Fall 2021 EE 313 Linear Systems and Signals Prof. Evans

Mini-Project #1: Sinusoidal Speech Synthesis

Assigned on Monday, September 13, 2021; Part 3(c) added September 18, 2021
Due on Tuesday, September 22, 2021, by 11:59pm pm via Canvas submission
Appendices added September 23, 2021. Appendix B includes hints conveyed to the class.

Late submission is subject to a penalty of two points per minute late.

Reading: McClellan, Schafer and Yoder, *Signal Processing First*, 2003, Chapter 3.
Companion Web site with demos and other supplemental information.

E-mail address for Mr. Faris Tabbara (TA) is firas.tabbara@utexas.edu. Lecture hours and office hours for Mr. Tabbara and Prof. Evans on Zoom (see links on the Canvas calendar) follow:

Time Slot	***Monday***	***Tuesday***	***Wednesday***	***Thursday***	***Friday***
9:30 am				Evans (Zoom)	
10:00 am				Evans (Zoom)	
10:30 am					
11:00 am		Evans (EER 1.516)		Evans (EER 1.516)	
11:30 am		Evans (EER 1.516)		Evans (EER 1.516)	
12:00 pm		Evans (EER 1.516)		Evans (EER 1.516)	
12:30 pm		Evans (Zoom)			
1:00 pm		Evans (Zoom)			
1:30 pm					
2:00 pm					Evans (Zoom)
2:30 pm					Evans (Zoom)
3:00 pm					Tabbara (Zoom)
3:30 pm			Tabbara (Zoom)		Tabbara (Zoom)
4:00 pm			Tabbara (Zoom)		Tabbara (Zoom)
4:30 pm			Tabbara (Zoom)		

You may work individually or with one partner. If you work with a partner, then create one report together. Each of you would submit the same report on Canvas. Be sure that the mini-project report represents the independent work of the author(s) on the report.

Please consider using headphones for this mini-project for the sanity of those around you.

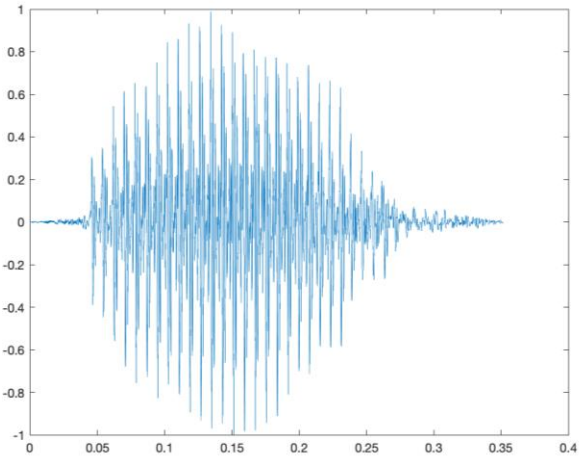
1.0 Introduction

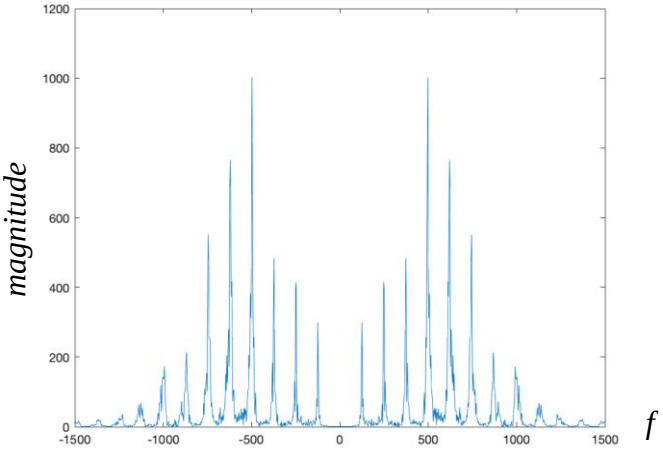
This lab will explore the use of a sum of sinusoids to synthesize a recorded vowel sound in English.

In English, vowel sounds have a nearly harmonic structure. We will model them as

$$x(t) = A_0 + \sum_{k=1}^{N} A_k \cos\left(2\pi f_k t + \phi_k\right)$$

as mentioned in Section 3.1 in *Signal Processing First* [1]. Our goal will be to take a recorded vowel sound, such as an "ah" sound,





and use enough sinusoidal terms to synthesize the vowel sound so that it is intelligible. Our first effort will to be use frequencies that are not harmonically related and then try to use ones that are.

2.0 Recording your speech and saving it to a file

There are many ways to create a digital audio signal, including recording from a microphone, reading from a file, and generating sound from a formula. In this section, we'll record our speech using a microphone, play it back, and save it to a file.

2.1 Standard Audio Sampling Rates

Use the following MATLAB code to record sound from your laptop's microphone. We'll use one of the standard audio sampling rates. Common audio sampling rates include

- 8000 Hz, 16000 Hz, and 32000 Hz for voice
- 11025 Hz and 22050 Hz for voice
- 44100 Hz for audio CD and its multiple 88200 Hz
- 48000 Hz for digital audio tape and its multiples 96000 Hz and 192000 Hz

For example, 2G cell phones had a single microphone that sampled voice at 8000 Hz; the phone company that provided voice service over landlines also sampled at 8000 Hz. Today's phones have multiple microphones and support many standard audio sampling rates. We'll sample at 8000 Hz.

2.2 Recording from a Microphone

Please copy and paste the MATLAB code below into a new script window and run it to record one second of your voice. When prompted, please utter a sustained vowel sound of your choice. Here are 15 vowel sound choices in English [2]. You could also choose a vowel sound in another language.

```
% UTAudioRecordAndPlayback.m

% Record from the microphone
fs = 8000;
numBits = 16;
numChannels = 1;
recordingTime = 1;
recObj = audiorecorder(fs, numBits, numChannels);
disp('Start recording...');
recordblocking(recObj, recordingTime);
disp('End recording.');

% Store data in double-precision floating-point array
myRecording = getaudiodata(recObj);

% Play back the recording with automatic scaling
soundsc(myRecording, fs);

% Remove DC value and normalize amplitude to [-1, 1]
myRecording = myRecording - mean(myRecording);
myRecording = myRecording / max(abs(myRecording));

% Save the data to a file
waveFilename = 'shortAudioClip.wav';
audiowrite(waveFilename, myRecording, fs );
```

2.3 Time-Domain Analysis of the Speech Segment

We now analyze the recorded speech in the time domain. Please copy and paste the following code into a new script window in MATLAB and run it.

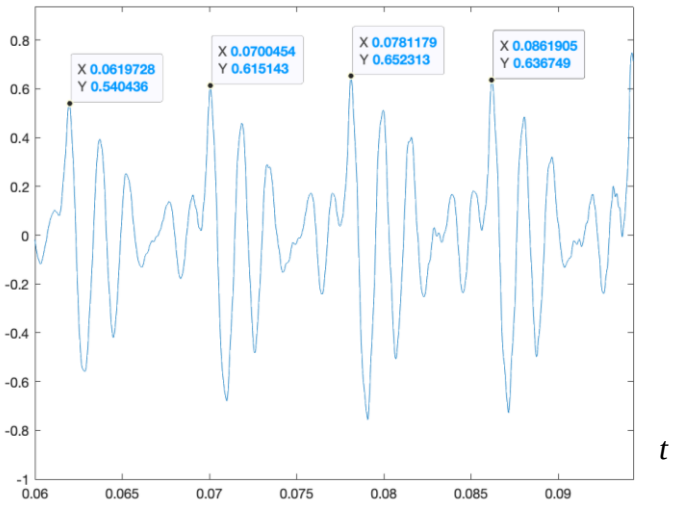
```
% UTAudioTimeDomainAnalysis.m

% Read the contents of the audio file
waveFilename = 'shortAudioClip.wav';
[myRecording, fs] = audioread(waveFilename);

% Play back the recording with automatic scaling
soundsc(myRecording, fs);

% Plot the waveform in the time domain
N = length(myRecording);
Ts = 1/fs;
t = 0 : Ts : (N-1)*Ts;
figure;
```

```
plot(t, myRecording);
xlabel('t');
```

Please submit the following
a) Vowel sound of your utterance.
b) Time-domain plot of the recording.
c) What is the average value of the signal? Use the `mean` command to validate your observation.
d) Please circle the quiet portions.
e) Estimate the pitch period per the instructions below.

Pitch Period. "Use the zoom function to zoom in to an interesting part of the [time-domain] waveform. Notice how the mouth and throat (the vocal tract) rings like a bell [damped sinusoid] each time the vocal folds close; the period between vocal fold closures is called the *pitch period*, and the frequencies at which the ringing occurs called *formant frequencies*." [3] For more information on the biological mechanisms for the human voice, please see [4].

Here's an example of determining the pitch period and hence the fundamental frequency using the "ah" recording in Section 1.0. I'll zoom into the time-domain plot to see the periodic "ringing" of the vocal tract. On the right, I've highlighted the starting times of the responses of four vocal fold closures by using the "Data Tips" tool available in the "Tools" menu in a plot window.



The plot on the right has three pitch periods that last from the first point to the fourth point, which is a total of 24.218ms for three pitch periods or 8.0727ms for one pitch period on average. For T_0 = 8.0727ms, f_0 = 123.9 Hz. The pitch frequency in a human voice is typically in the 100-300 Hz range. An individual's normal voice under non-stressful conditions would have a pitch frequency that is roughly constant.

2.4 Frequency and Time-Frequency Analysis of the Speech Segment

We can now analyze the recorded speech in the frequency and time-frequency domains. Please copy and paste the following code into a new script window in MATLAB and run it.

```
% UTAudioFreqDomainAnalysis.m

% Read the contents of the audio file
waveFilename = 'shortAudioClip.wav';
[myRecording, fs] = audioread(waveFilename);

% Plot the magnitude of the frequency content
% using a discrete-time version of the Fourier series
% Zoom the frequency axis to [-1500 Hz, 1500 Hz]
% to focus on the strongest frequency components
```

```
fourierSeriesCoeffs = fft(myRecording);
N = length(myRecording);
freqResolution = fs / N;
ff = (-fs/2) : freqResolution : (fs/2)-freqResolution;
figure;
plot(ff, abs(fftshift(fourierSeriesCoeffs)));
xlabel('f');
xlim( [-1500, 1500] );

% Plot the spectrogram
figure;
blockSize = round(N/4);
overlap = round(0.875 * blockSize);
spectrogram(myRecording, blockSize, overlap, blockSize, fs, 'yaxis');
```

Please submit the following:

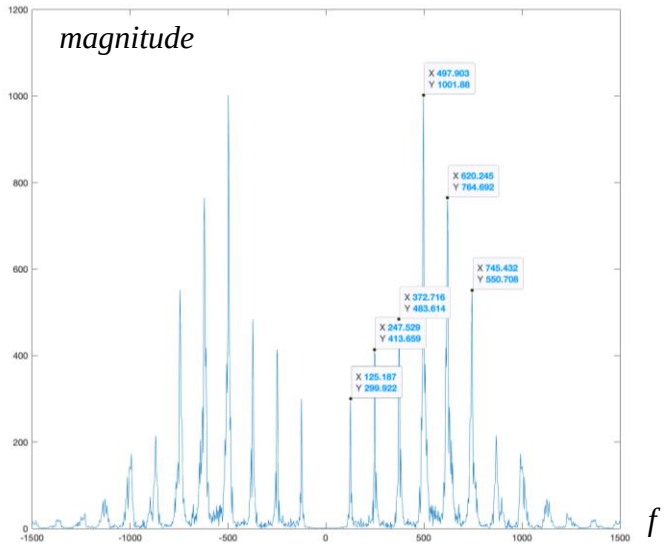
a) Frequency-domain plot of the recording.
b) Spectrogram of the recording.
c) From the spectrogram, give the approximate value of the strongest positive frequency component.
d) From the frequency-domain plot, please give the gain, frequency, and phase corresponding to the highest peak in positive frequencies. You'll likely need to modify the code for the computation of the frequency content. Explain how you determined the gain, frequency, and phase values, including any MATLAB code you've written. Please see Appendix A for more information.

3.0 Synthesizing the Vowel Sound

Based on your analysis of the speech utterance recorded in the previous section, you'll next try to synthesize the speech utterance using a sum of sinusoids.

$$x(t) = A_0 + \sum_{k=1}^{N} A_k \cos\left(2\pi f_k t + \phi_k\right)$$

On the right is a closer look at the frequency domain representation for the "ah" sound in the introduction. I've noted the frequencies at which the first six positive peaks in the magnitude plot occur.



For your recording, synthesize the vowel sound using the sum of sinusoids formula for $x(t)$:

(a) The frequencies $f_1, f_2, \ldots, f_N$ do not have to be harmonically related. Please use the positive frequencies at the peaks. Please plot the recorded vowel sound as well as the sinusoidal model $x(t)$ for $N \in \{ 1, 2, 4, 6, 8, 10 \}$. Please play them for your lab partner to see if you lab partner can understand the vowel sound. At what value of N is the vowel understandable (if any)?
(b) The frequencies $f_1, f_2, \ldots, f_N$ should be harmonically related. Please use the positive frequencies at the peaks. Please plot the recorded vowel sound as well as the sinusoidal model $x(t)$ for $N \in \{ 1, 2, 4, 6, 8, 10 \}$. Please play them for your project partner to see if they can understand the vowel sound. At what value of N is the vowel understandable (if any)? Please see below for more info.
(c) Use the UTAudioSynthUsingInvFFT.m script below to determine the smallest number of positive

and negative frequencies needed to synthesize the vowel sound through trial-and-error by changing the value of `Nkeep`. The script keeps the `Nkeep` strongest frequency components.

```
% UTAudioSynthUsingInvFFT.m
% Needs fourierSeriesCoeffs vector from UTAudioFreqDomainAnalysis.m

Nseries = length(fourierSeriesCoeffs);
fourierSeriesCoeffsAbs = abs(fourierSeriesCoeffs);

% Nkeep must be even to have an equal number of negative and positive freq.
Nkeep = 10;
synthSoundCoeffs = zeros(Nseries, 1);

% Find the Nkeep strongest positive and negative frequency components
for n = 1:Nkeep
    [ak, k] = max(fourierSeriesCoeffsAbs);
    synthSoundCoeffs(k) = fourierSeriesCoeffs(k);
    fourierSeriesCoeffsAbs(k) = 0;
end

% Convert Fourier series coefficients to time domain using inverse FFT
synthSound = ifft(synthSoundCoeffs);
soundsc(synthSound, fs);
```

Analysis for part (b). We can use the "ah" frequency domain plot on the previous page as an example of how to proceed. Note that the first peak on the above frequency-domain plot occurs at 125.187 Hz, which is roughly equal to the pitch frequency f_0. The five other peaks are approximately (but not exactly) harmonics of a common frequency. If we assume that the frequencies are harmonically related, we can estimate the pitch frequency f_0 by using the six frequencies:

$1f_0 = 125.187$
$2f_0 = 247.529$
$3f_0 = 372.716$
$4f_0 = 497.903$
$5f_0 = 620.245$
$6f_0 = 745.432$

```
freqs = [125.187, 247.529, 372.716, 497.903, 620.245, 745.432];
f0est = mean( freqs ./ [1 2 3 4 5 6] );
```

The estimated value for f_0 which is `f0est` is 124.3 Hz.

References

[1] James H. McClellan, Ronald W. Schafer & Mark A. Yoder, *Signal Processing First*, Prentice-Hall, ISBN 978-0130909992, 2003. Errata. On-line Companion.
[2] "Introduction to English Vowel Sounds", accessed Sept. 14, 2021.
[3] Mark Hasegawa-Johnson, "Lab 1", *ECE 498H Signal & Image Analysis*, University of Illinois Urbana-Champaign, Fall 2014.
[4] "Human Voice", *Wikipedia*, accessed Sept. 14, 2021.
[5] Robert J. McAulay and Thomas F. Quatieri, "Speech Analysis/Synthesis Based on a Sinusoidal Representation", *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-34, no. 4, Aug. 1986, pp. 744-754.

Appendix A: Connections Between Continuous-Time and Discrete-Time Fourier Series

Continuous-Time Fourier Series. A continuous-time periodic signal $x(t)$ with period T_0 sec is composed of a constant term plus frequency components at integer multiples (harmonic) of a fundamental frequency f_0 where $f_0 = 1 / T_0$. Fourier series analysis computes the constant term a_0 plus the magnitude and phase of each frequency term a_k where k is any integer:

$$x(t) = \sum_{k=-\infty}^{\infty} a_k e^{j2\pi k f_0 t} \quad \text{where} \quad a_0 = \frac{1}{T_0}\int_0^{T_0} x(t)\,dt \quad \text{and} \quad a_k = \frac{1}{T_0}\int_0^{T_0} x(t) e^{-j2\pi k f_0 t}\,dt$$

An *infinite* number of coefficients could be needed to exactly represent $x(t)$.

Discrete-Time Fourier Series. A discrete-time periodic signal $x[n]$ with period N samples is composed of a constant term plus frequency components at integer multiples (harmonic) of a fundamental frequency of $2\pi/N$ rad/sample which is equivalent to f_s / N in Hz. Fourier series analysis computes the constant term $X[0]$ plus the magnitude and phase of each frequency term $X[k]$ for $k = 0, 1, \ldots, N\text{-}1$.

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]\, e^{j\left(k\frac{2\pi}{N}\right)n} \quad \text{where} \quad X[0] = \sum_{n=0}^{N-1} x[n] \quad \text{and} \quad X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j\left(k\frac{2\pi}{N}\right)n}$$

A *finite* number of Fourier series coefficients would always be needed to exactly represent $x[n]$.

Normalization. The scaling of the Fourier series coefficients is different in continuous-time vs. discrete-time. When using the discrete-time Fourier series to compute continuous-time Fourier series coefficients, we would have to divide the discrete-time Fourier series coefficient by N due the $(1/N)$ term in the equation for $x[n]$.

Fast Fourier Transform (FFT) is a fast algorithm to compute the N discrete-time Fourier series coefficients $X[k]$ from the N samples of a discrete-time signal $x[n]$. As with the continuous-time Fourier series, the discrete-time Fourier series assumes that the N samples of $x[n]$ represents the fundamental period of an infinitely long signal in the time domain. Unlike the continuous-time Fourier series, the discrete-time Fourier series always has a finite number of terms, N. One of the reasons for this is due to the Sampling Theorem, which says that the sampling rate $f_s > 2\, f_{max}$ where f_{max} is the highest frequency of interest and hence $f_{max} < ½\, f_s$. Sampling only captures continuous-time frequencies $(-½\, f_s, ½\, f_s)$ whereas continuous-time signals have frequencies. You can think of the discrete-time Fourier series as computing the Fourier series coefficients for frequency components from $-½\, f_s$ to $-½\, f_s$, which is a finite number because $f_s > 0$.

How fast is the FFT? The FFT to compute $X[k]$ above requires $2M \log_2 M$ real-valued multiplications and additions and $4M$ words of memory instead of the $4\, M^2$ and $M^2 + 4M$, respectively, for the direct matrix-vector implementation of the discrete-time Fourier series. A direct matrix-vector multiplication to compute $X[k]$ would create a complex-valued N x N matrix of the term $\exp(-j\ (2\pi/N)\ kn)$ for $k = 0, 1, \ldots, N\text{-}1$ in one dimension and $n = 0, 1, \ldots, N\text{-}1$ in the other, form a column vector of $x[0], x[1], \ldots, x[N\text{-}1]$, and multiply the matrix and vector to find the column vector of $X[0], X[1], \ldots, X[N\text{-}1]$.

Appendix B: Report Format and Hints for Mini-Project #1

Report Format: Please write a self-contained narrative report. The audience is the other students in the class. The report should provide references to the textbook and other sources as needed. Please refer to the hints above, which apply to homework assignments and mini-project reports, as well as the following additional guidelines for the mini-project:

1. Introduction -- explain in your own words and and with appropriate references the larger topic of project. Build on your experiences so far in the class. You can also use ideas from the Introduction section in the lab assignment. Probably half of a page for this section.
2. Overview -- explain in your own words and with appropriate references the general idea of the topic of mini-project, including mathematical models. You can also use ideas from other sections in the lab assignment. Not more than a page for this section.
3. Analyzing a vowel sound -- provide the MATLAB code and answers to the questions in Section 2.
4. Synthesizing the vowel sound -- provide the MATLAB code and answers to the questions in Section 3.
5. Conclusion -- draw conclusions from your work and explanations in the earlier sections. Probably half of a page for this section.

Section 2.2 Recording the vowel sound. When recording the vowel sound, we're using a standard speech sampling rate of 8000 Hz or equivalently 8000 samples/s. Since we record for one second, we have 8000 samples in our recorded data. We save the recorded data as a wave file so that no data is lost. That is, a wave file is a lossless format where the original audio samples are fully retained. This is not the case for lossy format such as MPEG-1 layer 3 (mp3) or MPEG-4 audio.

Section 2.3 Time-Domain Analysis. The MATLAB code introduces a variable N for the number of samples in the recorded speech. This will clash with the N used in the sum of sinusoids formula that appears in Sections 1 and 3.

In this problem, you'll estimate the pitch period of the recorded voice. The pitch period is the fundamental period T_0. We can invert the pitch period to obtain the pitch frequency which is also the fundamental frequency f_0. You'll be able to validate the estimate for by using the frequency-domain analysis in section 2.4. Typically, the first peak in the plot of the magnitude of the Fourier series coefficients for positive frequencies is f_0.

Section 2.4 Frequency-Domain Analysis. When computing the Fourier series coefficients for a continuous-time signal, we assume that the observed signal is the fundamental period of a periodic signal, whether it really is or not. This is also the case when working with a discrete-time signal, as we're doing with the recorded audio signal.

To compute the exponential Fourier series coefficients for our discrete-time signal, we'll use the Fast Fourier Transform (FFT). The FFT takes the vector of N samples for the recorded signal, and returns the N exponential Fourier series coefficients. In the discrete-time case, we have a

finite number of Fourier series coefficients unlike the continuous-time Fourier series which has infinite terms.

Here's the use of the fft function in MATLAB in line 11 of UTAudioFreqDomainAnalysis.m

```
fourierSeriesCoeffs = fft(myRecording);
```

In `fourierSeriesCoeffs` vector, the first half contains the Fourier series coefficients for non-negative frequencies, and the second half contains the Fourier series coefficients for negative frequencies. The more detailed format of the vector follows, keeping in mind that the first element in a vector in MATLAB is at index 1 and `fourierSeriesCoeffs` has $N = 8000$ discrete-time Fourier series coefficients X_k. We are going to ignore the normalization between discrete-time and continuous-time Fourier series coefficients (see Appendix A) and handle the difference at audio playback using `soundsc` command:

- At index 1, it contains $X_0 = a_0$. Should be real-valued and relatively small in magnitude.
- At index 2, it contains $X_1 = a_1$ for frequency $f_s / N = 1$ Hz.
- At index 3, it contains $X_2 = a_2$ for frequency $2\ f_s / N = 2$ Hz.
- At index 4000, it contains $X_{3999} = a_{3999}$ for frequency $3999\ f_s / N = 3999$ Hz.
- At index 4001, it contains $X_{-4000} = a_{-4000}$ for frequency $-4000\ f_s / N = -4000$ Hz.
- At index 8000, it contains $X_{-1} = a_{-1}$ for frequency $-1\ f_s / N = -1$ Hz.

For example, we extract the exponential Fourier series coefficient for 246 Hz by accessing the 247th element of the vector `fourierSeriesCoeffs`, which is a complex number.

When plotting the Fourier series coefficients computed by the FFT, we use the FFT shift command `fftshift` to swap the halves of the `fourierSeriesCoeffs` vector.

3.0 Synthesizing the Vowel Sound. Each gain A_k and phase f_k are obtained from the appropriate element of the array `fourierSeriesCoeffs`.

When using the FFT, the gains for A_k will need to be multiplied by $1/T_0$ where T_0 is the fundamental period (a.k.a. pitch period).

A common approach to parts (a) and (b) is to identify frequencies of the 10 peaks in the plot of the magnitude of the Fourier series coefficients where the frequencies are close to being harmonically related. For each frequency, one can look up the Fourier series coefficient in the vector `fourierSeriesCoeffs` and here's how. First, we've recorded 1s of speech at a sampling rate of 8000 samples/s, which gives a total of 8000 samples. Because of this, the element at index n+1 in `fourierSeriesCoeffs` corresponds to a frequency of n Hz for n in [0, 3999]. Once we have the Fourier coefficient, we can compute its magnitude A_k using the abs command and its phase f_k using the angle command. For part (b), after computing the magnitude and phase of the Fourier series coefficient, each frequency would be changed to be a harmonic of an estimated value of the fundamental frequency f_0.