Steps

First, users can enter the value of the control parameter for dynamics, which can control both the sound level and the spectral slope of the source. The larger the parameter is, the larger the sound level is and the flatter the slope is. The amplitude of the high-frequency harmonics would be larger when there is a larger control parameter. Also, the users can decide whether they would like to add the vibrato modulation.

```
display('Please enter the value for the control parameter (0, 1].')
display('It can control both the sound level and the spectral slope of the source.')
display('The larger the parameter is, the larger the sound level is and the flatter the slope is.')
dynamics = input('Please enter the value: ');
vibrato_op = input('Do you want to add a vibrato modulation? (y/n): ', 's');
```

```
Please enter the value for the control parameter (0, 1].
It can control both the sound level and the spectral slope of the source.
The larger the parameter is, the larger the sound level is and the flatter the slope is.
Please enter the value: 5/9
Do you want to add a vibrato modulation? (y/n): y
```

Then, I initialized some variables which will be used in the following steps, including the sampling rate, the MIDI note numbers of the melody, the note frequency function, formant frequencies and the duration of each note.

```
% Set up some variables
fsamp = 32000;      % sampling rate

% the relationship between the midi number, m, and the fundamental freauency, f_m
% f_m = 2^((m-69)/12)*440
stars = [60 60 67 67 69 69 67 65 65 64 64 62 62 60];
stars_time = [0.8 0.8 0.8 0.8 0.8 0.8 1.6 0.8 0.8 0.8 0.8 0.8 0.8 1.6];

note = stars;
note_frequency = 2.^((note-69)./12).*440;
note_time = stars_time;
pause_zero = [20 20 20 20 20];

pOut_all = [];
fN = [700, 1050, 2300, 2500, 2800;  % male
      499, 1022, 3162, 3856, 5640;  % o
      386, 899, 2851, 4039, 5160;   % u
      700, 1016, 3279, 4059, 6000]; % a
fNrow = [4 4 2 2 4 4 2 4 3 4 3 4 4 3];
```

I used a for-loop to synthesize a melody. In every iteration of this for-loop, the note of the melody would be generated respectively.

vibAmp is the amplitude of the vibrato modulation; vibFreq is the frequency of the the vibrato modulation. A is the amplitude for the sawtooth signal and the minus sign would flip the resulting sawtooth signal upside down to make it look more like the glottal flow with rapid closing. N_harmonics represents the number of harmonics. Fsamp/2 is Nyquist frequency, so the frequencies of harmonics should not exceed this

value. Therefore, the number of harmonics is as below, floor(fsap/(2*f_0)). f_0 is the fundamental frequency.

```
vibAmp = 0.001;      % the amplitude for vibrato
vibFreq = 6;         % the frequency for vibrato

% A for amplitude
% The minus sign flips the resulting sawtooth upside down,
% to look more like a glottal flow with a rapid closing
% (it sounds the same).
A = -0.8;

% Number of harmonics, staying below Nyquist
% Nyquist frequency = fsamp/2
% Nyquist frequency/fundamental frequency = the number of harmonics
N_harmonics = floor(fsamp/(2*f_0));
```

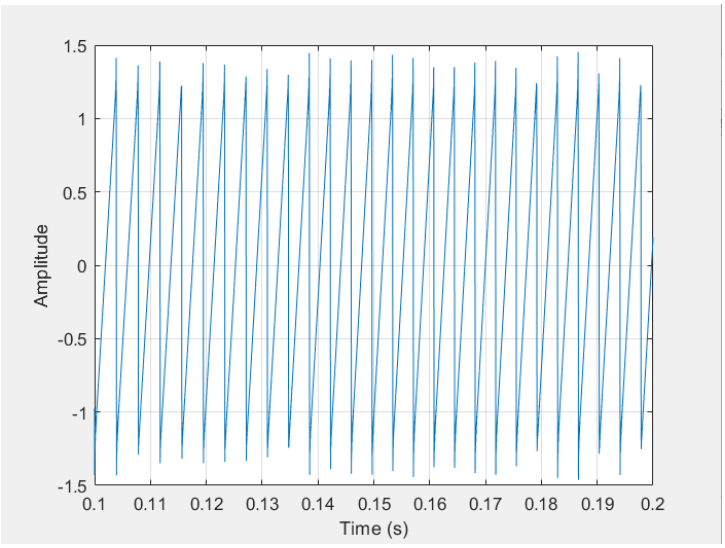
The control parameter the users entered at the beginning would be used to control the spectrum slope parameter of the source, a_slope. While controlling the spectrum slope, it also influences the sound level at the same time.

Then I generated the signals of harmonics, whose methods are different according to the users' selection on whether to add the vibrato modulation. If the users would like to add the vibrato modulation, a vector storing each fundamental frequency at each time point, f_0vib, would be generated. Then f_0vib would be used in making the sine waves, so the generated signals would have different fundamental frequencies at each time point, which implements the vibrato modulation.

```
% Spectrum slope parameter
a_slope = 1*(-1.8*dynamics+2);

% Make a time axis vector
t = linspace(0.001, totalTime, totalTime*fsamp);

% Make the sine waves and add them together
p = zeros(size(t));
for n = 1 : N_harmonics
    if vibrato_op == 'y'
        % f_0vib = f_0 + (vibAmp ./ t) .* sin(2*pi*vibFreq*t) * f_0
        f_0vib = (1 + (vibAmp ./ t) .* sin(2*pi*vibFreq*t)) * f_0;
        p = p + A*(n^(-a_slope))*sin(2*pi*n*f_0vib .* t);   % vibrato
    elseif vibrato_op == 'n'
        p = p + A*(n^(-a_slope))*sin(2*pi*n*f_0 .* t);     % no vibrato
    end
end
% p is now the source signal, representing glottal flow
```

After the above steps, now the source signal, p, has been generated. From the waveform, we can observe that it looks like a sawtooth. In the following steps, I will

apply five formant filters which are based on resonance two-pole filters on this source signal.



First, initialize the parameters for the formant filters according to “JoLi filters cheat sheet”. I used the Lowpass Formant Resonator.

```
%% Table of formant frequencies
% Table of formant bandwidths
bN = [25, 40, 60, 80, 100];

% Table of Q factors
qN = fN(fNrow(i), :) ./ bN;

% Compute the second order filter coefficients for the above five filters
% using the Liljencrants 'cheat sheet'
T = 1/fsamp;
for j = 1 : 5
  betas(j) = fN(fNrow(i), j)*2*pi;
  beta0s(j) = betas(j)*sqrt(1+1/(4*qN(j)^2));
  alphas(j) = beta0s(j)/(2*qN(j));
  aOnes(j) = -2*exp(-alphas(j)*T)*cos(betas(j)*T);
  aTwos(j) = exp(-2*alphas(j)*T);
  Gs(j) = 1 + aOnes(j) + aTwos(j);
end
```

I applied the generated filters on the source signals and acquire the resulting singing signals.

```
% Apply to "p" the desired number of resonances, in series.
% The recursive filter formula is (lowpass formant resonator)
% Yn = G*X(n) - aOne*Y(n-1) - aTwo*Y(n-2)
pOut = p;
pOut(1) = 0.0;
pOut(2) = 0.0;

for j = 1 : 5
    for n = 3 : length(pOut)
        pOut(n) = Gs(j)*pOut(n) - aOnes(j)*pOut(n-1) - aTwos(j)*pOut(n-2);
    end
end

pOut_all = [pOut_all, pOut];
```

To play the generated audio data.

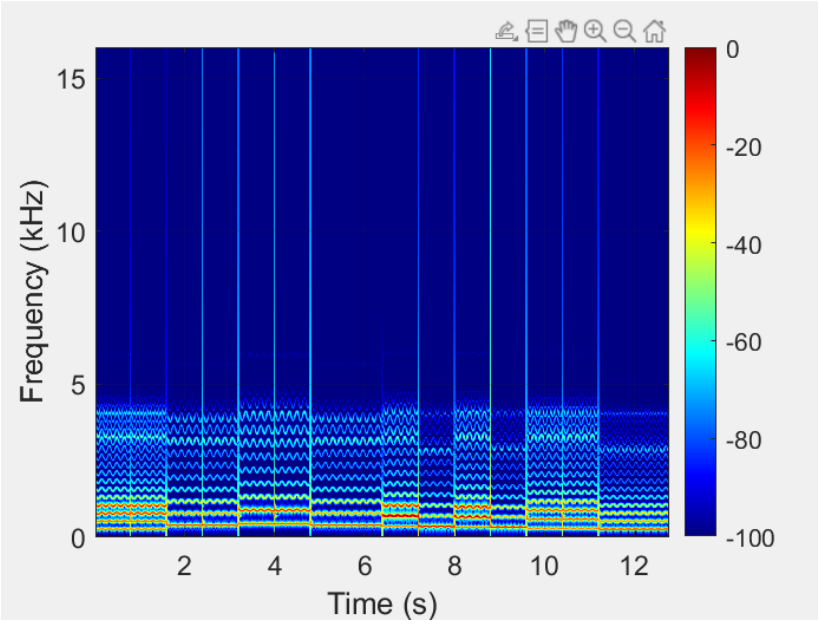
```
volume = 0.01;
% audioplayer(Y,Fs) creates an audioplayer object for signal Y, using sample rate Fs.
a = audioplayer(pOut_all*volume, fsamp);
a.play;
```

Before plotting the spectrogram, a hanning window is multiplied to attenuate the sides of the window of the signal. This is done because transients at the limits of the window will produce undesired artifacts in the spectrogram.

```
wsize=0.05; % specify window size in seconds
wsize=round(wsize*fsamp); % convert to sample size

window=hanning(wsize); % use a Hanning window for analysis
% window=rectwin(wsize);

overl=0.9;  %The amount of overlap defined as a fraction of the window size
overl=round(overl*wsize); % convert to sample size
%{
overl is the number of the overlap sample
%}

fftres=wsize*2;                         %Same considerations for the
                                        %resolution as for the spectrum
```

At the end, plot the spectrogram. The result is as below.

```
% compute the spectrogram
% Notice that we also get the corresponding frequency and time axes
[SG,Faxis,Taxis] = spectrogram(pOut_all,window,overl,fftres,fsamp);

%{
fftres is cyclical frequencies, specified as a vector. The units of f are specified by the sample rate, fs.
SG i spectrogram
Faxis is frequency axis
Taxis is time axis
%}

% plot sound level of spectrogram as an image and as a function of time and
% frequency
figure;
imagesc(Taxis,Faxis/1000,20*log10(abs(SG)./max(max(abs(SG)))));
axis xy; % set the axis the right way around
ylabel('Frequency (kHz)'); xlabel('Time (s)')
grid on;
set(gca,'FontSize',14);
colormap('jet'); % choose a color map! set help graph3d for some options
set(gca,'clim',[-100 0]); % scale the color limit for the color map
colorbar; % create a colorbar as a legend!
```



Problems and Possible Improvements

The result approaches the singing and can hear the differences between different pitches and vowels. However, I notice that the change between vowels/notes in the melody is not smooth and I can hear some impulse sounds when they change. I think that is because I generated every note respectively and concatenated them to be the whole signal. The generated note doesn't have any fade-in or fade-out so that it causes the concatenating point to have some impulse sounds. A possible improvement is to apply other filters on the resulting signal to generate the fade-in and fade-out effects. Another problem is that this singing synthesis model can only synthesize the

monophthong. In order to synthesize the diphthong, we need to implement the formant transition from one vowel to another vowel.