Assignment 1: Speech Synthesis (PDF Version)

▼ **Table of Contents**

Abstract

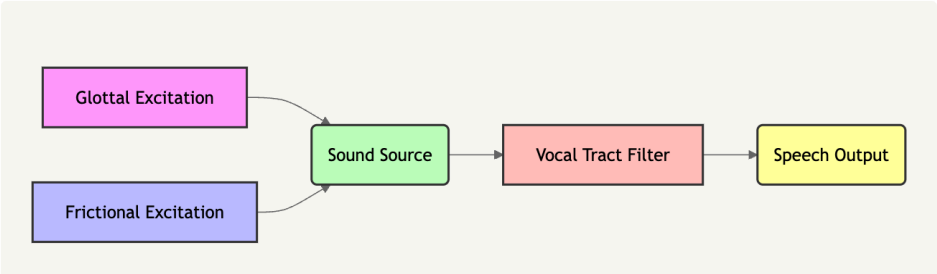
For a more engaging and interactive experience, visit the website here:

Assignment 1: Speech Synthesis.

The website features audio samples that complement this work.

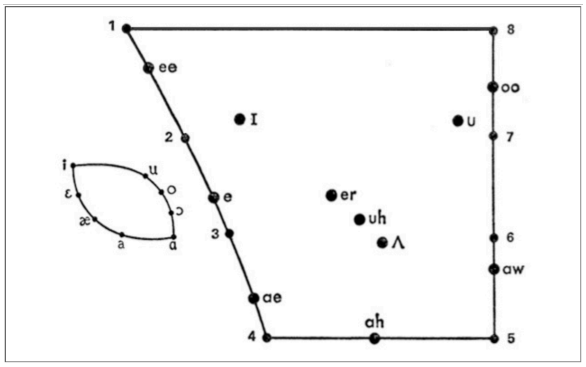
The code referenced in this assignment is here: https://github.com/frankcholula/sapr

A simple acoustic model of the voice begins with an airstream from the lungs. This stream undergoes periodic modulation by **the vocal folds** before being filtered through the **vocal tract**, which functions as a variable resonator. This process is represented by the source-filter model, illustrated below.

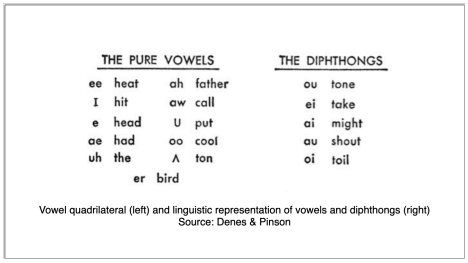


The source-filter model

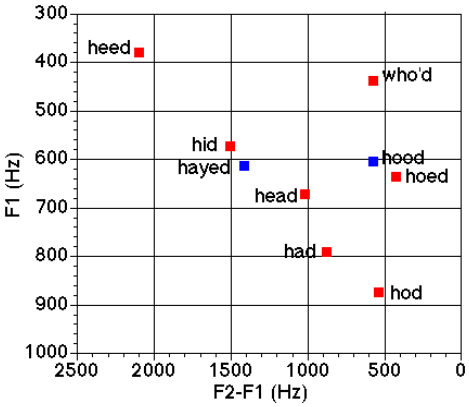
It's also interesting to examine the resonant system (our vocal tract) based on the tongue position and relate it to the vowel quadrilateral below. **Front vowels**, produced with the tongue body forward in the mouth, are characterized by a significant frequency gap between F1 and F2. **Back vowels**, formed with the tongue body retracted in the mouth, have F1 and F2 frequencies closer together.



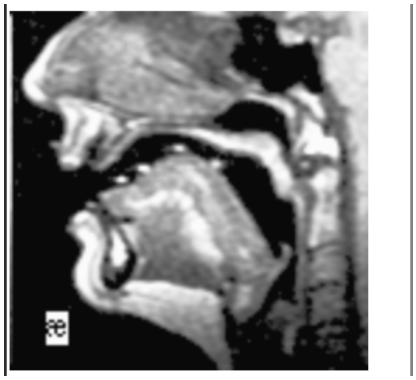
Vowel quadrilateral by Truax [4]



Linguistic representation of vowels and diphthongs by Truax [4]



Formant frequencies of vowels by Goldstein [5]



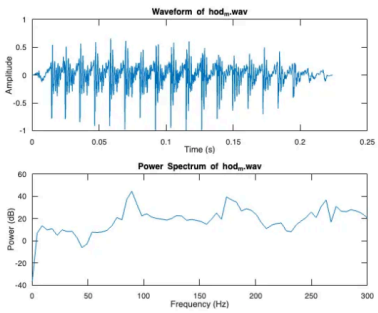
MR image of heed by Goldstein [5]

For the assignment, I'll focus on the **glottal excitation** of voiced speech by synthesizing only pure vowels (**monophthongs**) due to the simple, fixed tongue position in the mouth (see the MR image above). This is done by generating periodic impulse trains and pass them through an all-pole filter to create a set of resonant frequencies known as **formants**. Each vowel's formant structure will then be estimated from real speech samples using **linear predictive coding** (LPC). Finally, I'll analyze the quality of the synthesized speech and identify potential improvements.

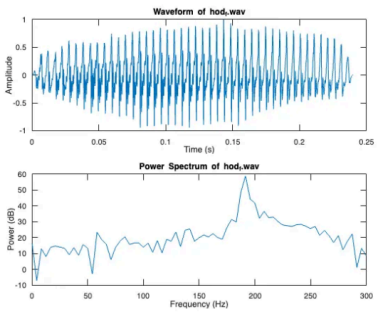
I'll be referencing the code I've written throughout this report. You can find all the code for the assignment at my GitHub link here: https://github.com/frankcholula/sapr.

Preliminary Analyses

First, we need to select a quasi-stationary segment of about **100ms in length** from each vowel for the LPC estimation. Upon inspection of the male and female hod audio samples, we can approximate this by taking the midpoint of each waveform.



Playing hod_m.wav at **24000 Hz** with duration **0.22 seconds** and **5380 samples** [1]

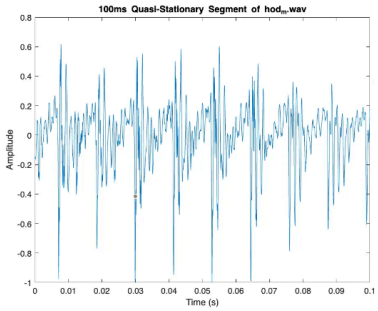


Playing hod_f.wav at **24000 Hz** with duration **0.24 seconds** and **5763 samples** [1]

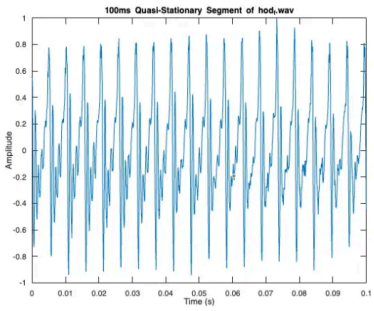
For the quasi-stationary segments, I've chosen the middle of the signal, which is a relatively safe assumption based on the waveform images shown above. These segments are characterized by consistent amplitude and frequency patterns.

Estimating F_0 by Inspection

I've selected the open back unrounded vowel `[a]` using the `hod` audio sample. Visual inspection of the power spectrum above reveals that the fundamental frequency peaks at approximately **90 Hz** for the male speaker and **180 Hz** for the female speaker. Note that I've purposely limited the x-axis to **0-300 Hz**, as this range typically encompasses the fundamental frequencies for both male and female speakers, allowing for clearer visualization of the F_0 peaks. These frequencies align with the typical ranges of human speech: **90 to 155 Hz** for adult males and **165 to 255 Hz** for adult females.



100ms quasi-stationary signal of `hod_m.wav` [1]



100ms quasi-stationary signal of `hod_f.wav` [1]

Estimating Formants by Inspection

Recall the *frequency resolution* and *time resolution* of a spectrogram are determined by the window size and the sampling rate.

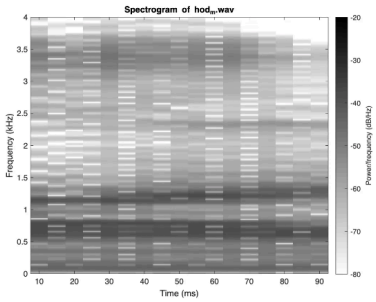
$$f_{res} = \frac{f_s}{N} \qquad t_{res} = \frac{N}{f_s}$$

These formulas illustrate the trade-off between frequency and time resolution in spectrogram analysis. A larger window size enhances frequency resolution but diminishes time resolution, and vice versa. Drawing from the lecture slides [6] and Spectrogram of Speech [2], I've generated the spectrogram using a 20ms window, considering that phonemes typically occur at a rate of 4–5 per second. To mitigate spectral leakage, I've applied a Hamming window, which smoothly tapers the signal's ends. The window size in samples is calculated by multiplying the sampling frequency with the window duration. Additionally, I've selected a hop size of **5ms** (75% overlap of the window size) to ensure smooth frame-to-frame transitions in energy distribution.

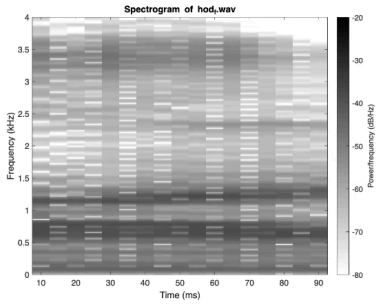
▼ **Code for Plotting the Spectrogram**

```
function plotSpectrogram (audio, fs, fileName)
    colormap('gray');
    map = colormap;
    imap = flipud(map);
    window_size = round(0.02 * fs);
    overlap = floor(window_size * 0.75);
    N = 2^nextpow2(4 * window_size);vv
    fprintf('Window Size: %d samples (%.2f ms)\n',
                      window_size, (window_size / fs) * 1000);
    fprintf('Overlap: %d samples (%.2f ms)\n', overlap, (overlap / fs) * 1000);
    fprintf('FFT Length (N): %d' , N);
```

```
    spectrogram(audio, hamming(window_size), overlap, N, fs, 'yaxis');
    colormap(imap);
    clim([-80, -20]); % Adjust dynamic range for more distinct blacks
    title(['Spectrogram of ', fileName]);
    ylim([0, 4]);
end
```



distinct black lines at **730 Hz**, **1100 Hz**, and **2400 Hz**. [1]



distinct black lines at **800 Hz**, **1200 Hz**, and **2500 Hz.** [1]

Upon examining the spectrograms, we can observe the first three formants for `hod_m.wav` at approximately **730 Hz**, **1100 Hz**, and **2400 Hz**. For `hod_f.wav`, the formants appear at about **800 Hz**, **1200 Hz**, and **2500 Hz**. These findings align with the table below, provided by Simon Fraser University's Sonic Research Studio. As these are back vowels, we expect F1 and F2 frequencies to be closer together.

		heed	head	had	hod	haw'd	who'd
Men	F1	270	530	660	730	570	300
	F2	2290	1840	1720	1090	840	870
	F3	3010	2480	2410	2440	2410	2240
Women	F1	310	610	860	850	590	370
	F2	2790	2330	2050	1220	920	950
	F3	3310	2990	2850	2810	2710	2670
Children	F1	370	690	1010	1030	680	430
	F2	3200	2610	2320	1370	1060	1170
	F3	3730	3570	3320	3170	3180	3260

Formant table by Truax [4]

Model Estimation

The LPC model predicts the current sample *s[n]* based on **a weighted sum of past samples**:

$$\hat{s}[n] \approx -\sum_{k=1}^{p} a_k s[n-k] + G \cdot u[n]$$

Where:

- ŝ[n] is the current sample
- p is the order of the LPC model
- a_k are the LPC coefficients
- s[n-k] are the past samples
- u[n] is the input excitation signal

After rearranging,

$$s[n] + \sum_{k=1}^{p} a_k \cdot s[n-k] - G \cdot u[n] = 0$$

The intuition is that in an ideal case where the prediction is perfect, the predicted sample ŝ[n] is exactly equal to s[n], leading to a prediction error of zero. Taking the Z transform of the difference equation lands us the transfer function of the LPC filter, which characterizes the `spectral envelope` of the speech signal:

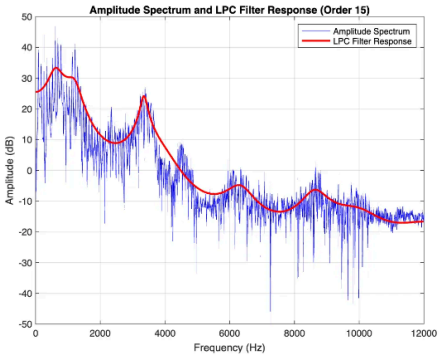
$$H(z) = \frac{G}{1 + \sum_{k=1}^{p} a_k z^{-k}}$$

Where:

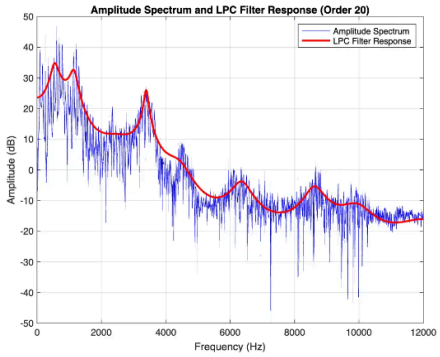
- G is the gain factor
- p is the order of the LPC filter
- a_k are the LPC coefficients
- z^{-k} represents a delay of k samples

Order Selection of the LPC Model

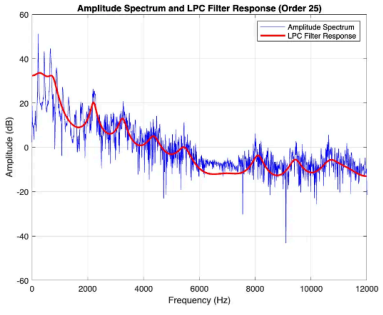
In practice, LPC orders of around 20–30 are sufficient to model the first few formants in speech. To illustrate the impact of different orders, I've generated plots comparing the amplitude spectrum to the filter response for orders ranging from 15 to 30, with increments of 5.



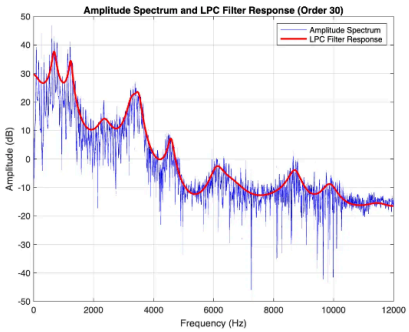
Order 15 filter response [1]



Order 20 filter response [1]



Order 25 filter response [1]



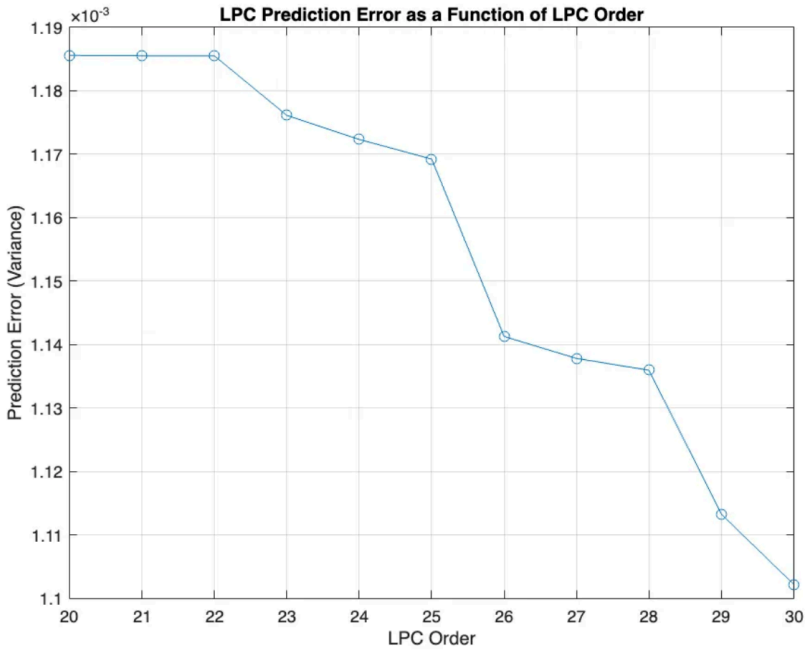
Order 30 filter response [1]

▼ **Code to Generate the Plots**

```
function plotLPCResponse(segment, fs, lpcOrder)
    [lpcCoeffs, g] = lpc(segment, lpcOrder)
    [h, f] = freqz(1, lpcCoeffs, 1024, fs);
    segmentFFT = fft(segment);
    segmentPowerSpectrum = abs(segmentFFT(1:floor(length(segment) / 2) + 1)).^2;
    freqAxis = (0:floor(length(segment) / 2)) * (fs / length(segment));

    figure;
```

```
    plot(freqAxis, pow2db(segmentPowerSpectrum), 'b');
    hold on;
    plot(f, mag2db(abs(h)), 'r', 'LineWidth', 2);
    title(['Amplitude Spectrum and LPC Filter Response (Order ',
               num2str(lpcOrder),')']);
    xlabel('Frequency (Hz)');
    ylabel('Amplitude (dB)');
    legend('Amplitude Spectrum', 'LPC Filter Response');
    grid on;
end
```

As the order *p* increases, the error between the AR fit and the speech signal decreases. The estimated coefficients yield a closer match to the actual spectral shape. At order 30, the LPC spectrum almost perfectly matches the spectral peaks. Below is a plot illustrating how the variance of the prediction error changes with increasing LPC order. We should observe an improved model fit with higher order. Consequently, **the prediction error variance typically decreases as the model explains more of the signal's energy**.

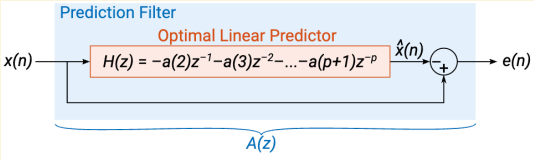


Prediction error vs. LPC order [1]

However, it's important to note that **excessively increasing the LPC order might lead to diminishing returns** in reducing prediction error variance. This is essentially overfitting, where the model begins to capture noise or minor fluctuations in the signal rather than its true underlying structure.

For reference, the `lpc` function, as documented on MathWorks, states that

> `[ a , g ] = lpc( x , p )` finds the coefficients of a `p` th-order linear predictor, an FIR filter that predicts the current value of the real-valued time series `x` based on past samples. The function also returns `g` , **the variance of the prediction error**. If `x` is a matrix, the function treats each column as an independent channel.



Prediction Error Model from MathWorks

The prediction error, $e(n)$, can be viewed as the output of the prediction error filter $A(z)$, where

- $H(z)$ is the optimal linear predictor.
- $x(n)$ is the input signal.
- $\hat{x}(n)$ is the predicted signal.

Recall the error(or residual) **e[n]** in LPC is defined as the difference between the actual sample **s[n]** and the predicted sample **ŝ[n]**:

$$e[n] = s[n] - \hat{s}[n] = s[n] + \sum_{k=1}^{p} a_k s[n-k]$$

This error term represents the residual signal after LPC prediction. Minimizing the mean squared error is the goal of the LPC algorithm.

$$MSE = \frac{1}{N}\sum_{n=1}^{N} e^2[n] = \frac{1}{N}\sum_{n=1}^{N}(s[n] - \hat{s}[n])^2$$

Assuming the error has no correlation with any past samples,

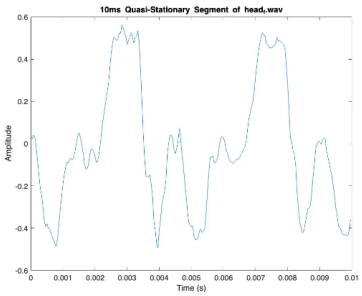
$$\sigma_e^2 = \frac{1}{N}\sum_{n=1}^{N} e[n]^2$$

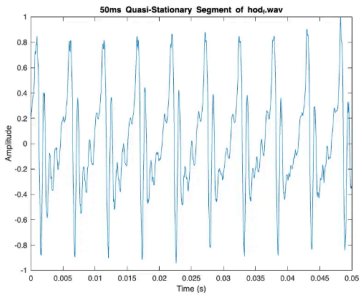
we can safely say that finding the MSE is equivalent to finding the variance of the prediction error provided by the `lpc` function!

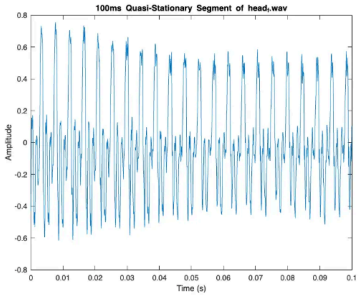
Varying Segment Length

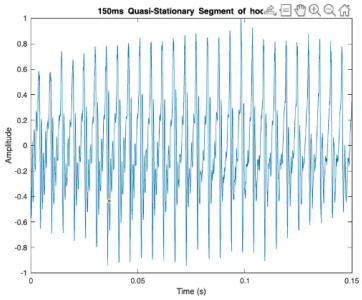
Instead of analyzing only a 100ms quasi-stationary segment, I experimented with various segment lengths and examined how the MSE changes as we adjust the segment duration. Below are the `hod_f.wav` segments of various lengths ranging from 10ms to 150ms to show that they are quasi-stationary.

▼ `hod_f.wav` **Segments of Various Lengths [1]**



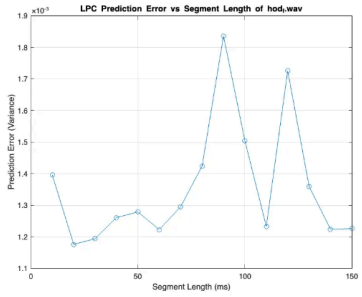


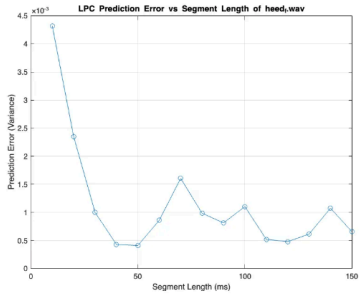


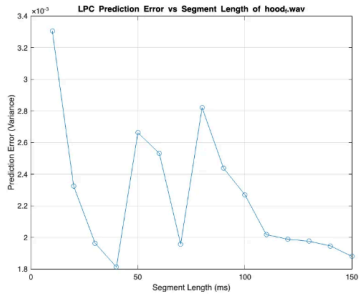


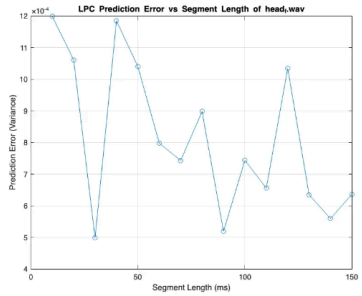
We can proceed to plot the prediction error variance as a function of segment length. For this analysis, I've fixed the LPC order at 30 and examined several different samples: `hod_f.wav`, `heed_f.wav`, `hood_f.wav`, and `head_f.wav`.

▼ **Varying Segment Lengths vs. LPC Error with an Order of 30 [1]**









Upon inspection, there doesn't appear to be a clear correlation between segment length and prediction error. However, across all the examined samples, **a 30ms segment length seems to consistently yield the lowest error variance**, suggesting it might be an optimal choice for analysis. This aligns with the aforementioned duration of phonemes, which generally range from 20 to 40 milliseconds.

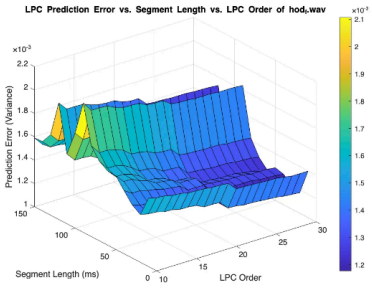
The variations in LPC error variance stem from the signal's stationarity and the selected LPC order. Shorter segments (10 to 30ms) capture formants and spectral characteristics more effectively, yielding lower prediction error variance—

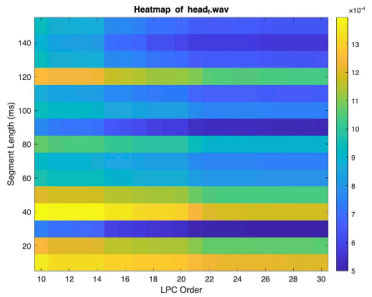
especially with an order of 30. As segment length increases, we encounter greater variability and potentially alternating patterns of stationary and non-stationary regions. Consequently, the LPC model performs well in some segments but poorly in others.

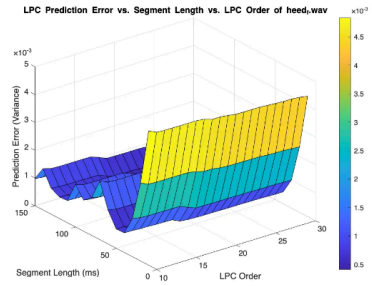
Segment Length vs. LPC Order

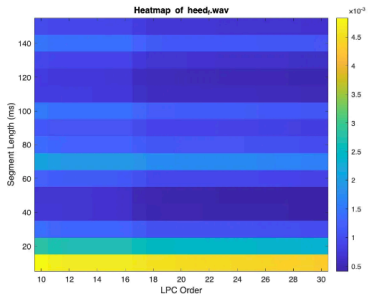
I was also curious about the effects of LPC error as a function of segment length and order, so I generated a 3D surface plot and a corresponding heatmap.

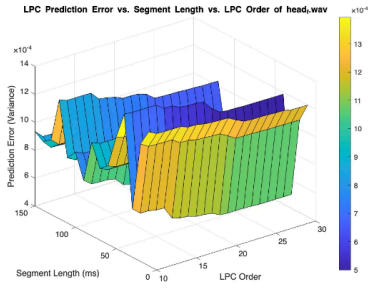
▼ **3D surface plot and heatmap [1]**

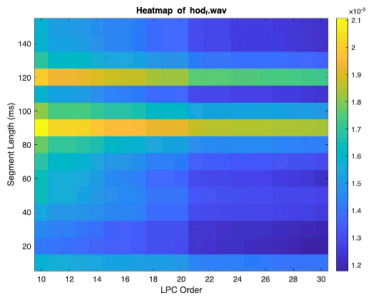


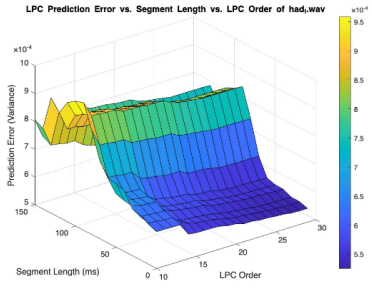


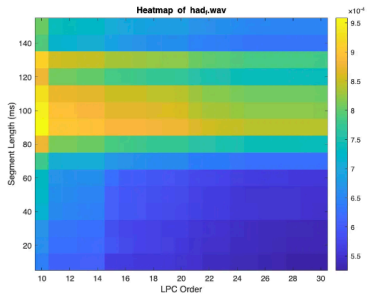












This further corroborates that segment lengths between 20–40ms and LPC orders around 20–30 consistently yield the best results. You can observe this by looking for the dark blue spots on the heatmap.

Estimating F_0 by Autocorrelation

LPC is not designed to determine the fundamental frequency of a speech signal, as it models the vocal tract's filter characteristics rather than the glottal source. A relatively simple and effective method to identify F_0 is using autocorrelation. To break down the steps:

1. Duplicate the original signal.
2. Shift the copy on a sample-by-sample basis.
3. Compute the correlation between the original and shifted signals.
4. Identify the lag with the highest correlation, indicating the period of F_0.

▼ **Below is the code, which includes plotting of the lags and peak values:**

```
function [f0_estimate] = estimateF0ByAutoCorrelation(segment, fs)
    % Compute autocorrelation and find peaks
    [autocorrValues, lags] = xcorr(segment, 'coeff');
    posLagIdx = lags >= 0;
    lags = lags(posLagIdx);
    autocorrValues = autocorrValues(posLagIdx);
    figure();
    plot(lags, autocorrValues);
    [peakValues, locs] = findpeaks(autocorrValues,
                                                    'MinPeakHeight', 0
                                        'MinPeakDistance', fs / 90,
                                        'MinPeakProminence', 0.2);

    if isempty(peakValues)
        f0_estimate = NaN; % Return NaN if no peaks are found
        disp('No peaks found for F0 estimation');
        return;
    end

    % Find the maximum peak value and its location
    [maxPeak, maxPeakIdx] = max(peakValues);
    peakLag = lags(locs(maxPeakIdx)); % Get the lag corresponding to the max peak

    % Convert lag of the maximum peak to frequency
    f0_estimate = fs / peakLag;
    fprintf('Estimated F0 using autocorrelation: %.2f Hz\n', f0_estimate);
end
```

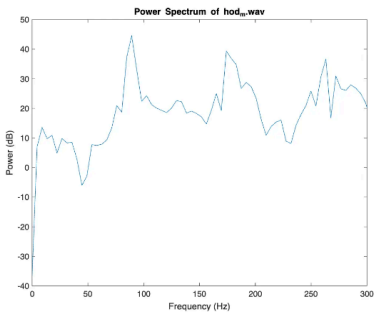
I utilized the `xcorr` function to compute the normalized autocorrelation of the input signal segment, yielding both autocorrelation values and their corresponding lag indices. It's worth noting that we need only examine the positive lags, as these represent rightward shifts—or time delays—in the signal.

After identifying the maximum peak and its corresponding lag, I calculate the fundamental frequency by dividing the sampling frequency by the peak lag, which represents the fundamental period:

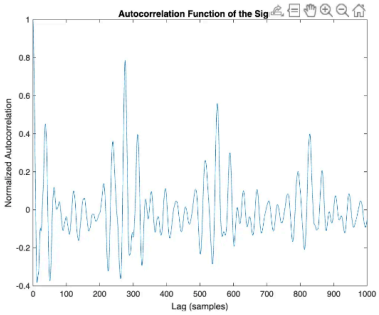
$$F_0 = \frac{f_s}{T_0}$$

Where:

- F_0 is the fundamental frequency in Hz
- f_s is the sampling frequency in Hz
- T_0 is the period of the fundamental frequency in samples



Power spectrum of `hod_m.wav` [1]

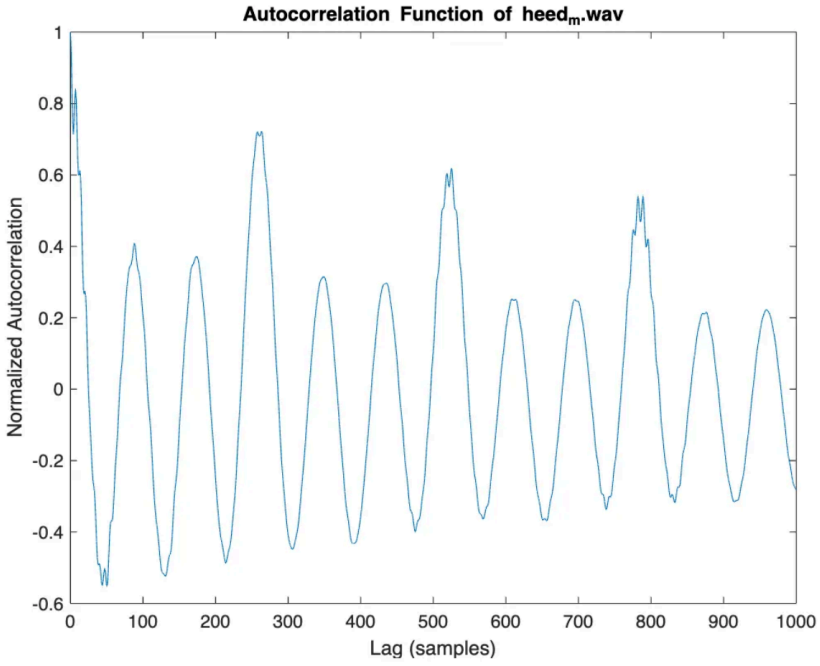


Autocorrelation vs lag for `hod_m.wav` [1]

The estimate of F_0 by autocorrelation yields **86.96 Hz** for `hod_m.wav`, compared to the built-in `pitch` function's result of **87.08 Hz**. This close match, along with the power spectrum shown above, corroborates my estimation.

Challenges and Resolutions

However, background noise or signal artifacts can distort the autocorrelation function, leading to an incorrect estimate. In the `heed_m.wav` example below, we've detected a false peak at a lag of **7 samples**. This leads to the incorrect estimate of **3428.57Hz**.

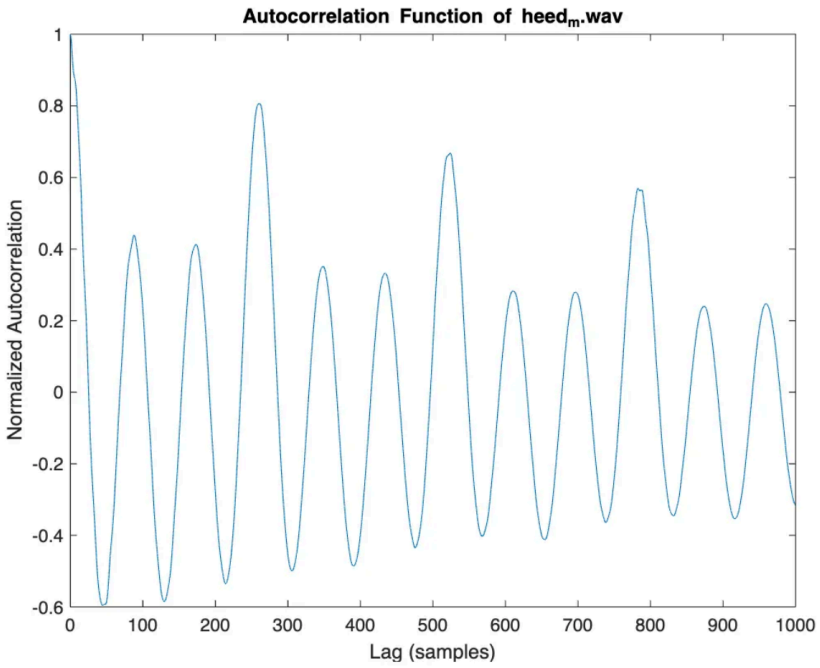


False peaks at around 7 samples [1]

This issue can be mitigated by smoothing the segment before performing the autocorrelation. To eliminate false peaks, I applied a moving average filter with a window size of 5. This resulted in a more stable and reliable autocorrelation output.

```
function [f0_estimate] = estimateF0ByAutoCorrelation(segment, fs, filename)
    segment = movmean(segment, 5);
    [autocorrValues, lags] = xcorr(segment, 'coeff');
    % rest of the code same as above
```

The effect of smoothing is evident in the results below. The estimated F_0 is now **91.25 Hz**, which aligns much more closely with the pitch function's result of **90.91 Hz**.



Smoothened autocorrelation curve [1]

To further prevent the detection of false peaks, we can set additional parameters in the `findpeaks` function. The three main parameters being

- `MinPeakHeight`: Sets the minimum value a peak must have to be considered. This essentially filters out smaller peaks that could be due to noise. I've set it to `0.3` of the maximum value of the autocorrelation.
- `MinPeakDistance`: Specifies the minimum number of samples between peaks. It prevents multiple close peaks from being considered as separate. This corresponds to the expected fundamental period in samples. In this case I set it to `fs/90` because males typically have a fundamental frequency between **90 to 155 Hz.**
- `MinPeakProminence`: Measures how much a peak stands out from its surrounding peaks. I've chosen `0.2`.

```
findpeaks(autocorrValues,
          'MinPeakHeight', 0.3,
          'MinPeakDistance', fs / 90,
          'MinPeakProminence', 0.2);
```

Alternatives for F_0 Estimations

Autocorrelation is particularly effective for voiced speech, as the periodic nature of these sounds produces a strong autocorrelation peak. However, **I have noticed in the case of a voiced signal with strong formants, it doesn't perform that well.** While I won't delve into the implementations, I'll briefly outline some alternative methods for F_0 estimation.

We can use **inverse filtering** and **residual analysis** as covered in lecture 3 [6]. Essentially, we create a filter that inverts the estimated vocal tract response and apply it to the original signal, isolating the residual signal. **This residual signal gives the underlying excitation that contains the periodicity of the speech**.

A Cepstrum-based F_0 estimation can also be effective in noisy environments. The log-magnitude spectrum efficiently **separates pitch information from formant structures**, with the added benefit of smoothing out many high-frequency noise components.

Formant Analysis and Pole Zero Plots

The filter's poles can be determined by finding the roots of the polynomial formed by the LPC coefficients. We then retain only the poles inside the unit circle, excluding purely real poles. To obtain the formant frequencies, we convert the angles of these poles to frequencies in Hz using the formula

$$f = \frac{\theta \cdot f_s}{2\pi}$$

Where:

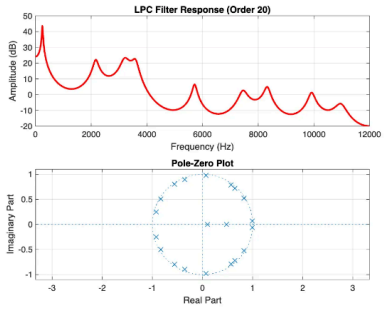
- f is the formant frequency in Hz
- θ is the angle of the pole in radians
- f_s is the sampling frequency in Hz

In formant analysis, we ignore poles on the real axis because they don't create specific frequency resonances or formants. Resonance requires oscillations, which need both magnitude and phase components. Real poles only contribute to general amplitude shaping and damping, without producing specific resonances.

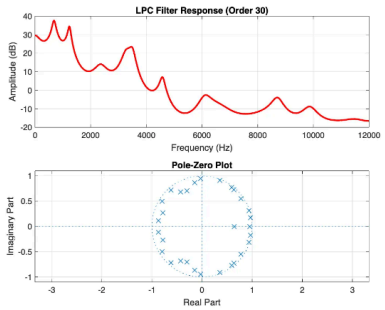
```
function [formants] = estimateFormants(segment, fs, lpcOrder)
    lpcCoeffs = lpc(segment, lpcOrder);
    poles = roots(lpcCoeffs);
    poles = poles(abs(poles) < 1 & imag(poles) ~= 0);
    angles = angle(poles);
    frequencies = abs(angles * (fs / (2 * pi)));

    uniqueFrequencies = unique(sort(frequencies));
    formants = uniqueFrequencies(1:min(3, length(uniqueFrequencies)));  % First three fo
    fprintf('Estimated Formant Frequencies (Hz):\n');
    for i = 1:length(formants)
        fprintf('F%d: %.2f Hz\n', i, formants(i));
    end
end
```

To visualize the poles and spectral peaks, I've created a pole-zero plot on the z-plane. Poles, which are points where the filter's transfer function approaches infinity in magnitude, correspond to the spectral peaks.

```
function plotLPCPoleZero(segment, fs, lpcOrder)
    lpcCoeffs = lpc(segment, lpcOrder);
    [h, f] = freqz(1, lpcCoeffs, 1024, fs);

    % Plot frequency response (Amplitude Spectrum)
    figure;
    subplot(2, 1, 1);
    plot(f, mag2db(abs(h)), 'r', 'LineWidth', 2);
    title(['LPC Filter Response (Order ', num2str(lpcOrder), ')']);
    xlabel('Frequency (Hz)');
    ylabel('Amplitude (dB)');
    grid on;

    % Pole-Zero Plot
    subplot(2, 1, 2);
    zplane(roots(1), roots(lpcCoeffs));
    title('Pole-Zero Plot');
    xlabel('Real Part');
    ylabel('Imaginary Part');
    grid on;
end
```



Frequency Response and the Pole-Zero Plot of `hod_m.wav` with an LPC order of 20. [1]



Frequency Response and the Pole-Zero Plot of `hod_m.wav` with an LPC order of 30. [1]

Prominent poles with strong resonance appear as **peaks in the frequency response** (magnitude spectrum) at frequencies corresponding to the pole angles. The closer these poles are to the unit circle, the **stronger the resonance**. In the left plot, we see **20 poles**, or **10 conjugate pairs**, corresponding to the 10 visible peaks. The rightmost conjugate pair, located in the first and fourth quadrants, exhibits the strongest magnitude—nearly touching the unit circle. On the other hand, the leftmost conjugate pair, located in the second and third quadrants, doesn't show as strong a magnitude.

It's also important to note the conjugate symmetry of the poles across the real axis. Each conjugate pair contributes a single peak at the frequency specified by the angle θ. The height and sharpness of this peak are determined by the poles' proximity to the unit circle.

Using this information, our estimate formant frequencies for `hod_m.wav` and `heed_f.wav` using an order of 25 are

```
Estimated Formnts for hod_m.wav:
F1: 671.18 Hz
F2: 1245.79 Hz
F3: 2713.61 Hz
```

```
Estimated Formnts for heed_f.wav:
F1: 370.81 Hz
F2: 2685.17 Hz
F3: 3261.04 Hz
```

Comparing these results to the reference values in the table above:

```
hod_m: 730 Hz, 1090 Hz, and 2440 Hz

heed_f: 310 Hz, 2790 Hz, and 3310 Hz
```

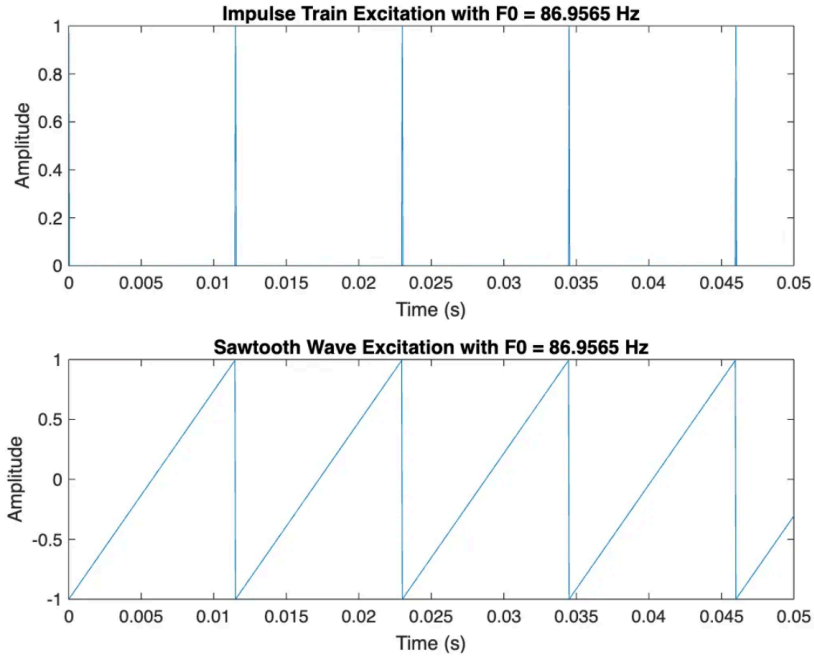
our estimates appear to be reasonably accurate.

Monophthong Synthesis

Having estimated the fundamental frequency and obtained the LPC coefficients, we can now proceed to generate our glottal excitation signal. I have generated two versions of synthesized speech: one using a periodic impulse train and another using a sawtooth signal. Both versions are based on the fundamental frequency of `hod_m.wav`.

```
    T0 = round(fs / f0);
    numSamples = round(duration * fs);

    impulseSignal = zeros(1, numSamples);
    impulseSignal(1:T0:end) = 1;
```

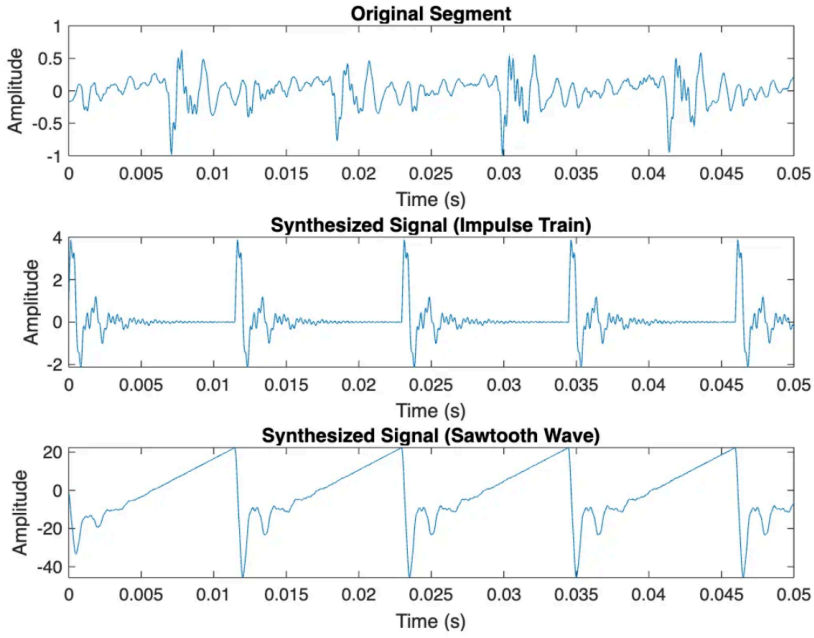
```
    t = (0:numSamples-1) / fs;
    sawtoothSignal = sawtooth(2 * pi * f0 * t);
```



Excitation signals [1]

And after passing the impulse through the LPC filter, I got the following results.

```
synthesizedImpulse = filter(1, lpcCoeffs, impulseSignal);
```

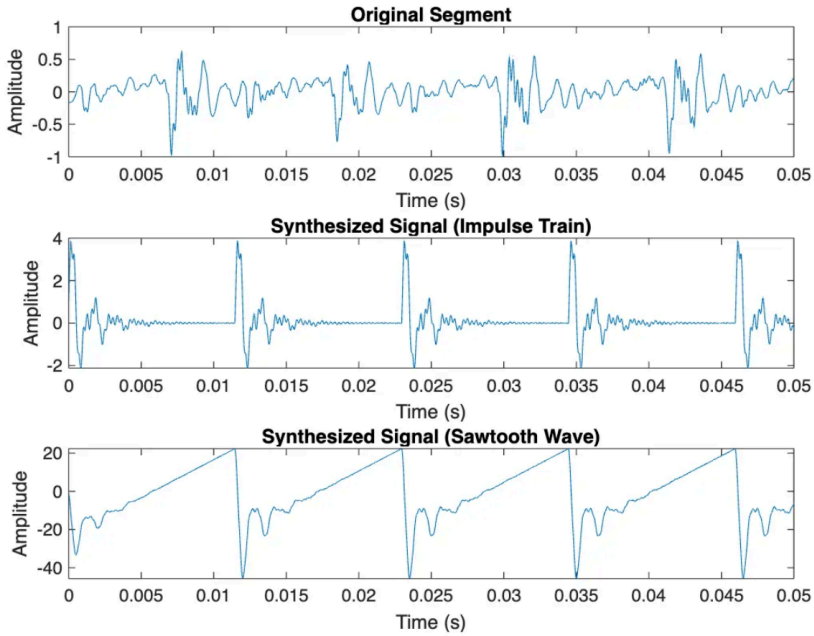


Synthesized signals vs. original signal, not normalized. [1]

Note that **the gain of the synthesized signal is significantly higher**. This occurs because the filter's resonant peaks amplify certain frequencies in the impulse train. To control the synthesized signal's amplitude, we can either normalize the output at the end or adjust the input impulse train's amplitude. I've chosen the first approach, normalizing by the original segment's peak amplitude as seen below.

```
originalPeak = max(abs(segment));
synthesizedImpulse = filter(1, lpcCoeffs, impulseSignal);
synthesizedImpulsePeak = max(abs(synthesizedImpulse));
synthesizedImpulse = synthesizedImpulse * (originalPeak/synthesizedImpulsePeak);


synthesizedSawtooth = filter(1, lpcCoeffs, sawtoothSignal);
synthesizedSawtoothPeak = max(abs(synthesizedSawtooth));
synthesizedSawtooth = synthesizedSawtooth * (originalPeak/synthesizedSawtoothPeak);
```

The resulting synthesized signal's perceived loudness matches that of the original signal much better.



Synthesized signals vs. original signal, not normalized. [1]

Interactive Audio Playground

The image below displays a PDF screenshot of the interactive audio playground. You can find interactive audio samples in the toggle sections on the website. Additionally, you can access the audio files in the `synthesized_sawtooth` and `synthesized_impulse` folders.

▼ hod

Male (Impulse)

0:00 / 0:00

Female (Impulse)

0:00 / 0:00

Male (Sawtooth)

0:00 / 0:00

Female (Sawtooth)

0:00 / 0:00

▼ heed

Male (Impulse)

0:00 / 0:00

Female (Impulse)

0:00 / 0:00

Male (Sawtooth)

0:00 / 0:00

Female (Sawtooth)

0:00 / 0:00

▼ hood

Male

0:00 / 0:00

Female

0:00 / 0:00

Male (Sawtooth)

0:00 / 0:00

Female (Sawtooth)

0:00 / 0:00

▼ head

Male

0:00 / 0:00

Female

0:00 / 0:00

Male (Sawtooth)

0:00 / 0:00

Female (Sawtooth)

0:00 / 0:00

Synthesized Speech Quality Analysis

While an impulse train-LPC combination is a useful approximation for basic voiced sounds, it is limited in replicating the natural variability of human speech. The produced audio samples, as heard above, sound relatively **monotone and robotic** due to the regular, unchanging excitation. Nevertheless, they are clear and intelligible enough to be effective. I can see this technique being useful for simple text-to-speech systems where clarity is more critical than natural sound quality.

I attempted to introduce some **timbre** to the synthesized speech by generating a sawtooth signal. Using the playground above, you should be able to hear **a richer harmonic content** and a "**brassier**" sound. The continuous nature of the sawtooth wave gives the synthesized signal a **smoother waveform** as opposed to the jagged appearance of the synthesized impulse train.

Potential Improvements

As covered in lecture 3 [6], **LPC is fundamentally designed to model the vocal tract as a resonant filter**. However, **it struggles to capture noise-like fricatives**—a limitation of the all-pole model. To address this, we could employ **Cepstrum analysis** to separate the excitation source (which shapes pitch and harmonics) from the vocal tract (which shapes formants). By using **liftering** for envelope extraction, we could potentially retain the noise-like characteristics of unvoiced sounds. However, this approach is beyond the scope of this assignment.

Another notable observation is that **a higher order of LPC typically yields better results for female voices**. This aligns with our intuitive understanding, as female voices often exhibit formants with wider bandwidths and slightly higher frequencies compared to male voices. To enhance the fidelity of LPC estimation, **we could implement a dynamic LPC order that adjusts based on gender**, allowing for a more precise match between the LPC order and the complexity of the signal.

References

[1] **Code** by *Frank Lü* , University of Surrey

https://github.com/frankcholula/sapr

[2] **Spectrogram Plotting** by *Julius Orion Smith III,* Stanford University CCRMA

Spectrogram of Speech
https://ccrma.stanford.edu/~jos/st/Spectrogram_Speech.html

[3] **Window Functions in Spectrum Analyzers** by *Tecktronixs*

Window Functions in Spectrum Analyzers
In this post, I'm going to dive into windowing, a spectrum analysis fine-tune setting that you may occasionally need.
https://www.tek.com/en/blog/window-functions-spectrum-analyzers

[4] **Speech Acoustics** by *Barry Truax,* Simon Fraser University Sonic Research Studio

Speech Acoustics
A) The acoustics of speech
production: vowels and consonants
https://www.sfu.ca/sonic-studio-webdav/cmns/Handbook%20Tutorial/SpeechAcoustics.html

[5] **General Phonetics** by *Louis Goldstein,* University of Southern California SAIL

General Phonetics Home Page
https://sail.usc.edu/~lgoldste/General_Phonetics/

[6] **Speech and Audio Processing Lecture 3** by *Wenwu Wang*, University of Surrey CVSSP