EE 779: Computing Assignment 2
Linear Prediction Analysis

Kalpesh Patil (130040019)

Problem A: Synthesized vowel

Vowel /a/ (formants: 730, 1090, 2440 Hz; bandwidths: 50 Hz) at two fundamental frequencies (120 Hz, 300 Hz) was produced. Levinson recursion was implemented for LP analysis. Following diagram shows Levinson algorithm outline.

$a_0(0) = 1$ and $\epsilon_0 = r_x(0)$

For $j = 0, \ldots, p-1$

1. $\gamma_j = r_x(j+1) + \sum_{i=1}^{j} a_j(i) r_x(j+1-i)$
2. $\Gamma_{j+1} = -\frac{\gamma_j}{\epsilon_j}$
3. For $i = 1, \ldots, j$,

$$a_{j+1}(i) = a_j(i) + \Gamma_{j+1} a_j(j-i+1)$$

4. $a_{j+1}(j+1) = \Gamma_{j+1}$
5. $\epsilon_{j+1} = \epsilon_j(1 - |\Gamma_{j+1}|^2)$

$b(0) = \sqrt{\epsilon_p}$

(src: EE 779 lcture slides)

- Code

```
clear all

//Function to calculate filter response using difference equation
function [y]=time_response(x, num, den)
   n_samples = length(x);
   y = zeros(n_samples,1)
   //numerator is constant (all pole filter)
   y(1) = num(1)*x(1)
   k = num(1)
   //response by taking coefficients for denominator

   for ii =2:n_samples
      temp = k*x(ii)
      for jj = 1:min(ii-1,length(den)-1)
         temp = temp - den(jj+1)*y(ii - jj)
      end
      y(ii) = temp
   end
endfunction
```

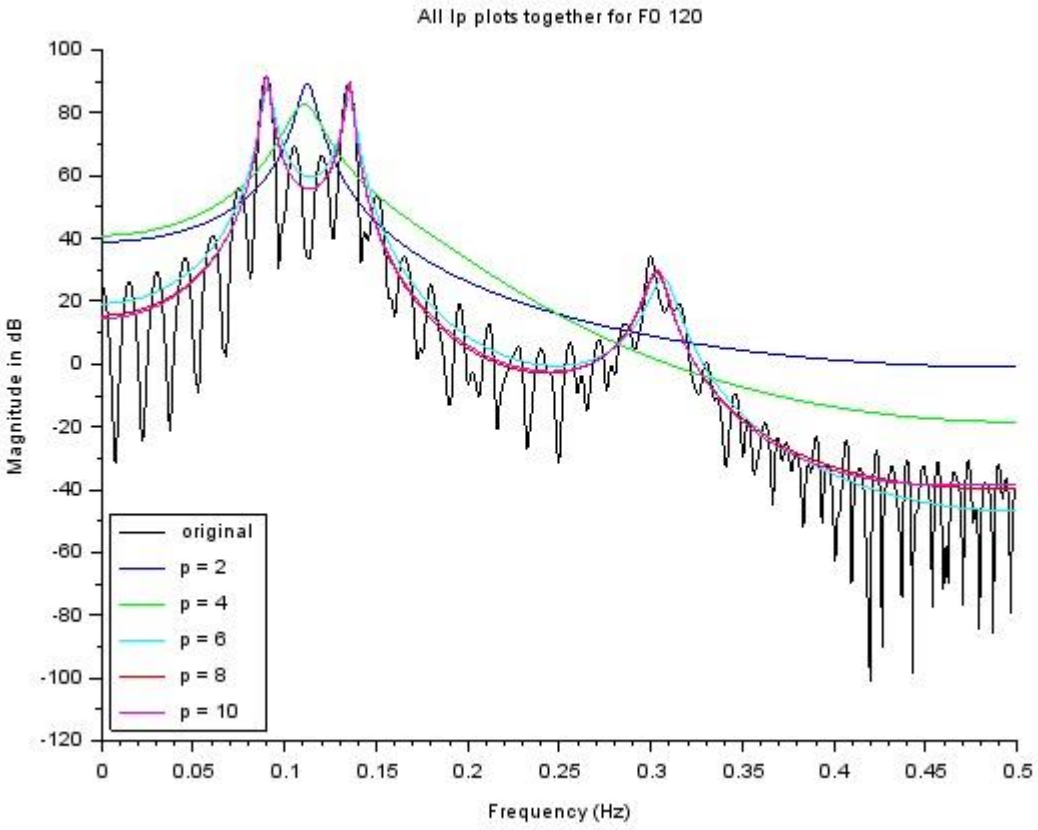
```
//function to find discrete coefficients of cascade of multiple filters
function [num, den]=find_cascade_filter(F_list, B_list, Fs)
    num = 1
    den = [1]
    for iter = 1:length(F_list)
        F = F_list(iter)
        B = B_list(iter)
        //finding poles for current iteration
        r = exp(-B*%pi/Fs)
        theta = 2*%pi*F/Fs
        //current filter
        num_curr = 1
        den_curr = [1 -2*r*cos(theta) r^2]
        //multiplication of polynomials can be computed using their
        //convolution. Numerator is constant(1). Hence only
        //denominator needs to be multiplied
        den = conv(den,den_curr)
    end
endfunction

//produce vowel /a/ for given F0
function a=produce_vowel(F0)
    vowel_formant_list = [730, 1090, 2440]; //F1,F2,F3 for vowell a
    Fs = 8000
    t_duration = 0.5
    //numerator of filter assumed to be 1
    k = 1
    //specifying input signal
    n_samples = t_duration*Fs + 1
    n_fft = n_samples*10;
    x = zeros(n_samples,1);
    //impulse train generation
    for i = 2:t_duration*F0
        x(floor((i-1)*Fs/F0))= 1
    end
    //Create delta input for finding impulse response
    delta_n = zeros(n_samples,1)
    delta_n(1) = 1
    F_list = vowel_formant_list
    //Bandwidth is constant for all formants
    B_list = [50,50,50]
    //find output of cascade filter
    [num,den] = find_cascade_filter(F_list,B_list,Fs)
    h = time_response(delta_n,num,den)
    y = time_response(x,num,den)
    a = y
    //plot responses and save sound files
//    plot_frequency_response(y,n_fft,F0,Fs)
//    plot_y(y,F0,Fs)
endfunction

//produce vowels
Fs = 8000
F0_list = [120, 300]
```

```
for iii = 1:length(F0_list)
        //window signal near centre
        window_time = 30
        F0 = F0_list(iii)
        x = produce_vowel(F0)'
        N = window_time*Fs/1000 + 1
        n_fft = 10*N
        //applying hamming window
        x_init = (length(x)+1)/2 - (N-1)/2
        win_hamming = window('hm', N);
        y = x(x_init: x_init + N - 1).*win_hamming;
        //finding autocorrelation
        r = zeros(length(y),1)
        for ki=0:length(x)
            for ni=ki:L-1
              r(ki+1)=r(ki+1)+y(ni+1)*y(ni-ki+1);
            end
        end

        delta_n = zeros(N,1)
        delta_n(1) = 1
        h_total = []
        //Levinson Durbin recursion
        p_max = 10
        e_vec = [r(1)]
        i = 1
        k = r(2)/e_vec(1)
        a_vec = [1,k]
        e_vec = [e_vec,(1-(k)^2)*e_vec(1)]

        for i = 2:p_max
           k = r(i+1)
           for j = 1:i-1
              k = k - a_vec(j+1)*r(i-j+1)
           end
           k = k/e_vec(i)

           a_vec_old = a_vec
           a_new = k
           for j = 1:i-1
              a_vec(j+1) = a_vec_old(j+1)-k*a_vec_old(i-j+1)
           end
           a_vec = [a_vec , a_new]
           e_vec(i+1) = (1-k^2)*e_vec(i)


           //Plotting LP approximation overlapping with the original spectrum
           if (modulo(i,2)==0)
              fig = scf()
              h_padded = zeros(n_fft,1)
              h_padded(1:length(y)) = y'
              H = fftshift(fft(h_padded))
              H_mag = abs(H(n_fft/2 +1:n_fft))
              freq_array = linspace(0,Fs/2,n_fft/2)

              plot(freq_array,20*log(H_mag))
```

```
            if(h_total == [])
               h_total = H_mag
            end
            ax = gca()
            original_frequencies = [730, 1090, 2440]
            temp_d = (n_fft*original_frequencies)/Fs
            temp_d2 = zeros(length(freq_array),1)
            temp_d2(floor(temp_d)+1) = max(ax.y_ticks.locations)

            plot(freq_array,temp_d2,'k-.')
            num = sqrt(e_vec(i+1))
            den = -a_vec(2:length(a_vec))
            den = [1,den]
            [h,w] = frmag(num,den,n_fft)
            [h2,w2] = frmag(num,den,n_fft/2)
            h_total = [h_total,h2']
            plot(w*Fs,20*log(abs(h)),'red')
            plot_title = strcat(['LP approximation of vowel a for F0 = ',string(F0),' p = ',string(i)])
            xtitle(plot_title,'Frequency (Hz)','Magnitude in dB')
            xs2jpg(gcf(), strcat(['../plots/Q1/',plot_title,'.jpg']));
         end
      end

      //plotting all LP together
      fig = scf()
      plot2d(w2,20*log(h_total),[1:(p_max/2+1)])
      plot_title = strcat(['All lp plots together for F0 ',string(F0)])
      xtitle(plot_title,'Frequency (Hz)','Magnitude in dB')
      legends(['original','p = 2','p = 4','p = 6','p = 8','p = 10'],[1:(p_max/2+1)],opt = 3)
      xs2jpg(gcf(), strcat(['../plots/Q1/',plot_title,'.jpg']));
end

xdel(winsid())
```

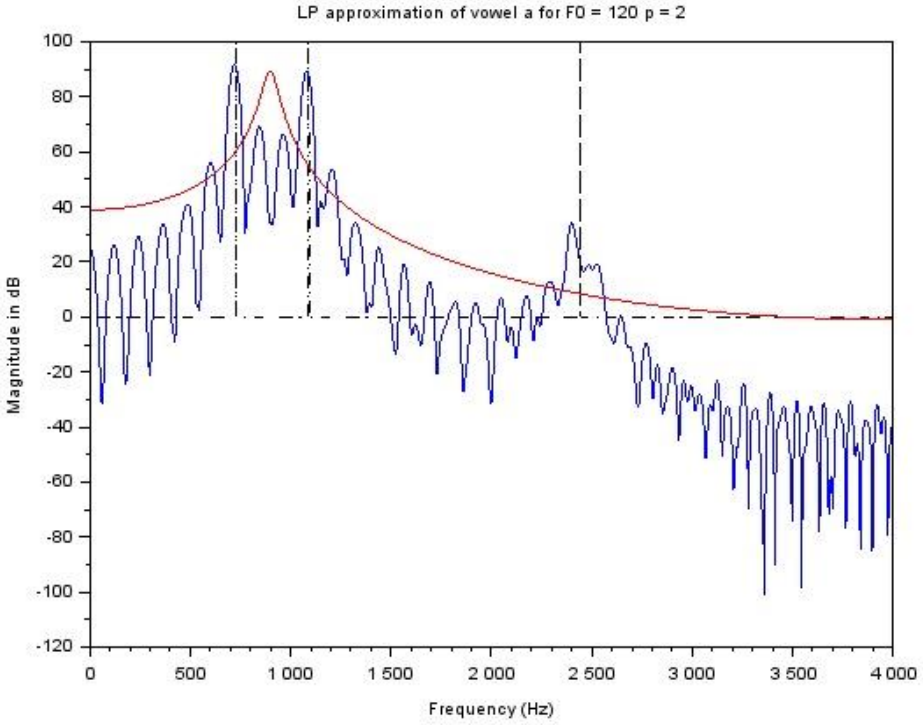
- Results & Discussion
  1. For lower values of p all the poles of the filter (formant frequencies) are not captured in the LP approximation
  2. Actual vowel sound has 6^{th} order transfer function (3 formants -> 6^{th} order). As we can observe from the combined plot of all p's together, for p >= 6, LP approximation was successful in capturing the formant frequencies.
  3. For higher values of p, LP approximation tries to envelope the signal more aggressively which results in deviation from ideal LP output for F0 = 300 Hz
  4. LP approximation is better for pitch frequency of 120 Hz than that of 300 Hz.

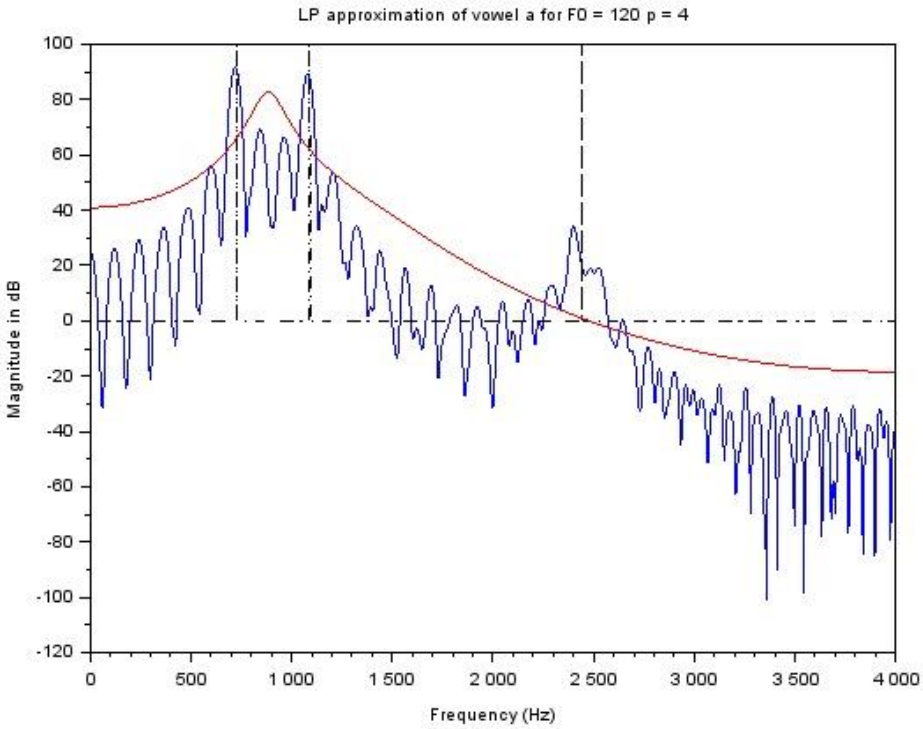
All plots together for different p's for F0 = 120 Hz

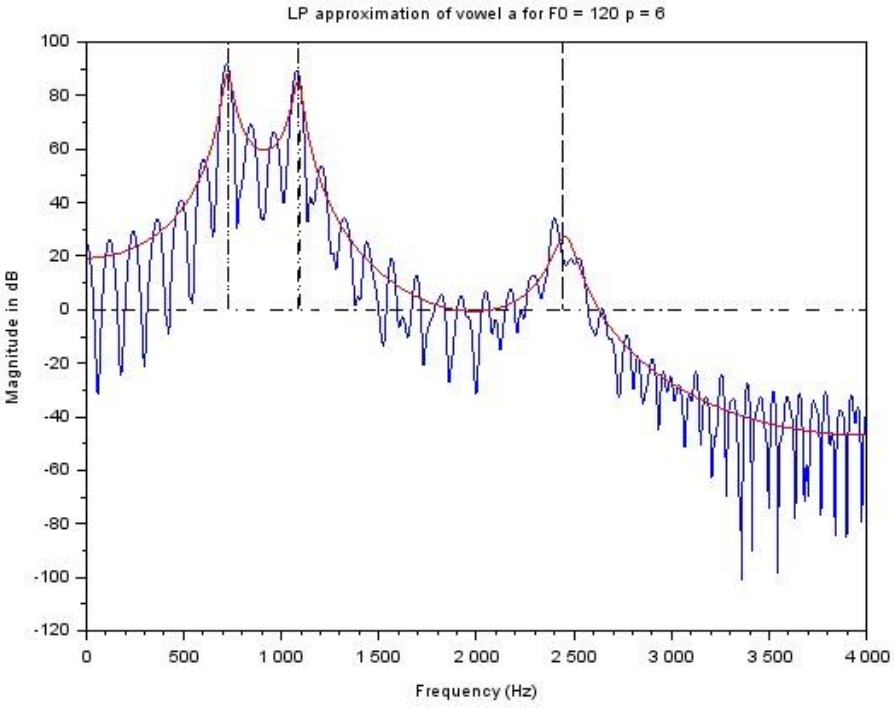


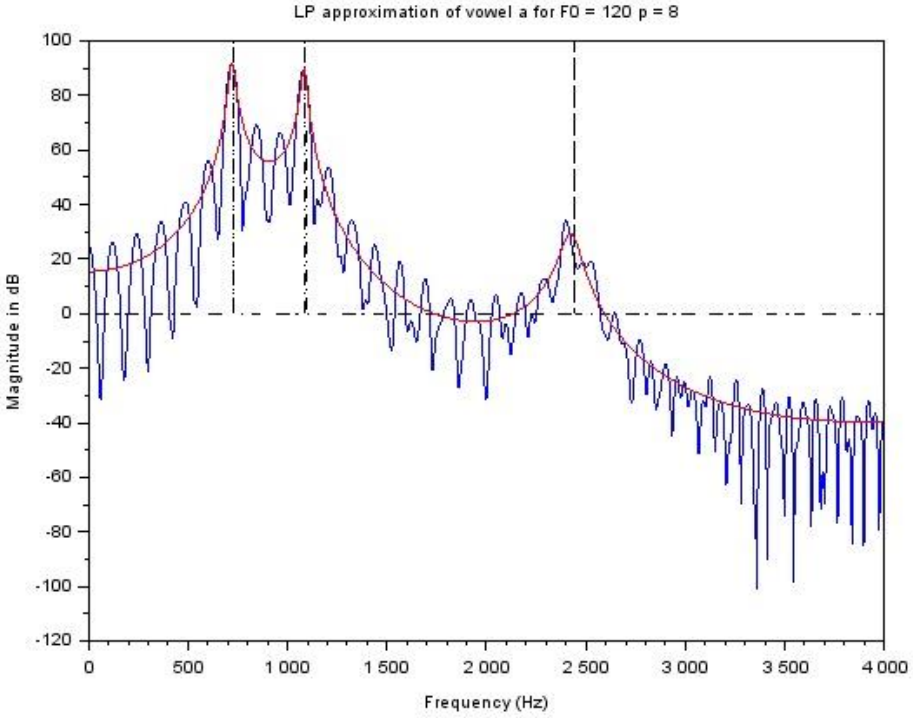
NOTE: Vertical lines depicting required discrete frequencies are shown in the individual plots for each p to avoid clutter in common plot

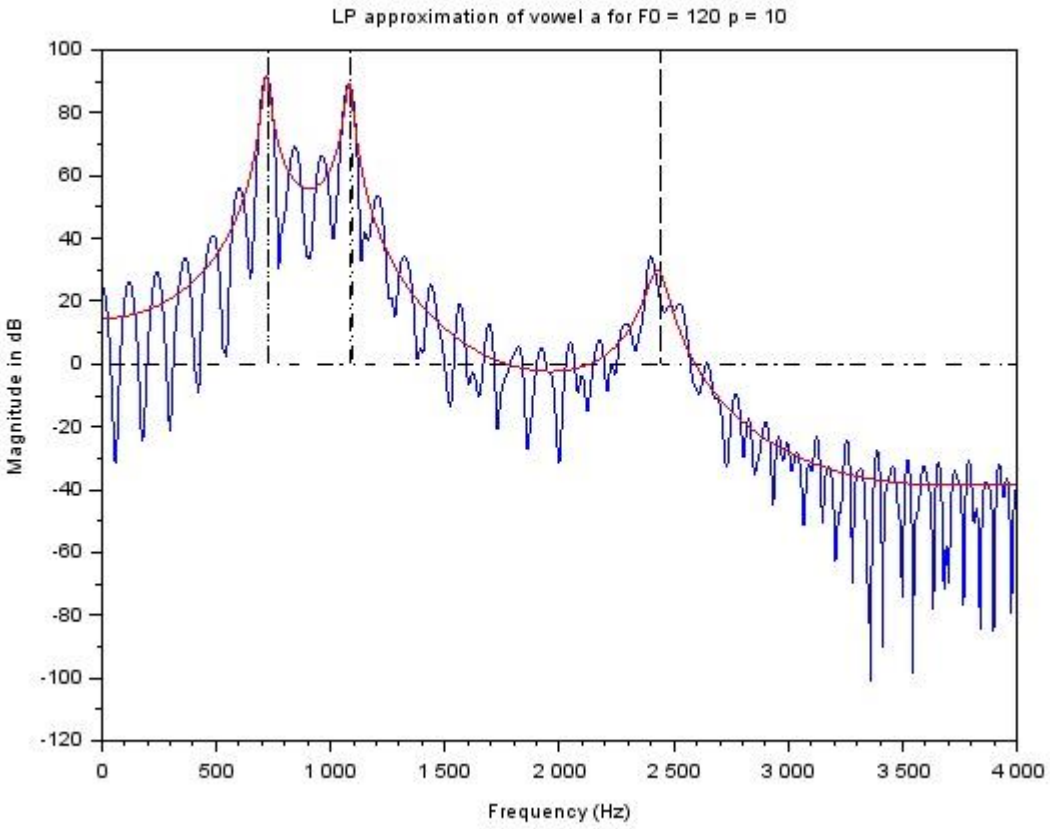
Plots for F0 = 120 Hz with different values of p



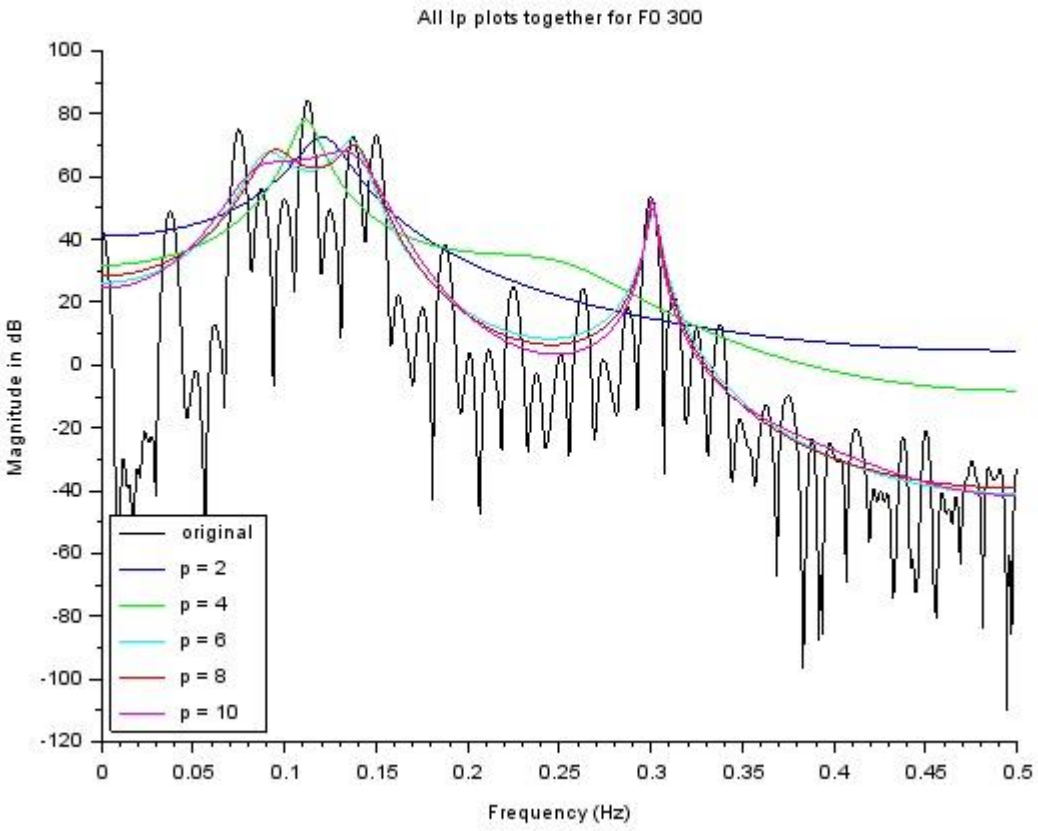



LP approximation of vowel a for F0 = 120 p = 6
Magnitude in dB
Frequency (Hz)
100
80
60
40
20
0
-20
-40
-60
-80
-100
-120
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000

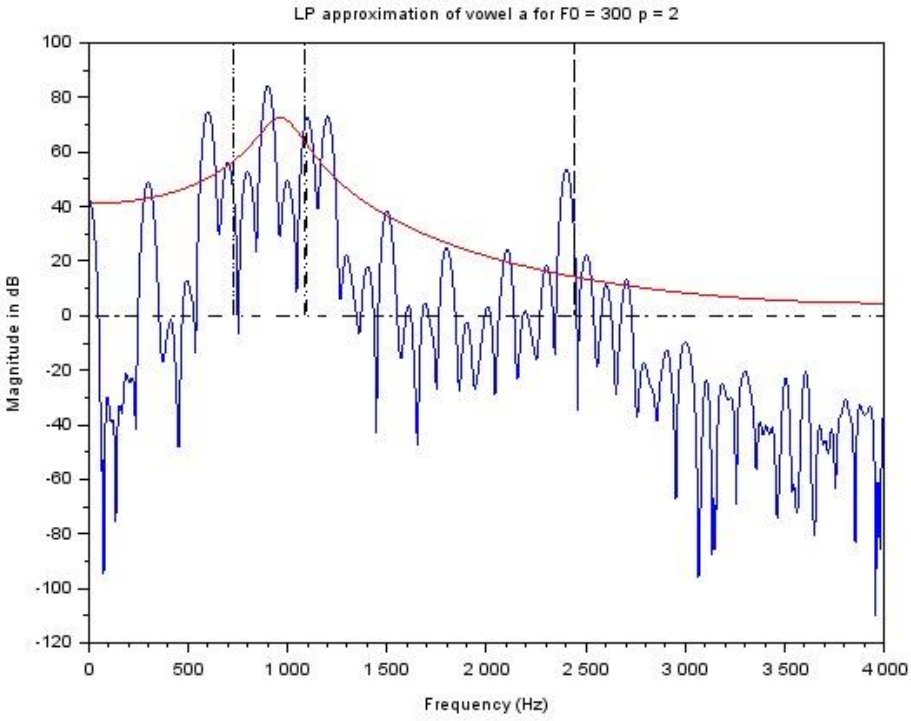

LP approximation of vowel a for F0 = 120 p = 8
Magnitude in dB
Frequency (Hz)
100
80
60
40
20
0
-20
-40
-60
-80
-100
-120
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000

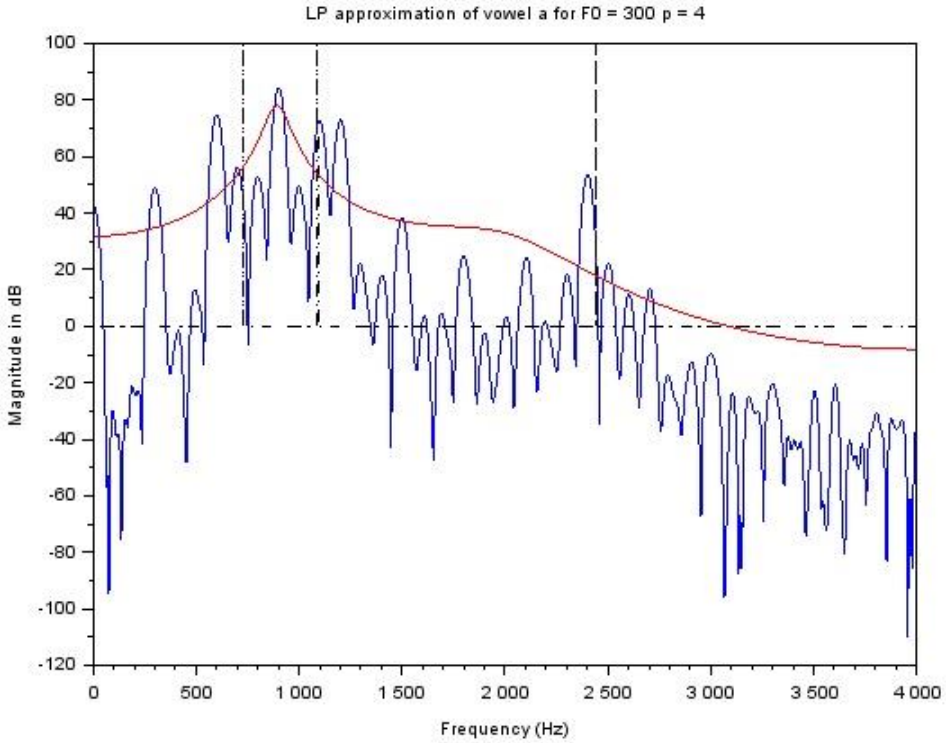

LP approximation of vowel a for F0 = 120 p = 10
Magnitude in dB
Frequency (Hz)
100
80
60
40
20
0
-20
-40
-60
-80
-100
-120
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000

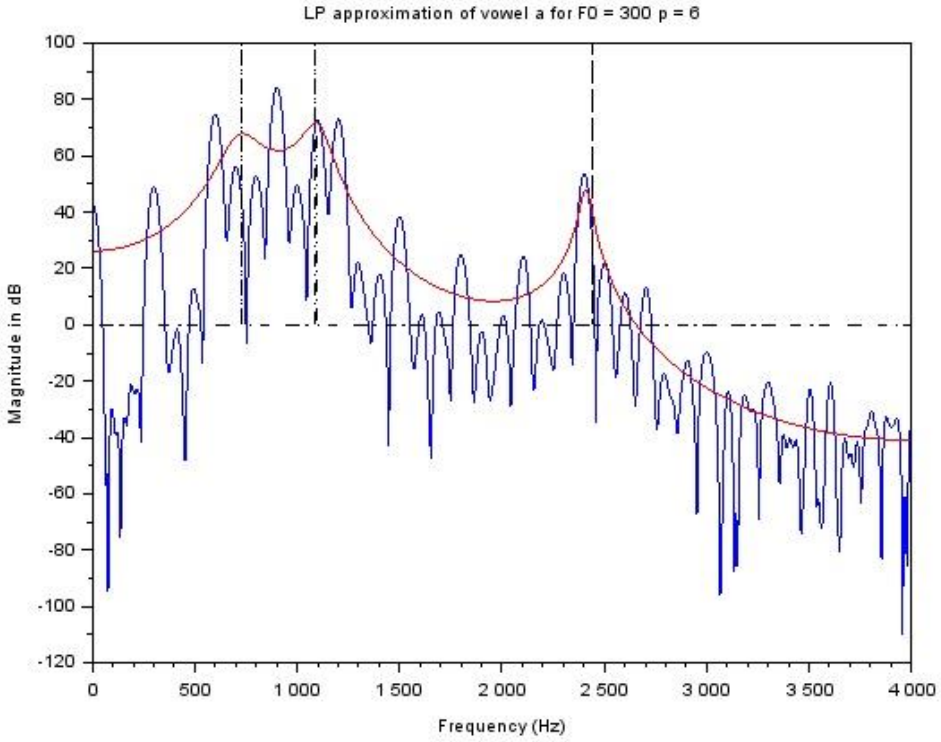
All plots together for different p's for F0 = 120 Hz

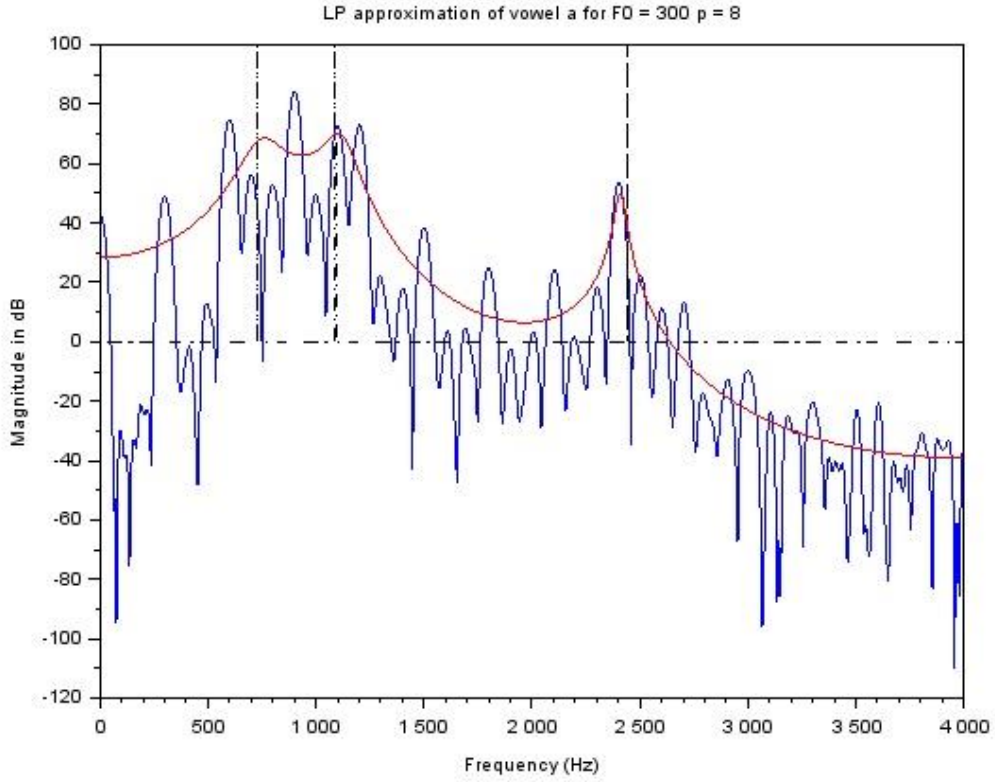


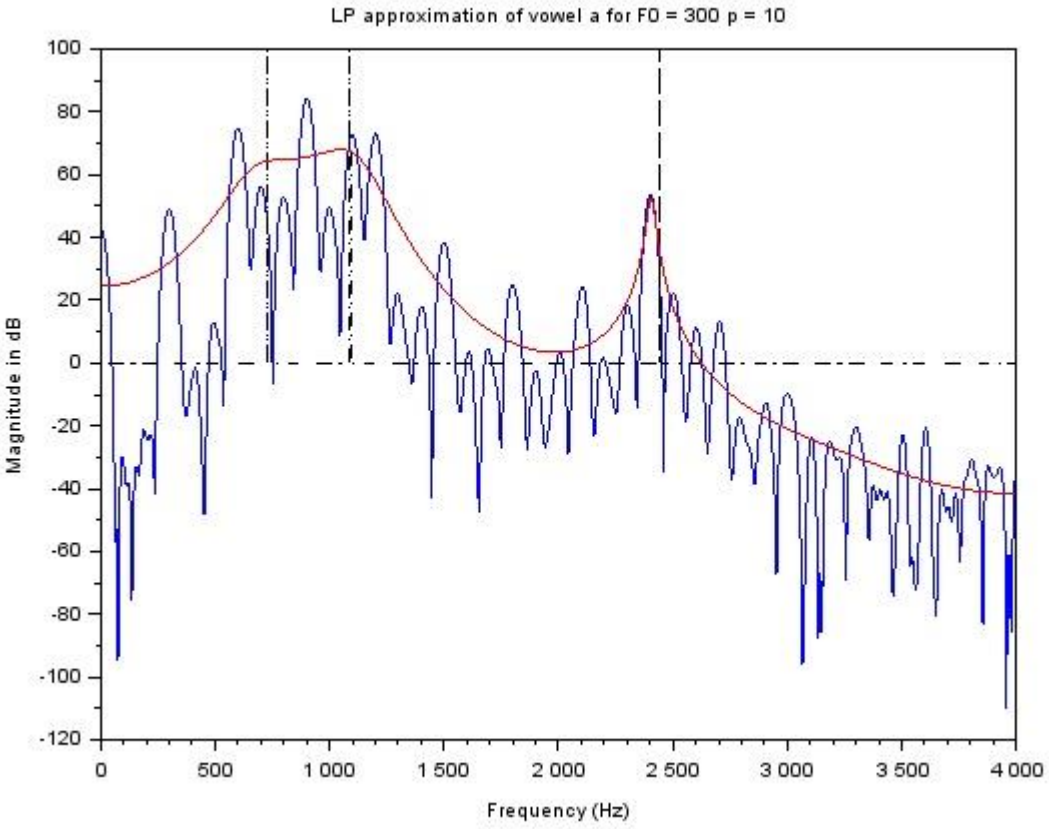
Plots for F0 = 300 Hz with different values of p






LP approximation of vowel a for F0 = 300 p = 6
Magnitude in dB
Frequency (Hz)
100
80
60
40
20
0
-20
-40
-60
-80
-100
-120
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000


LP approximation of vowel a for F0 = 300 p = 8
Magnitude in dB
Frequency (Hz)
100
80
60
40
20
0
-20
-40
-60
-80
-100
-120
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000


LP approximation of vowel a for F0 = 300 p = 10
Magnitude in dB
100
80
60
40
20
0
-20
-40
-60
-80
-100
-120
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000
Frequency (Hz)

Problem B: Natural Speech

Q1. Pre-emphasis

Following function implements pre-emphasis filter using difference equation. Note that pre-emphasis has been done only for the voiced sounds (/a/, /n/ and /i/). Unvoiced sounds (/s/) doesn't require pre-emphasis to be done

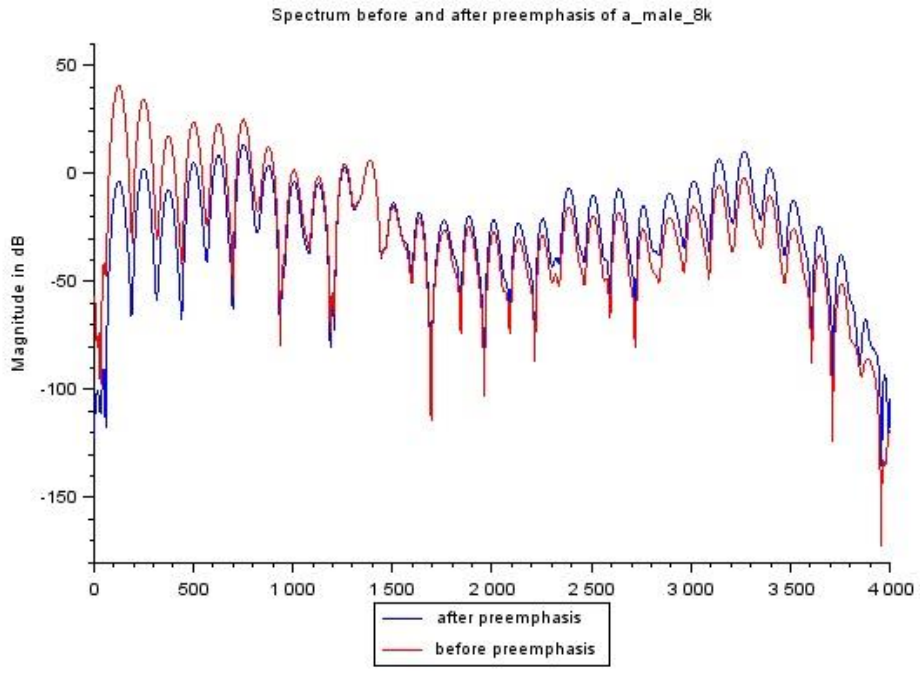
- Code

```
//pre_emphasis filter
function y=pre_emphasis_filter(x, alpha)
   y = zeros(length(x))
   y(1) = x(1)
   for i = 2:length(x)
      y(i) = x(i)-alpha*x(i-1)
   end
endfunction
```

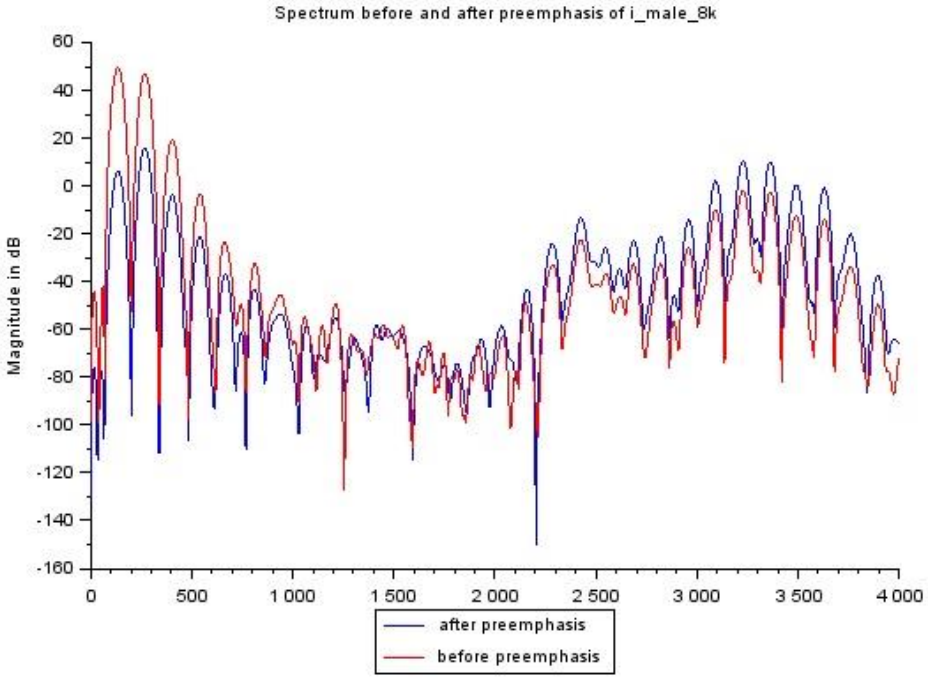
- Results & Discussion

Pre-emphasis is done compensate for the spectral roll off. Higher frequencies are boosted and lower frequencies are suppressed (high pass filter) due pre-emphasis. This phenomenon can be observed in the following graphs. Note that /s/ is an unvoiced sound and hence it is not passed through pre-emphasis filter.

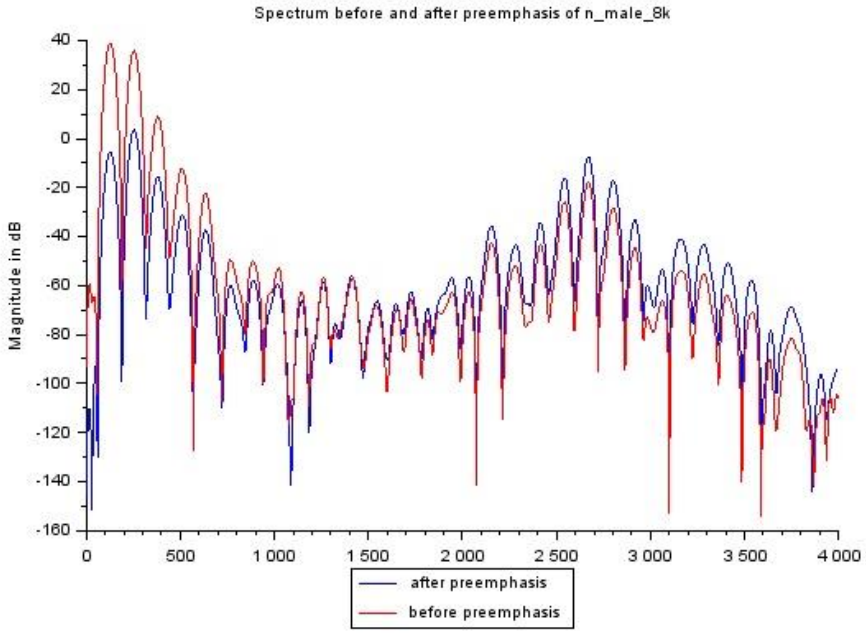
/a/ before and after pre-emphasis



/i/ before and after pre-emphasis



/n/ before and after pre-emphasis



Q2. LP analysis using Levinson method

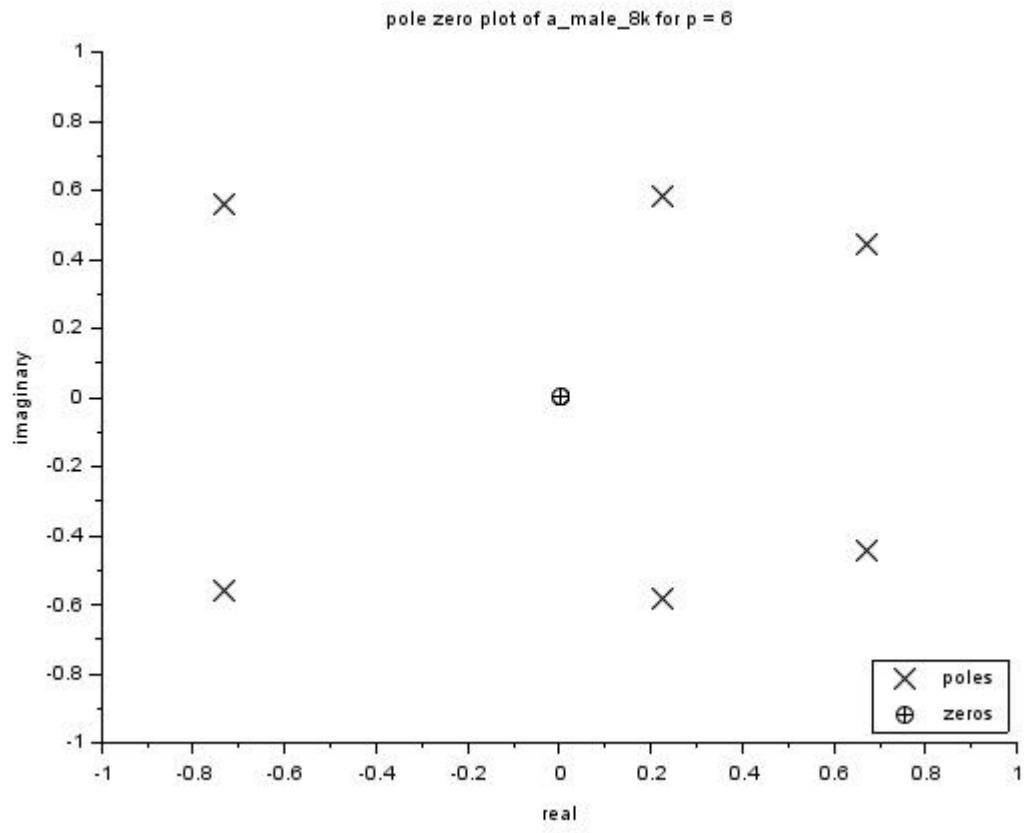
Using the Levinson algorithm described in Problem A, LP analysis was performed on windowed pre-emphasized input signal

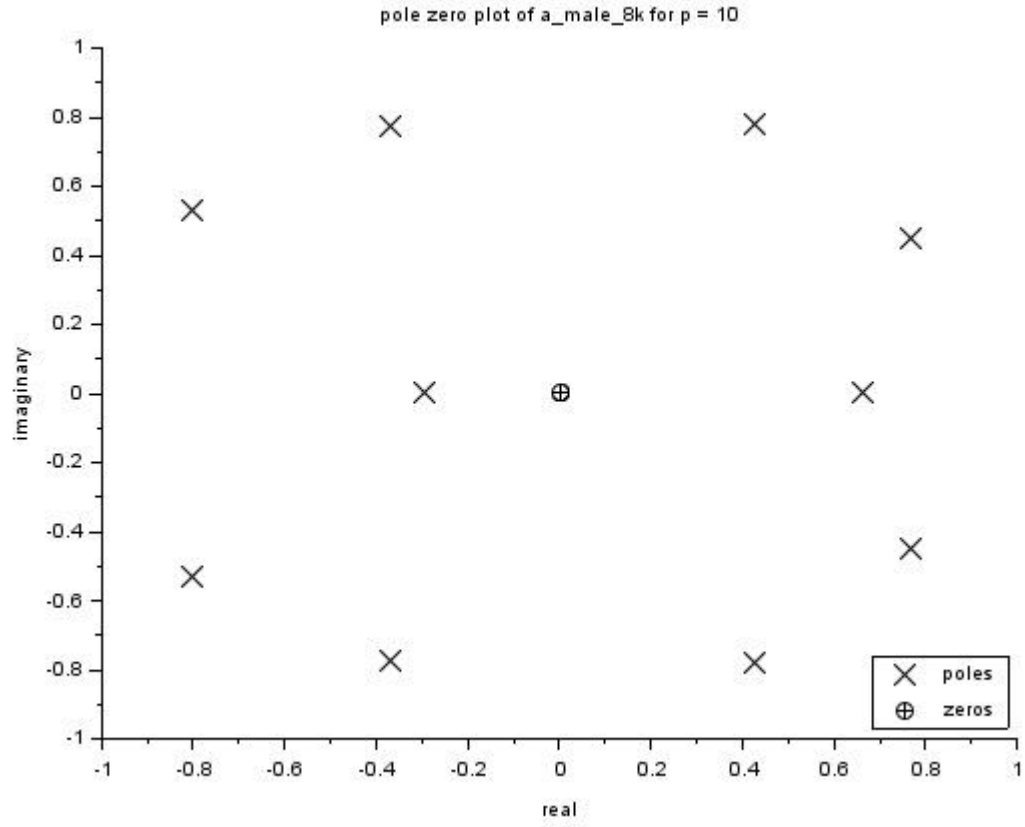
Code for Q1, Q2 and Q3 is at the end.

- Results & Discussion

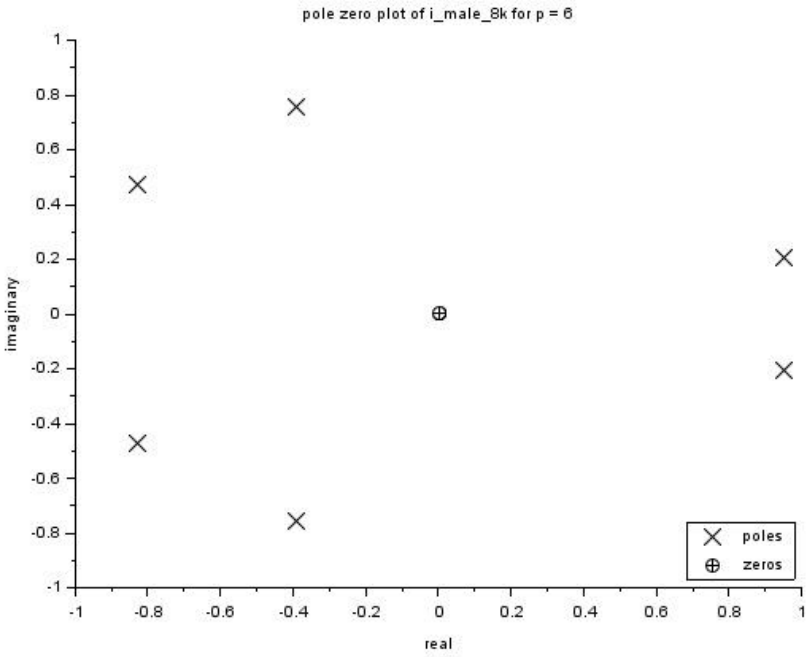
a. Pole zero plots
   Pole zero plots are plotted for given values of p. It can be noted that all poles and zeros of the LP filter obtained using autocorrelation model are inside unit circle. This is expected since autocorrelation method indeed produces a stable-causal system unlike covariance (covariance method can produce poles at any location).

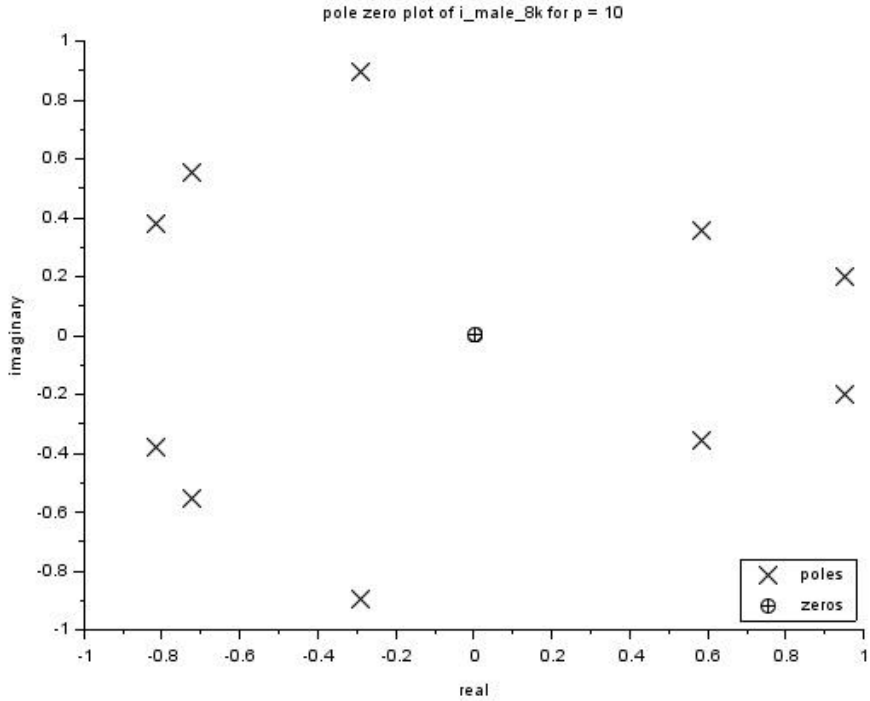
Pole zero plots for /a/ for different values of p = 6 and p = 10



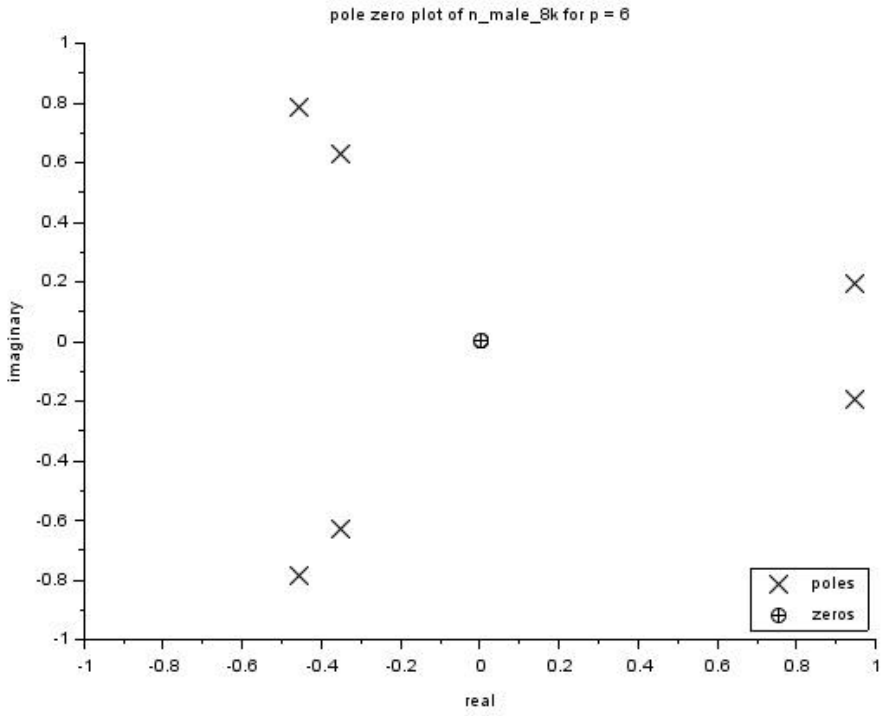

pole zero plot of a_male_8k for p = 10
imaginary
real
poles
zeros
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

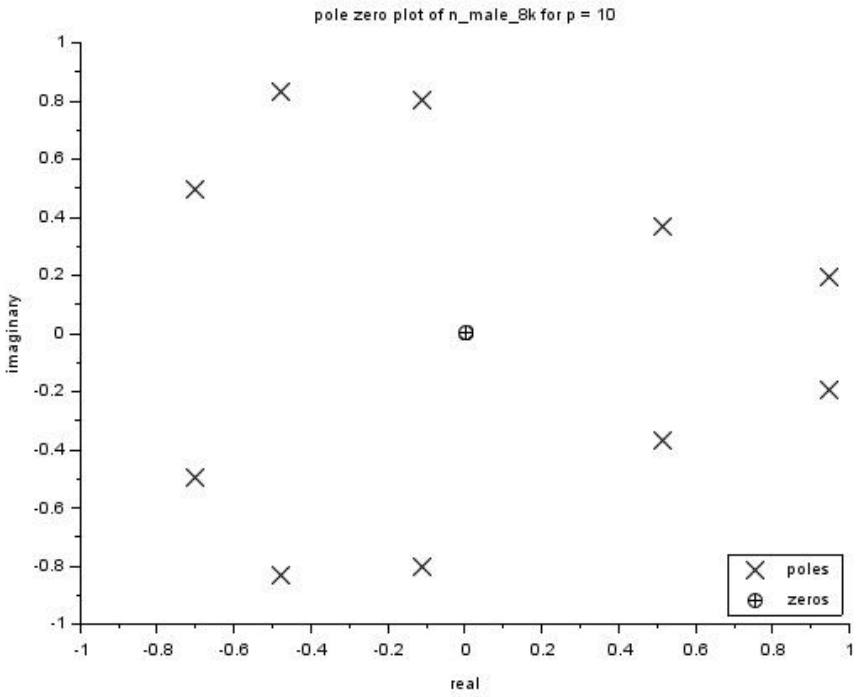
Poles zero plot for /i/ for p = 6 and p = 10



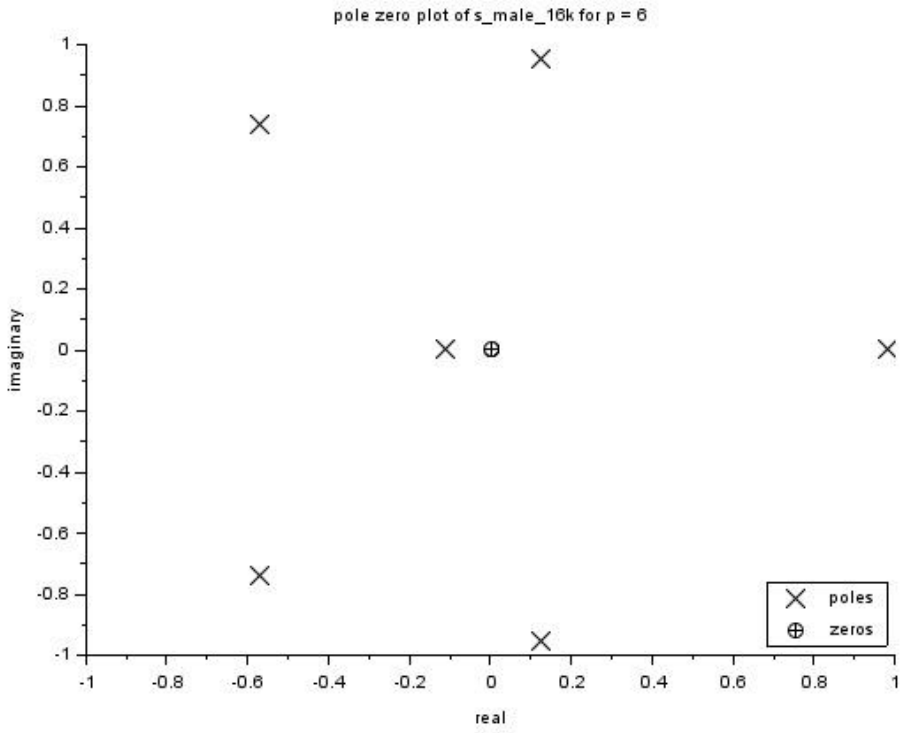


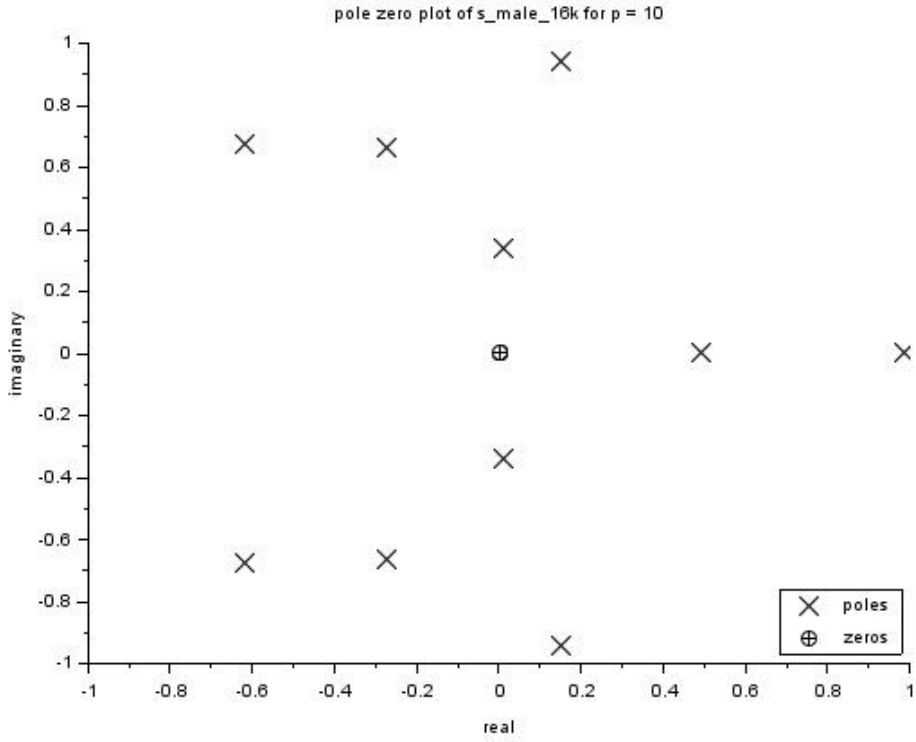
Pole zero plot for /n/ for p = 6 and p = 10





Pole zero plot for /s/ for p = 6 and p = 10

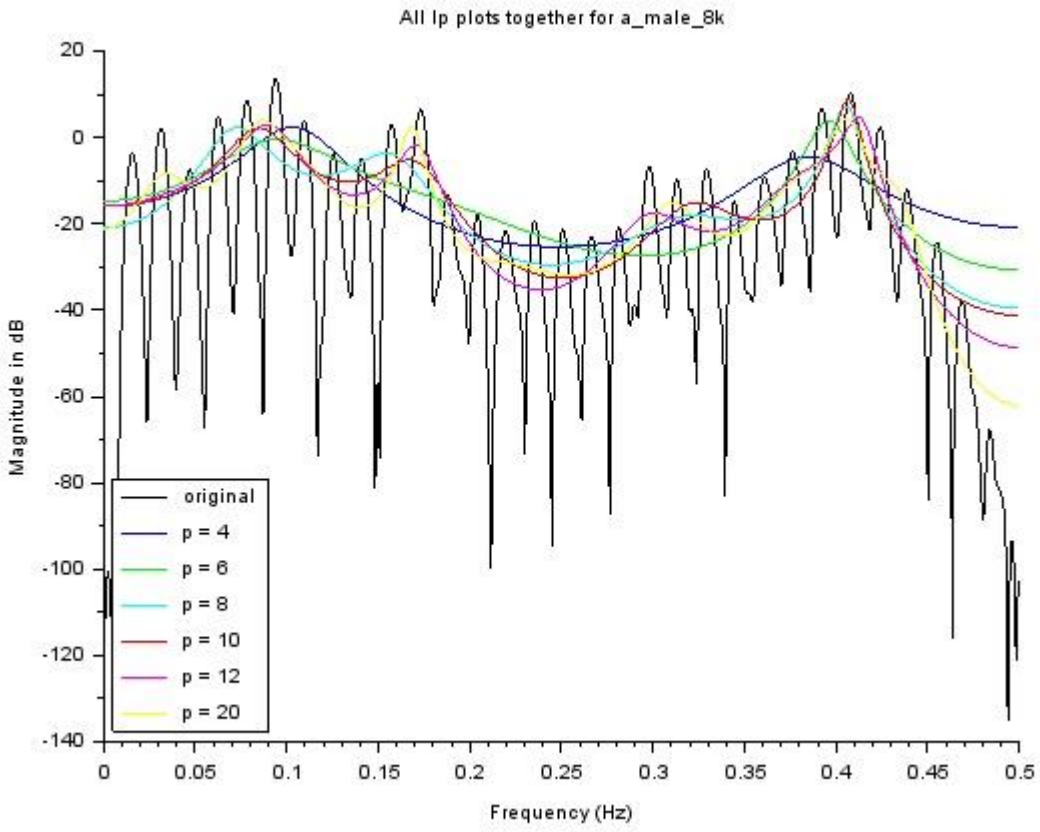




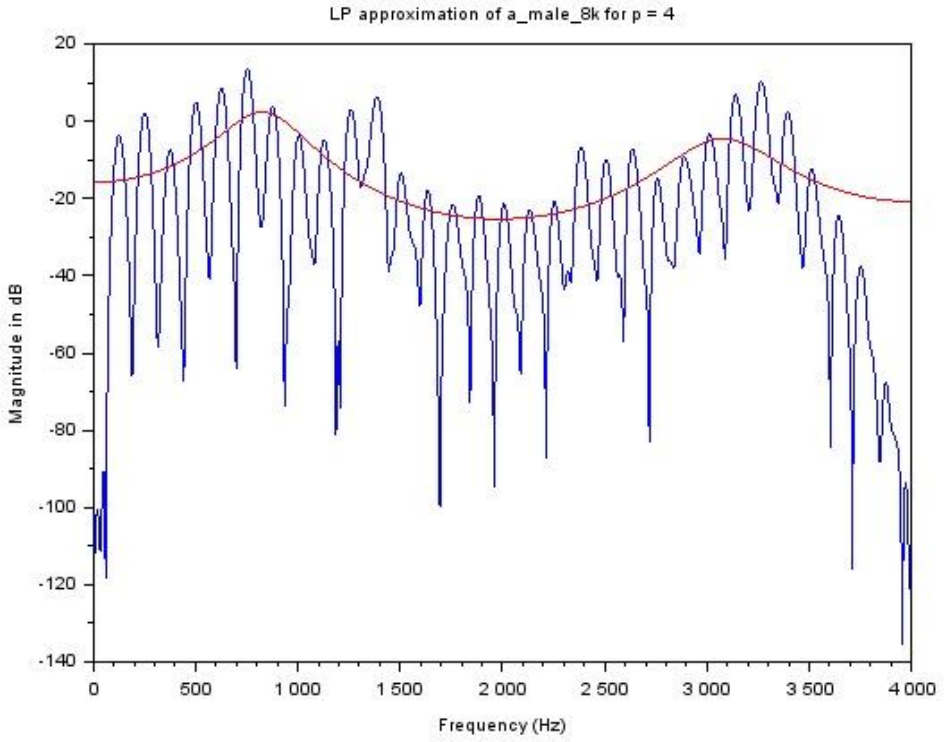
b. LPC spectrum

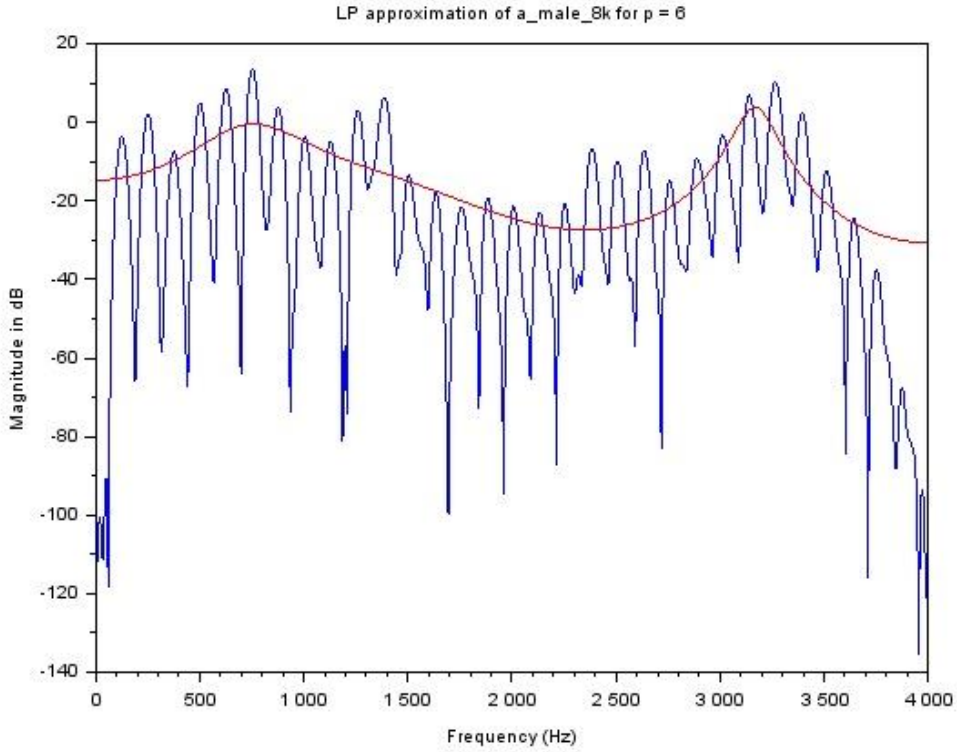
In general it can be observed that lower values of p are not able to capture all the poles while higher values of p try to envelope the signal more aggressively. Also the error due to approximation is more at the valleys in the spectrum than peaks. Vowels like /a/ and /i/ are inherently all pole transfer functions, hence they can be modelled pretty well using LP. For nasal sound /n/ higher order LP is required since it contains zeros as well. Fricatives (/s/) also require higher order LP.

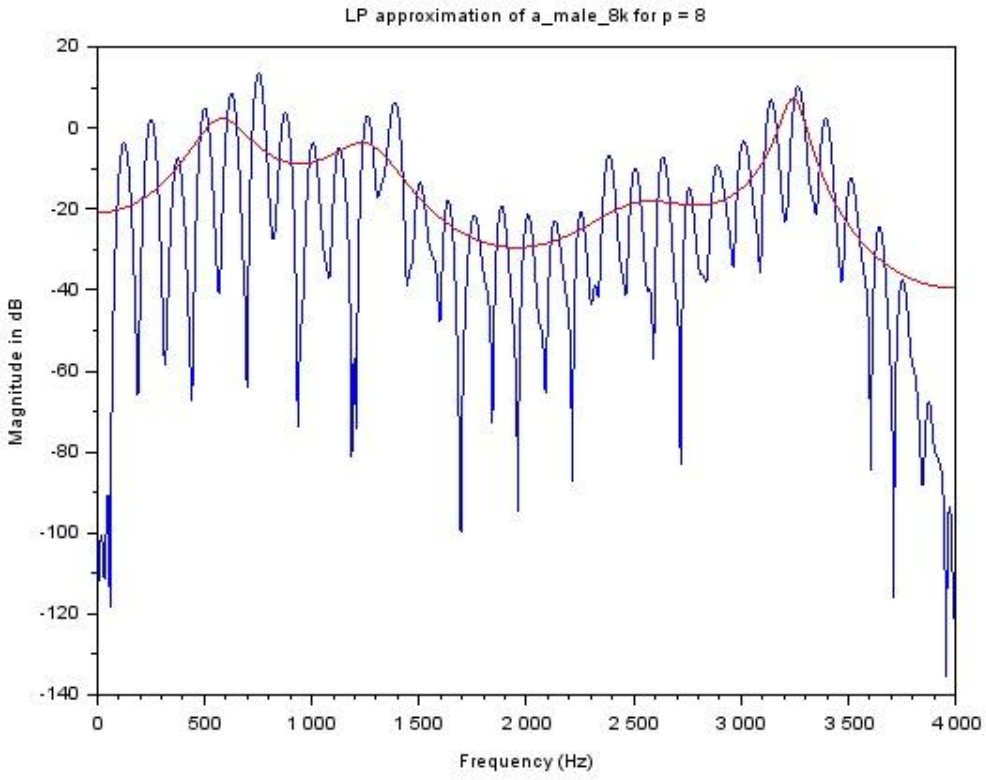
Combined LP spectrum for all p's for /a/

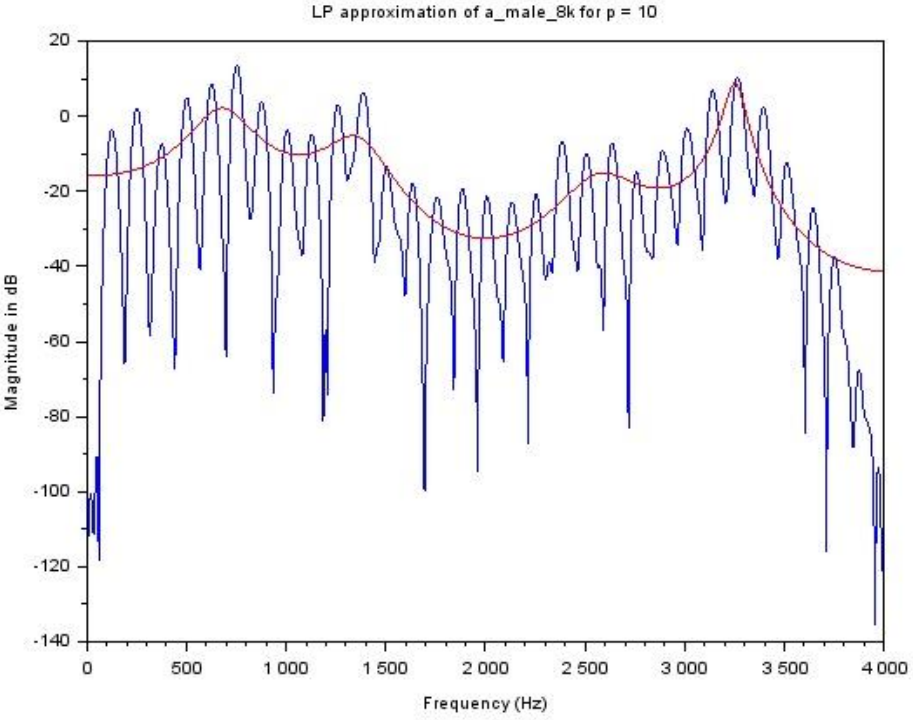


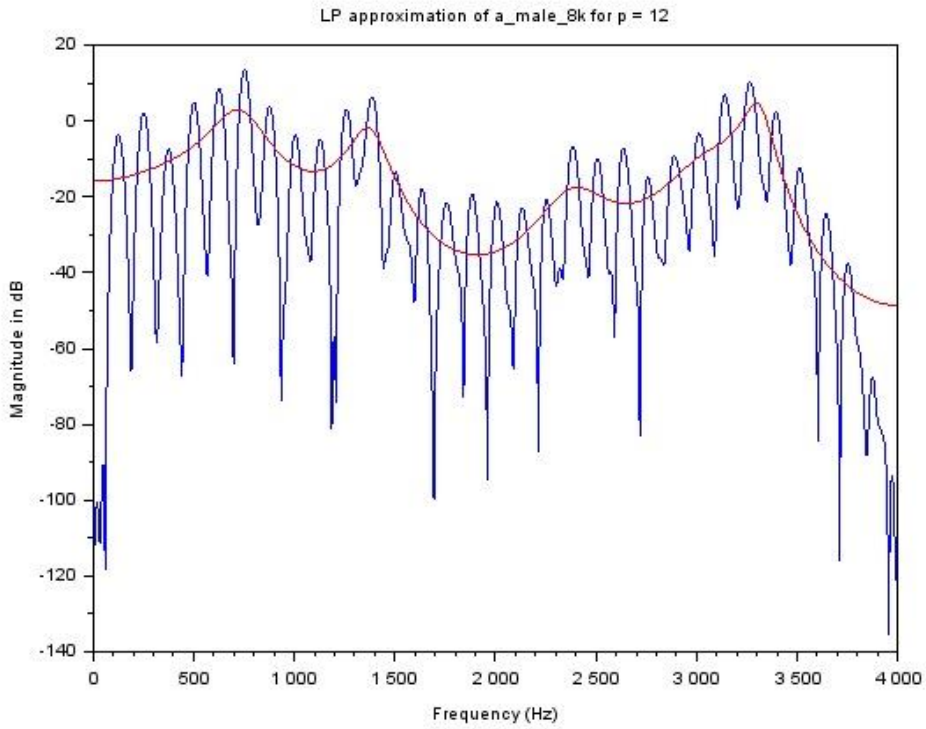
LPC magnitude spectrum for /a/ for different values of p

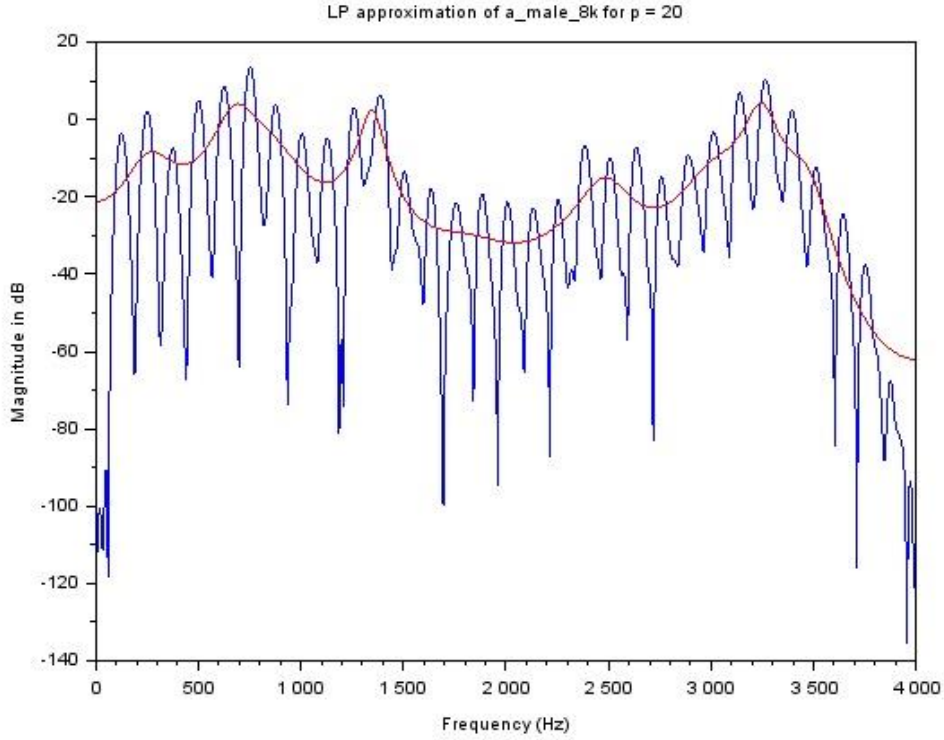




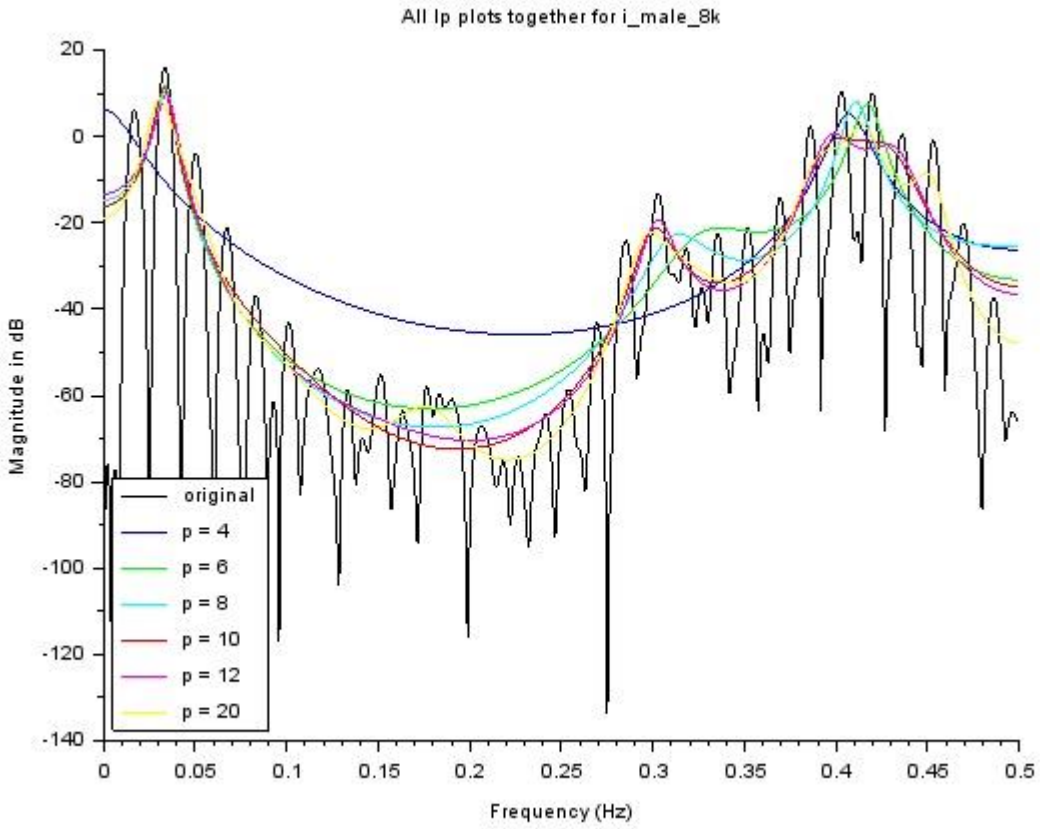

LP approximation of a_male_8k for p = 8
Magnitude in dB
20
0
-20
-40
-60
-80
-100
-120
-140
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000
Frequency (Hz)


LP approximation of a_male_8k for p = 10
Magnitude in dB
20
0
-20
-40
-60
-80
-100
-120
-140
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000
Frequency (Hz)

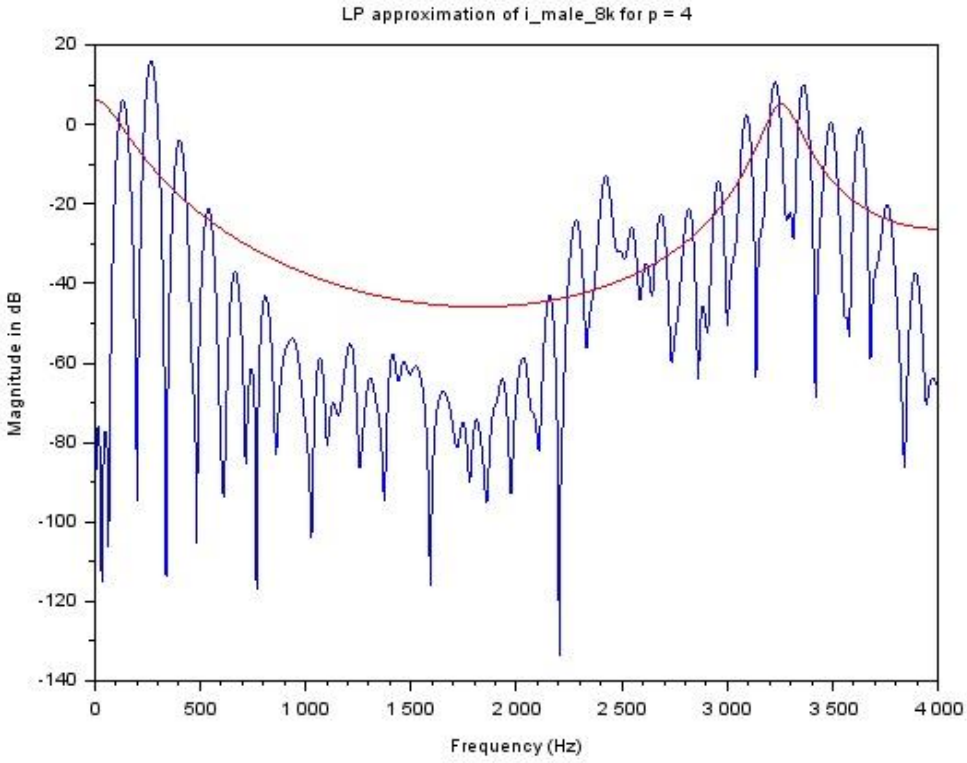

LP approximation of a_male_8k for p = 12
Magnitude in dB
20
0
-20
-40
-60
-80
-100
-120
-140
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000
Frequency (Hz)

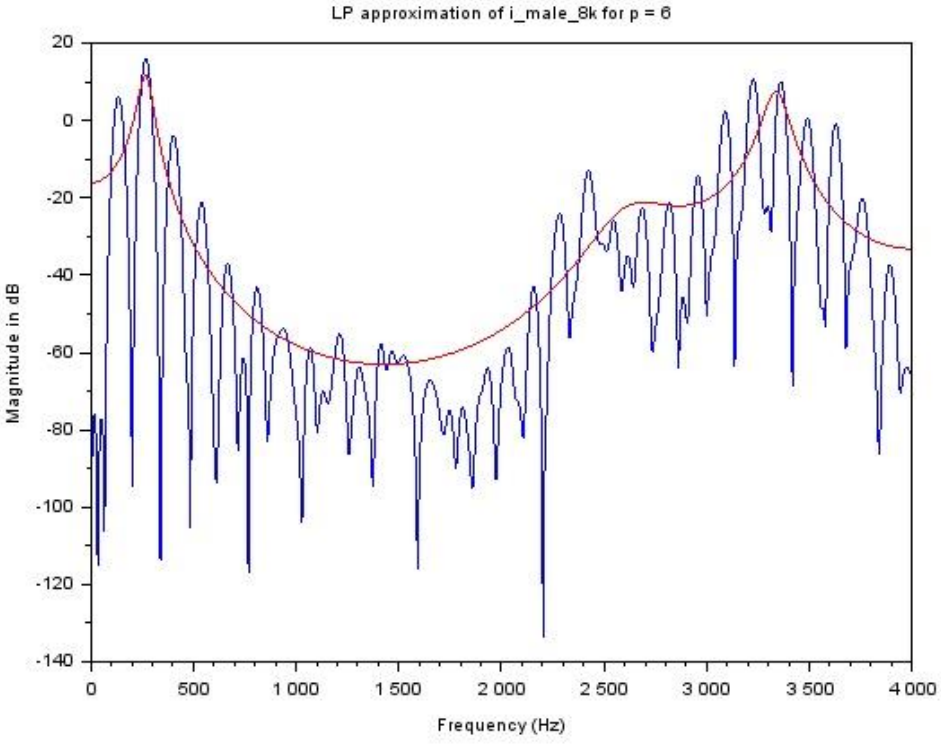

LP approximation of a_male_8k for p = 20
Magnitude in dB
20
0
-20
-40
-60
-80
-100
-120
-140
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000
Frequency (Hz)

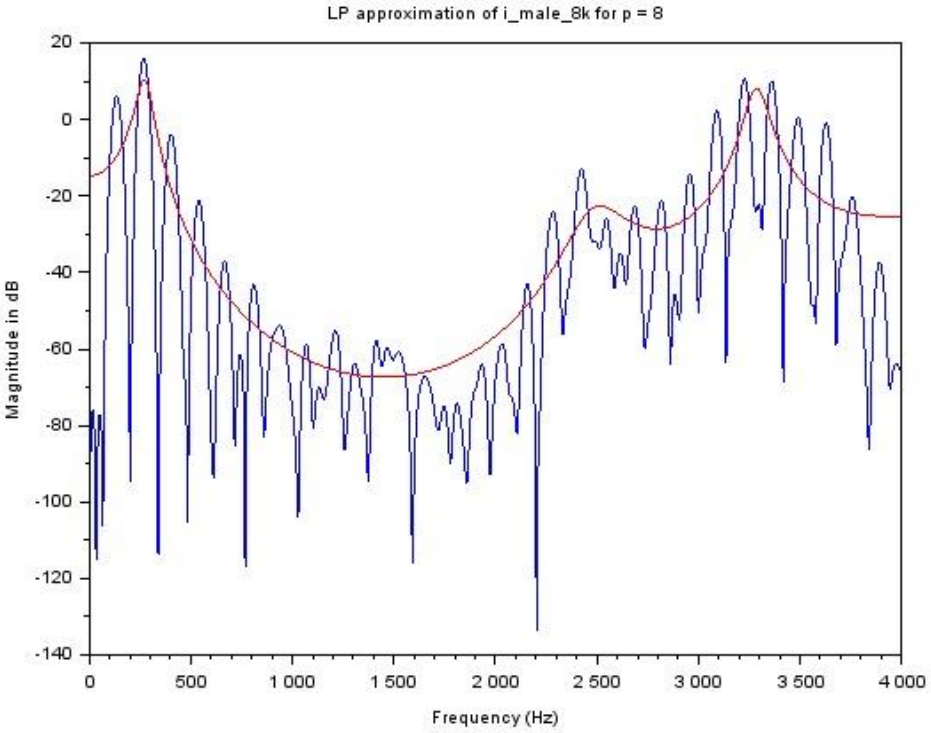
Combined LP spectrum for all p's for /i/

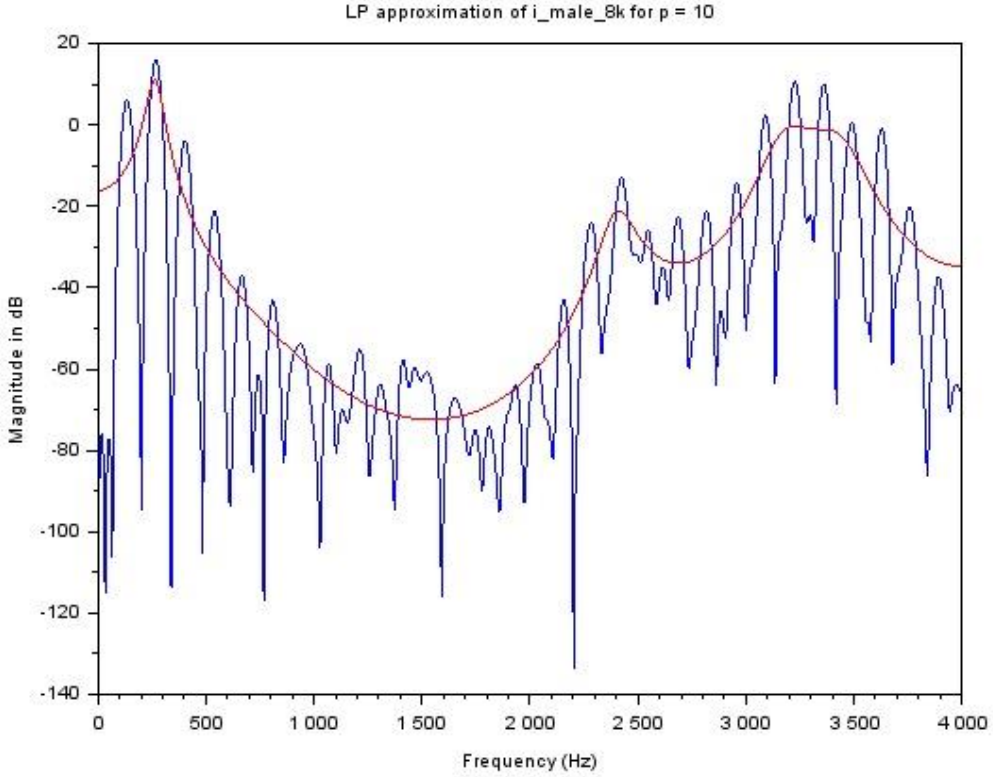


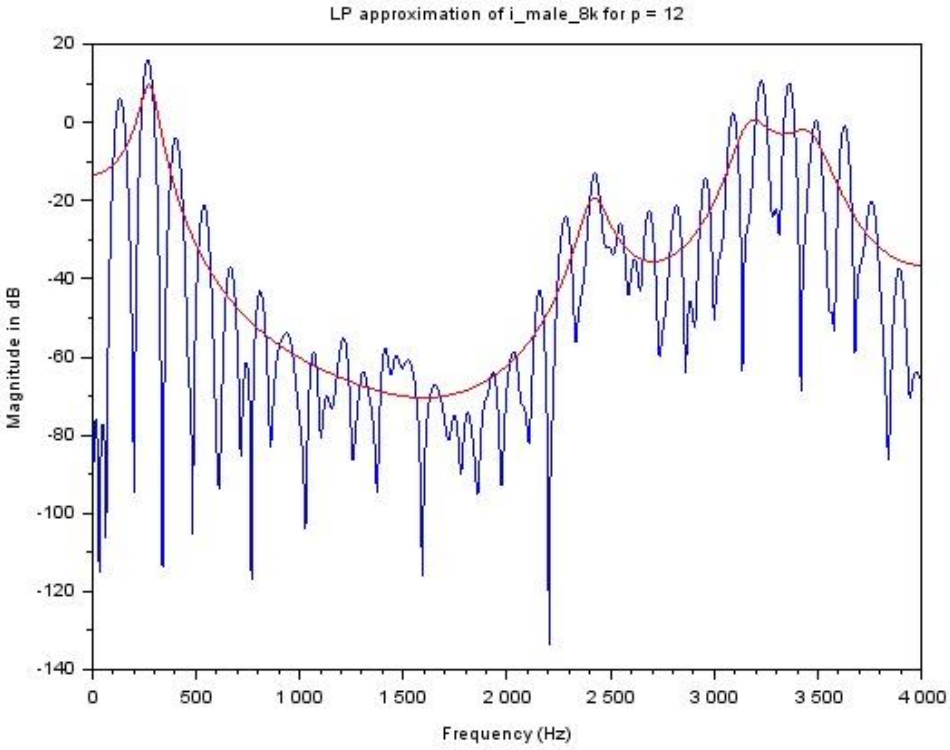
LPC magnitude spectrum for /i/ for different values of p

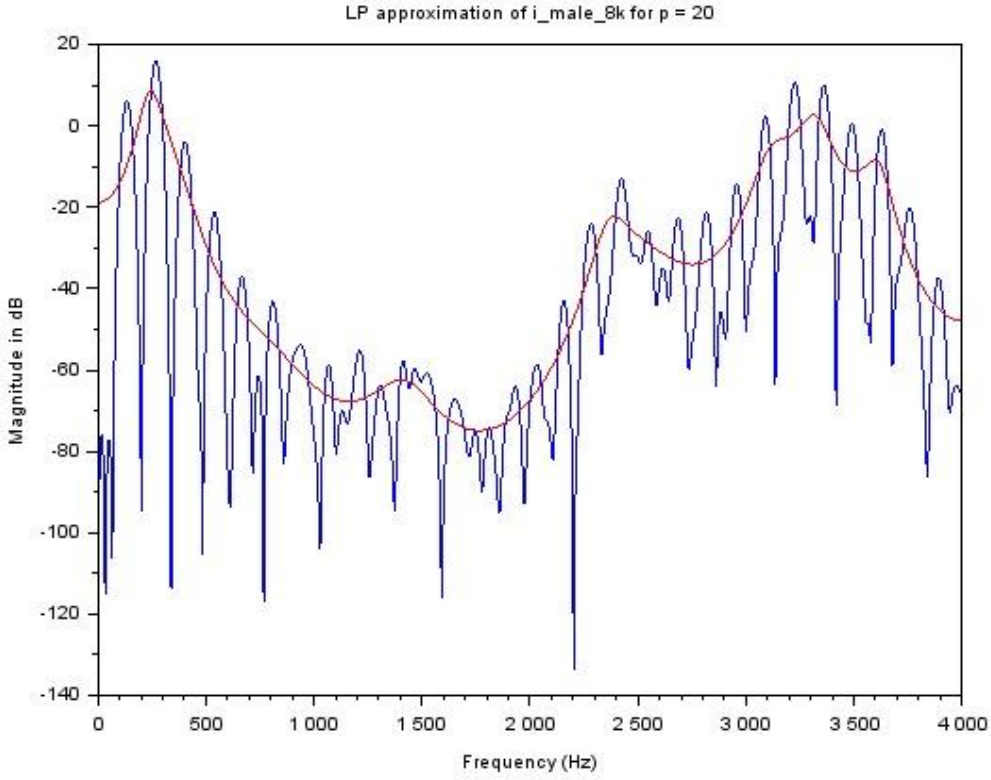




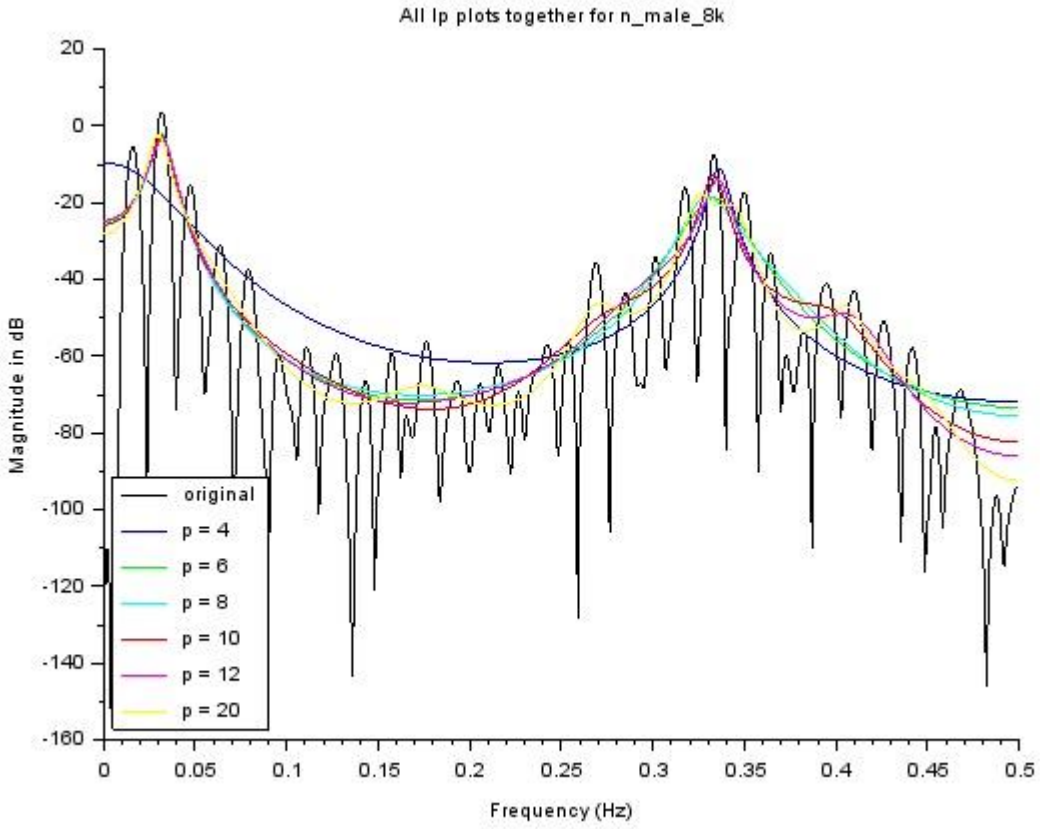

LP approximation of i_male_8k for p = 8
Magnitude in dB
Frequency (Hz)
20
0
-20
-40
-60
-80
-100
-120
-140
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000


LP approximation of i_male_8k for p = 10
Magnitude in dB
Frequency (Hz)
20
0
-20
-40
-60
-80
-100
-120
-140
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000

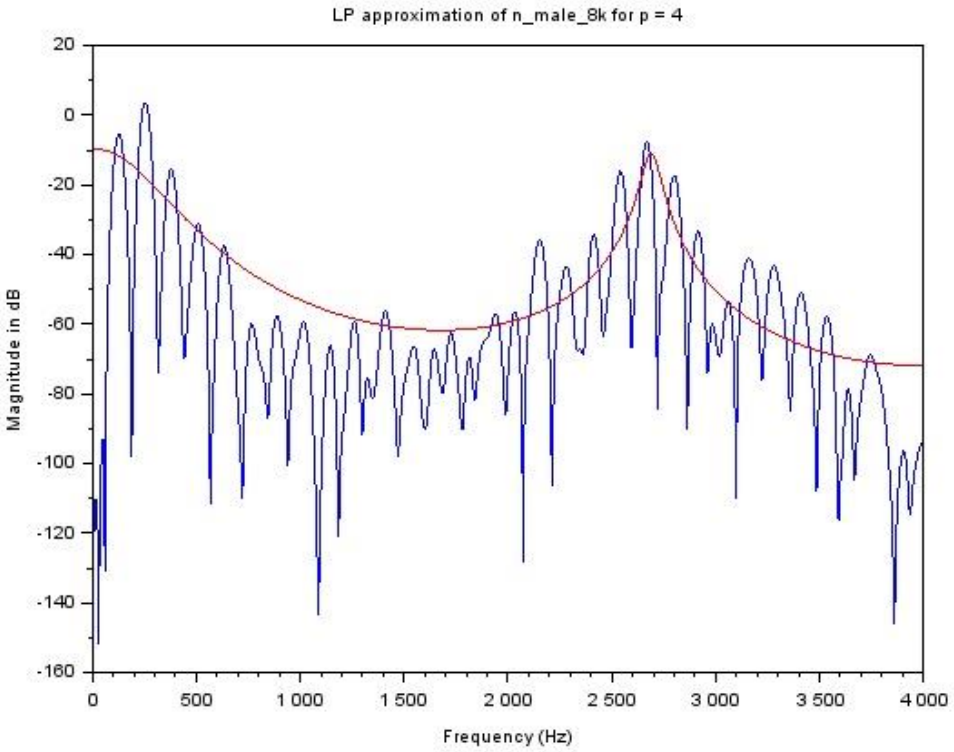

LP approximation of i_male_8k for p = 12
Magnitude in dB
Frequency (Hz)
20
0
-20
-40
-60
-80
-100
-120
-140
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000

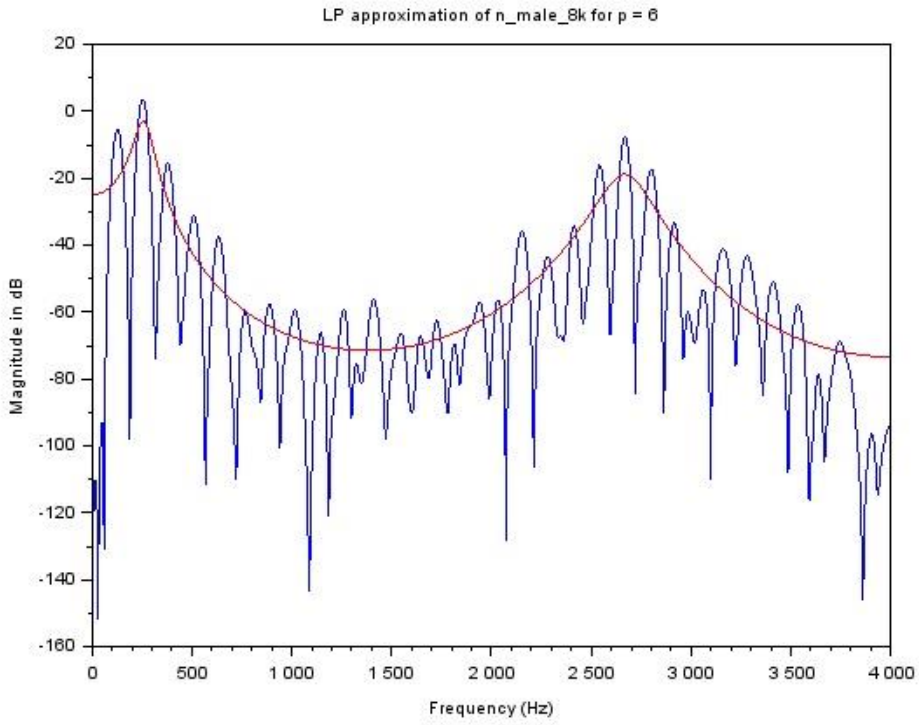

LP approximation of i_male_8k for p = 20
Magnitude in dB
Frequency (Hz)
20
0
-20
-40
-60
-80
-100
-120
-140
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000

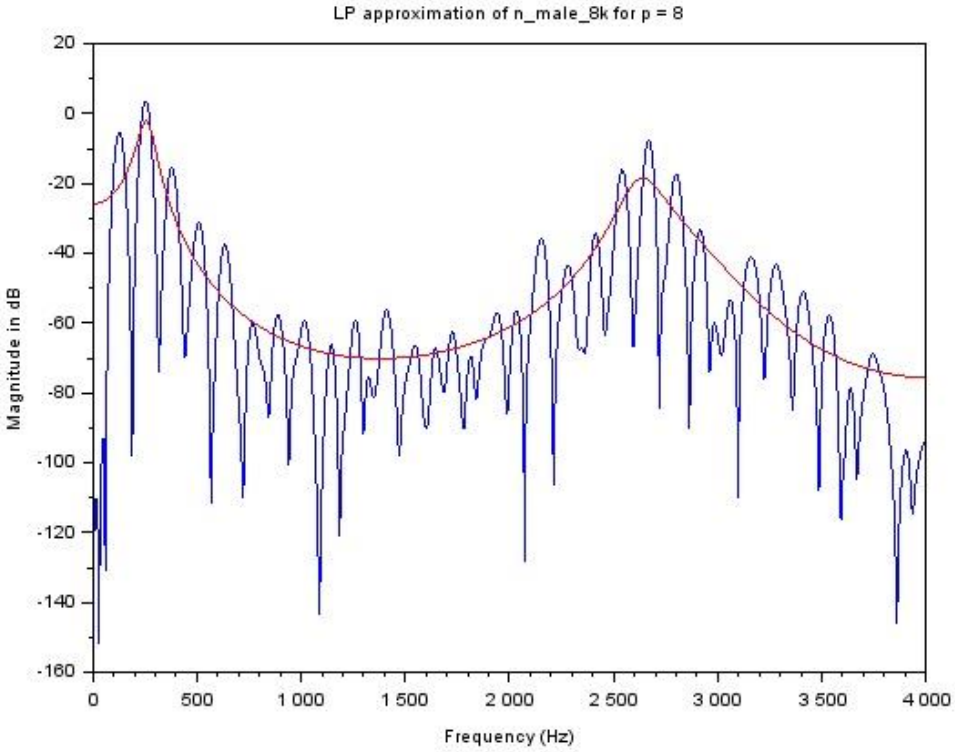
LPC magnitude spectrum for /i/ for different values of p

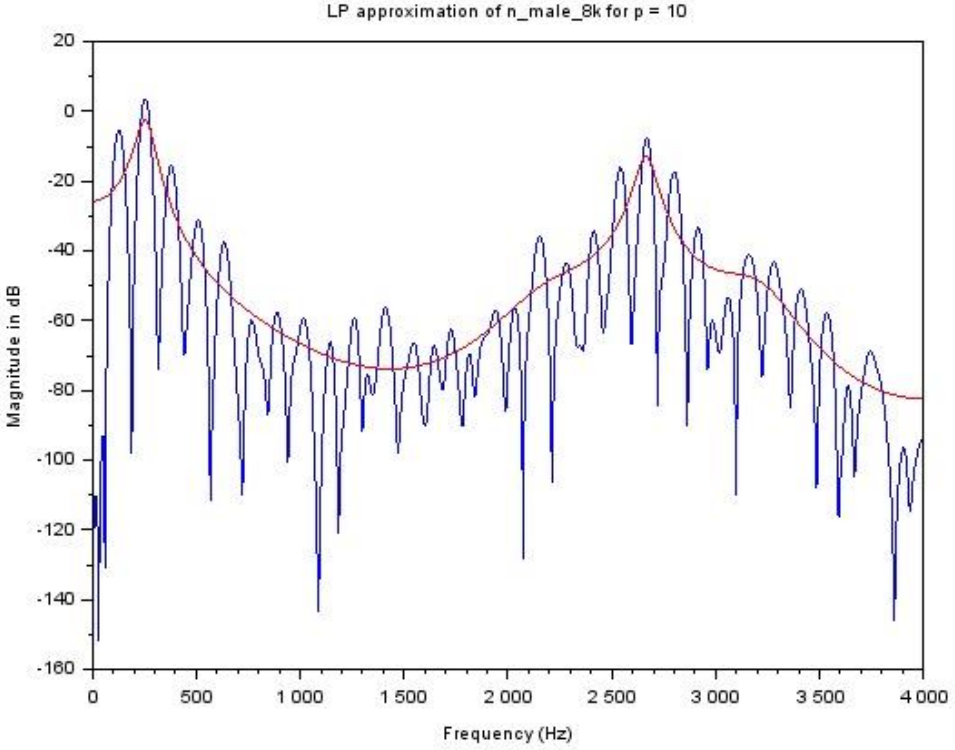


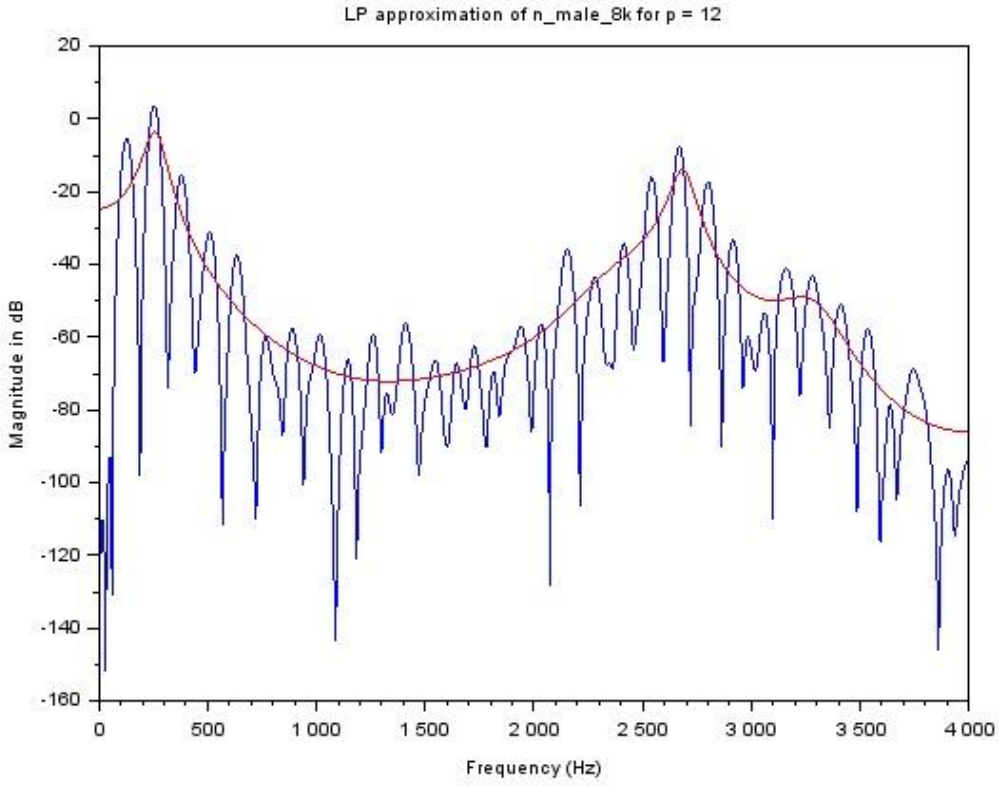
LPC spectrum for /n/ for different values of p

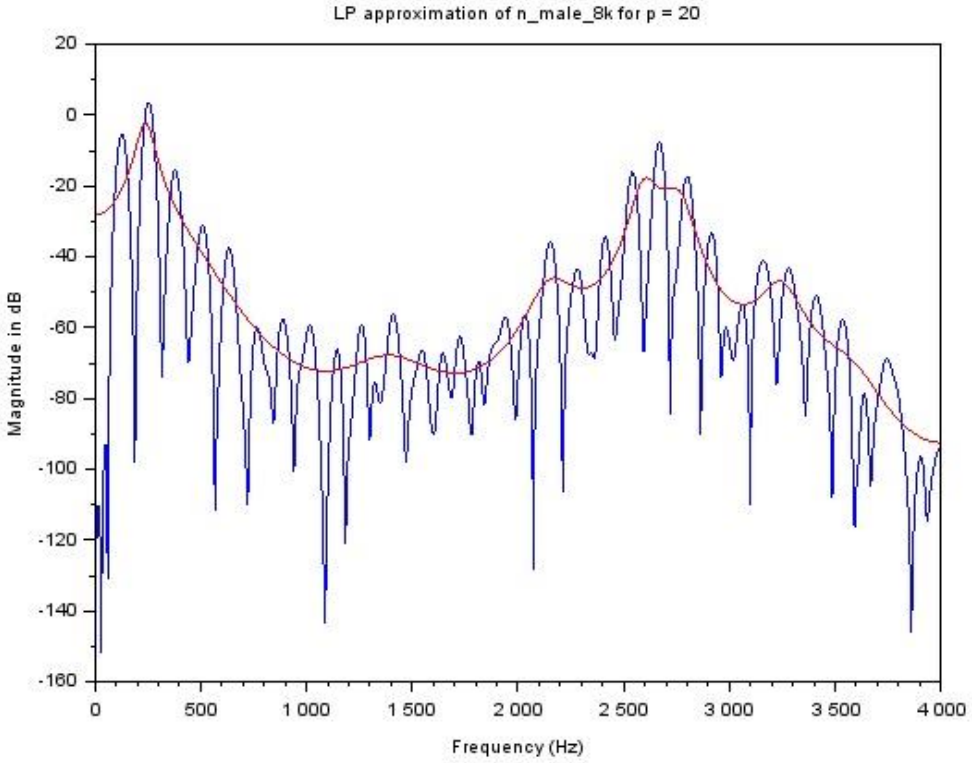




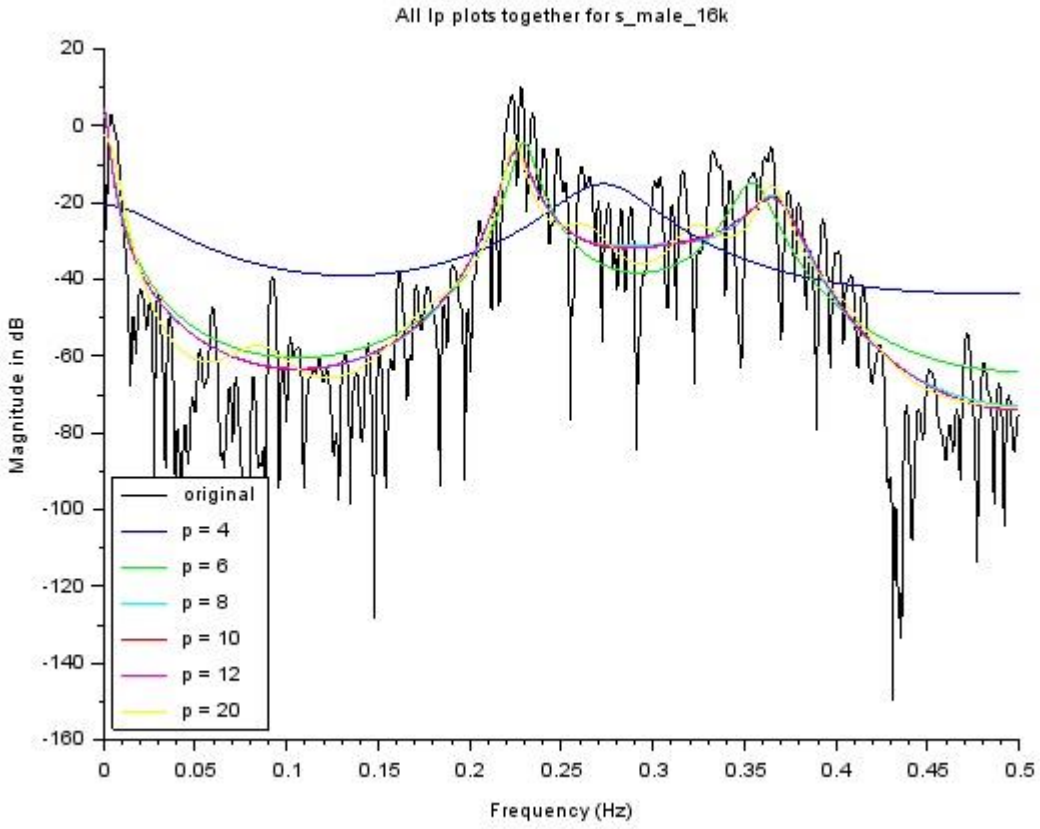

LP approximation of n_male_8k for p = 8
Magnitude in dB
20
0
-20
-40
-60
-80
-100
-120
-140
-160
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000
Frequency (Hz)


LP approximation of n_male_8k for p = 10
Magnitude in dB
20
0
-20
-40
-60
-80
-100
-120
-140
-160
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000
Frequency (Hz)

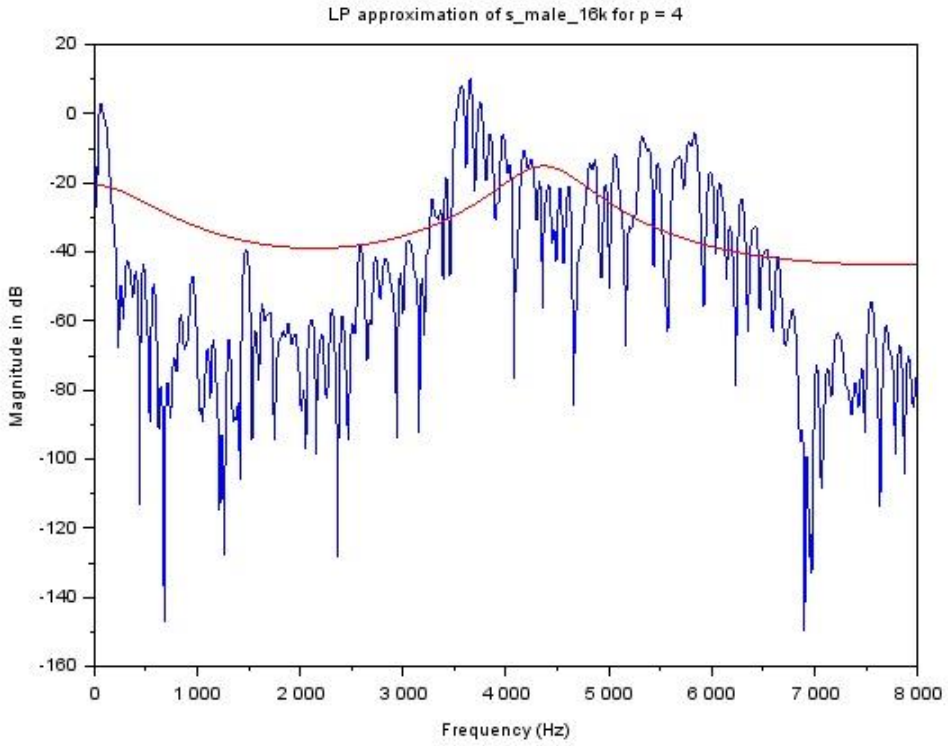

LP approximation of n_male_8k for p = 12
Magnitude in dB
20
0
-20
-40
-60
-80
-100
-120
-140
-160
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000
Frequency (Hz)

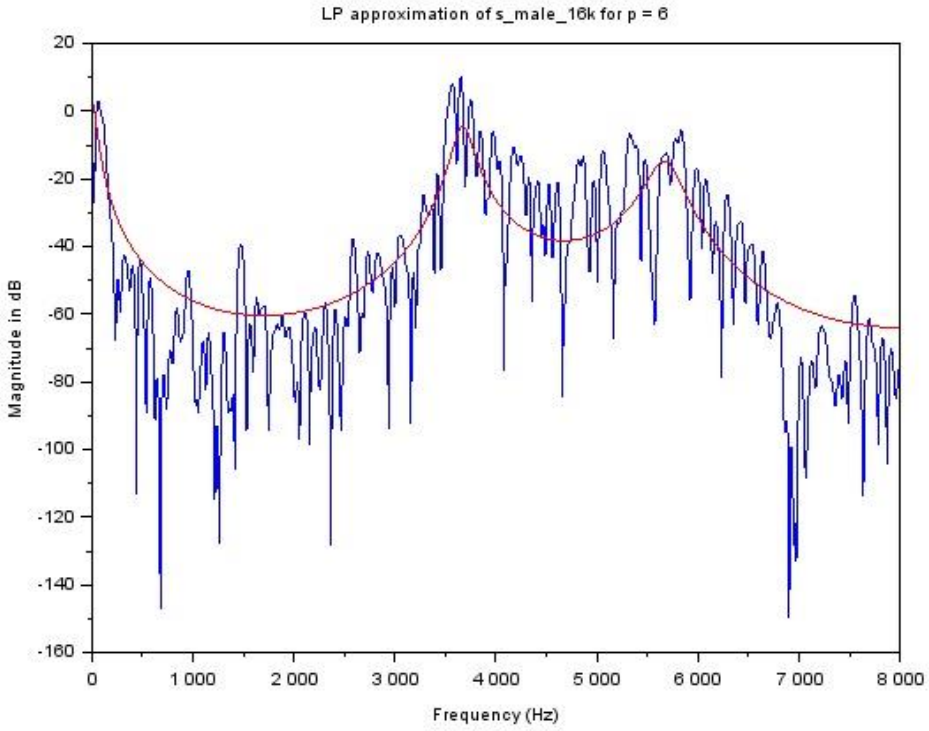

LP approximation of n_male_8k for p = 20
Magnitude in dB
20
0
-20
-40
-60
-80
-100
-120
-140
-160
0
500
1 000
1 500
2 000
2 500
3 000
3 500
4 000
Frequency (Hz)

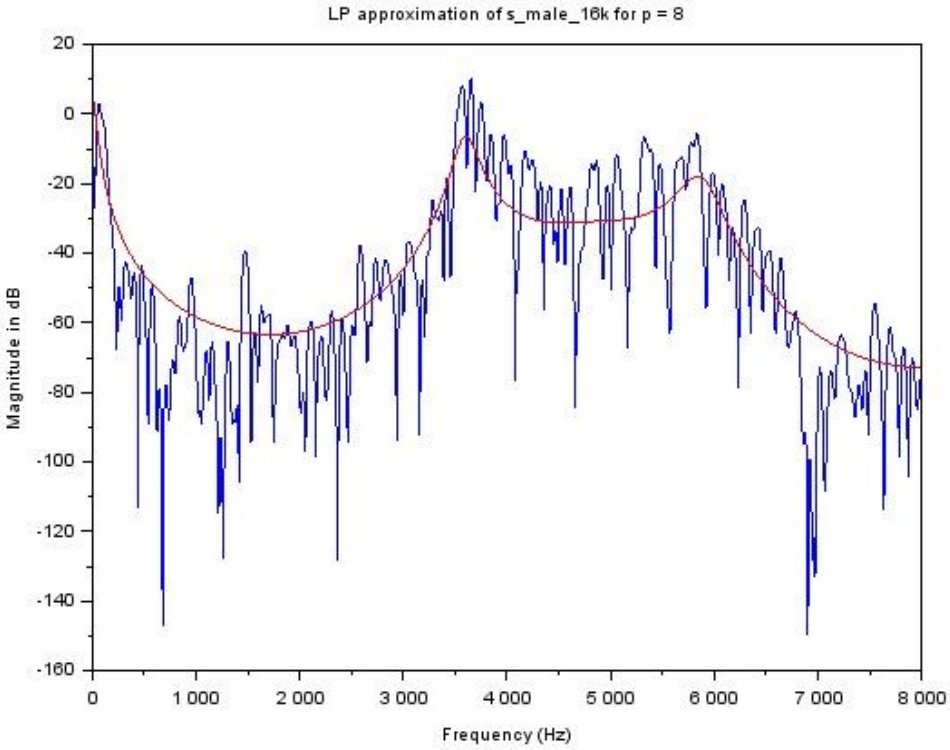
LPC magnitude spectrum for /s/ for different values of p

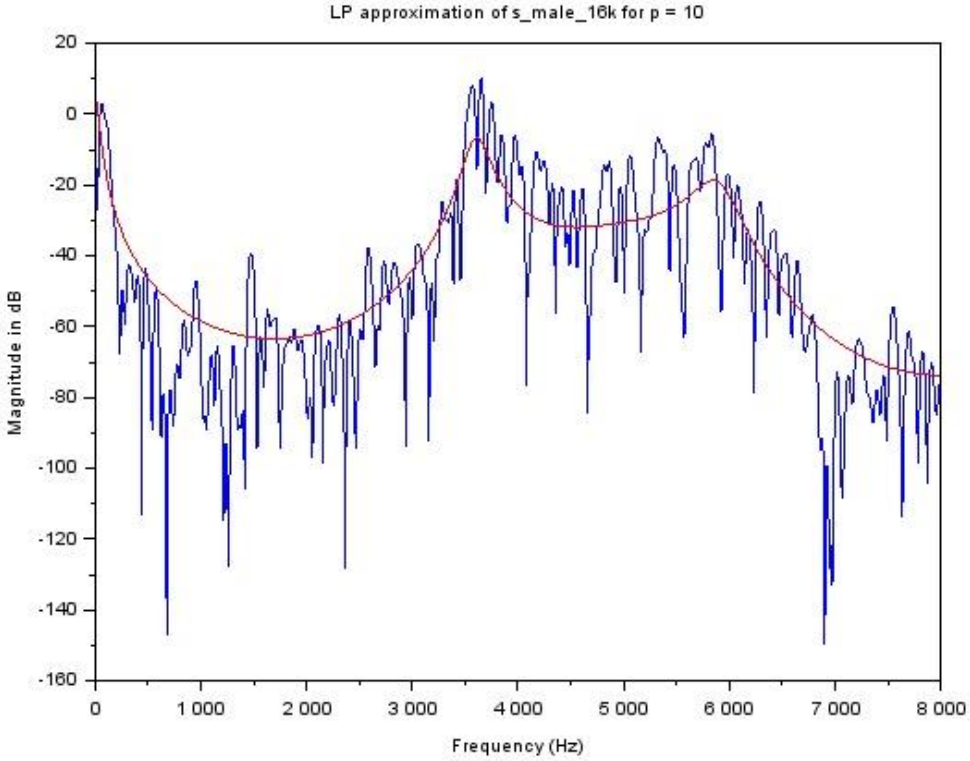


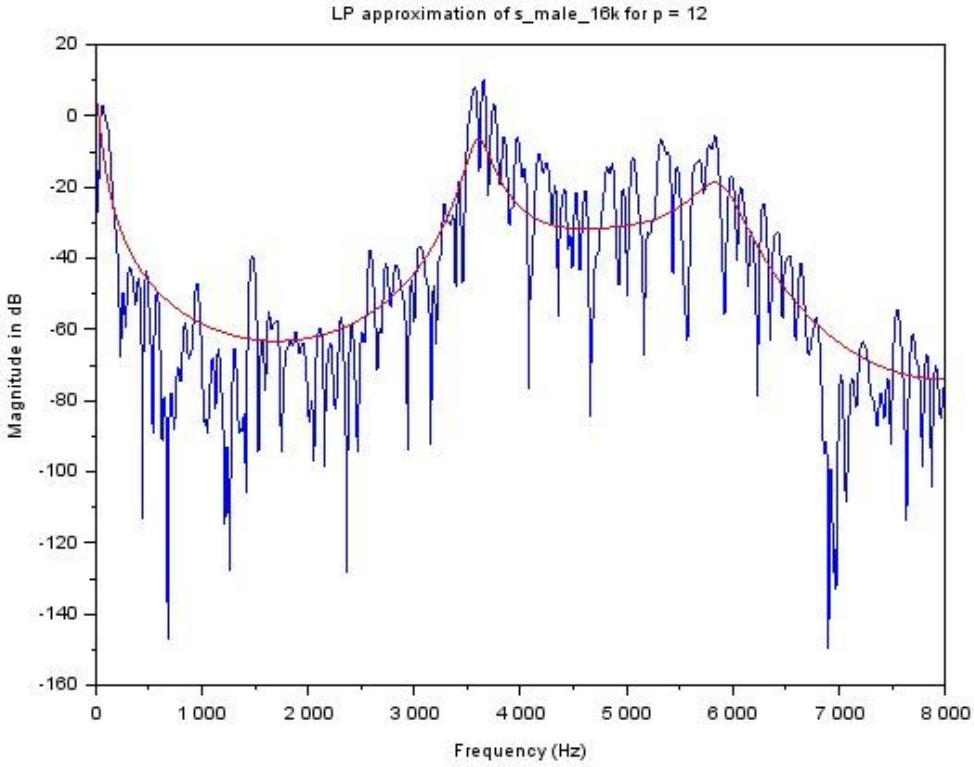
LPC spectrum for /s/ for different values of p

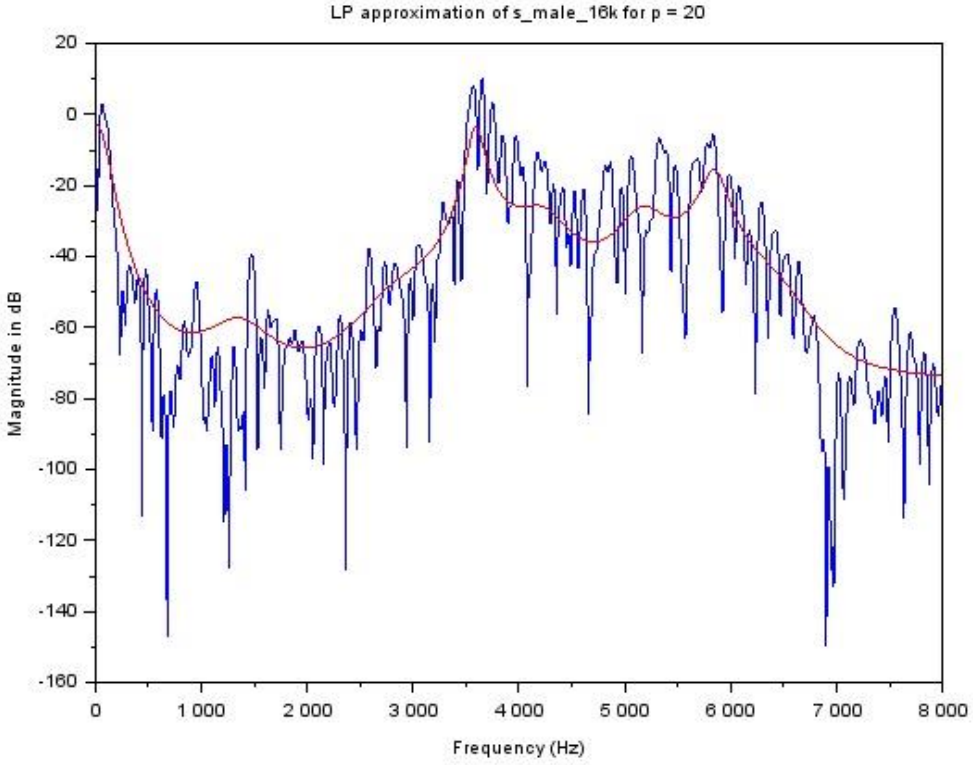





LP approximation of s_male_16k for p = 8
Magnitude in dB
20
0
-20
-40
-60
-80
-100
-120
-140
-160
0
1 000
2 000
3 000
4 000
5 000
6 000
7 000
8 000
Frequency (Hz)


LP approximation of s_male_16k for p = 10
Magnitude in dB
20
0
-20
-40
-60
-80
-100
-120
-140
-160
0
1 000
2 000
3 000
4 000
5 000
6 000
7 000
8 000
Frequency (Hz)

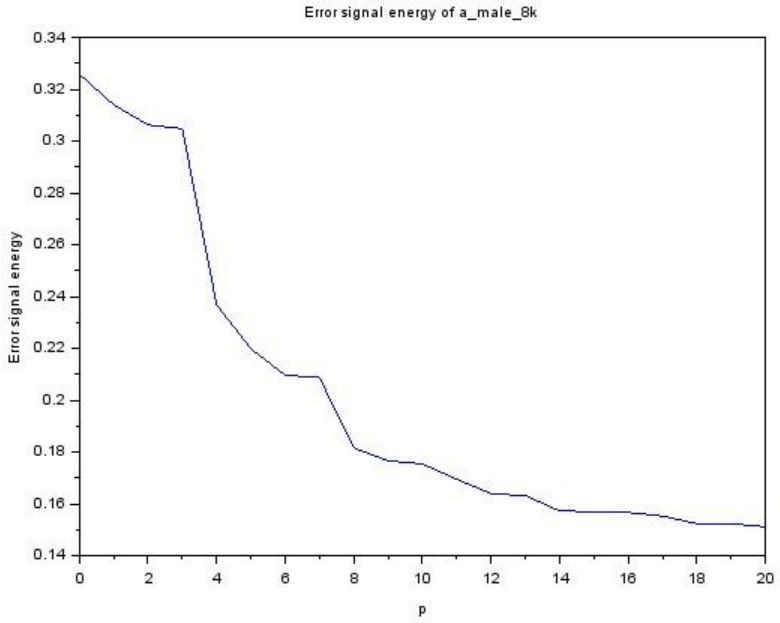

LP approximation of s_male_16k for p = 12
Magnitude in dB
20
0
-20
-40
-60
-80
-100
-120
-140
-160
0
1 000
2 000
3 000
4 000
5 000
6 000
7 000
8 000
Frequency (Hz)


LP approximation of s_male_16k for p = 20
Magnitude in dB
20
0
-20
-40
-60
-80
-100
-120
-140
-160
0
1 000
2 000
3 000
4 000
5 000
6 000
7 000
8 000
Frequency (Hz)

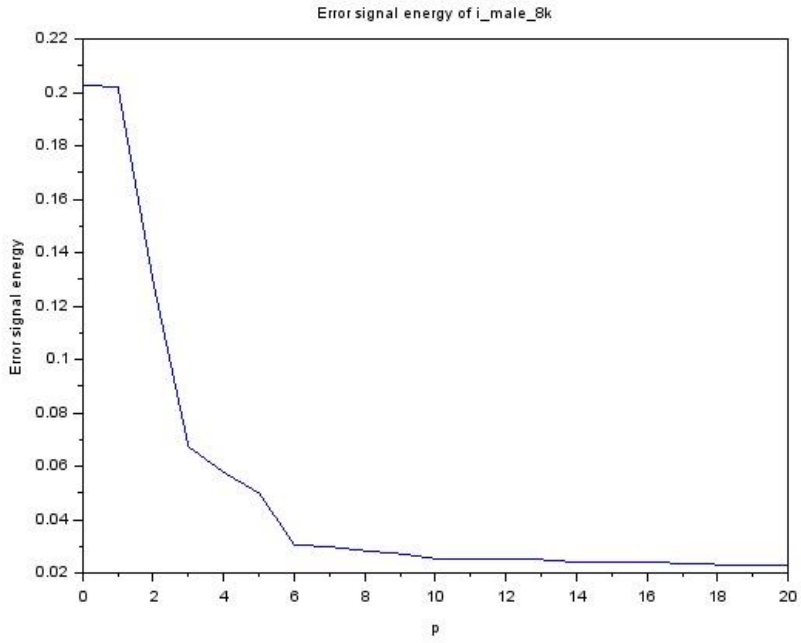
c. Error vs p

Error signal energy is plotted for given sounds. As p increases, error signal energy decreases. This is expected as a result of Levinson recursion. Note that absolute value of error energy is more for /a/, /i/ compared to /s/ and /n/, because vowels generally tend to have higher energy (abs(x[n]) is large -> acf(x[n]) will have larger values -> error will be more)

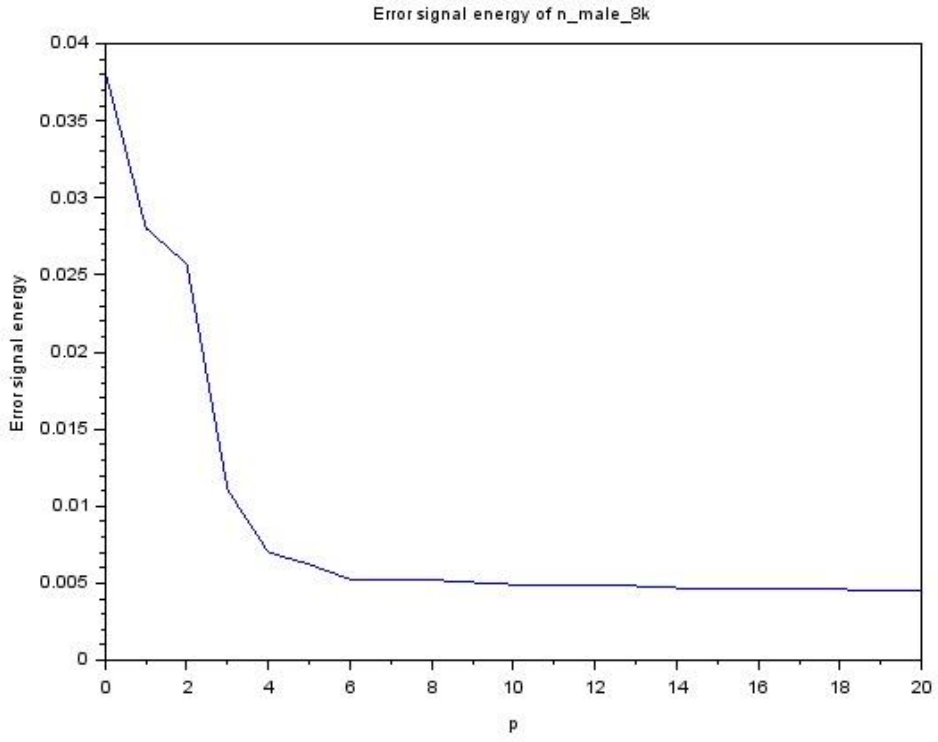
Error vs p plot for /a/



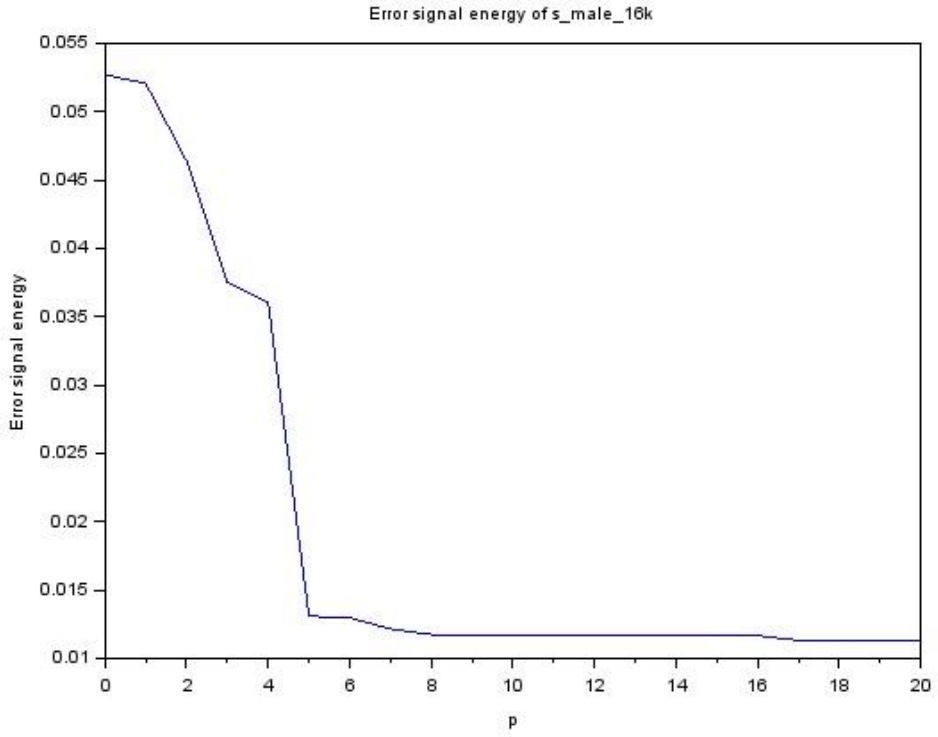
Error vs p plot for /i/



Error vs p plot for /n/



Erros vs p for /s/



Q2. Inverse Filtering

Following function implements inverse filter using linear difference equation.

- Code

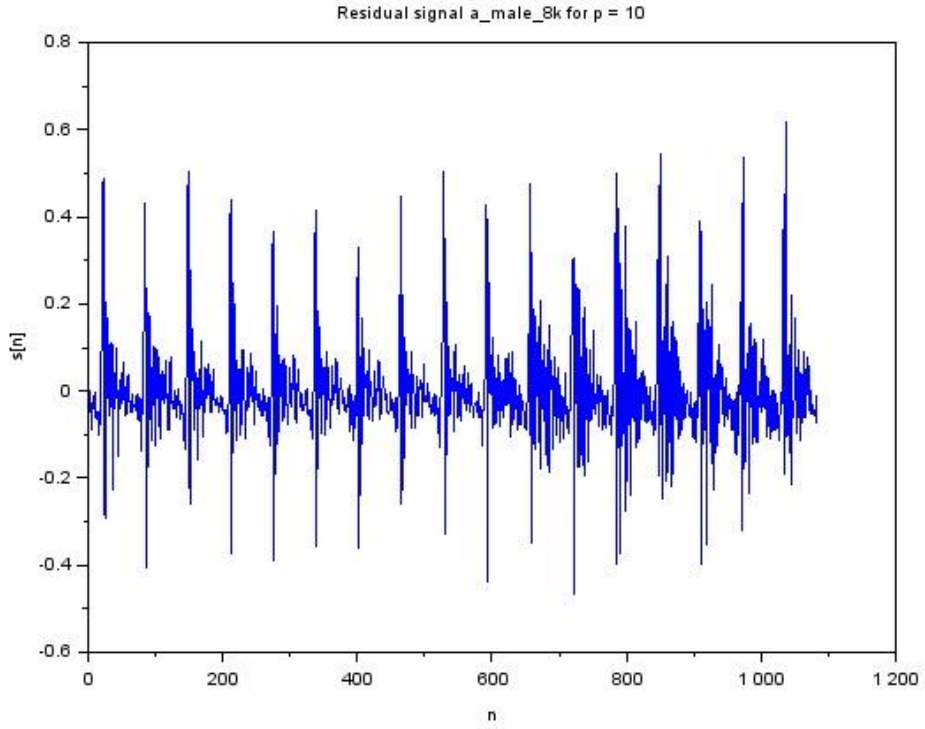
```
//inverse filter
function y=inverse_filter(x, num, den)
   y = zeros(length(x),1)
   for i = 1:length(x)
      for j = 1:min(length(den),i)
         y(i) = y(i) + den(j)*x(i-j+1)
      end
      y(i) = y(i)/num
   end
endfunction
```

- Results & Discussion

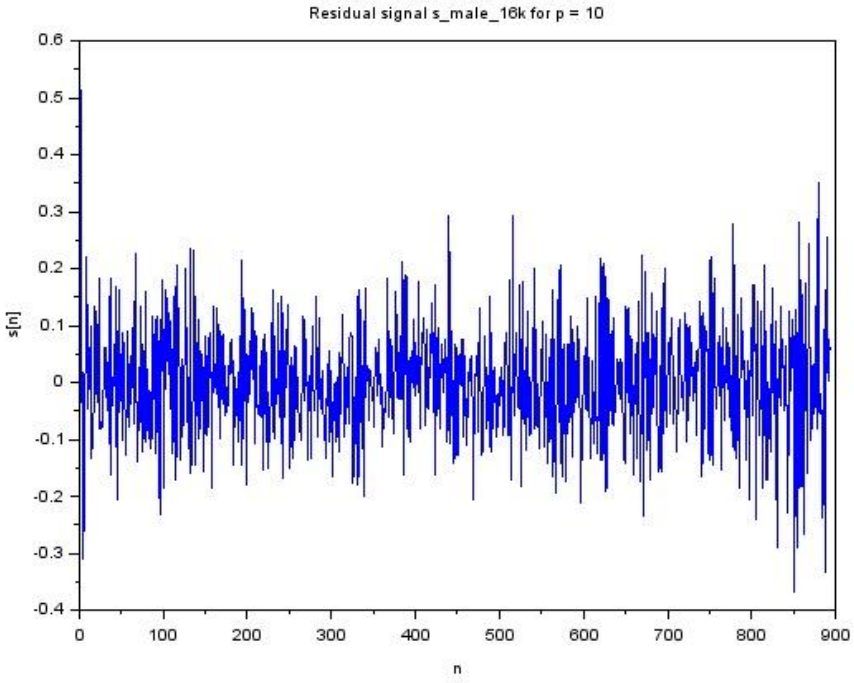
Residual signal is computed as the output of the corresponding inverse filter for /a/ and /s/. This residual signal approximates the source excitation signal. For voiced sound (/a/) source excitation is impulse train. Source excitation is white noise for unvoiced sound (/s/). This can be seen from the plots of residual signals. The pitch of the voiced sound (/a/) can be estimated using autocorrelation function of the residual signal. We compute the distance between successive peaks in the autocorrelation function of residual signal of /a/. The distance between the peaks was found to be 64. Therefore pitch frequency is 8000/64 = **125 Hz** which corresponds to pitch period of **8 ms.**

We observe that magnitude spectrum of residual /a/ contains harmonics (due to voiced nature). Such harmonics are not observed in case of /s/

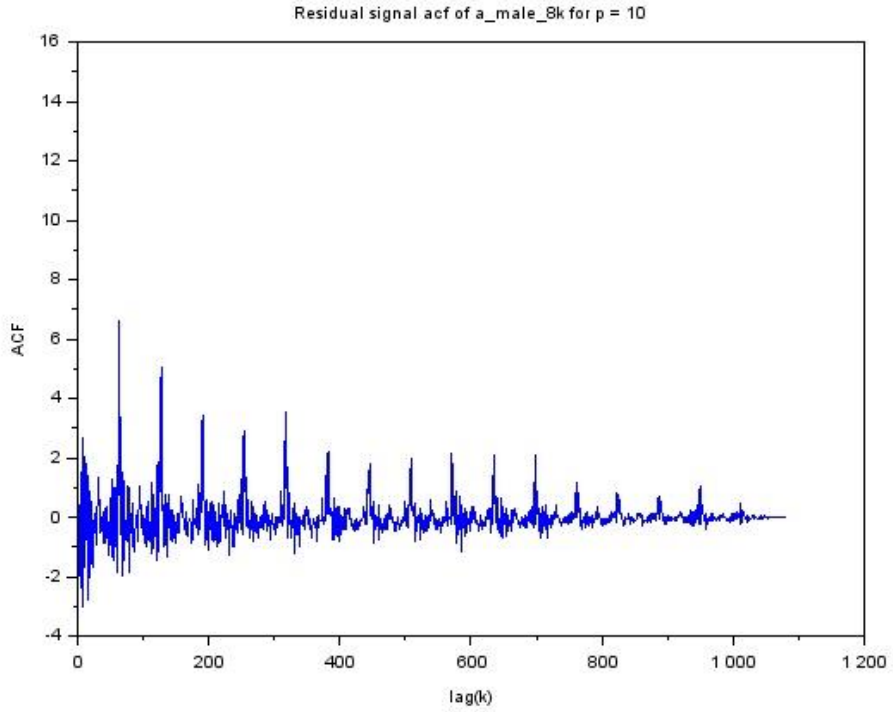
Residual signal for /a/



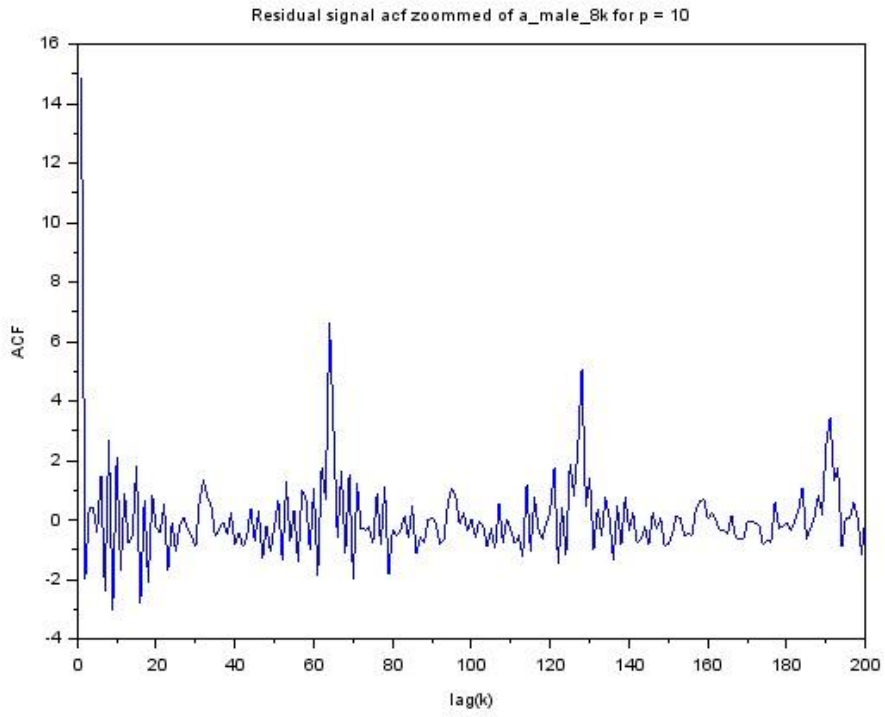
Residual signal for /s/



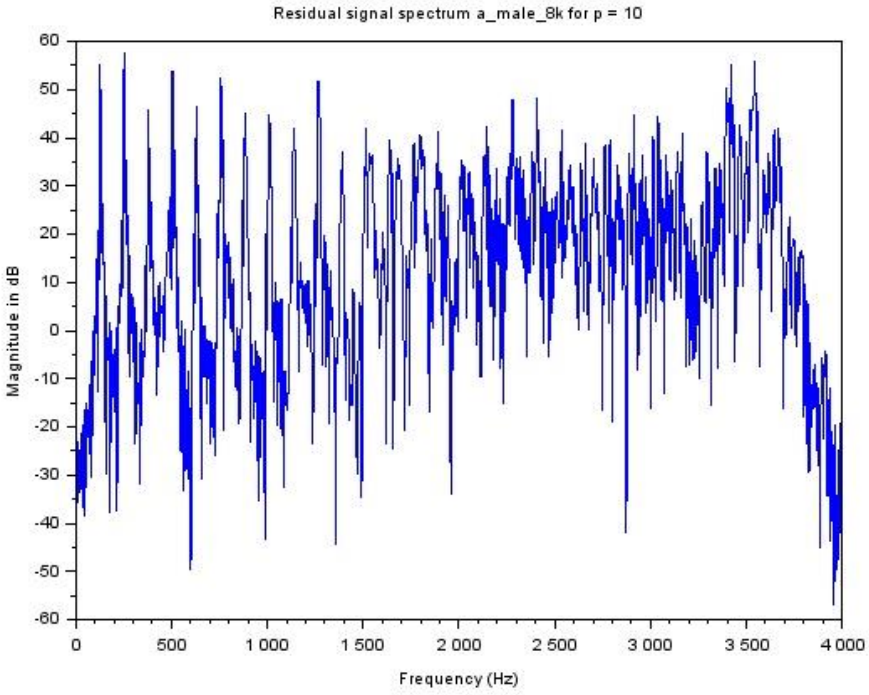
ACF of residual signal for /a/



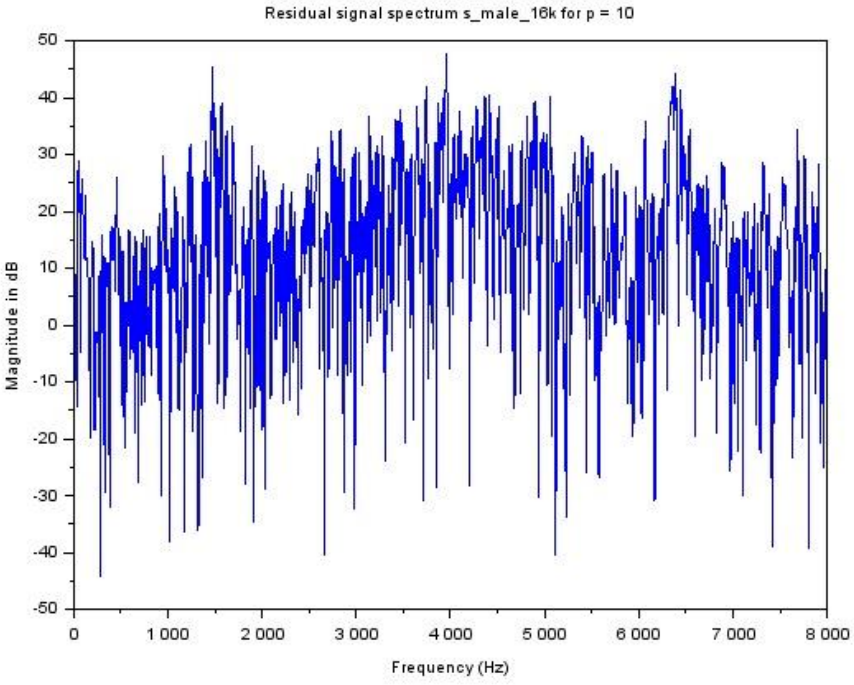
ACF of residual signal zoomed (for pitch estimation)



Magnitude spectrum of residual signal of /a/



Magnitude spectrum of residual signal of /s/



- Entire Code for Problem B

```
clear all

//function to read previously generated wavefiles
function y=read_file(wavefile)
   y = loadwave(strcat(['../wav_files/Q2/',wavefile]));
endfunction

//pre_emphasis filter
function y=pre_emphasis_filter(x, alpha)
   y = zeros(length(x))
   y(1) = x(1)
   for i = 2:length(x)
      y(i) = x(i)-alpha*x(i-1)
   end
endfunction

//de_emphasis filter
function y=de_emphasis_filter(x, alpha)
   y = zeros(length(x))
   y(1) = alpha*x(1)
   for i = 2:length(x)
      y(i) = y(i-1) + alpha*x(i)
   end
endfunction

//inverse filter
function y=inverse_filter(x, num, den)
   y = zeros(length(x),1)
   for i = 1:length(x)
      for j = 1:min(length(den),i)
         y(i) = y(i) + den(j)*x(i-j+1)
      end
      y(i) = y(i)/num
   end
endfunction

//find windowed fft
function [X_mag]=find_windowed_FFT(x, window_type, N, n_fft)
   //we can put this window anywhere on the signal. We will put it
   //at the centre
   x_init = (length(x)+1)/2 - (N-1)/2
   select window_type
   case 'hamming' then
      win_hamming = window('hm', N);
      windowed_x = x(x_init: x_init + N - 1).*win_hamming;
   case 'rect' then
      windowed_x = x(x_init: x_init + N - 1)
   end
   //zero padding
   pad = zeros(1,ceil((n_fft - N)/2));
   padded_x = [pad windowed_x pad]
   //finding fft
```

```
    freq_x = fftshift(fft(padded_x))
    X_mag = abs(freq_x(n_fft/2 +1:n_fft))
endfunction

//1. Compute and plot the narrowband spectrum using a Hamming window
// of duration = 30ms before and after pre-emphasis.

files_list = ["a_male_8k","i_male_8k","n_male_8k","s_male_16k"]
Fs_list = [8000,8000,8000,16000]

for i = 1:length(Fs_list)
    x = read_file(files_list(i))
    window_time = 30
    Fs = Fs_list(i)
    N = window_time*Fs/1000 + 1
    n_fft = 10*N

   [X_mag] = find_windowed_FFT (x,'hamming',N,n_fft)
   freq_array = linspace(0,Fs/2,n_fft/2)
   fig = scf()
   x_preemph = pre_emphasis_filter(x,0.95)
   [X_preemph_mag] = find_windowed_FFT (x_preemph','hamming',N,n_fft)
   plot2d(freq_array,[20*log(X_preemph_mag)',20*log(X_mag)'],[2,5])
   plot_title = strcat(['Spectrum before and after preemphasis of ',files_list(i)])
   xtitle(plot_title,'Frequency (Hz)','Magnitude in dB')
   legends(['after preemphasis','before preemphasis'],[2,5],opt = 6)
   xs2jpg(gcf(), strcat(['../plots/Q2/',plot_title,'.jpg']));

end

//2. Using a 30 ms Hamming window centered in the segment of the
//  waveform preemphasised for the voiced sounds)

do_emphasis_list = [1,1,1,0]
p_list = [4,6,8,10,12,20]
pole_zero_list = [6,10]
do_inverse_filtering = [1,0,0,1]
for iii = 1:length(Fs_list)
       Fs = Fs_list(iii)
       x = read_file(files_list(iii))'
       if(do_emphasis_list(iii)==1)
          x = pre_emphasis_filter(x,0.95)
       end
       x = x'
       window_time = 30
       N = window_time*Fs/1000 + 1
       n_fft = 10*N
       x_init = (length(x)+1)/2 - (N-1)/2
       win_hamming = window('hm', N);
       y = x(x_init: x_init + N - 1).*win_hamming;

       r = zeros(length(y),1)
       for ki=0:length(x)
           for ni=ki:L-1
             r(ki+1)=r(ki+1)+y(ni+1)*y(ni-ki+1);
           end
```

```
end

delta_n = zeros(N,1)
delta_n(1) = 1

h_total = []
//LP recursion
p_max = 20
e_vec = [r(1)]
i = 1
k = r(2)/e_vec(1)
a_vec = [1,k]
e_vec = [e_vec,(1-(k)^2)*e_vec(1)]

for i = 2:p_max
   k = r(i+1)
   for j = 1:i-1
      k = k - a_vec(j+1)*r(i-j+1)
   end
   k = k/e_vec(i)
   a_vec_old = a_vec
   a_new = k
   for j = 1:i-1
      a_vec(j+1) = a_vec_old(j+1)-k*a_vec_old(i-j+1)
   end
   a_vec = [a_vec , a_new]
   e_vec(i+1) = (1-k^2)*e_vec(i)

   //overlapping LP approximation plot on windowed fft
   if (find(i==p_list) ~= [])
      fig = scf()
      h_padded = zeros(n_fft,1)
      h_padded(1:length(y)) = y'
      H = fftshift(fft(h_padded))
      H_mag = abs(H(n_fft/2 +1:n_fft))
      if(h_total == [])
         h_total = [h_total,H_mag]
      end
      freq_array = linspace(0,Fs/2,n_fft/2)
      plot(freq_array,20*log(H_mag))
      num = sqrt(e_vec(i+1))
      den = -a_vec(2:length(a_vec))
      den = [1,den]
      [h,w] = frmag(num,den,n_fft)
      [h2,w2] = frmag(num,den,n_fft/2)
      h_total = [h_total,h2']
      plot(w*Fs,20*log(abs(h)),'red')
      plot_title = strcat(['LP approximation of ',files_list(iii),' for p = ',string(i)])
      xtitle(plot_title,'Frequency (Hz)','Magnitude in dB')
      xs2jpg(gcf(), strcat(['../plots/Q2/',plot_title,'.jpg']));

   end

   //plotting pole zero plots
   if(find(i == pole_zero_list)~=[])
      fig = scf()
```

```
        den = -a_vec(2:length(a_vec))
        den = [1,den]
        poles = roots(den)
        plot2d([real(poles),zeros(length(poles),1)],[imag(poles),zeros(length(poles),1)],[-2,-3])
        legends(['poles','zeros'],[-2,-3],opt = 'lr')
        plot_title = strcat(['pole zero plot of ',files_list(iii),' for p = ',string(i)])
        xtitle(plot_title,'Frequency (Hz)','Magnitude in dB')
        ax=get("current_axes")
        ax.data_bounds = [-1,-1;1,1]
        xs2jpg(gcf(), strcat(['../plots/Q2/',plot_title,'.jpg']));
    end

    //inverse filtering
    if((do_inverse_filtering(iii)) & (i == 10))
        den = -a_vec(2:length(a_vec))
        den = [1,den]
        num = sqrt(e_vec(i+1))
        residual_signal = inverse_filter(x,num,den)
        fig = scf()
        plot(residual_signal)
        plot_title = strcat(['Residual signal ',files_list(iii),' for p = ',string(i)])
        xtitle(plot_title,'n','s[n]')
        xs2jpg(gcf(), strcat(['../plots/Q2/',plot_title,'.jpg']))
        fig = scf()
        h_padded = zeros(n_fft,1)
        h_padded(1:length(residual_signal)) = residual_signal
        H = fftshift(fft(h_padded))
        H_mag = abs(H(n_fft/2 +1:n_fft))
        freq_array = linspace(0,Fs/2,n_fft/2)
        plot(freq_array,20*log(H_mag))
        plot_title = strcat(['Residual signal spectrum ',files_list(iii),' for p = ',string(i)])
        xtitle(plot_title,'Frequency (Hz)','Magnitude in dB')
        xs2jpg(gcf(), strcat(['../plots/Q2/',plot_title,'.jpg']));

        //3. finding pitch period for voiced signals using ACF of residual signal
        // and plotting magnitude spectrum of residual signal
        if(do_emphasis_list(iii)==1)
            temp_res = length(residual_signal)*xcorr(residual_signal,["biased"])
            acf_residual = temp_res(length(residual_signal):length(temp_res))
            fig = scf()
            plot(acf_residual)
            plot_title = strcat(['Residual signal acf of ',files_list(iii),' for p = ',string(i)])
            xtitle(plot_title,'lag(k)','ACF')
            xs2jpg(gcf(), strcat(['../plots/Q2/',plot_title,'.jpg']));
            //63.5 is period
            acf_zoomed = acf_residual(1:200)
            fig = scf()
            plot(acf_zoomed)
            plot_title = strcat(['Residual signal acf zoommed of ',files_list(iii),' for p = ',string(i)])
            xtitle(plot_title,'lag(k)','ACF')
            xs2jpg(gcf(), strcat(['../plots/Q2/',plot_title,'.jpg']));
            pitch_period = Fs_list(iii)*63.5
        end
    end
end
```

```
    //plotting error vs p graph
    fig = scf()
    plot([0:p_max],e_vec)
    plot_title = strcat(['Error signal energy of ',files_list(iii)])
    xtitle(plot_title,'p','Error signal energy')
    xs2jpg(gcf(), strcat(['../plots/Q2/',plot_title,'.jpg']));

    //plotting all LP together
    fig = scf()
    plot2d(w2,20*log(h_total),[1:(length(p_list)+1)])
    plot_title = strcat(['All lp plots together for ',files_list(iii)])
    xtitle(plot_title,'Frequency (Hz)','Magnitude in dB')
    legends(['original','p = 4','p = 6','p = 8','p = 10','p = 12','p = 20'],[1:(length(p_list)+1)],opt = 3)
    xs2jpg(gcf(), strcat(['../plots/Q2/',plot_title,'.jpg']));

end
xdel(winsid())
```