ISS/VSG Project 2021/22

Aleksandr Verevkin (xverev00)
December 27, 2021

Introduction

The project was done by using programming language `Python` with the following libraries:

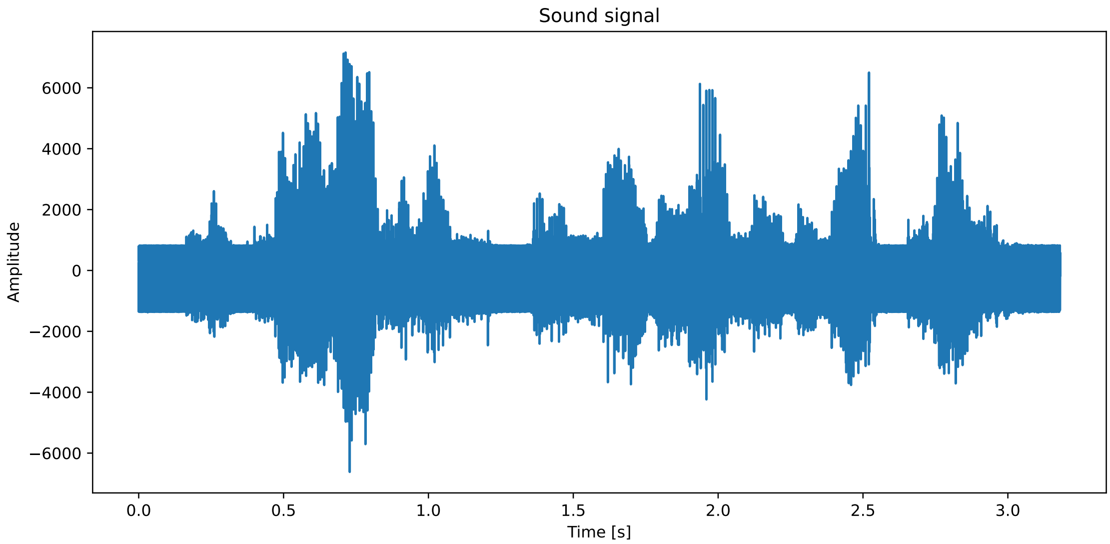
- numpy
- matplotlib.pyplot
- math
- scipy
- IPython

The archive with the project contains folder `src`, with python file, and folder `audio`, with saved audio files.

4.1

Signal was read by the command `wavfile.read`, from `scipy.io` library. Count of samples was determined by `.size` method, length in seconds by division of samples count on sampling frequency. Minimum and maximum values were determined by `.min` and `.max` methods.

File	Samples	Seconds	Min. value	Max. value
xverev00.wav	50893	3.1808125	-6619	7161



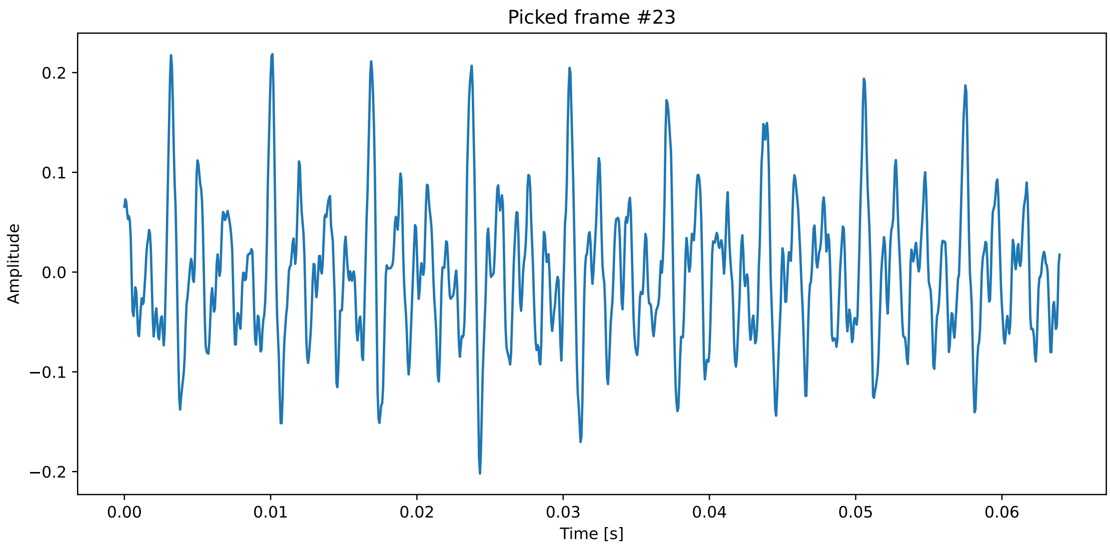
As we can see there is a monotonous, disturbing sound throughout the whole recording.

4.2

Signal was normalized to dynamic scale (-1, 1), by subtraction of his mean value(`np.mean()`) and division by maximum of absolute value. Then cutted into frames, each with 1024 samples and overlap 512 samples:

```
# Cut signal into frames
count_of_frames = math.floor(s.size / 512)
frame_arr = [[0 for x in range(1024)] for y in range(count_of_frames)]
frame_start = 0
frame_end = 1024
frame_counter = 0
while frame_counter != count_of_frames:
    if frame_counter + 1 == count_of_frames:
        frame_arr[frame_counter] = s[frame_start: s.size - 1]
    else:
        frame_arr[frame_counter] = s[frame_start: frame_end]
    frame_start += 512
    frame_end += 512
    frame_counter += 1
```

Then, manually was selected ”beautiful” frame №23/99, with vowel in it:

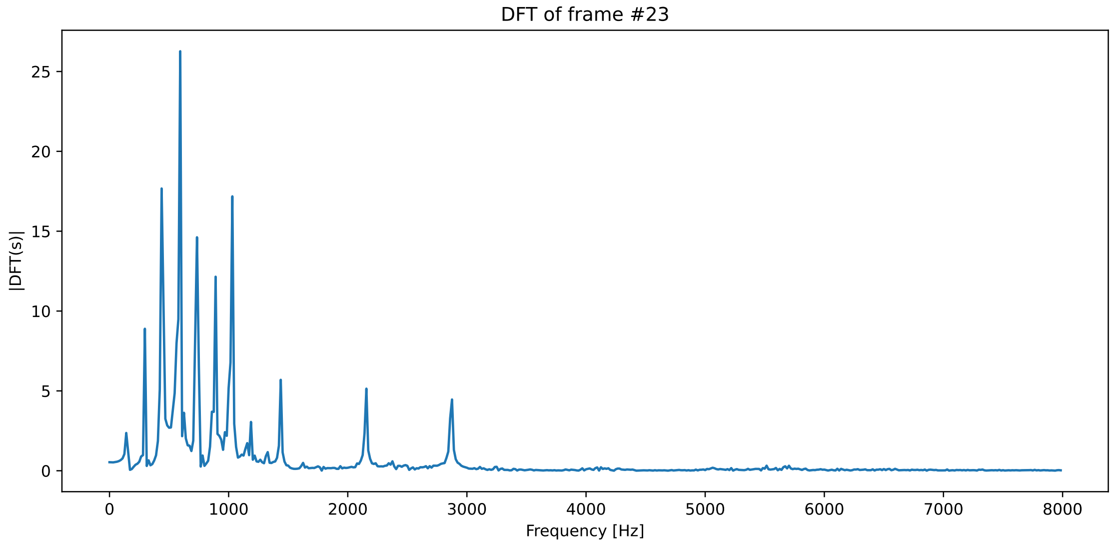


4.3

For DFT calculation was implemented own function:

```
def dft(frame):
    length = len(frame)
    if length != 1024:
        frame = np.append(frame, np.zeros(1024 - length))
    dft_s = []
    for i in range(length):
        appended = 0
        for j in range(length):
            appended += frame[j] * np.exp(-2j * np.pi * i * j / length)
        dft_s.append(appended)
    return dft_s
```

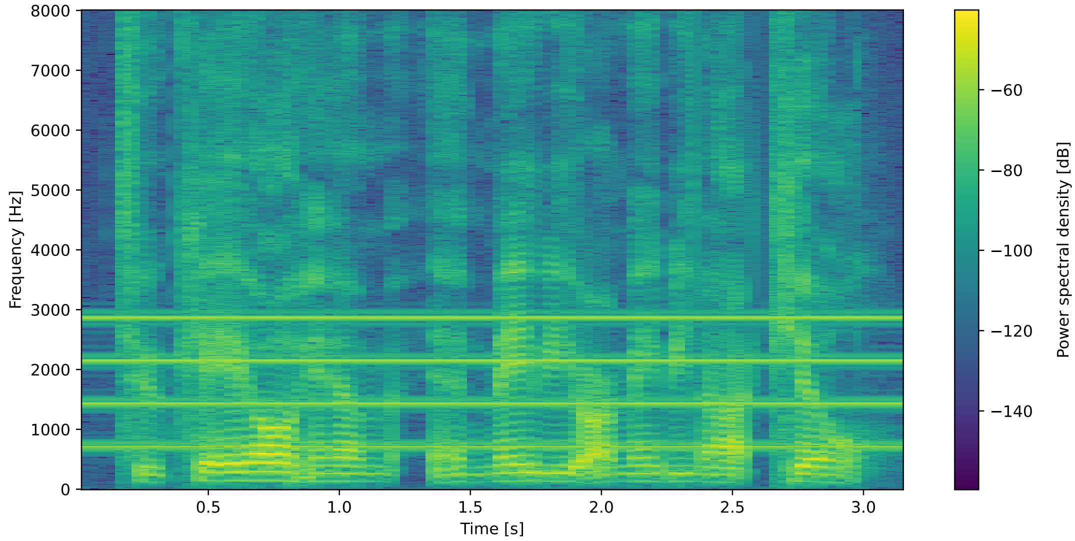
Selected frame was passed trough implemented function. Module(`np.abs()`) of the frame after:



Second half of the DFT module will be symmetric to the first. Result is equal to the `fft` function of the library `numpy`.

4.4

With help of function `spectrogram` from `scipy.signal` library, with optional arguments `nperseg=1024` and `noverlap=512` for the right length of frames, was plotted spectrogram for the whole signal:



As we can clearly see, spectrogram of the signal contains 4 disturbing components, all on the different frequencies.

4.5

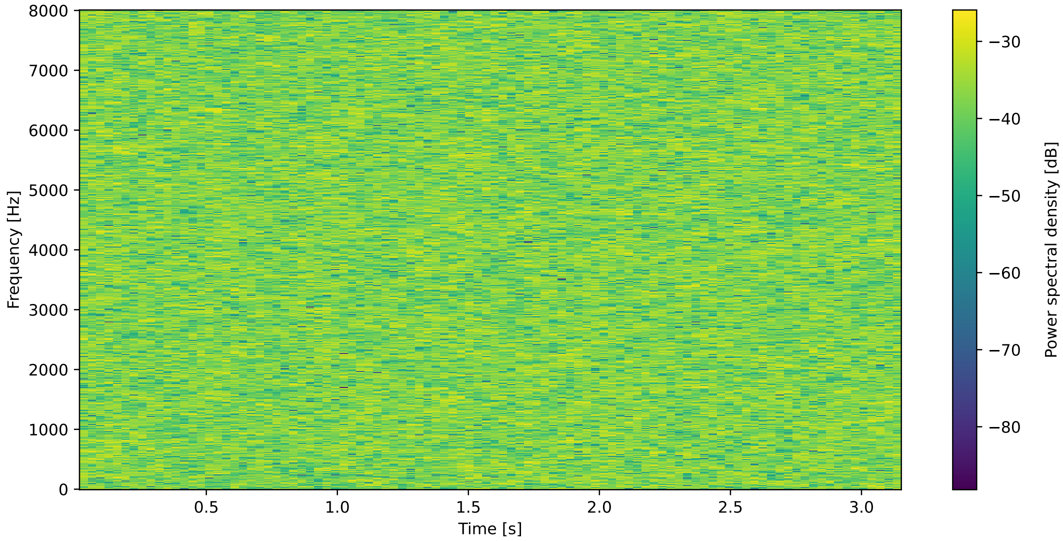
Manually were determined frequencies of disturbing components:

Frequency	**Value[Hz]**
f1	720
f2	1440
f3	2160
f4	2880

All components are harmonically related, all of them are multiplications of the lowest one.

4.6

For each frequency was generated cosine of the same length as original signal have. Cosines were generated with `cos` function of the `numpy` library. All cosines were mixed into one by simple addition of the signals. Spectrogram of the generated signal:



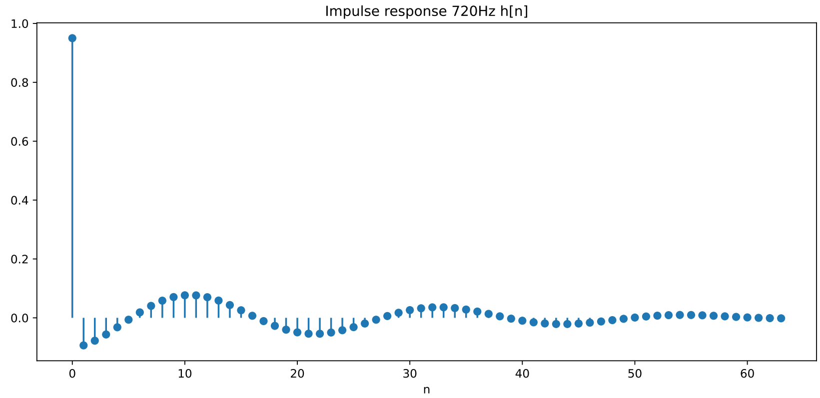
As we can see on the spectrogram, and by listening of generated audio (`/audio/4cos.wav`), signal only contain loud, disturbing noise. That only can mean that frequencies were selected correctly and signal was generated right.

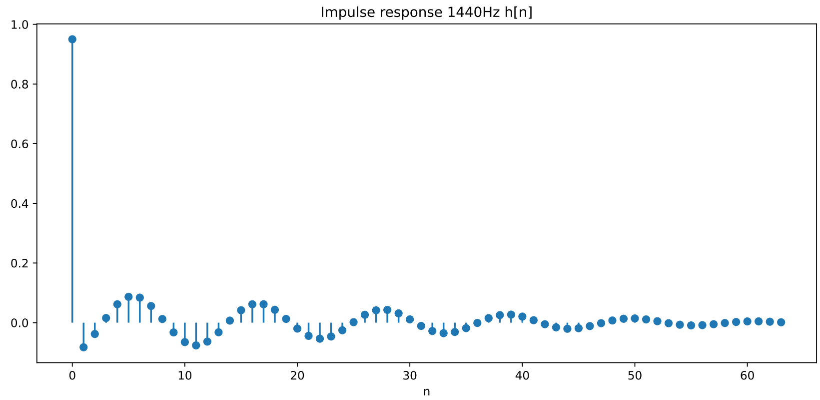
4.7

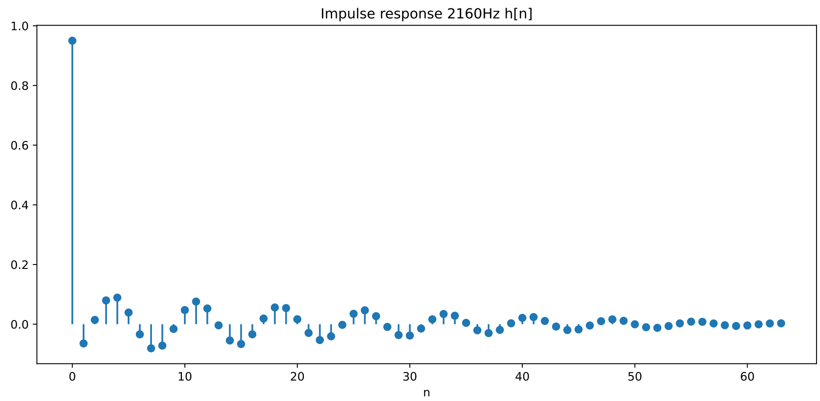
To clear the signal were selected **band-stop** filters with **stop-bands** near needed frequencies. Filters were designed with help of `buttord` and `butter` functions of the `scipy.signal` library. Functions were used for generation of the filter coefficients. Filters coefficients:

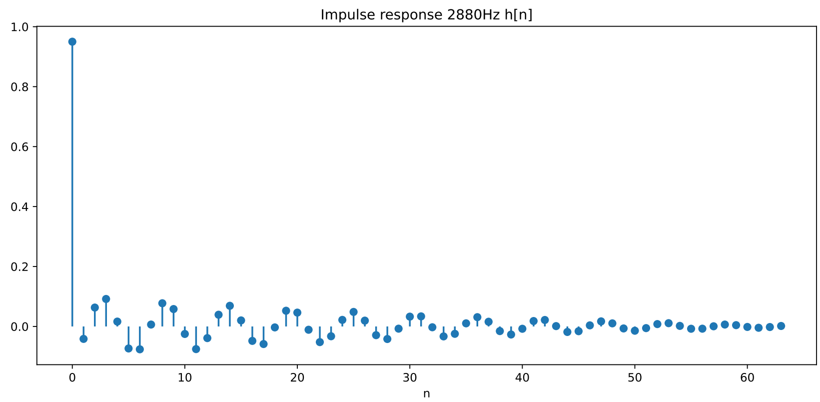
```
b1 coefficients: [  0.94998178  -7.29949911  24.83297015 -48.83422064  60.70157271
 -48.83422064  24.83297015  -7.29949911   0.94998178]
a1 coefficients: [  1.          -7.58527203  25.47470187 -49.45579144  60.68984114
 -48.20303915  24.20046818  -7.02333689   0.90246539]
b2 coefficients: [  0.94998178  -6.41800631  20.05976724 -37.56239748  45.9501987
 -37.56239748  20.05976724  -6.41800631   0.94998178]
a2 coefficients: [  1.          -6.66926909  20.5779826  -38.03995768  45.94056198
 -37.07638717  19.54868678  -6.17519365   0.90246539]
b3 coefficients: [  0.94998178  -5.02684277  13.77478218 -23.87754402  28.55894704
 -23.87754402  13.77478218  -5.02684277   0.94998178]
a3 coefficients: [  1.          -5.22364196  14.13036075 -24.18048043  28.55206864
 -23.56798914  13.42358021  -4.83666204   0.90246539]
b4 coefficients: [  0.94998178  -3.23648443   7.93480831 -12.05730155  14.469582
 -12.05730155   7.93480831  -3.23648443   0.94998178]
a4 coefficients: [  1.          -3.3631917    8.13926564 -12.20980362  14.46526661
 -11.90053825   7.73216454  -3.11403839   0.90246539]
```

Impulse responses of the designed filters:





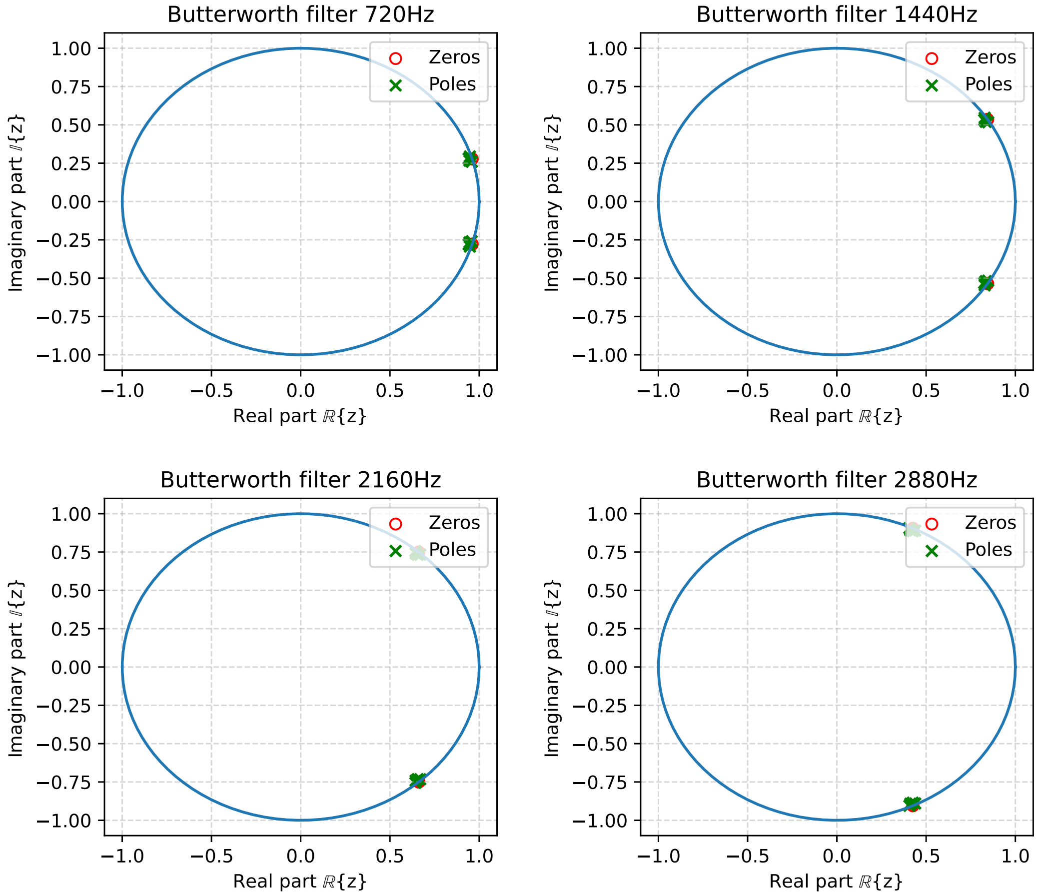


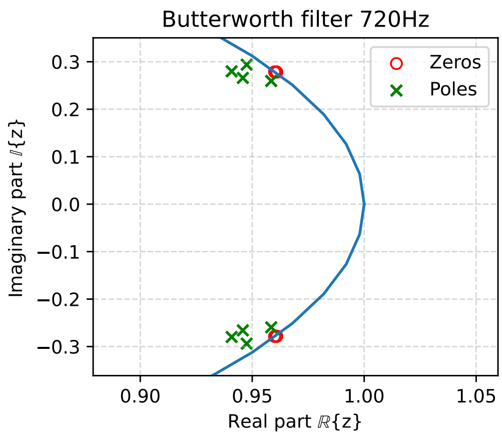


Signals were filtered by using `lfilter` from `scipy.signal`.

4.8

Zeroes and poles were calculated and plotted with usage of function `tf2zpk` of the `scipy.signal` library. Zeroes and poles of the filters:

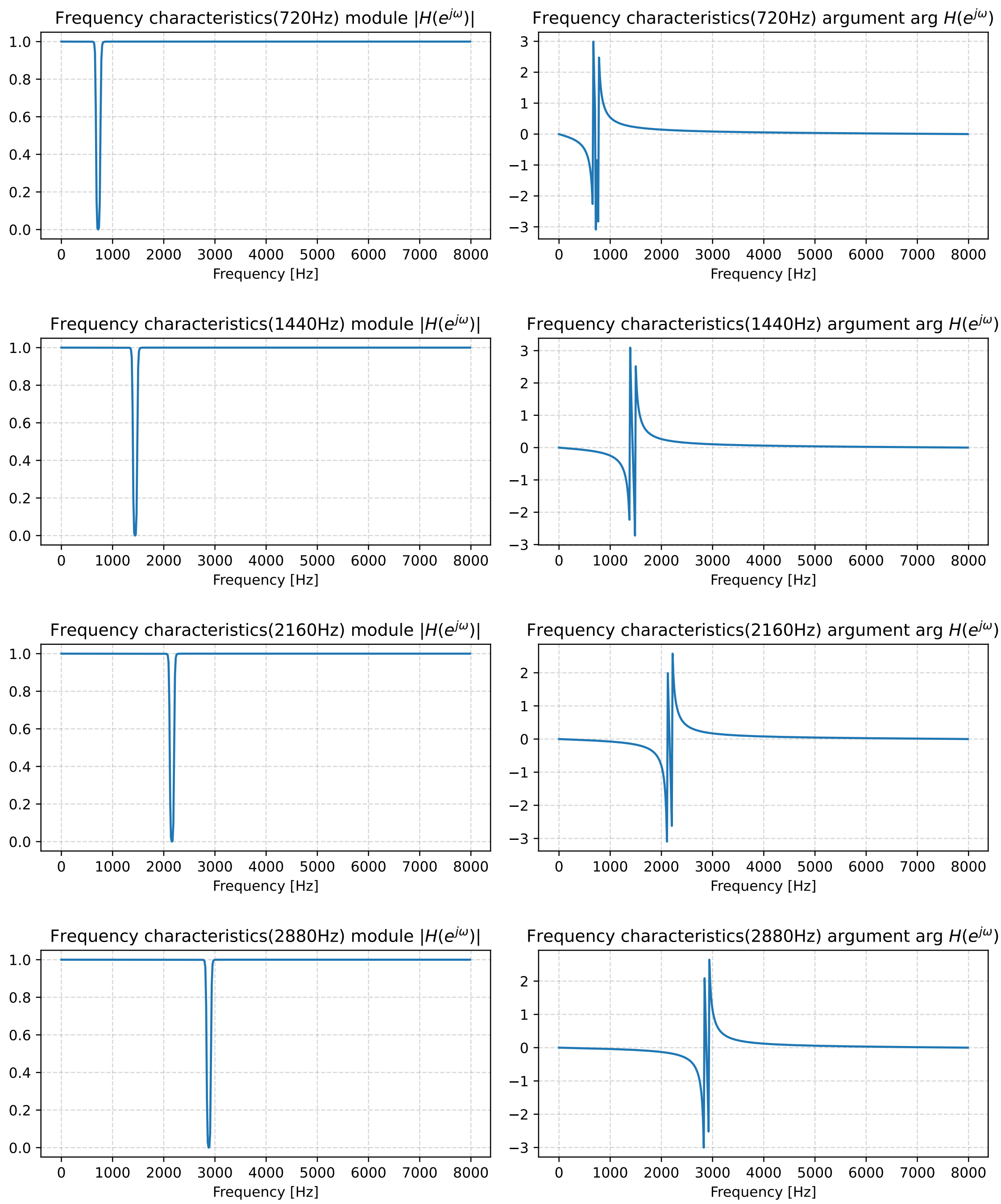




In the zoomed version of one of the figures we can see, that poles do not overlap zeros, but ”wrap” them.

4.9

Frequency characteristics were calculated and plotted as module and argument parts, with help of function `freqz` from `scipy.signal` library.



As we can see on frequency characteristics, filters pushing disturbing signals on the correct frequencies.

4.10

Finally, filter out our signal with generated filters, again with usage of function `lfilter`. Output audio was saved in the audio folder(`audio/clean_bandstop.wav`). By listening to output audio we can assure that signal was cleared out of the disturbing artifacts.