Physical Computing Group 7E: Beatrix Zöllner, Jaspreet Singh, Tsungai Chipato

Lab 3: Processing

Exercise 1: *Write a MATLAB function that acquires an image and returns the number of large objects (BLOBs) in it. The function should also plot (in a mosaic of windows) all the intermediate images used to obtain the result (by using the MATLAB function subplot). Your standard solution counts only round objects.*
Answer: In order to obtain an image, we used the following command:

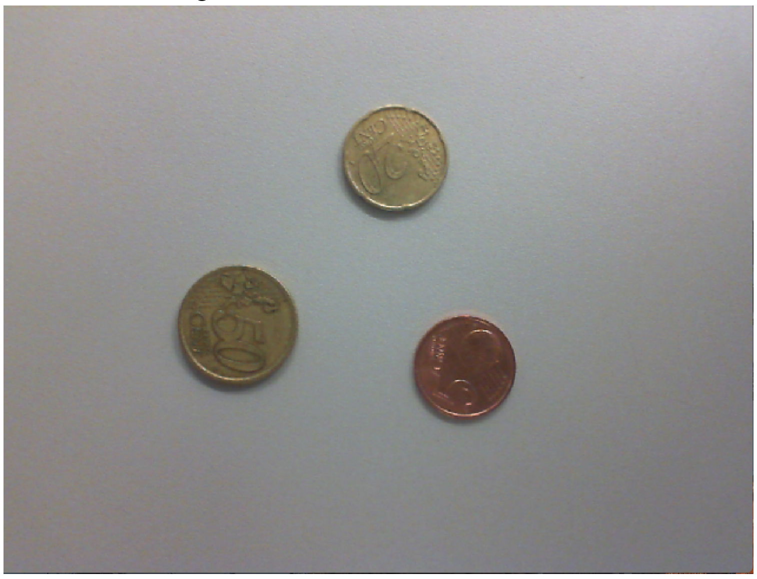
```
>> % We want a window which
% shows what the camera sees,
% so we can take a good picture of the coins:
vid = videoinput ('winvideo',1);
start (vid);
preview (vid);
```

```
>> % Now, we take a picture:
im = getsnapshot(vid);
stop(vid);
delete(vid);
```

```
>> % We look at the image:
imshow(im);
% We save the image (im):
imwrite (im, 'image.bmp', 'bmp')
```

We did not write these commands in our MATLAB function, because we wanted to place the camera in a good spot, so that we would be able to take a good picture. The MATLAB function describes the process once we have the image. The MATLAB function can thus be used by anyone who has the 'image.bmp' file.

This is our image:



There is a function generated by matlab that allows us to access and import the data from a specified file.We used this function so that we could retrieve the data from the files that we had saved on our computer so that we could use it directly. This is the function that we used:

```
function importfile(fileToRead1)
%IMPORTFILE(FILETOREAD1)
%  Imports data from the specified file
%  FILETOREAD1:  file to read

%  Auto-generated by MATLAB on 17-Nov-2017 14:41:40

% Import the file
rawData1 = importdata(fileToRead1);

% For some simple files (such as a CSV or JPEG files), IMPORTDATA might
% return a simple array.  If so, generate a structure so that the output
% matches that from the Import Wizard.
[~,name] = fileparts(fileToRead1);
newData1.(matlab.lang.makeValidName(name)) = rawData1;

% Create new variables in the base workspace from those fields.
vars = fieldnames(newData1);
for i = 1:length(vars)
    assignin('base', vars{i}, newData1.(vars{i}));
end
```

We already had the image saved on our workspace so we could use it directly. But, to use our MATLAB function in order to count the number of objects in the image above, one can use the following command:

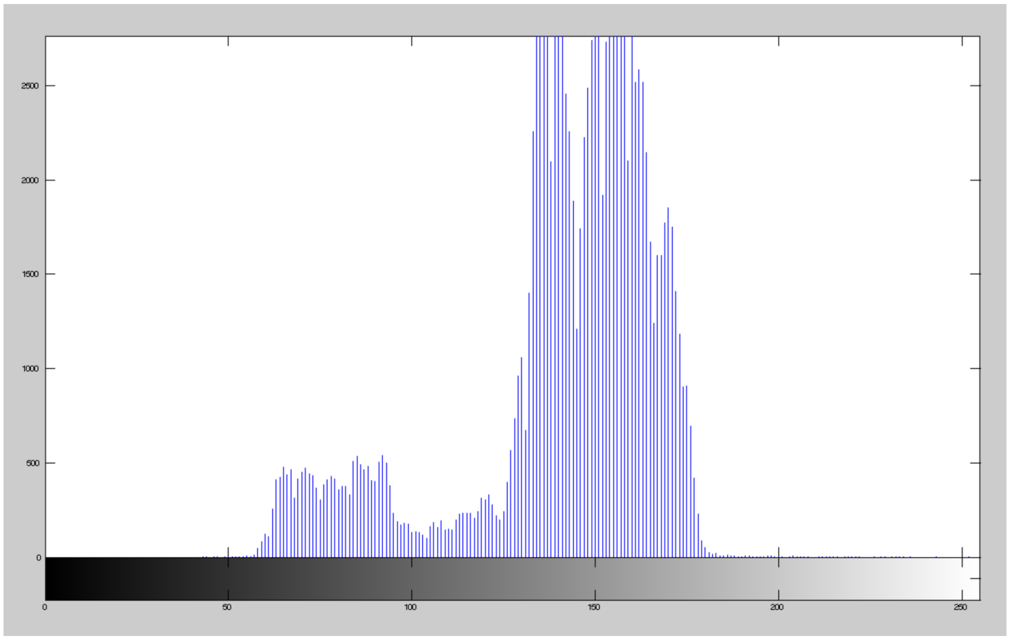
```
>> im=imread('image.bmp');
```

This is our MATLAB function:

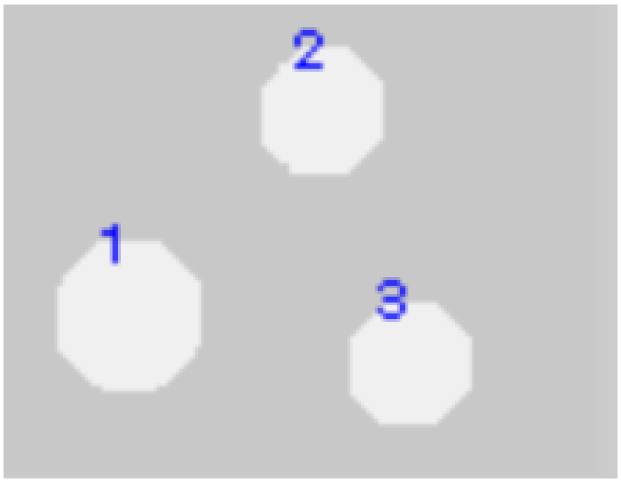
```
im = imread('image.bmp');
imshow(im);
pause(2);
% We determine the coordinates where
% we want to start cropping the image (im):
% We put imtool(im) in comments, because
% we only needed this during the process
% of counting the objects. But, as we write
% the function, we already know the coordinates
% for cropping the image. Same goes for
% imtool(im5) (later in the code).
% For im it would be:
% imtool(im);
% We crop the original image (im).
% We call the cropped image im1:
im1 = imcrop(im, [115, 60, 410, 325]);
% We look at the cropped image (im1):
imshow(im1);
% For 2 seconds:
pause(2);
% We save the cropped image (im1):
imwrite (im1, 'image1.bmp', 'bmp');
% We convert im1 to a grayscale image,
% which we call im2:
```

```
im2 = rgb2gray(im1);
% We look at im2:
imshow(im2);
% For 2 seconds:
pause(2);
% We save im2:
imwrite (im2, 'image2.bmp', 'bmp');
% We want a histogram of im2:
imhist(im2);
pause(2);
% We want MATLAB to determine
% a good threshold of im2.
% We assign the value of the threshold
% to the variable called level:
level = graythresh(im2);
% Now, we will get to know
% the threshold:
disp (['Threshold: ', num2str(level)]);
% disp(level);
% Now, we know what the threshold is (0,4588).
% So, we convert im2 to a black and white image
% using the threshold (variable called level,
% which is equal to 0,4588). The black and white image
% is called im3:
im3 = im2bw(im2, level);
% We look at im3:
imshow(im3);
% For 2 seconds:
pause(2);
% We save the black and white image (im3):
imwrite (im3, 'image3.bmp', 'bmp');
% We use an element for im3,
% we can cover the white holes:
im4 = imopen(im3, strel('disk',10));
% We look at im4:
imshow(im4);
% For 2 seconds:
pause(2);
% We save im4:
imwrite (im4, 'image4.bmp', 'bmp');
% We use the MATLAB imcomplement function
% in order to complement im4:
im5 = imcomplement(im4);
% We look at im5:
imshow(im5);
% For 2 seconds
pause(2);
% We save image im5:
imwrite (im5, 'image5.bmp', 'bmp');
% We want to identify round large objects
% (BLOBs) in our image.
% So, we use imtool to determine
% what a good radius would be:
% imtool (im5);
% We will separate only large round objects
```

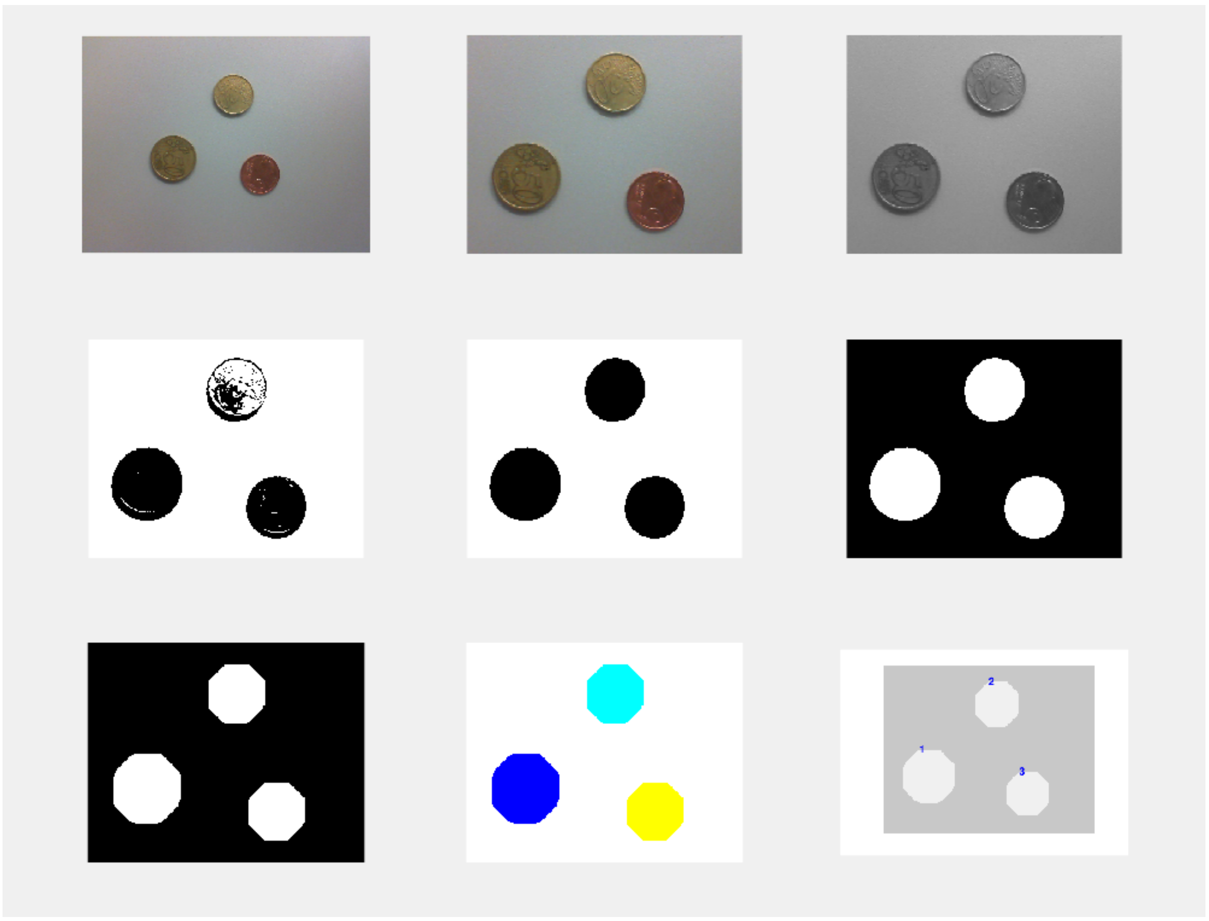
```matlab
% and we can apply an opening operation:
im6 = imopen(im5, strel('disk', 40));
% We look at im6:
imshow(im6);
% For 2 seconds:
pause(2);
imwrite (im6, 'image6.bmp', 'bmp');
% We want determine how many large round objects
% exist on the image. We use a connected
% components labeling algorithm. The MATLAB function
% bwlabel will identify the BLOBs. It will return an array
% with two elements. The first element is a labeled image.
% In this image, pixels that belong to the background are
% labeled with the value 0. Pixels belonging to the first
% BLOB are assigned a value of 1, pixels belonging to
% the second BLOB are assigned a value of 2, etc.
% The other element contains the number of BLOBs
% that are found in the image:
[labels, numlabels] = bwlabel(im6);
% We display the number of BLOBs:
disp(['Numlabels: ', num2str(numlabels)]);
disp([' ']);
% Each BLOB will be displayed using a different color.
% To do so, we draw the label matrix in colour (im7):
im7 = label2rgb(labels);
% We look at im7:
imshow(im7);
% For 2 seconds:
pause(2);
% We save im7:
imwrite (im7, 'image7.bmp', 'bmp');
% To visualize the BLOBs according to their label,
% we use the vislabels function. The vislabels function
% can be found in the vislabels.m file on Canvas:
vislabels(labels);
pause(2);
% As the image generated is a figure we save this image
% as a new image with extension .bmp:
img = getframe(gcf);
imwrite(img.cdata, ['newimage', '.bmp']);
% We  make use of the function importfile.
% We open this new from our folder with all the saved images.
% We assign the image to im8:
im8 = imread('newimage.bmp');
% We display the sentence:
% "Number of large objects found is: " <numlabels>:
disp (['Number of large objects found is: ', num2str(numlabels)]);
% We subplot im, im1 ... im8 using
% the MATLAB function subplot:
subplot(3, 3, 1); imshow(im);
subplot(3, 3, 2); imshow(im1);
subplot(3, 3, 3); imshow(im2);
subplot(3, 3, 4); imshow(im3);
subplot(3, 3, 5); imshow(im4);
subplot(3, 3, 6); imshow(im5);
subplot(3, 3, 7); imshow(im6);
subplot(3, 3, 8); imshow(im7);
subplot(3, 3, 9); imshow(im8);
set(figure(1), 'Position', [100, 100, 1000, 800]);
```



This is the histogram of the grayscale image (in the code: `imhist(im2)`).



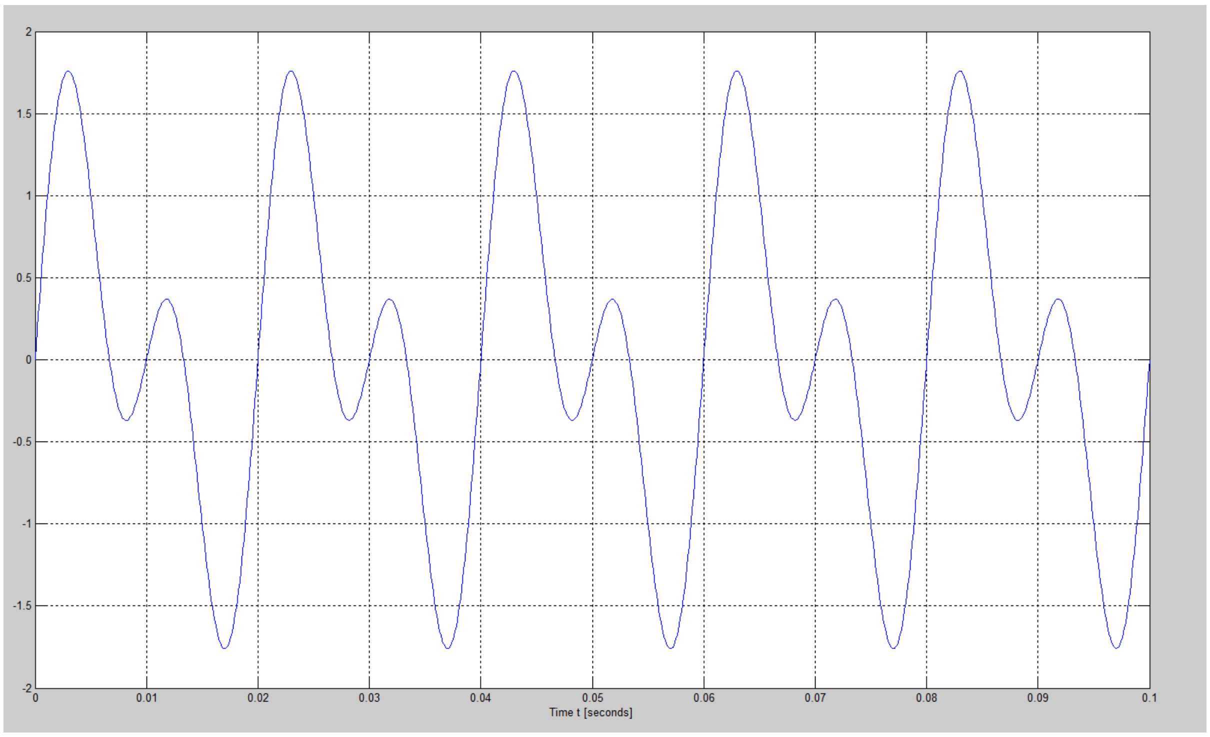
This is the figure we get when we use the `vislabels` function (in the code: `vislabels(labels)`).


Mosaic of all the images

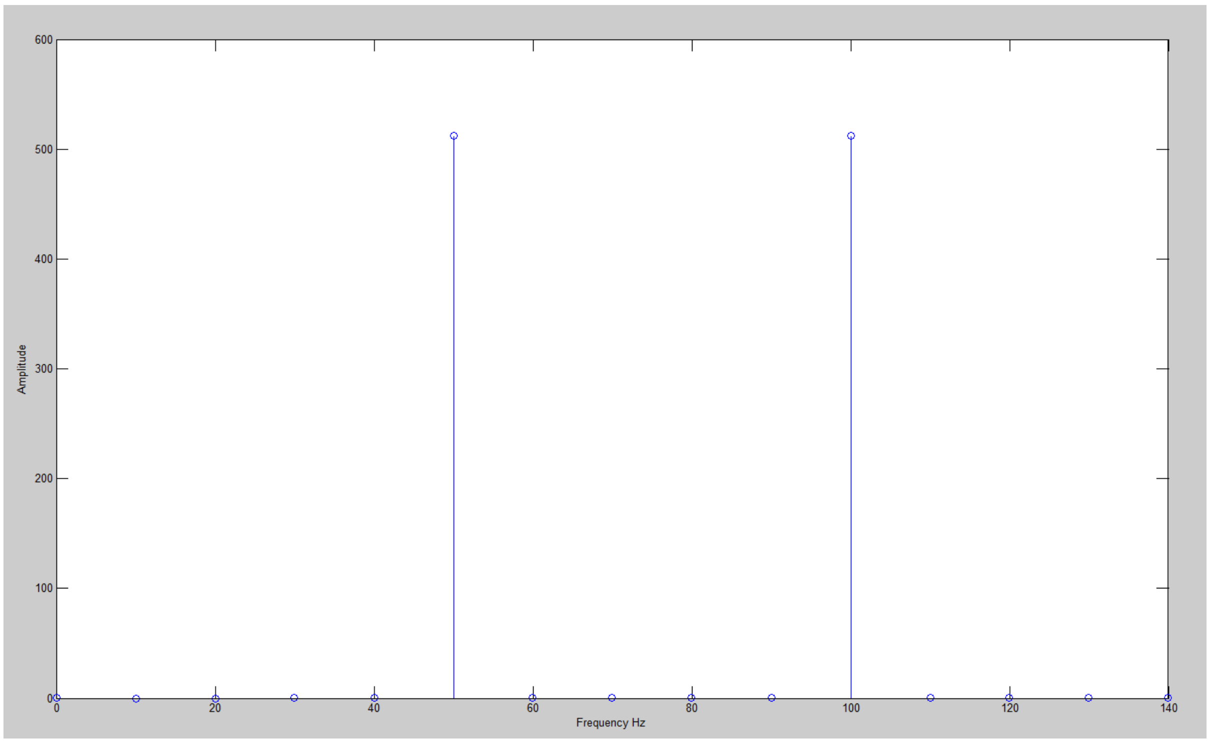
Exercise 2: *Write a MATLAB function that takes as input an one-dimensional waveform, and plots its frequency spectrum and its spectrogram.*
a) *A sum of 2 sinusoidal signals (of, for example, 50 Hz and 100 Hz).*
Answer: This is our MATLAB function:

```
% We want the sum of two sinusoidal signals.
% We want a signal with a frequency of 50 Hz:
f = 50;
% And another signal with a frequency of 100 Hz:
f2 = 100;
% The period of the signal is 0,1 seconds:
Td = 0.1;
% The total number of samples is equal to 1024:
N = 1024;
% The sampling period Ts [s] is Td divided by N:
Ts = Td/N;
% Time axis, where we start at t = 0 and end at t = Td = 0.1.
% We take steps of Ts = 0,1 / 1024 seconds.
t = 0:Ts:Td;
% We generate the signal: (The amplitude A = 1,
% that is why we do not use a variable for A.)
```

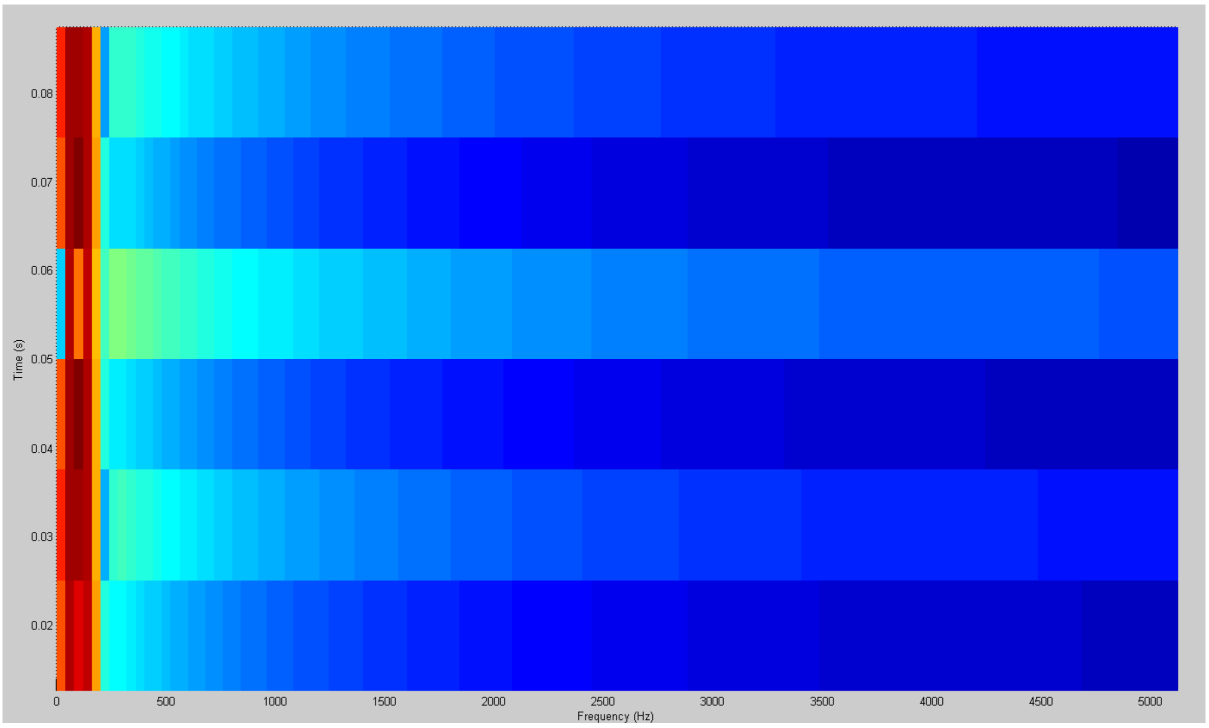
```matlab
s = sin(2*pi*f*t) + sin(2*pi*f2*t);
grid on;
xlabel ('Time t [seconds]');
% We plot the signal s as a function of t:
plot(t,s);
% For 3 seconds:
pause(3);
% We define the sampling frequency Fs:
Fs = 1/Ts;
% We apply FFT on vector s with N = 1024 samples.
% The returned vector is fft_vector with 1024 elements:
fft_vector = fft(s,N);
% fft_vector is a complex function, so we only take the real part:
m = abs(fft_vector);
% Only the first half is useful:
halfN = N/2;
% We make the frequency axis for
% the first half of the returned vector:
x = (0:halfN-1)*Fs/N;
m = m(1:halfN);
% A general frequency spectrum plot:
plot(x,m);
% For 1 second:
pause(1);
% We want to plot only the first 12 harmonics from 0 to 110 Hz:
x=x(1:12);
m=m(1:12);
% Plot frequency spectrum from 0 to 110 Hz:
stem(x,m);
% For 3 seconds:
pause(3);
xlabel('Frequency Hz');
ylabel('Amplitude');
fft_vector = fft(x,N);
% We are only interested in the amplitude:
m = abs(fft_vector);
% The frequency axis with the first 1000 elements:
x = (0:999)*Fs/N;
% We plot the frequency spectogram:
stem(x, m(1:1000));
% For 1 second
pause(1);
xlabel ('Frequency [Hz]');
% We plot the spectrogram:
spectrogram (s,256,128,256,Fs,'xaxis');
```



The sum of two sinusoidal signals



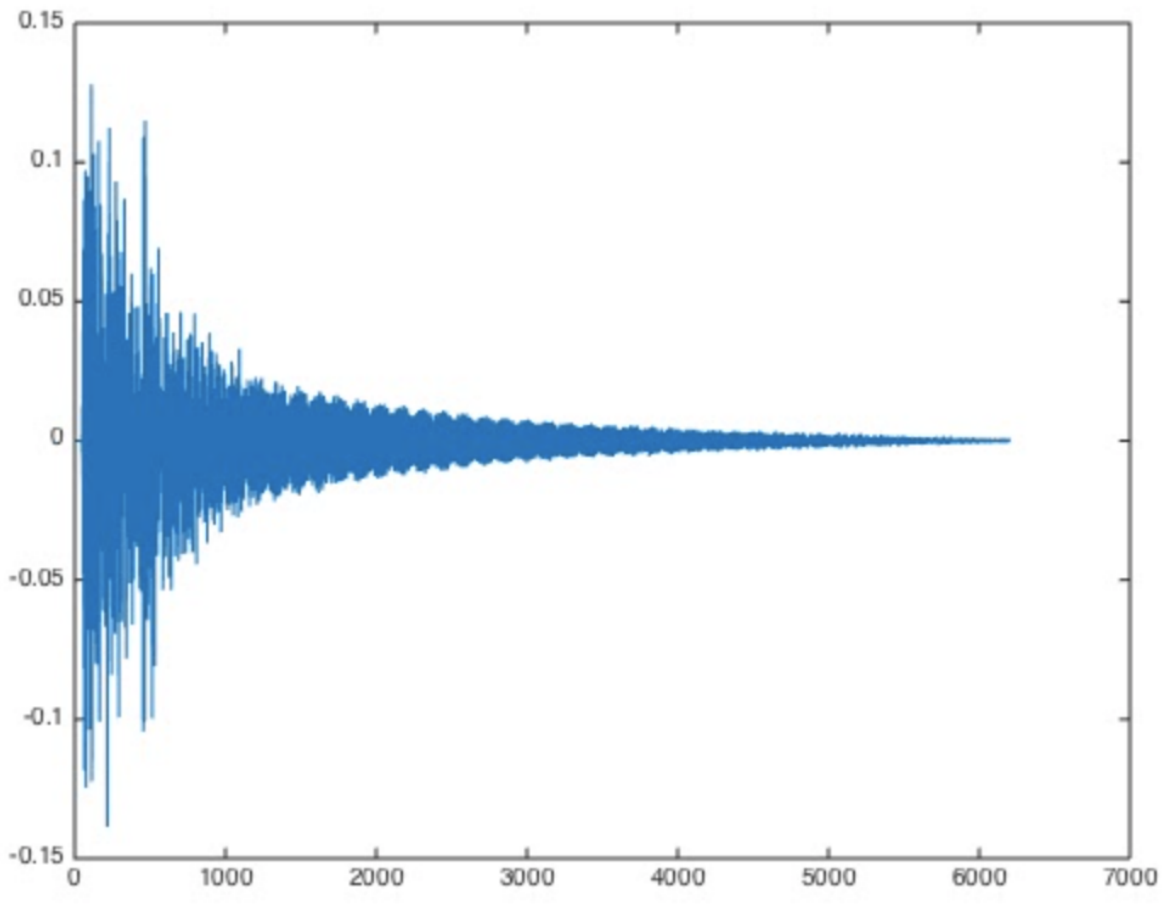
The frequency spectrum



The spectrogram

2b) The tuning fork signal recorded with a microphone, as you already did in Lab 1.
Answer: The commands we used to record the sound of the tuning fork are the following:

```
% We used a sampling frequency of Fs       whereby Fs is:
Fs=8000 ;
rec = audiorecorder(8000,16,1);
recordblocking (rec, 8);
% We store the data abstracted from our recording
% into tuningfork
tuningfork = getaudiodata (rec);
% We took a smaller segment of tuningfork and we
% stored that into a new vector tuningfork2
tuningfork2 = tuningfork (12000:19000);
% We took a smaller segment of tuningfork2 and we
% stored that into a new vector tuningfork4
tuningfork4 = tuningfork2 (900:6100)
```



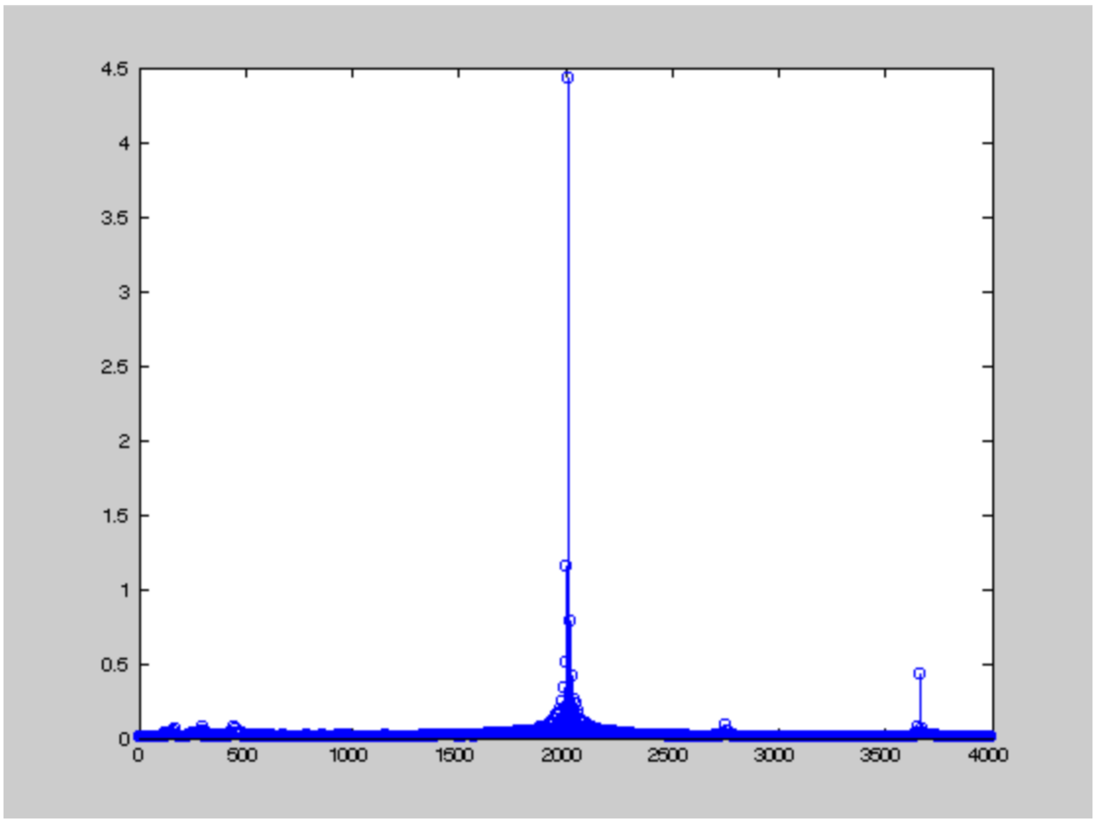
The sound of the tuning fork

This is our MATLAB function:

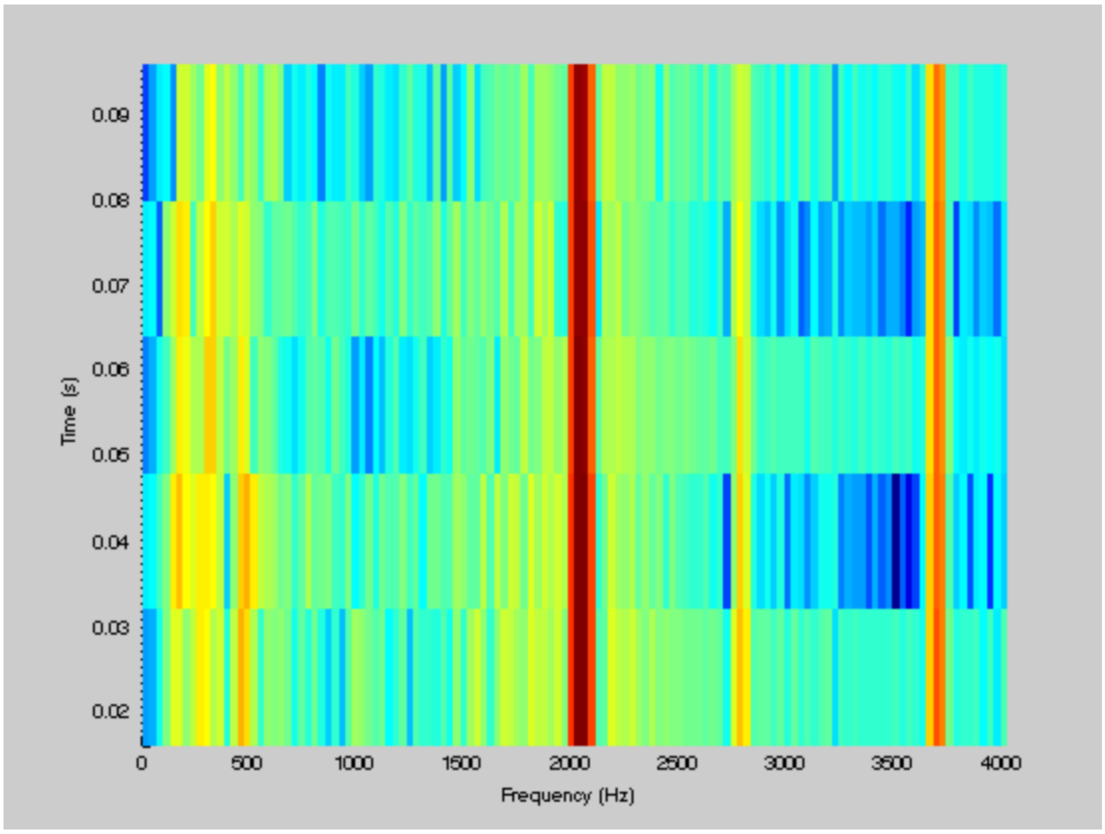
```
% We used a sampling frequency of Fs whereby FS is:
Fs = 8000;
% The total number of samples N is equal to 1000 :
N = 1000;
% We took a small segment of the signal from the
% tuning fork and put it into a vector y1:
y1 = tuningfork4 (2000:3000);
% We apply FFT on vector y1 with N = 1000 samples.
% The returned vector is fft_vector with 1000 elements:
fft_vector = fft(y1, N);
% We take a smaller segment of our vector fft_vector.
% We store that segment into a new vector y2.
y2 = fft_vector(1:500);
% We are only interested in the amplitude:
m = abs(y2);
% The frequency axis with the first 500 elements:
x = (0:499)*Fs/N;
% Plot frequency spectrum:
```

```
stem(x, m(1:500));
% For one second
pause(1);
xlabel ('Frequency [Hz]')
% We plot the spectrogram where by the frequency is on
% the 'xaxis' and the time is on the y axis.
spectrogram (y1, 256, 128, 256, Fs, 'xaxis')
```

The result:

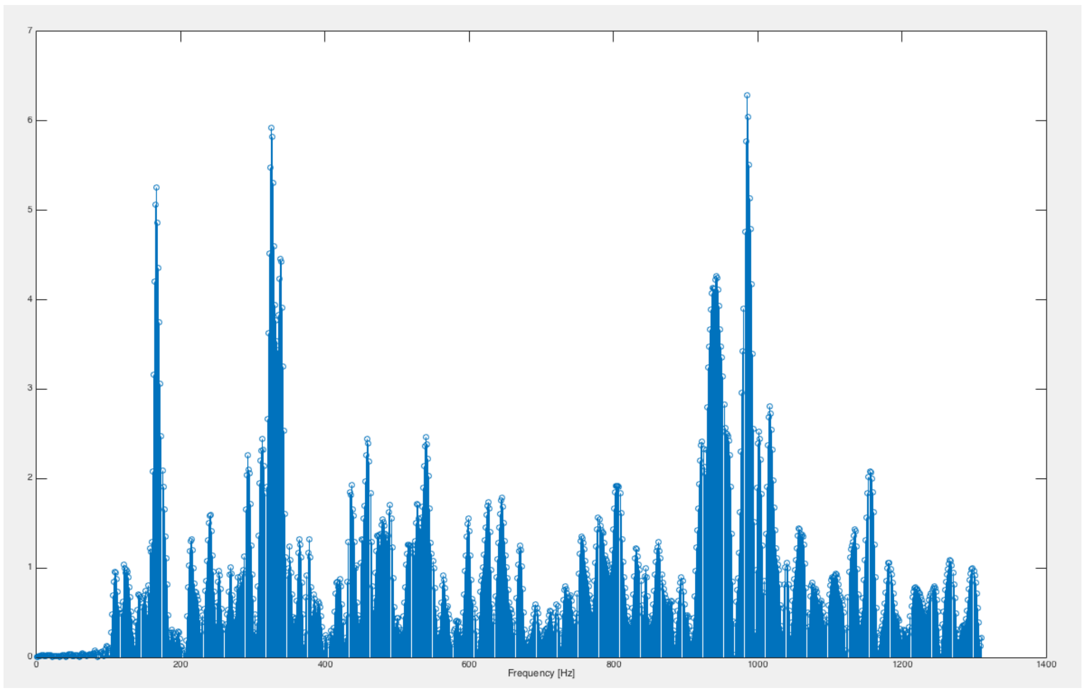


The frequency spectrum

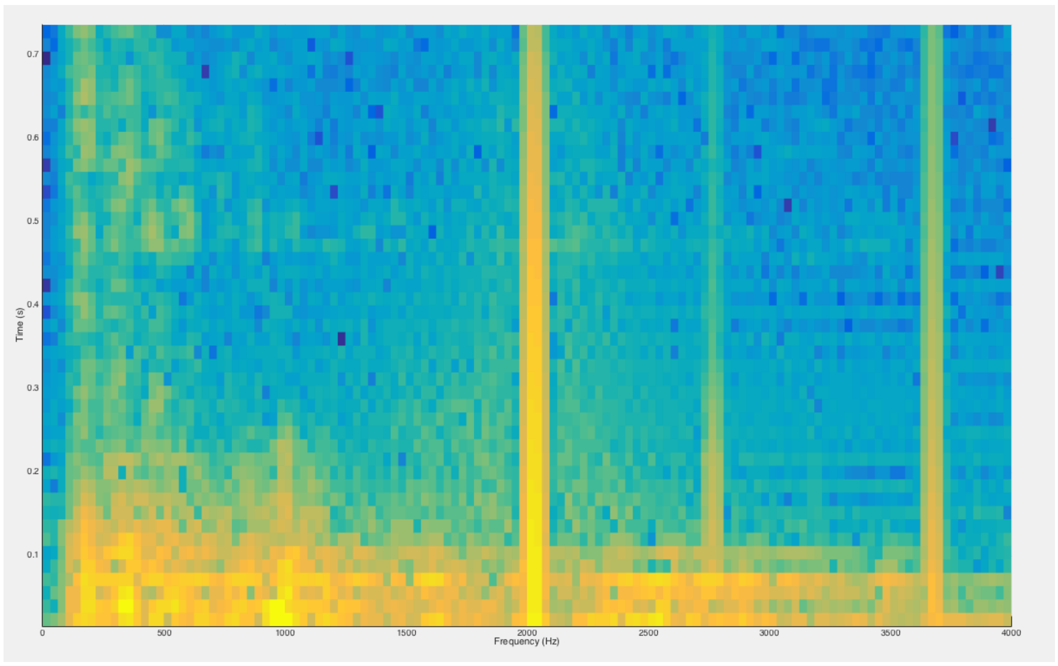


The spectrogram

This was the whole sound:



The frequency spectrum

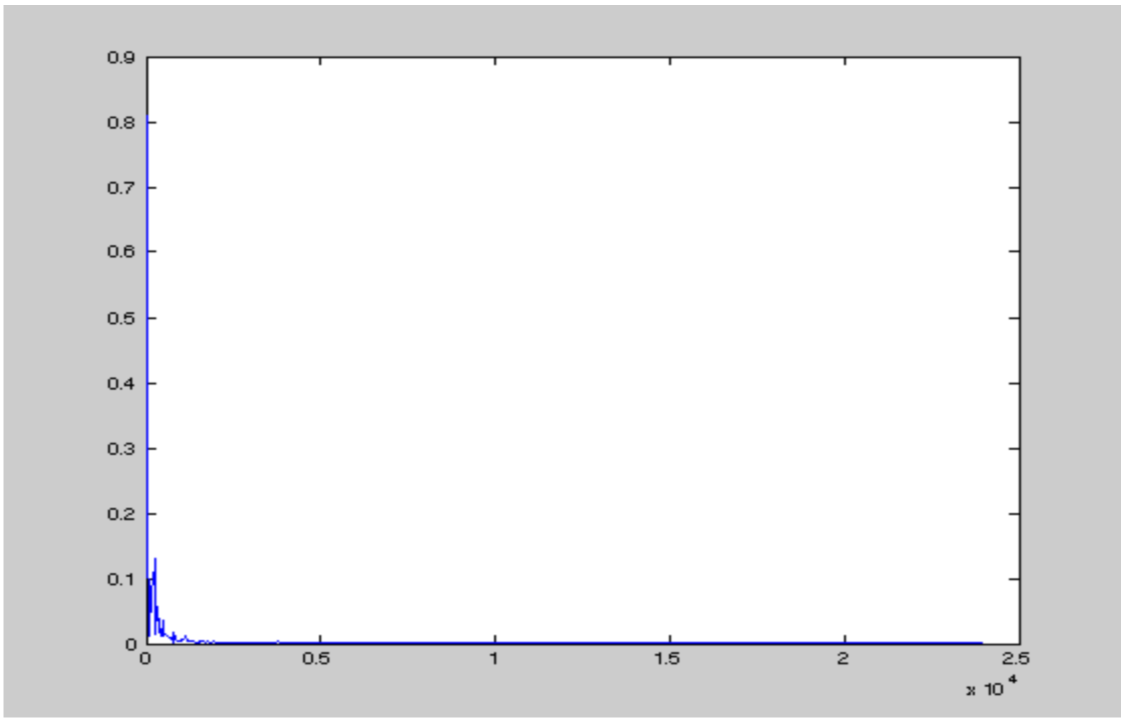


The spectrogram

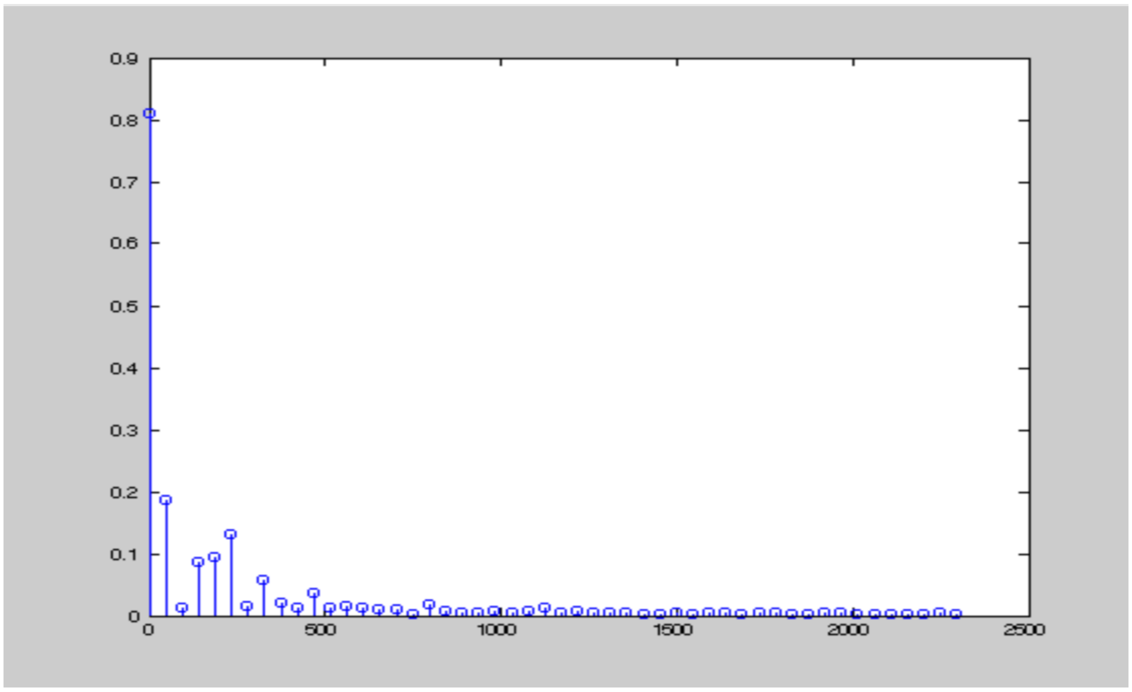
2c) *A piano waveform that can be found on Canvas or another instrument tone recorded by yourself.*

Answer: This is our MATLAB code:

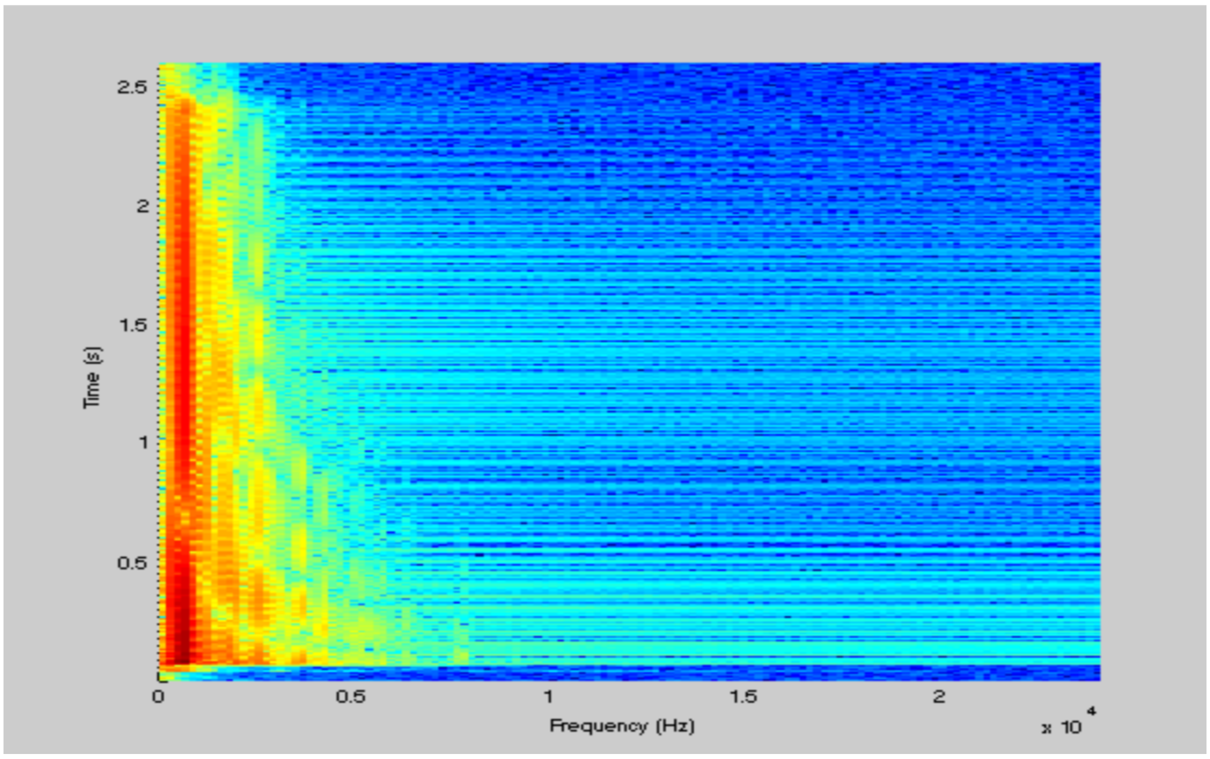
```
% We downloaded the sound we needed from Canvas.
% It was saved as ?pia60.wav?.
% Then, we read the soundwave file.
% s is the vector with samples.
% fs is the sampling frequency:
[s, fs] = audioread('pia60.wav');
% The total number of samples is equal to 1024:
N = 1024; %Total number of samples is 1024
% We apply FFT on vector s with N = 1024 samples.
% The returned vector is fft_vector with 1024 elements:
fft_vector = fft(s,N);
% fft_vector is a complex function, so we only take the real part:
m = abs(fft_vector);
% Only the first half is useful:
halfN = N/2;
% We make the frequency axis for
% the first half of the returned vector:
x = (0:halfN-1)*fs/N;
m = m(1:halfN);
% A general frequency spectrum plot:
plot(x,m);
% For 3 seconds:
pause(3);
% We want to plot only the harmonics from 0 to 2500 Hz:
x=x(1:50);
m=m(1:50);
% Plot frequency spectrum:
stem(x,m);
% For 3 seconds:
pause(3);
xlabel('Frequency Hz');
ylabel('Amplitude');
fft_vector=fft(x,N);
% We are only interested in the amplitude:
m = abs(fft_vector);
% The frequency axis with the first 1000 elements:
x = (0:999)*fs/N;
% We plot the frequency spectogram:
stem(x, m(1:1000));
% For 1 second:
pause(1);
xlabel ('Frequency [Hz]');
spectrogram (s,256,128,256,fs,'xaxis');
```



The signal



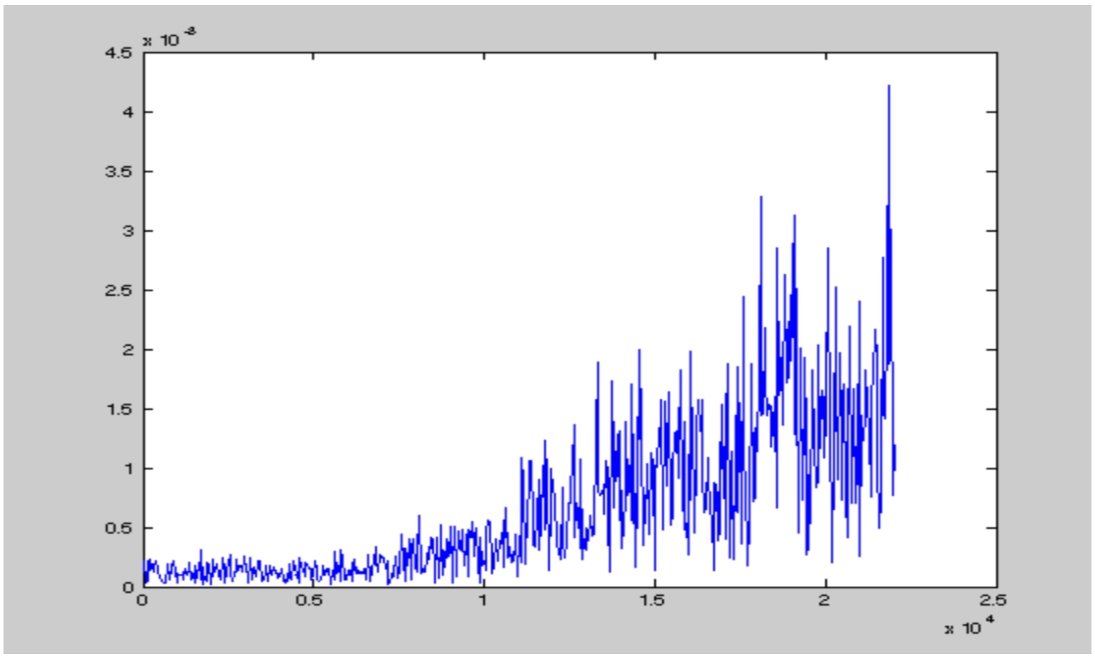
The frequency spectrogram



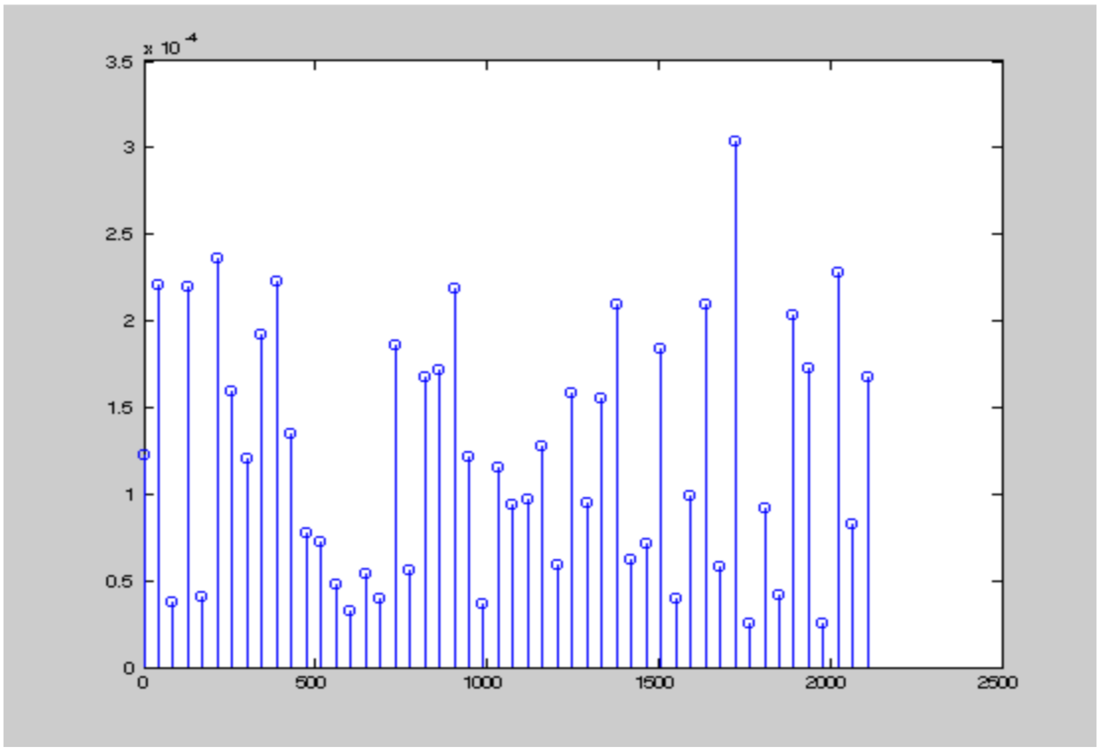
The spectrogram

2d) w*A middle C music scale that can be found in Canvas or one recorded by yourself.*
Answer: This is our MATLAB function:

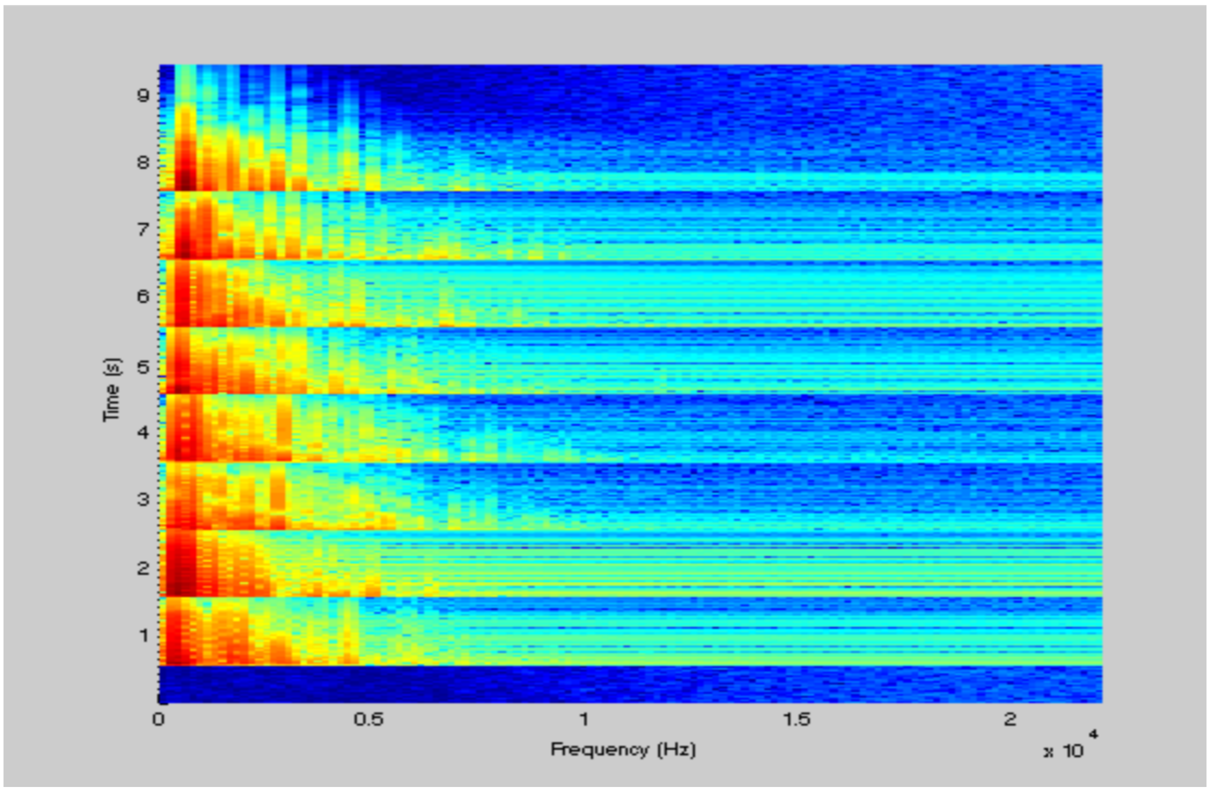
```
% We downloaded the sound we needed from Canvas.
% We saved it as ‘fishmusic.wav’.
% Then, we read the soundwave file.
% s is the vector with samples.
% fs is the sampling frequency:
[s, fs] = audioread('fishmusic.wav');
% We know what we have 416879 samples.
% We make another vector var with
% the elements of vector s:
var = s(1:416879);
% The total number of samples is equal to 1024:
N = 1024;
% We apply FFT on vector s with N = 1024 samples.
% The returned vector is fft_vector with 1024 elements:
fft_vector = fft(s,N);
% fft_vector is a complex function, so we only take the real part:
m = abs(fft_vector);
% Only the first half is useful:
halfN = N/2;
% We make the frequency axis for
% the first half of the returned vector:
x = (0:halfN-1)*fs/N;
m = m(1:halfN);
% A general frequency spectrum plot:
plot(x,m);
% For 3 seconds:
pause(10);
% We want to plot only the harmonics from 0 to 2500 Hz:
x=x(1:50);
m=m(1:50);
% Plot frequency spectrum:
stem(x,m);
% For 3 seconds:
pause(3);
xlabel('Frequency Hz');
ylabel('Amplitude');
fft_vector=fft(x,N);
% We are only interested in the amplitude:
m = abs(fft_vector);
% The frequency axis with the first 1000 elements:
x = (0:999)*fs/N;
% We plot the frequency spectogram:
stem(x, m(1:1000));
% For 1 second:
pause(1);
xlabel ('Frequency [Hz]');
spectrogram (var,256,128,256,fs,'xaxis');
```



The signal



The frequency spectrum



The spectrogram