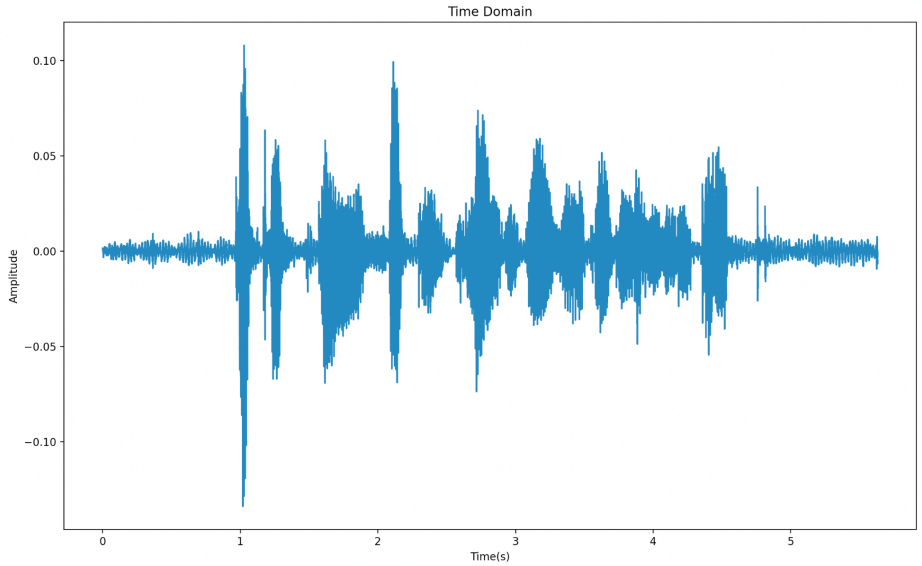
DSP Assignment 1 - Fourier Transform

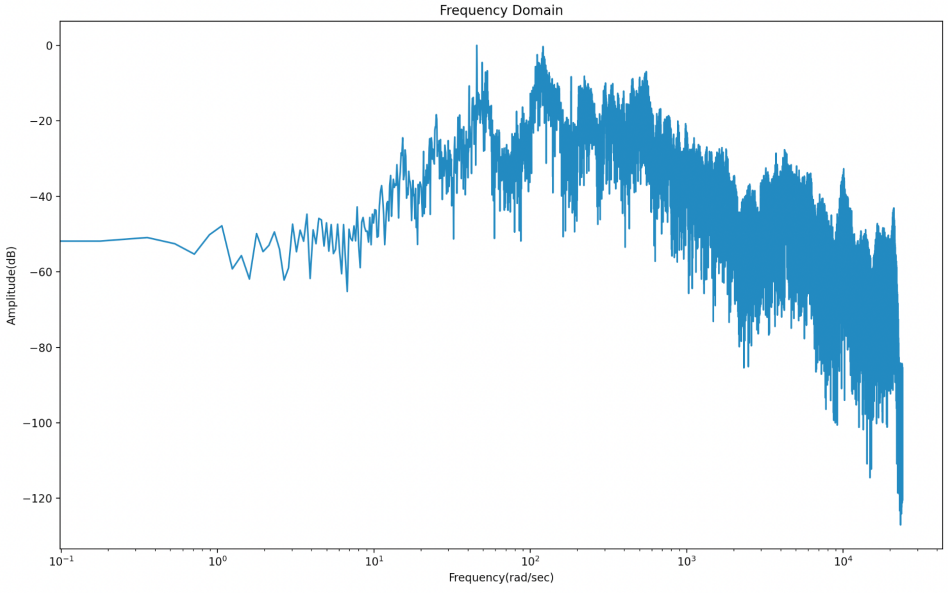
Question 1

Audio signal plot - File used original.wav at 48 KHz



Note: Achieved normalization by dividing all amplitude values by (2^16)/2

Logarithmic FFT plot

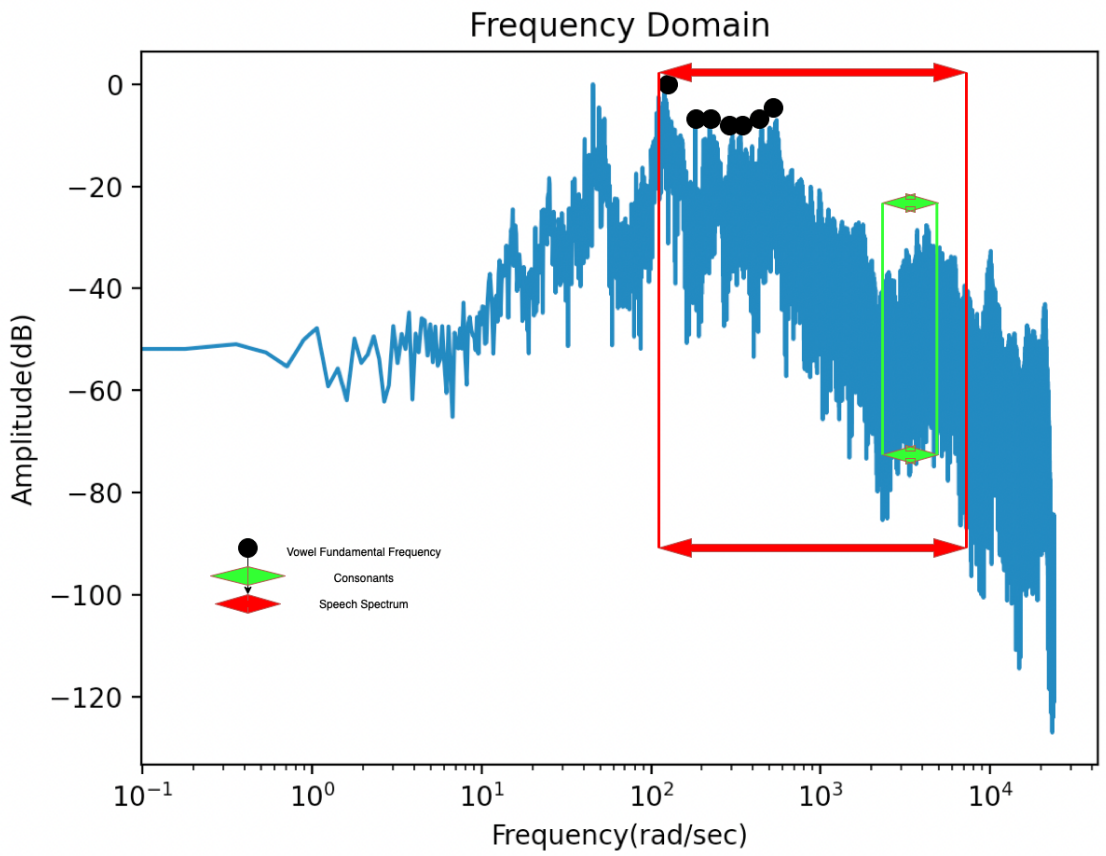


Note: dBFS calculated using $dB = 20\log_{10}(\frac{A}{A_0})$ [3]

Question 2

1. Spoken vowels range from 250 Hz to 2 KHz [1] but affect the whole voice. When compared with our own research, it's safe to assume that they cover a region from 250 Hz to 2 KHz.
2. Consonants are found between 2 KHz and 5KHz [4].
3. The speech spectrum would begin at 100 Hz, as human speech is recognized from 100 Hz to 8 Khz [1], and would end at 8 KHz after which there is only high frequency noise which doesn't register with humans.

Plot with Markups



Question 3

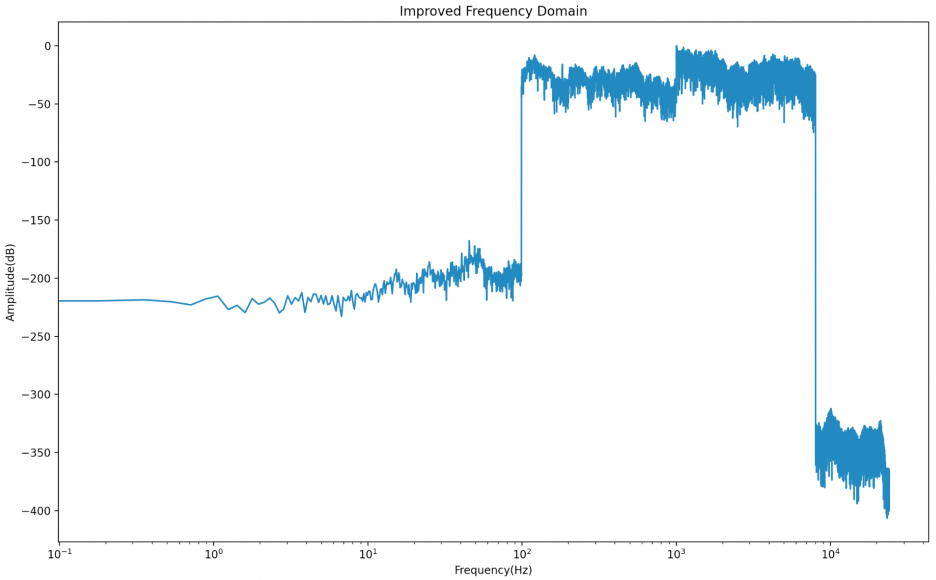
The region of highest harmonics is from 1 KHz to 8 KHz. The region was broken down into four chunks, 1 KHz to 2500 Hz to capture the vowels, 2501 Hz to 5000 Hz to capture the consonants, 5001 Hz to 7500 Hz and 7501 Hz to 8 KHz to capture the unvoiced consonants.

Amplification was carried out in these four regions by multiplying each chunk by a unique amplification factor to boost certain aspects of the audio file both before and after the nyquist frequency by using the mirroring method.

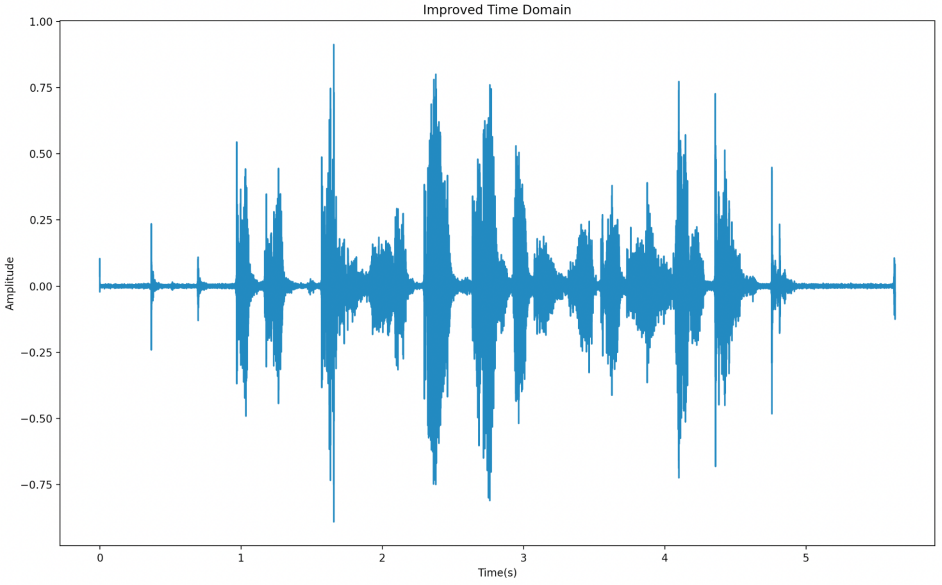
Noise reduction was carried out on the regions 0 Hz to 100 Hz and 10 KHz to 24 KHz, by dividing the data set by a large value, unique values were used for each frequency range, found before and after the Nyquist frequency using the mirroring method to nullify the effect of noise on the sound file.

An improved audio file was extracted from the above process by performing an inverse fourier transform on the modified FFT data set. In order to save the audio file, we had to take the real part of the inverse fourier transform since imaginary parts will not cancel out during the inverse FFT process.

The modified logarithmic FFT plot



Improved audio signal



Question 4

Section 4 will give an overview of the design process and the operating principles of the vowel detector algorithm specified in question 4. Design decisions and limitations of this approach will also be justified and discussed.

4.1. Recording library

Initially a library of recordings was created in order to gain insight and inform the development of the vowel detection algorithm. For this library, short words containing only one phonetic vowel were recorded. In order to test variability due to enunciation, accent etc, certain words were recorded several times, spoken by a male and female voice and or repeated several times. The following table shows all words in the library.

Overview of the content of the phonetic library used in the development of this algorithm.

Word	**Single recording One repetition**	**Single recording Multiple Repetitions**	**Male**	**Female**	**Phonetic vowel**
on	true	true	true	true	ɒ
door	true	true	true	true	ɔ:
floor	true	true	true	false	ɔ:
bed	true	false	true	false	e
far	true	false	true	false	a:
cat	true	false	true	false	æ
shoot	true	true	true	false	u:
sheep	true	true	true	true	i:
her	false	true	true	false	
learn	false	true	true	false	
work	false	true	true	false	

4.2. Plot normalisation

All plots in this section will use linear axes since it is easier to see the magnitude of difference between signals. Since the region of interest in the frequency domain is only that related to vowels, plots will focus only on the sub 2kHz range based on the findings from prior sections.

To allow for better comparison of signals in these plots (for example to compensate for varying length of signal, vowel prominence, recording volume etc.) signals have been rescaled such that the total spectral power in the frequency range of vowels (0-2kHz) is always equal to unity. The Y axis, as such, is fairly arbitrary, and is significant only insofar as it allows for the comparison of waves on the same plot. This is desirable for the purpose of developing the algorithm since all scoring must be done on the basis of relative thresholds and scores to negate the effect of variations in recording volume, vowel prominence and recording length.

From [6] Parseval's Theorem and the definition of average signal power give us:

$$P_C = \sum_{k=-\infty}^{\infty} |C(k)|^2 \tag{1}$$

$$C(k) = 1/N \sum_{n=-N/2}^{N/2} x(n)\, e^{-j(2\pi/N)kn} = 1/N\, f \tag{2}$$

Where C(k) is the kth normalised fourier coefficient, Tp is the time period of the data is the number of samples and Fs is the sampling frequency

Substituting (1) into (2) we get

$$P_C = 1/N^2 \sum_{k=-N/2}^{N/2} |fft(x)(k)|^2 = 2/N^2 \sum_{k=0}^{N/2} |fft(x)(\tag{3}$$

The spectral power in a particular range can now be defined by changing the summation bounds. Setting these between 0 and 2kHz we can derive the signal power associated with only the vowel. To normalise the plots against another a scaling A is applied element wise to the output of the fft(x) and the spectral power in the vowel range is set to unity:

$$P_C = 1 = 2/N^2 \sum_{k=0}^{N_v/2} |A\, fft(x)(k)|^2 = 2/(N^2 A^2) \sum_{k=0}^{N_v/2} |fft(x)(k) \tag{4}$$

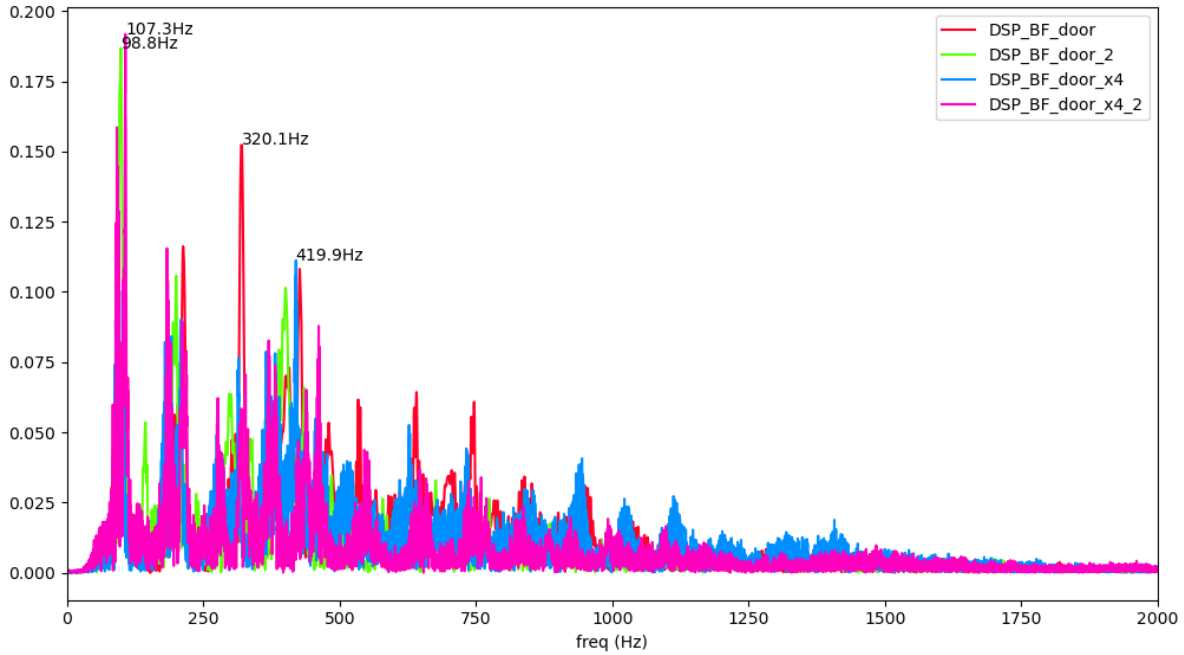
$$A = \sqrt{\frac{N^2}{2\sum_{k=0}^{Nv/2} |fft(x)(k)|^2}} \tag{5}$$

Where Nv is the index of highest frequency less than or equal to 2kHz Thus the spectral power in the range 0-2kHz for all recorded signals is normalised to 1 prior to plotting.

4.3. Phonetic vowel data

4.3.1 Differences between recordings

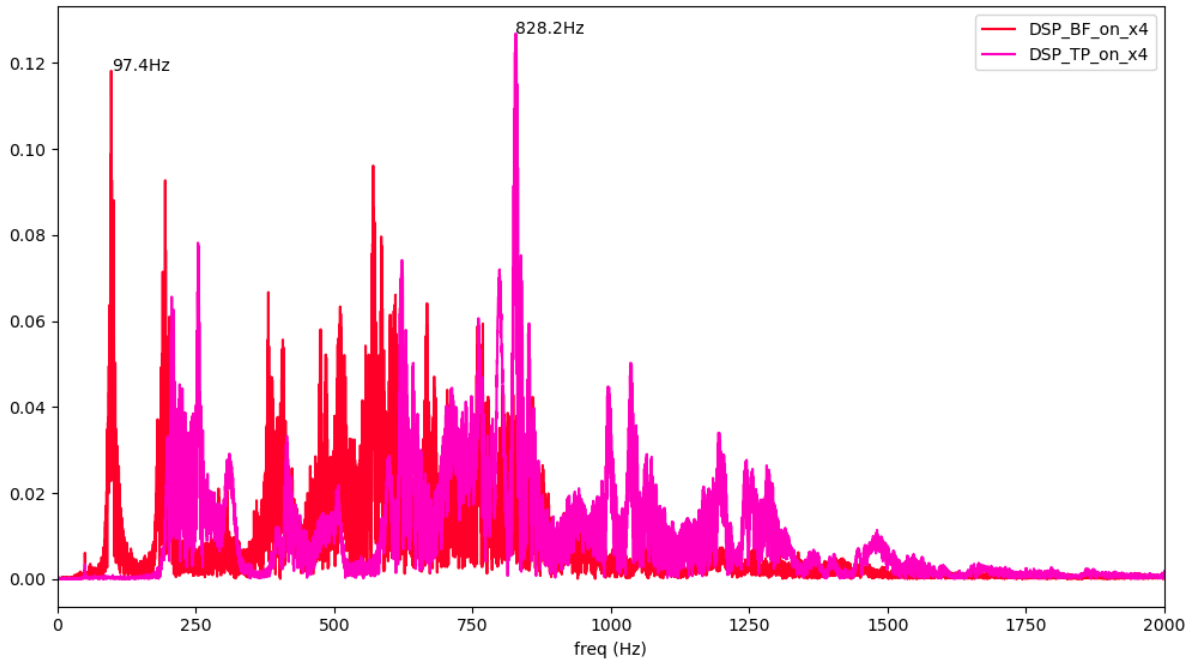
From the two plots shown below it is clear that between individual utterances of words containing the same phonetic vowel, there were substantial changes in the magnitude and position of individual harmonic peaks. This characteristic generally precludes the use of a peak finding algorithm. The only characteristic seemingly consistent between recordings is the general average distribution in particular regions. This characteristic will be exploited for vowel differentiation.



Four recordings of the word “door” by the same person, with one and four repetitions.

4.3.2 Variation in Individuals’ Enunciation

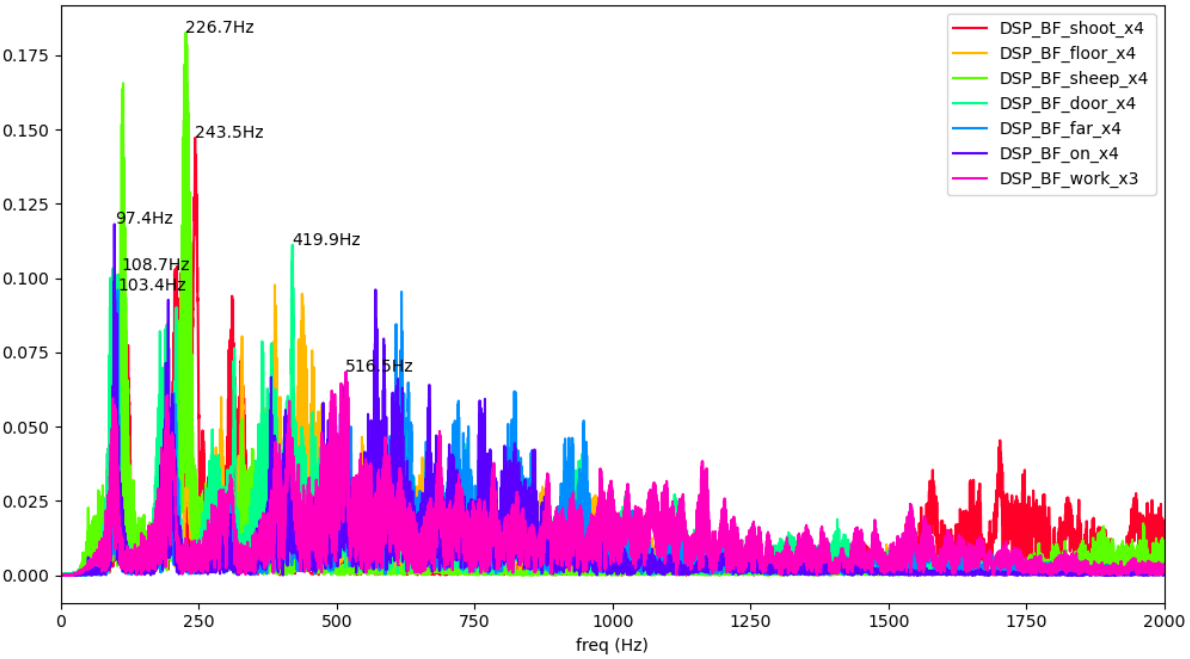
The characteristic distribution of spectral energy is also subject to which person is speaking as can be seen in the plot below. The average distribution of amplitudes tends to shift in frequency for one voice though it seems to display the same general pattern. Considering only one voice will simplify the design of the algorithm.



Two recordings of separate people saying the word “on” four times

4.3.3. Vowel Selection

To accurately distinguish spoken vowels in different recording conditions and by different people, selected vowels would ideally exhibit a dense group of high amplitudes in a region where none of the other selected frequencies exhibit this. The algorithm would then be able to consider only this particular frequency range for the purpose of differentiating the signals.



Seven recordings each representing a different phonetic vowel.

The final selection of vowels was narrowed down to the three shown in the table below. Since the exercise requires printing the name of each vowel to the terminal, the alphabetic

vowel corresponding to the selected phonetic vowel will henceforth be used when discussing the three selected phonetic vowels.

Mapping between alphabetic vowel printed by algorithm and selected phonetic vowel

Alphabetic vowel Printed	Phonetic vowel	Example word
e	iː	sheep
o	ɔː	door
a	aː	far

The exercise requires printing a single alphabetic vowel into the terminal; however, a single alphabetic vowel may have many phonetic variants associated with it. Since we aim only to differentiate 3 vowels the selection of phonetic vowels can be such that each selected phonetic vowel maps to a different alphabetic vowel:

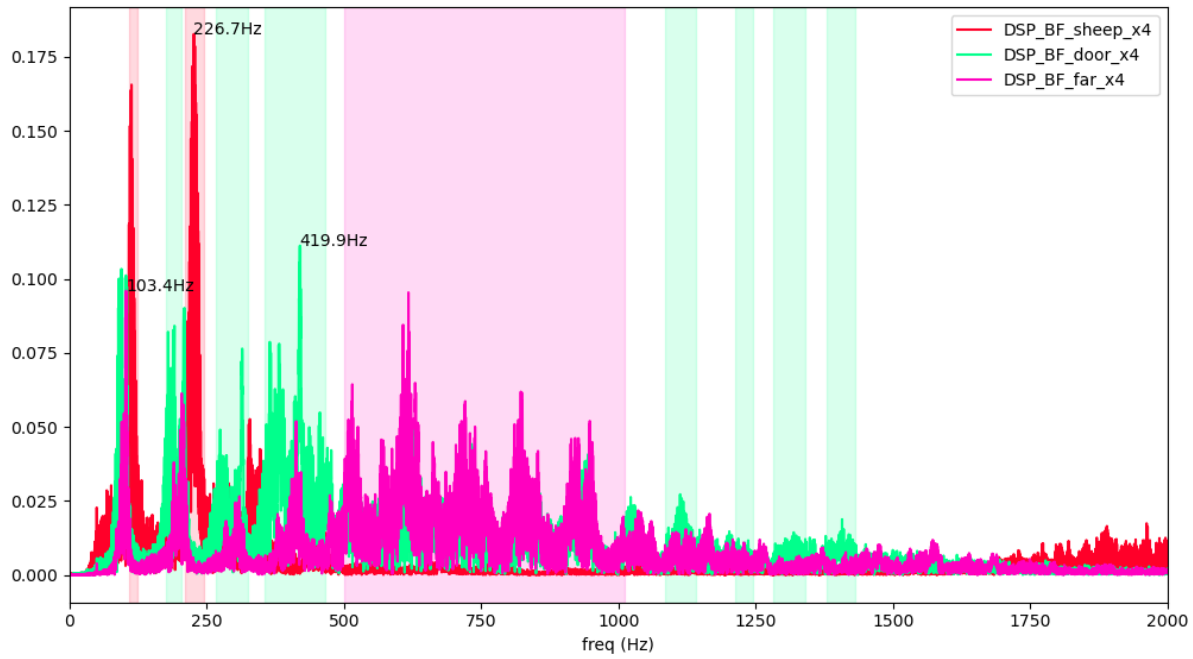
4.4. Algorithm working principles

To exploit the characteristic average distribution of harmonics of each selected vowel observed in the prior section, the algorithm calculated the spectral energy within a tuned range of frequencies which are selected based on the average dominance of a vowel in a particular range for a set of calibration data. The spectral power within the frequency range is then divided against the total within the vowel range to give a fractional score from 0-1.The frequency ranges are manually determined from the normalised plots by inspection.

Frequency bounds for selected vowels

O bounds (Hz)	E bounds (Hz)	A bounds (Hz)
176-203	108-125	500-1011
267-325	210-245	
335-465		
1085-1140		
1212-1244		
1281-1340		
1380-1430		

The figure below shows these ranges plotted with the normalised calibration signals for each vowel.



Selected vowel calibration data with the highlighted frequency bound for each vowel.

This is repeated 3 times, for every set of frequency bounds returning 3 scores. The set of bounds that returns the highest ratio can be thought of as the best matching result.

Since some signals are spread more evenly than others, the best ratio achievable will differ for every signal, this is shown by the maxim ratio achieved by each of the calibration signals with the correct vowel frequency range.

Ratio of spectral power in each vowel's frequency range to the total present in the range from 0-2kHz for a calibration recording containing that vowel

Vowel	Training Data Ratio
o	0.4789494436157115
a	0.7346054181089477
e	0.771509570998525

To compensate for this disparity the final score is derived by dividing each calculated ratio by the maximum score achieved by the calibration data. Thus the final score of any calibration data will simply be unity. The score with the highest value is then selected as the identified vowel.

This final correction step could likely be avoided with a more optimal selection of frequency ranges.

4.5. Accuracy/Testing

Testing of the algorithm was conducted using all recordings corresponding to the final selected phonetic vowels. It is worth noting that recordings postfixed with _CR and _PS were recorded with a different microphone, furthermore volume and recording time was not controlled.

The table below shows the power ratio returned by a recording for reach of the frequency ranges tested. And then the final corrected score for the actual vowel a recording represents. Calibration recordings are highlighted in purple, while the ratio corresponding to the true vowel present in the recording is highlighted in green. Where this ratio is not the largest, the incorrect ratio that is greater is highlighted in red.

Ratios final scores and success of algorithm when tested on all recordings containing one of the selected vowels.

Input File	Correct detection	Name	True vowel	e	o	a	Final Score in True vowel (%)
DSP_BF_far.wav	true	Ben	a	0.002	0.082	0.809	110.1
DSP_BF_door.wav	true	Ben	o	0.104	0.469	0.258	97.9
DSP_BF_door_2.wav	true	Ben	o	0.022	0.453	0.063	94.5
DSP_BF_floor_x4.wav	true	Ben	o	0.13	0.445	0.186	92.9
DSP_BF_sheep_x4.wav	true	Ben	e	0.772	0.057	0.004	100.0%
DSP_BF_door_x4	true	Ben	o	0.021	0.482	0.212	100.0%
DSP_BF_door_x4_2.wav	true	Ben	o	0.145	0.35	0.101	73%
DSP_BF_far_x4.wav	true	Ben	a	0.01	0.097	0.735	100%
DSP_TP_sheep_x4.wav	false	Tiia	e	0.179	0.314	0.022	23.2%
DSP_TP_door_x4.wav	true	Tiia	o	0.047	0.529	0.276	110.5%
DSP_CR_door.wav	true	Cameron	o	0.276	0.454	0.088	94.9%
DSP_CR_far.wav	true	Cameron	a	0.062	0.041	0.848	115.5%
DSP_CR_sheep.wav	true	Cameron	e	0.166	0.066	0.004	21.5%
DSP_PS_door.wav	true	Peter	o	0.12	0.181	0.228	37.8%
DSP_PS_far.wav	true	Peter	a	0.202	0.073	0.385	52.4%

Out of the 15 recordings tested, only one gave the wrong answer, though the magnitude of scores given decreased substantially with recordings from different people and/or recording equipment.

With recording "*DSP_PS_door.wav*" it is worth noting that although the ratio returned favours vowel 'a' when the ratio is corrected by the 'ideal score' derived from the calibration data, the final score still favours the correct vowel.

For all of Ben's recordings, the correct vowel was returned, with ratios associated with the correct vowel being substantially larger than those not, indicating a high degree of confidence in the prediction.

4.6. Limitations

Beyond the detection of only a few vowels this approach is non-ideal for several reasons. As more vowels need to be distinguished and/or if there is a requirement to be capable of detecting variations due to enunciation or accent, the range of frequencies at which all variants of the vowel you wish to detect are dominant become increasingly narrow. Furthermore many vowels are very similar in average amplitude for large regions making them particularly difficult to distinguish from another.

A further limitation stems from the manual iterative tuning of the detection regions, where accuracy is heavily biased towards data that one is calibrating against.

The scoring within the algorithm itself is flawed in that it considers only the ratio of spectral energy in the regions of interest relative to that in the region related to vowels. Thus the algorithm will always return a score regardless if there are vowels to be detected in a word or not. Effectively the algorithm contains an implicit assumption of the presence of the vowel in a recording. This could be altered by considering the energy in the ranges of interest against the total energy of the signal rather than just the vowel region and having a minimum threshold (such as 25% of total signal power) before a positive detection is issued.

Finally, this algorithm works on arbitrary length recordings meaning that the amplitude of frequencies associated with a particular vowel will vary with the relative length of the vowel in the entire sentence, if the recording is large there may be multiple vowels in the recording yet the algorithm will only give one output. The solution would of course be to use a fixed length window swept through the recording.

References

[1] Ecophon. N/A. “Generating and Understanding Speech.” www.ecophon.com. https://www.ecophon.com/en/about-ecophon/acoustic-knowledge/basic-acoustics/generating-and-understanding-speech/.

[2] Nave, Carl R. 2000. “Vowel Sounds.” hyperphysics.phy-astr.gsu.edu. http://hyperphysics.phy-astr.gsu.edu/hbase/Music/vowel.html.

[3] RS-MET. 2008. “Amplitude and Decibels.” rs-met.com. http://rs-met.com/documents/tutorials/AmplitudeAndDecibels.pdf.

[4] Russel, James. 2020. “The Human Voice and the Frequency Range.” blog.accusonus.com. https://blog.accusonus.com/pro-audio-production/human-voice-frequency-range/.

[5] UCL. 2003. “Acoustics of Vowels.” www.phon.ucl.ac.uk. https://www.phon.ucl.ac.uk/courses/spsci/acoustics/week2-3.pdf.

[6] John G. Proakis Dimitris G. Manolakis “Digital Signal processing Principles algorithms and applications ”

Appendix

voice_enhancer.py:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
import os

# Constants:

ns_1 = 10**8
ns_2 = 50**8
amp_1 = 35
amp_2 = 25
amp_3 = 35
amp_4 = 30
max_values = 2 ** 16

# Functions:
def mirrorAmplify(arr, start, stop, amp):
    N = len(arr)
    arr[start:stop+1] *= amp
    arr[(N-stop):(N-start)+1] *= amp

# Question 1

"""Import an audio file in .wav format at 48KHz"""
path = os.getcwd()
file_name = 'original.wav'

location = os.path.join(path, file_name)
samplerate, data = wavfile.read(location)

"""Create the FFT of the data(np array of numbers) obtained from the audio
file"""
data_fft = np.fft.fft(data)

"""Create an array of half range of values from the fourier transform so we
can plot up to the nyquist frequency"""
half_rangeValues = data_fft[0:int(len(data_fft) / 2 - 1)]

"""Define the frequency and time domains"""
f = np.linspace(0, samplerate / 2, len(half_rangeValues))
t = np.linspace(0, len(data) / samplerate, len(data))

"""Plot the Time domain spectrum"""
plt.subplot(2, 2, 1)
plt.plot(t, data / (max_values / 2))
plt.title('Time Domain')
plt.xlabel('Time(s)')
```

```
plt.ylabel('Amplitude')

"""Plot the Frequency domain spectrum"""
dB = 20 * np.log10(half_rangeValues / max(half_rangeValues))    # Using dB =
20log(A/A_0)
plt.subplot(2, 2, 2)
plt.plot(f, dB)
plt.xscale('log')
plt.title('Frequency Domain')
plt.xlabel('Frequency(rad/sec)')
plt.ylabel('Amplitude(dB)')

# Question 3

# Amplification and noise reduction

"""Values for lower frequency noise at 0 - 100Hz """
noise_1 = int(len(data_fft) / samplerate * 0)  # array location at 0Hz
noise_2 = int(len(data_fft) / samplerate * 99)  # array location at 99Hz

"""Values for the harmonic frequency at 1KHz to 8Khz"""

"""k1 and k2 capture the vowels"""
k1 = int(len(data_fft) / samplerate * 1000)  # array location at 1000Hz
k2 = int(len(data_fft) / samplerate * 2500)  # array location at 2500Hz

"""k3 and k4 capture the consonants"""
k3 = int(len(data_fft) / samplerate * 2501)  # array location at 2501Hz
k4 = int(len(data_fft) / samplerate * 5000)  # array location at 5000Hz

"""k5 and k6 capture the unvoiced consonants"""
k5 = int(len(data_fft) / samplerate * 5001)  # array location at 5001Hz
k6 = int(len(data_fft) / samplerate * 7500)  # array location at 7500Hz

"""k7 and k8 capture the remaining unvoiced consonants"""
k7 = int(len(data_fft) / samplerate * 7501)  # array location at 7501Hz
k8 = int(len(data_fft) / samplerate * 8000)  # array location at 8000Hz

"""Values for the higher frequency noise at 8KHz - 24KHz"""
noise_3 = int(len(data_fft) / samplerate * 8001)  # array location at 8001Hz
noise_4 = int(len(data_fft) / samplerate * 24000)  # array location at
24000Hz

"""Perform noise reduction by a factor so the resulting DB is almost negated.
Call the get_noiseReduction function on
both the area before N/2 and after to obtain a mirroring effect"""
mirrorAmplify(data_fft, noise_1, noise_2, 1 / ns_1) # Lower frequency noise
reduction
mirrorAmplify(data_fft, noise_3, noise_4, 1 / ns_2) # Higher frequency noise
reduction

"""Perform amplification by a chosen factor so the resulting audio file is
clearer than the original.
```

```
Call the get_amplification function on both the area before N/2 and after to
obtain a mirroring effect"""
mirrorAmplify(data_fft, k1, k2, amp_1)
mirrorAmplify(data_fft, k3, k4, amp_2)
mirrorAmplify(data_fft, k5, k6, amp_3)
mirrorAmplify(data_fft, k7, k8, amp_4)

"""Plot the improved fft"""
new_halfRange = data_fft[0:int(len(data_fft) / 2 - 1)]
new_dB = 20 * np.log10(new_halfRange / max(half_rangeValues))
plt.subplot(2, 2, 4)
plt.plot(f, new_dB)
plt.xscale('log')
plt.title('Improved Frequency Domain')
plt.xlabel('Frequency(rad/sec)')
plt.ylabel('Amplitude(dB)')

"""Plot the improved sound wave by performing inverse fourier transform"""
raw_data = np.fft.ifft(data_fft)
new_data = np.real(raw_data)
wav_file = new_data.astype(np.int16)
plt.subplot(2, 2, 3)
plt.plot(t, new_data / (max_values / 2))
plt.title('Improved Time Domain')
plt.xlabel('Time(s)')
plt.ylabel('Amplitude')


"""Create a new audio file that stores the improved version of the initial
audio file"""
name = str('Improved ' + file_name)
wavfile.write(name, samplerate, wav_file)

# Subplot adjustments
plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)

plt.show()
```

Vowel_detector.py

```
import os
import numpy as np
from scipy.io import wavfile
```

```
def voweldetector(targetwavefile):
    '''prints the detected vowel (a, o, e)'''

    doPrintDebug = False  # print debug messages
    # define some constants

    vowelMaxF = 2000  # Hz maximum expected vowel freq

    # the range of frequencies where 'o' is dominant
    doorFbounds = [
        [176, 203],
        [267, 325],
        [355, 465],
        [1085, 1140],
        [1212, 1244],
        [1281, 1340],
        [1380, 1430],
    ]

    # the score achieved by the test data for this range
    doorMaxScore = 0.4789494436157115

    # the range of freqs where 'e' is dominant
    sheepFbounds = [
        [108, 125],
        [210, 245],
    ]

    # the score achieved by the test data for this range
    sheepMaxScore = 0.771509570998525

    # the range of freqs where 'a' is dominant
    farFbounds = [
        [500, 1011],
        ]

    # the score achieved by the test data for this range
    farMaxScore = 0.7346054181089477

    # stick all of the above into list that we can loop through
    vowels2Check = ['e',  # sheep
                    'o',  # door
                    'a']  # far

    bounds = [sheepFbounds,
              doorFbounds,
              farFbounds]

    maxScores = [sheepMaxScore,
                 doorMaxScore,
                 farMaxScore]

    # main section
```

```
    baseFileName = os.path.basename(targetwavefile)
    if doPrintDebug:
        print("Reading: ", baseFileName, "from:\n", targetwavefile, "\n")

    # read in data
    fs, data = wavfile.read(targetwavefile)
    if len(data.shape) > 1:  # make sure we only have one ch
        data = data[:, 1]

    data = np.int64(data)  # recast this incase we run out of bits
    # tSignalPower = np.sum(abs(data) ** 2)

    # define some constants
    N = len(data)  # Number of samples
    df = fs/N  # frequency resolution
    # ts = 1/fs  # sample period
    # fn = fs/2  # nyquist frequency

    # define frequency/time vectors
    freqs = np.arange(0, N) * df
    idxVowels = freqs < vowelMaxF
    # t = np.arange(0, N) * ts

    # take fft
    dataFft = np.fft.fft(data)
    # fSignalPower = 1/N * np.sum(abs(dataFft) ** 2)

    # here we look at the sig Pow in only the vowel range 0->2000Hz
    vowelSignalPower = 2 * 1/N * np.sum(abs(dataFft[idxVowels]) ** 2)
    score = np.array([0, 0, 0], float)

    # loop through all of different vowels we are testing for
    for i in range(3):
        thisVowelFreqBounds = bounds[i]
        thisVowelIdStr = vowels2Check[i]

        # here we create a logical array which represents the idxs we have
selected
        idxFreqsOfInterest = np.full(N, False, bool)  # create a blank logical
arr 2 pop
        for ii in range(len(thisVowelFreqBounds)):
            theseBounds = thisVowelFreqBounds[ii]
            lowBound = min(theseBounds)
            uppBound = max(theseBounds)
            theseIdxs = (freqs > lowBound) & (freqs < uppBound)
            idxFreqsOfInterest = idxFreqsOfInterest | theseIdxs

        # calculate the signal power in the region we are interested
        maskedSignalPower = 2 * 1/N * np.sum(abs(dataFft[idxFreqsOfInterest])
** 2)
        pRatio = (maskedSignalPower/vowelSignalPower)  # the ifraction of sig
P in the freq range
```

```
        score[i] = pRatio / maxScores[i]  # the ratio of pRatio to a perfect
score in %
        # this will be 100% for training data

        # print out the scores for each of the tree vowels
        if doPrintDebug:
            print(round(score[i] * 100, 1), '%', thisVowelIdStr, "in ",
                  baseFileName, " ... ", " maskedPow/vowPow -> ",
                  round(pRatio, 3))

    # now we decide which vowel it was based on the score
    idxMax = score.argmax()
    print(vowels2Check[idxMax])


currentFilePath = os.getcwd()
dir_1 = 'Recordings/WordsWithSinglePhonetic/DSP_BF_door.wav'
dir_2 = 'Recordings/WordsWithSinglePhonetic/DSP_BF_far.wav'
path_1 = os.path.join(currentFilePath, dir_1)
path_2 = os.path.join(currentFilePath, dir_2)

voweldetector(path_1)
voweldetector(path_2)
```