create-4d-array-of-images?#answer_1176085
List_of_files  = dir(fullfile('spectogramfiles_lib\', '*.png'));
List_of_matches = {};
for a = 1:numel(List_of_files)
    % Get library spectogram and its name
    lib_image = imread(fullfile(List_of_files(a).folder, List_of_files(a).name));
    if contains(List_of_files(a).name(), person_name, 'IgnoreCase', true)
        Letter = List_of_files(a).name(end-4)
        comparison_image = imread(spectrogramfile_name, "png");

        ssimval = ssim(lib_image, comparison_image)
        peaksnr = psnr(lib_image, comparison_image)
        MSE_err = immse(lib_image, comparison_image)

        matching_score = ssimval * 100;
        %disp(Letter + matching_score);
        % Store result in a list
        List_of_matches = [List_of_matches; {Letter, matching_score}];
    end
end
clear a;
```

Determining which letter was said

Once the list "*List_of_matches*" is filled, the program finds the best match, buy a simple code, which takes the max SSIM value in the list. Telling the user which letter they have said, is the final step of this program.

```
highestValue = 0;
% Loop through the list of pairs to find the highest value
for i = 1:size(List_of_matches,1)
    % Get the current value
    currentValue = List_of_matches{i,2};

    % Check if the current value is higher than the highest value found so far
    if currentValue > highestValue
        highestValue = currentValue;
        highestValueIndex = i;
    end
end
disp(person_name + ", you said the letter: " + List_of_matches{highestValueIndex,1})
clear person_name; clear Letter; clear i; clear spectrogramfile_name;
```

Methods

Fourier Transformation

For the Fourier transformation, the Matlab function fft() is used. Calculates the frequencies in a signal, but no absolute amplitudes or frequencies. To get more samples out of fft() we add zeros at the end of the signal.

Spectrograms

There is a built-in function to render a spectrogram however, for this project we decided to implement our own function. All functions used are described under Own functions.
The principal of a spectrogram is to see, how the frequency components of a signal change over time. To achieve this, the signal is split into time intervals and Fourier transformed. For the visualisation of the relation between the frequencies, colour is used.

SSIM, PSNR and MSE

- **SSIM, Structural Similarity Index**:
  Source https://ch.mathworks.com/help/images/ref/ssim.html
  This function computes a value between 0 and 1, which indicates the structural similarity of two grayscale images.
- **PSNR, Peak Singal-to-Noise Ratio**:
  Source: https://ch.mathworks.com/help/images/ref/psnr.html
  This function is a tool to compare original images with compares images to determine their quality regarding the noise generated by compression. In our application, this can have a use, since we want to find the difference (or noise) between two images.
- **MSE, Mean Squared Error**:
  Source: https://ch.mathworks.com/help/images/ref/immse.html
  This is a very basic function, which computes the squared error for each entry in the array (if you consider images being very big array containing colour values). The lower the returned value, the smaller the difference between the two images.

Which method to use?

We have experimented with all three methods and concluded that these almost always agree with each other regarding the images we gave them. Therefore, we have opted to use the SSIM exclusively for determination of the spoken letter. For testing and further analysis, all three values are computed and can be inspected in the output of the program.

Usage

Filling up the database:

The user has to make sure, that the microphone of their device (laptop / PC) is properly set up. To fill the database, the user can work through the "*fill_database_audio_reader.mlx*" live script. It will tell which actions to take and when to speak. **Speak loud and clearly.**

Analysing spoken letters:

The user has to make sure, that the microphone of their device (laptop / PC) is properly set up. To analyse their voice, the user can work through the "*analyze_audio_program.mlx*" live script. It will tell which actions to take and when to speak. **Speak loud and clearly.**

Conclusion

In this project we have applied aspects of the lecture, such as images and fft, which helped us get to the result. Our program is good at recognising vowels and certain consonants. We have filled the database with letters A-Z, however the more letters the database has, the less reliable the program becomes. Nonetheless, there are some aspects, which could be improved for this project. One of them is the cutting of the audio signal to a shorter length. This is done with hardcoded parameters that are fairly well tested, but since every recording is unique, an algorithm for proper cutting of the audio signal could be implemented. Another aspect, which can be improved is finding the best matching spectrograms. In our code, we are only considering the SSIM values, however there are two more we are calculation and Matlab contains even more image comparison tools, which are not further mentioned here.
Regardless of those minor issues, the code functions in a way we have wanted it to, and thus are happy with the outcome.