International Institute of Information Technology
Bangalore

Digital Signal Processing
Course Project Report

Male Female Voice Conversion

Barath S Narayan, Pandey Shourya Prasad , Nitheezkant R

April 22, 2023

Analysis of Male and Female Voice

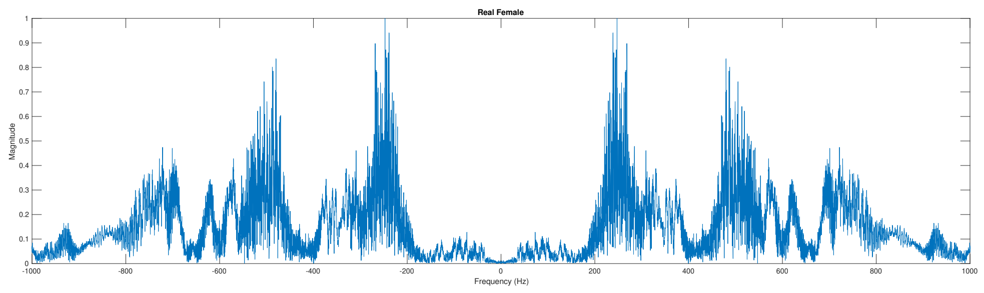
.

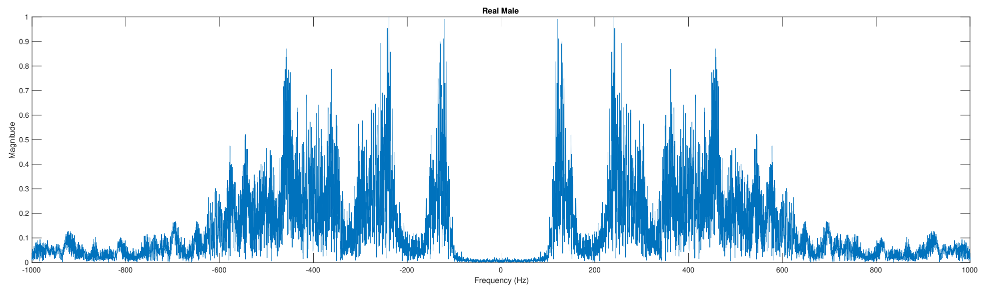
Data Set

- For analysing the voice of males and females, we both recorded and used several samples available on the Internet.
- For this report, we have used 2 samples male.wav and female.wav, that we recorded. The sound says "This is an audio sample used for digital signal processing project".
- These audios have been sampled at a rate of 44100Hz.
- The sample is a stereo, but only one channel is used for all the analysis and conversion.

Fourier Transform

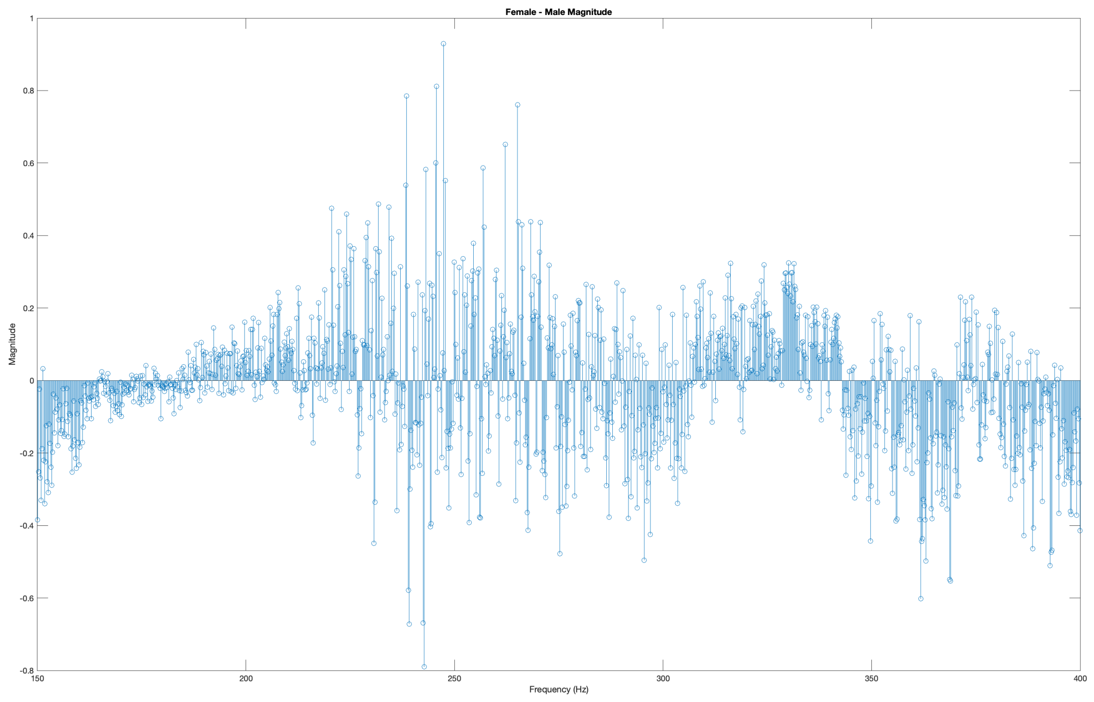
- One of the distinctive property that distinguishes male voice and female voice is its frequency contents.
- The magnitude of each frequency component gives good information on the gender of the voice.
- The Fourier transform of these samples was taken using the fast Fourier transform method.
- The magnitude of the frequency spectrum has been normalized by dividing it by the maximum amplitude.





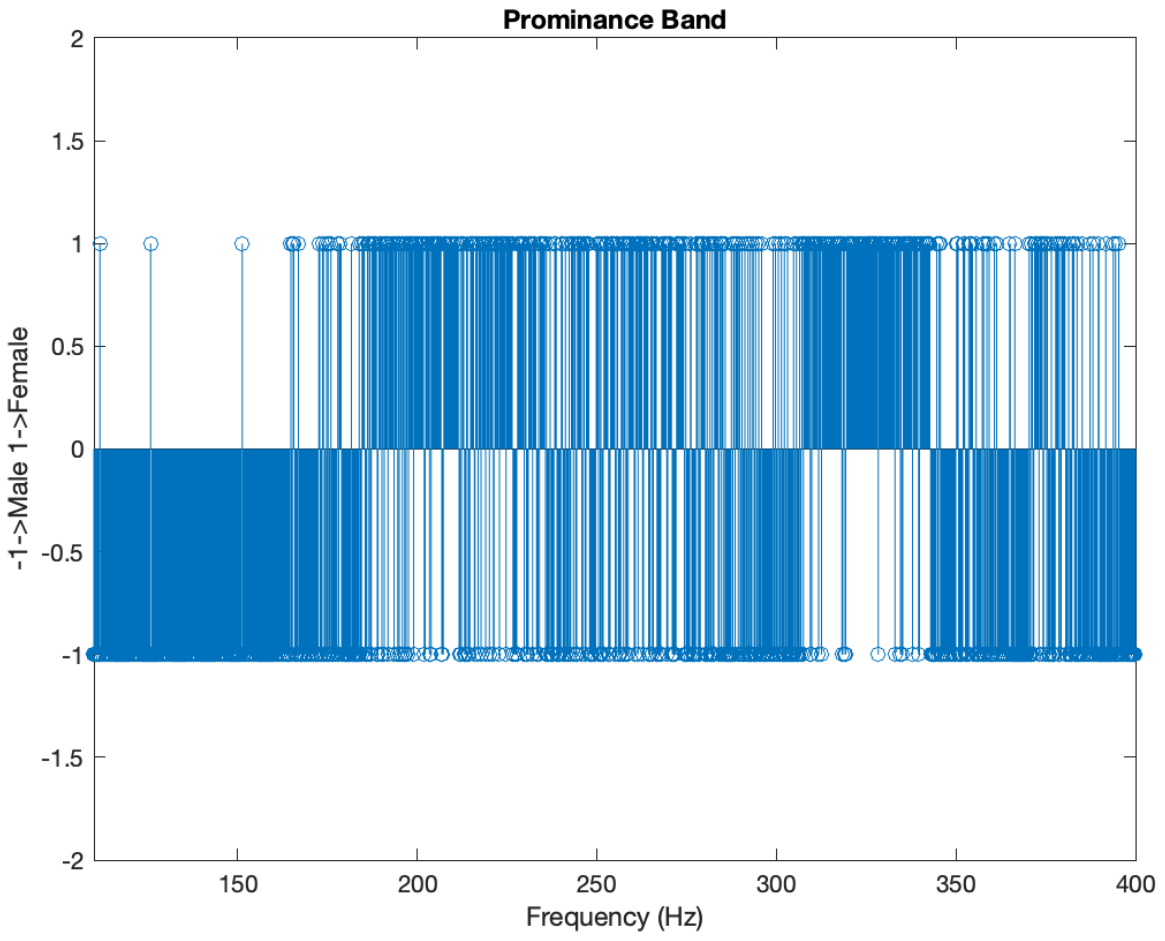
Comparison of frequency spectrum

- It is clear from the above 2 graphs that the fundamental frequency and the harmonics of the male voice is less than the female voice.
- For a better picture of the frequency content analysis, the normalized magnitude of male subtracted from the female is plotted below.
- This shows the relative dominance of the 2 voices at a particular frequency.



Prominent frequency bands

- Something that would also be useful is the approximate band of frequencies at which male/female frequencies are dominant over the other.
- This has been plotted in the following figure. We observe that until around 165Hz, the male voice is more dominant.
- This graph has been plotted by comparing the male and female amplitude at each frequency. 1 is plotted if the female's magnitude is more. -1 is plotted if the male's magnitude is more.



Matlab Code

```
close all;
% Load the audio sample
[y2, Fs] = audioread('female.wav');
y2=y2(:,1);
[y3, Fs] = audioread('male.wav');
y3=y3(:,1);
%
if length(y3) > length(y2)
    y2 = [y2.', zeros(1, length(y3) - length(y2))];
    y2=y2.';
else
    y3 = [y3.', zeros(1, length(y2) - length(y3))];
    y3=y3.';
end

% Apply FFT
Y = fft(y2);

% Shift FFT
Yshifted2 = fftshift(Y);
```

```
% Plot FFT
figure;
f = linspace(-Fs/2, Fs/2, length(y2));
ff2= abs(Yshifted2)/max(abs(Yshifted2));
plot(f, abs(Yshifted2)/max(abs(Yshifted2)));
xlim([-1000 1000]);
title('Real Female');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

% Load the audio sample

% Apply FFT
Y = fft(y3);

% Shift FFT
Yshifted3 = fftshift(Y);

% Plot FFT
f = linspace(-Fs/2, Fs/2, length(y3));
figure;
ff3= abs(Yshifted3)/max(abs(Yshifted3));
plot(f, abs(Yshifted3)/max(abs(Yshifted3)));
xlim([-1000 1000]);
title('Real Male');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%Prominant Bands

% initialize the third array with zeros
C = zeros(size(ff2));

% loop through the arrays and set the corresponding value in C
for i = 1:length(ff3)
    if ff2(i) > ff3(i)
        C(i) = 1;
    elseif ff2(i) < ff3(i)
        C(i) = -1;
    end
end

figure;
stem(f, C);
ylim([-2 2]);
xlim([110 400]);
title('Prominance Band');
xlabel('Frequency (Hz)');
ylabel('-1->Male 1->Female');
```

```
%Female-Male

figure;
stem(f,ff2-ff3);
xlim([150 400]);
title('Female - Male Magnitude');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
```

Our approach to voice conversion-PSOLA

.

The PSOLA synthesis algorithm scheme involves three steps:

- An analysis of the original speech waveform in order to produce intermediate non parametric representation of the signal.
- Modification brought to the intermediate representation.
- Synthesis of the modified signal from the modified intermediate representation.

Pitch-synchronous analysis

The intermediate representation of the digitized speech waveform x(n) consists of a sequence of short-term signals $x_m(n)$, obtained by multiplying the signal by a sequence of pitch-synchronous analysis windows $h_m(n)$:

$$x_m(n) = h_m(t_m - n)x(n)$$

These windows are centered around the successive instants tm, called pitch marks, which are set at a pitch synchronous rate on the voiced portions of the signal and at a constant rate in the unvoiced portions.

The windows $h_m(n)$ are of Hanning type and they are always longer than one single pitch period, so that neighbouring short-term signals (ST-signals) always involve a certain overlap. Their lengths are usually set to be proportional to the local pitch period, with a proportionality factor μ ranging from $\mu = 2$, for short analysis windows, to $\mu = 4$, for longer ones. These values correspond to 50% and 75% window overlapping factors, respectively. The proportionality rule for window length can be expressed as follows:

$$h_m(n) = h(n/\mu P)$$

where h(t) is the window with the length normalized to unity and P denotes the local pitch period.

Pitch synchronous modifications

The stream of analysis ST-signals xm(n) is converted into a modified stream of synthesis ST signals $\tilde{x}_q(n)$ synchronised on a new set of synthesis pitch marks $\tilde{t}_q$. Such a conversion involves three basic operations: a modification of the number of ST-signals, a modification of the delays between the ST-signals, and possibly, a modification of the waveform of each individual ST-signal. The number of synthesis pitch-marks t_q depends on the pitch-scale and time-scale modifications factors called β and γ, respectively. The delays $\tilde{t}_q - \tilde{t}_{q-1}$ between two successive pitch-marks must be equal to the local synthesis pitch period. The algorithm works out a mapping $\tilde{t}_q \rightarrow \tilde{t}_m$ between the synthesis and analysis pitch-marks, specifying which analysis ST-signal $x_m(n)$ is to be selected to produce any given synthesis ST-signal $\tilde{x}_q(n)$. In the Time-Domain PSOLA (TD-PSOLA) approach, the synthesis ST-signals are obtained by simply copying a version of the corresponding analysis signal, so that the algorithm consists in selecting a certain number of analysis ST-signals $x_m(n)$ and translating them by the sequence of delays $\delta_q = \tilde{t}_q - t_m$:

$$\tilde{x}_q(n) = x_m(n - \delta_q) = x_m(n + t_m - \tilde{t}_q)$$

In the Frequency-Domain PSOLA (FD-PSOLA) approach, the synthesis ST-signals are obtained by a frequency-domain transformation of the translated signal $x_m(n - \delta_q)$.

Pitch-synchronous overlap-add synthesis

Several overlap-add (OLA) synthesis procedures are available to obtain the final synthetic speech. For instance, the synthetic signal $\tilde{x}(n)$ can be obtained by means of the least-square overlap- add synthesis scheme:

$$\tilde{x}(n) = \frac{\sum_q \alpha_q \tilde{x}_q(n) \tilde{h}_q(\tilde{t}_q - n)}{\sum_q \tilde{h}_q^2(\tilde{t}_q - n)}$$

where $\tilde{h}_q(n)$ denotes the sequence of synthesis windows. The additional normalization factor α_q is introduced to compensate for the energy modifications related to the pitch modification procedure. The spectral interpretation of this synthesis scheme is that it minimizes the quadratic error between the spectra of the synthesis ST-signals .$\tilde{x}_q(n)$ and the corresponding short-time spectra of the synthetic speech $\tilde{x}(n)$. An alternative synthesis scheme is the simple overlap-add procedure:

$$\tilde{x}(n) = \frac{\sum_q \alpha_q \tilde{x}_q(n)}{\sum_q \tilde{h}_q(\tilde{t}_q - n)}$$

As in the least-square synthesis formula, the denominator of this formula plays the role of a time variable normalization factor: it compensates for the energy modifications due to the variable overlap between the successive windows. Under narrow band conditions, this factor is nearly constant. Under wide band conditions, it can also be kept constant, in particular when the synthesis window length is chosen to be equal to twice the synthesis pitch period. In such cases, and assuming a q = 1, the synthesis formula is reduced to the simplified overlap-add scheme:

$$\tilde{x}(n) = \sum_q \tilde{x}_q(n)$$

In this last formula , the synthetic signal appears as a simple linear combination of windowed and translated versions of the original signal. All the operations involved are linear except the windowing operation . Subsequently, when combining the PSOLA speech modification scheme with a linear filter such as a LPC-filter or a low-pass filter, the order of the operations cannot be changed without modifying the behaviour of the overall system.

Implementation of PSOLA

The PSOLA algorithm first tries to find the peaks of the signal. For this purpose it assumes a window size and finds the dominant frequency in the window. Once that is done we then take the average of the the periods obtained by dividing the sampling frequency with the dominant frequency of that anlaysis sequence. Once that is done, the algorithm tries to find the position of the new peaks based on the old peaks and the ratio by which we want to change the frequencies of the signal. This way we find the new peaks of the signal. The reason to this is then to use hanning windows of the signal to get a particular sequence and then add them to get a pitch shifted signal which will change the characteristics to be close to the new values of frequencies we wanted. The hanning window ensures that there will be overlap between consecutive sequences.

Step by Step Procedure:

- Find the peaks in the signal
- Find the new peaks (location) using the frequency ratio
- Use a hanning window at every new peak and multiply with the signal and add the windowed outputs.

The new signal obtained by this procedure will be the output signal we need.

The transformation leads to the following characteristics:

- If we use a frequency ratio >1 then the frequency contents of the signal shift to the higher values.
- If we use a frequency ration <1 then the frequency contents of the signal shift to the lower values.

The further away from 1 the frequency ratio the more distorted the signal will be as the voice quality is limited and the overlap add might lead to a distorted version.

Python Code

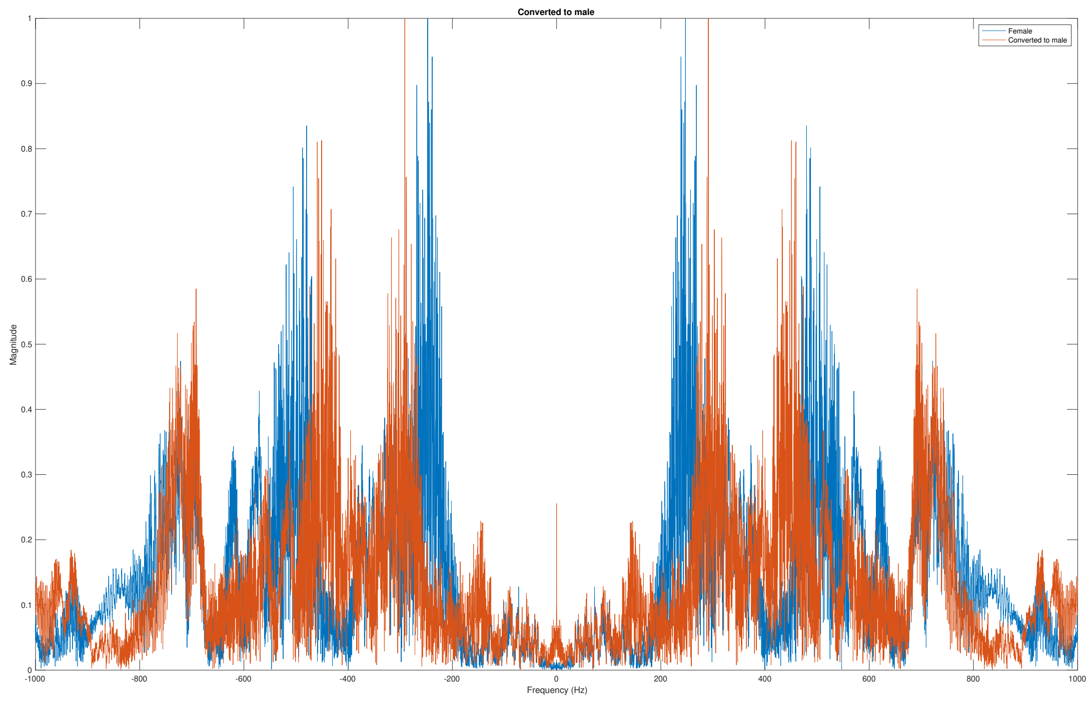
```
import soundfile as sf
import numpy as np
from numpy.fft import fft, ifft
import matplotlib.pyplot as plt  # These are the imports which we needed
import librosa
from scipy import signal


def pitch_shift(signal, fs, f_ratio):
    '''
    This function helps in shifting the pitch in the time domain by a factor "f_ratio"
    f_ratio is ratio between new frequncy needed and original frequency
    '''
    peaks = find_peaks(signal, fs)
    new_signal = psola(signal, peaks, f_ratio)
    return new_signal


def find_peaks(signal, fs, max_hz=950, min_hz=75, analysis_win_ms=40, max_change=1.05, min_change=0.95):
    '''
    The function first computes the pitch periodicity by dividing the signal into small overlapping windows and
    computing the period of each window.Then it finds the peaks in the signal that correspond to the fundamental
    frequency by searching for the maximum values near the expected location. The function returns an array of the
    peak positions in the input signal.
    '''
    N = len(signal)
    min_period = fs // max_hz
    max_period = fs // min_hz


    sequence = int(analysis_win_ms / 1000 * fs)
    periods = periods_per_seq(signal, sequence, min_period, max_period)

    peaks = [np.argmax(signal[:int(periods[0]*1.1)])]
    while True:
        prev = peaks[-1]
        idx = prev // sequence
        if prev + int(periods[idx] * max_change) >= N:
            break
        peaks.append(prev + int(periods[idx] * min_change) + np.argmax(signal[prev + int(periods[idx] * min_chan
        ge): prev + int(periods[idx] * max_change)]))
    return np.array(peaks)


def periods_per_seq(signal, sequence, min_period, max_period):
    '''
    The function computes the pitch periodicity of the signal by dividing it into small windows
    and computing the period of each window using autocorrelation.The autocorrelation peak period is found
    by computing the FFT of each window, setting the DC component to zero, taking the inverse FFT, and finding the
    maximum value in the specified period range.The function returns an array of the autocorrelation peak periods
    for each window sequence.
    '''
    offset = 0  # current sample offset
    periods = []  # period length of each analysis sequence

    while offset < N:
        fourier = fft(signal[offset: offset + sequence])
        fourier[0] = 0  # remove DC component
        auto_correlation = ifft(fourier * np.conj(fourier)).real
        auto_correlation_peak = min_period + \
            np.argmax(auto_correlation[min_period: max_period])
        periods.append(auto_correlation_peak)
        offset += sequence
    return periods


def psola(signal, peaks, f_ratio):
    '''
     The function first performs linear interpolation to change the number of peak indices to match the desired
     time-stretch factor. Then by iterating over each new peak index, finding the corresponding old peak index,
     and constructing an overlapping window centered at the new peak that is applied to a portion of the input
     signal centered at the old peak. The resulting windowed signals are then added together to produce the output
     signal with the desired time-stretch factor. This functions return the final sginal.
    '''
    N = len(signal)

    new_signal = np.zeros(N)
    print("The value of f_ratio is", f_ratio)
    new_peaks_ref = np.linspace(0, len(peaks) - 1, int(len(peaks) * f_ratio))
    new_peaks = np.zeros(len(new_peaks_ref)).astype(int)

    for i in range(len(new_peaks)):
        weight = new_peaks_ref[i] % 1
        left = np.floor(new_peaks_ref[i]).astype(int)
        right = np.ceil(new_peaks_ref[i]).astype(int)
        new_peaks[i] = int(peaks[left] * (1 - weight) + peaks[right] * weight)


    for j in range(len(new_peaks)):
        # find the corresponding old peak index
        i = np.argmin(np.abs(peaks - new_peaks[j]))
        # get the distances to adjacent peaks
        P1 = [new_peaks[j] if j == 0 else new_peaks[j] - new_peaks[j-1],N - 1 - new_peaks[j] if j == len(new_peak
        s) - 1 else new_peaks[j+1] - new_peaks[j]]
        # edge case truncation
        if peaks[i] - P1[0] < 0:
            P1[0] = peaks[i]
        if peaks[i] + P1[1] > N - 1:
            P1[1] = N - 1 - peaks[i]
        # linear OLA window
        window = list(np.linspace(0, 1, P1[0] + 1)[1:]) + list(np.linspace(1, 0, P1[1] + 1)[1:])
        # center window from original signal at the new peak
        new_signal[new_peaks[j] - P1[0]: new_peaks[j] +
        P1[1]] += window * signal[peaks[i] - P1[0]: peaks[i] + P1[1]]
    return new_signal




sample_rate = 44100                            # These are parameters for lowpass filter
cutoff_freq = 2000  # Hz
nyquist_freq = 0.5 * sample_rate
cutoff_normalized = cutoff_freq / nyquist_freq
order = 5


b, a = signal.butter(order, cutoff_normalized, btype='low')   # Create the Butterworth filter

orig_signal, fs = librosa.load("Voice2.wav", sr=44100)            #loading the voice

orig_signal = signal.lfilter(b, a, orig_signal)                    #using the LPF filter
N = len(orig_signal)

f_ratio = 0.6                                                       # Pitch shift amount as a ratio

new_signal = pitch_shift(orig_signal, fs, f_ratio)                  # Shift pitch

new_signal = 10*new_signal                                          #Increase the amplitude

sf.write('female_scale_transposed.wav', new_signal, fs)              #Write the output
```

Result on the sample

Female to Male

- The file female.wav was passed through a low-pass filter with a cutoff frequency of 2000Hz so that the background noise is filtered out to some extent.
- The filtered file is then sent through the PSOLA-based sex changer.

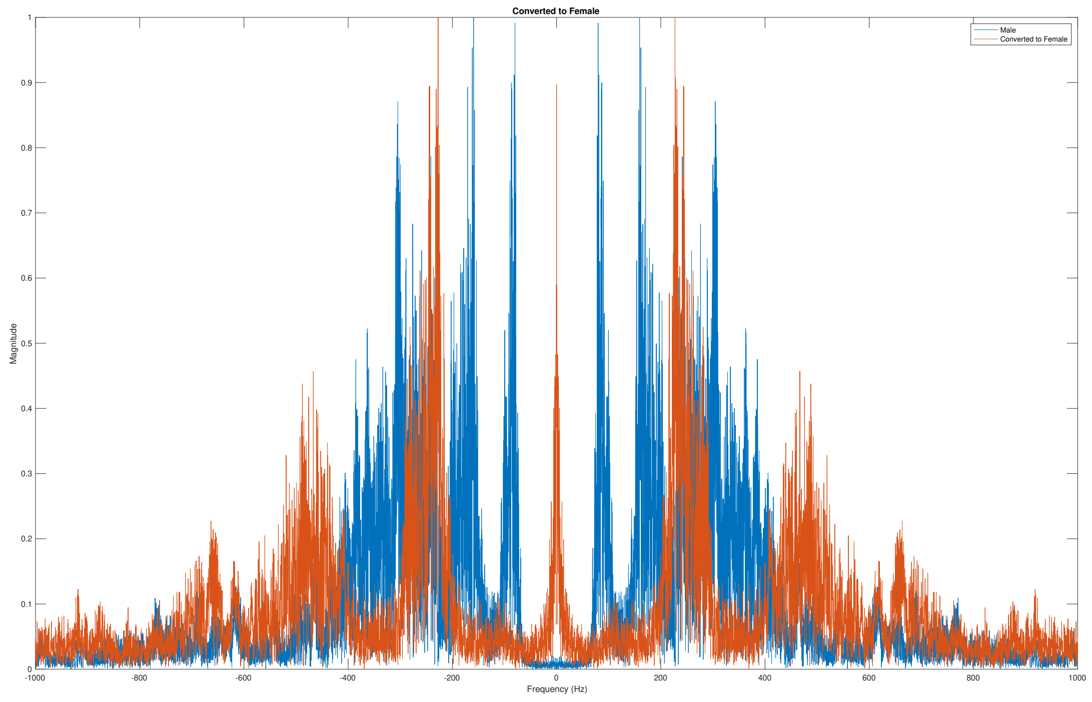
- The frequency comparison of this converted to male voice and the original voice is given below.



Male to Female

- The file male.wav was passed through a low-pass filter with a cutoff frequency of 2000Hz so that the background noise is filtered out to some extent.
- The filtered file is then sent through the PSOLA-based voice sex changer.

- The frequency comparison of this converted to female voice and the original voice is given below.



Matlab Code

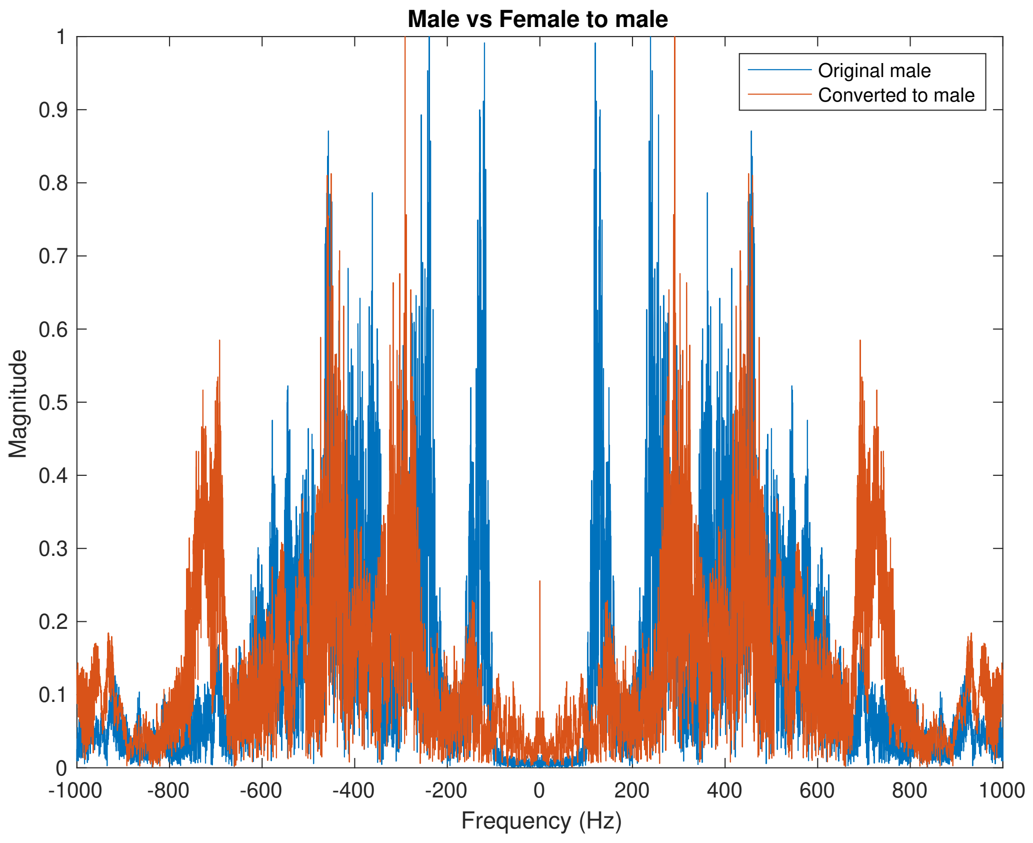
```
close all;
% Load the audio sample
[y1, Fs] = audioread('female_to_male.wav');
[y4, Fs] = audioread('male_to_female.wav');
[y2, Fs] = audioread('female.wav');
y2=y2(:,1);
[y3, Fs] = audioread('male.wav');
y3=y3(:,1);
if length(y3) > length(y2)
    y2 = [y2.', zeros(1, length(y3) - length(y2))];
    y2=y2.';
else
    y3 = [y3.', zeros(1, length(y2) - length(y3))];
    y3=y3.';
end

if length(y1) > length(y2)
    y2 = [y2.', zeros(1, length(y1) - length(y2))];
    y2=y2.';
else
    y1 = [y1.', zeros(1, length(y2) - length(y1))];
    y1=y1.';
end

if length(y1) > length(y4)
```

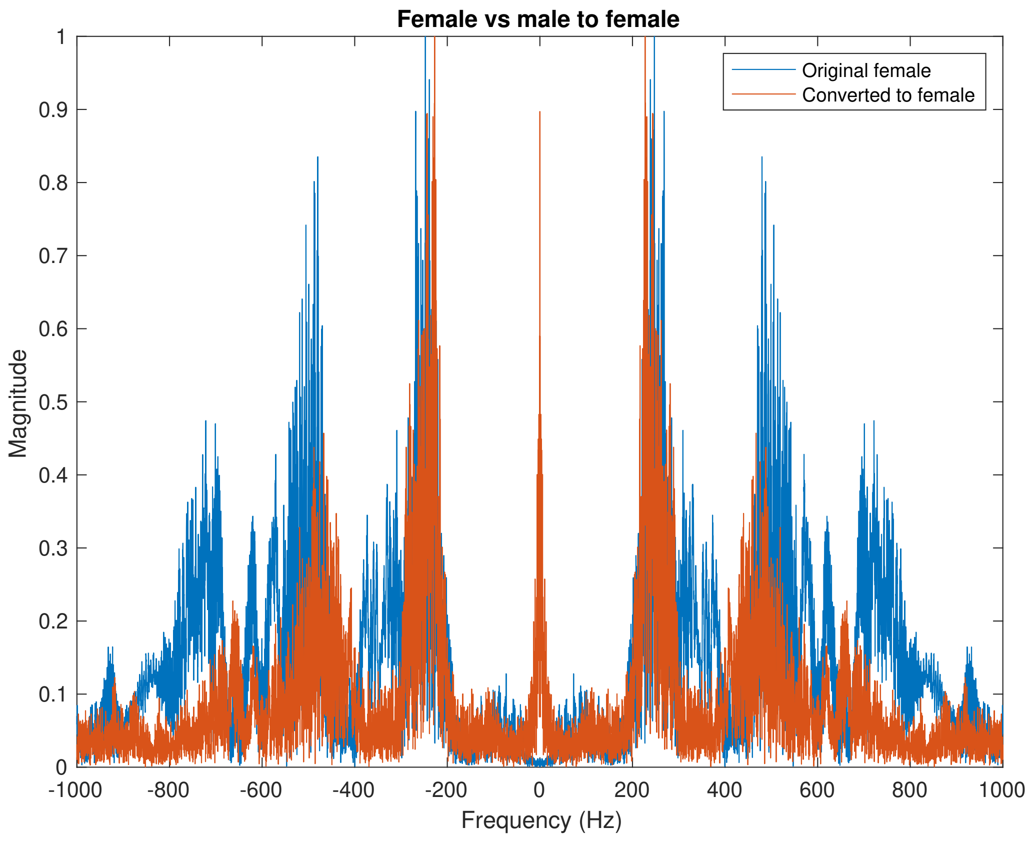
```
    y2 = [y2.', zeros(1, length(y1) - length(y4))];
    y4=y4.';
else
    y1 = [y1.', zeros(1, length(y4) - length(y1))];
    y1=y1.';
end



%female_to_male

% Apply FFT
Y = fft(y2);

% Shift FFT
Yshifted2 = fftshift(Y);
Fs
% Plot FFT
figure;
f = linspace(-Fs/2, Fs/2, length(y2));
ff2= abs(Yshifted2)/max(abs(Yshifted2));
plot(f, abs(Yshifted2)/max(abs(Yshifted2)));hold on;
xlim([-1000 1000]);

xlabel('Frequency (Hz)');
ylabel('Magnitude');

% Apply FFT
Y = fft(y1);

% Shift FFT
Yshifted1 = fftshift(Y);

% Plot FFT
f = linspace(-Fs/2, Fs/2, length(y1));
ff1= abs(Yshifted1)/max(abs(Yshifted1));
plot(f, abs(Yshifted1)/max(abs(Yshifted1)));
xlim([-1000 1000]);
title('Converted to male');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
legend('Female','Converted to male')


%male_to_female

% Apply FFT
Y = fft(y3);

% Shift FFT
```

```
Yshifted3 = fftshift(Y);
Fs
% Plot FFT
figure;
f = linspace(-Fs/3, Fs/3, length(y3));
ff3= abs(Yshifted3)/max(abs(Yshifted3));
plot(f, abs(Yshifted3)/max(abs(Yshifted3)));hold on;
xlim([-1000 1000]);

xlabel('Frequency (Hz)');
ylabel('Magnitude');

% Apply FFT
Y = fft(y4);

% Shift FFT
Yshifted4 = fftshift(Y);

% Plot FFT
f = linspace(-Fs/2, Fs/2, length(y1));
ff1= abs(Yshifted4)/max(abs(Yshifted4));
plot(f, abs(Yshifted4)/max(abs(Yshifted4)));
xlim([-1000 1000]);
title('Converted to Female');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
legend('Male','Converted to Female')
```

Performance verification

- To check how well the converted voice matches the target gender, it can be compared to the reference voice of the target gender.
- Here we compare female to male to male.wav and male to female to female.wav.
- Through the FFT, we can see that they do approximately match.in terms of frequency contents.


Male vs Female to male
Original male
Converted to male
Magnitude
Frequency (Hz)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-1000
-800
-600
-400
-200
0
200
400
600
800
1000


Female vs male to female
Original female
Converted to female
Magnitude
Frequency (Hz)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-1000
-800
-600
-400
-200
0
200
400
600
800
1000

Matlab Code

```
close all;
% Load the audio sample
[y1, Fs] = audioread('female_to_male.wav');
[y4, Fs] = audioread('male_to_female.wav');
[y2, Fs] = audioread('female.wav');
y2=y2(:,1);
[y3, Fs] = audioread('male.wav');
y3=y3(:,1);
if length(y3) > length(y2)
    y2 = [y2.', zeros(1, length(y3) - length(y2))];
    y2=y2.';
else
    y3 = [y3.', zeros(1, length(y2) - length(y3))];
    y3=y3.';
end

if length(y1) > length(y2)
    y2 = [y2.', zeros(1, length(y1) - length(y2))];
    y2=y2.';
else
    y1 = [y1.', zeros(1, length(y2) - length(y1))];
    y1=y1.';
end

if length(y1) > length(y4)
    y2 = [y2.', zeros(1, length(y1) - length(y4))];
    y4=y4.';
else
    y1 = [y1.', zeros(1, length(y4) - length(y1))];
    y1=y1.';
end


% Apply FFT
Y = fft(y3);

% Shift FFT
Yshifted3 = fftshift(Y);
Fs
% Plot FFT
figure;
f = linspace(-Fs/2, Fs/2, length(y3));
ff3= abs(Yshifted3)/max(abs(Yshifted3));
plot(f, abs(Yshifted3)/max(abs(Yshifted3)));hold on;
xlim([-1000 1000]);
xlabel('Frequency (Hz)');
ylabel('Magnitude');
```

```
% Apply FFT
Y = fft(y1);

% Shift FFT
Yshifted1 = fftshift(Y);

% Plot FFT
f = linspace(-Fs/2, Fs/2, length(y1));
ff1= abs(Yshifted1)/max(abs(Yshifted1));
plot(f, abs(Yshifted1)/max(abs(Yshifted1)));
xlim([-1000 1000]);
title('Male vs Female to male');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
legend('Original male','Converted to male')

% Apply FFT
Y = fft(y2);

% Shift FFT
Yshifted2 = fftshift(Y);
Fs
% Plot FFT
figure;
f = linspace(-Fs/2, Fs/2, length(y2));
ff2= abs(Yshifted2)/max(abs(Yshifted2));
plot(f, abs(Yshifted2)/max(abs(Yshifted2)));hold on;
xlim([-1000 1000]);
xlabel('Frequency (Hz)');
ylabel('Magnitude');

% Apply FFT
Y = fft(y4);

% Shift FFT
Yshifted4 = fftshift(Y);

% Plot FFT
f = linspace(-Fs/2, Fs/2, length(y4));
ff4= abs(Yshifted4)/max(abs(Yshifted4));
plot(f, abs(Yshifted4)/max(abs(Yshifted4)));
xlim([-1000 1000]);
title('Female vs male to female');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
legend('Original female','Converted to female')
```