Fall 2023 EE 313 Linear Systems and Signals Prof. Evans

Mini-Project #1: Music Synthesis

By Prof. Brian Evans and Mr. Dan Jacobellis

Assigned on Tuesday, September 12, 2023, 7:00am
Due on Tuesday, September 18, 2023, by 11:59pm pm via Canvas submission

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: http://dspfirst.gatech.edu/

E-mail address for TA Elyes Balti is ebalti@utexas.edu. Please consider posting questions on Ed Discussion, which can be answered by anyone in the class. You can post anonymously.

Lecture and office hours for Mr. Balti and Prof. Evans follow:

Time Slot	***Monday***	***Tuesday***	***Wednesday***	***Thursday***	***Friday***
11:00 am		**Evans (ECJ 2.104)**		**Evans (ECJ 2.104)**	
11:30 am		**Evans (ECJ 2.104)**		**Evans (ECJ 2.104)**	
12:00 pm		**Evans (ECJ 2.104)**		**Evans (ECJ 2.104)**	
12:30 pm					
1:00 pm					
1:30 pm					
2:00 pm			**Evans (EER 6.882 and Zoom)**	**Evans (EER 6.882 and Zoom)**	**Balti (EER 3.648)**
2:30 pm			**Evans (EER 6.882 and Zoom)**	**Evans (EER 6.882 and Zoom)**	**Balti (EER 3.648)**
3:00 pm			**Evans (EER 6.882 and Zoom)**	**Evans (EER 6.882 and Zoom)**	**Balti (EER 3.648)**
3:30 pm		**Balti (EER 3.648)**			
4:00 pm		**Balti (EER 3.648)**			
4:30 pm		**Balti (EER 3.648)**			
5:00 pm				**Balti (EER 3.648)**	
5:30 pm				**Balti (EER 3.648)**	
6:00 pm				**Balti (EER 3.648)**	

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.

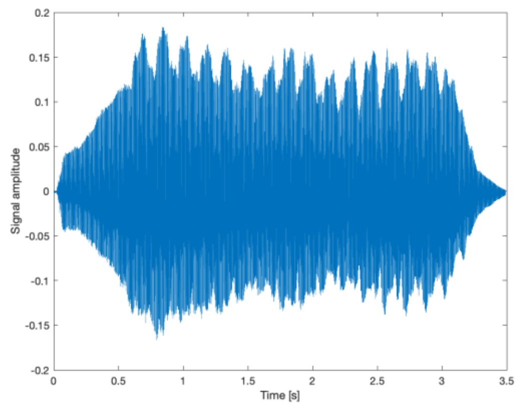
Please consider using headphones for this miniproject for the sanity of those around you.

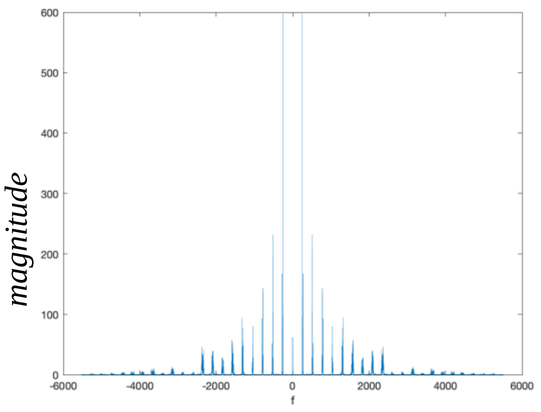
1.0 Introduction

This lab will explore the use of a sum of sinusoids to analyze and synthesize a recorded note played by a musical instrument. We will model the recorded sound 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 sound and use enough sinusoidal terms to synthesize the sound so that it is recognizable. Our first effort will to be use frequencies that are not harmonically related and then try to use ones that are.





2.0 Signal Download, Playback, and Analysis

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 download a recording of an acoustical instrument, play the recording, and analyze the recording in the time and frequency domains.

2.1 Download the Recording

Prof. Dan Ellis at Columbia University has created a collection of single notes of different principal frequencies played by different acoustic instruments at

https://www.ee.columbia.edu/~dpwe/sounds/instruments/

From the page: "The examples are excerpted from The McGill University Master Samples collection, a fabulous set of CDs of instruments playing every note in their range, recorded in studio conditions. The examples below have been downsampled to 11 kHz mono, but the original data is much higher quality."

Please download the recording 'violin-C4.wav' of a violin playing 'C' in the fourth octave ('C4') on the Western scale. Please place the file in your MATLAB directory or in a directory on your MATLAB path. The principal frequency of 'C4' is 261.63 Hz.

2.2 Time-Domain Analysis

We will now analyze the recorded acoustic sound (audio signal) in the time domain.

Please play and plot the audio signal by running the following MATLAB code in a new script window:

```
% Read the contents of the audio file
waveFilename = 'violin-C4.wav';
[instrumentSound, fs] = audioread(waveFilename);

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

% Plot the waveform in the time domain
N = length(instrumentSound);
Ts = 1/fs;
Tmax = (N-1)*Ts;
t = 0 : Ts : Tmax;
figure;
plot(t, instrumentSound);
xlabel('Time [s]');
ylabel('Signal amplitude');
```

Please submit the following

a) The time-domain plot of the recording generated by the above MATLAB code. In the MATLAB figure window, you can save the plot as a PNG file that can include in your writeup.

b) What is the average value of the signal? Use the `mean` command to validate your observation.

c) Estimate the principal frequency of the note being played from the time-domain plot. Zoom the time-domain plot to 1.5s to 1.6s. You can do this using the MATLAB command

```
xlim( [1.5 1.6] );
```

or the Zoom tool which is under the Tools menu in the figure window. To estimate the principal frequency, count the number of periods, estimate the fundamental period by dividing the duration of time by the number of periods, and invert the estimate of the fundamental period. Use the Data Tips tool to click on the start of the first period and the end of the last period to estimate the duration of time. See the next page for an example.

   i. Submit the zoomed plot from 1.5s to 1.6s with the two Data Tips visible.

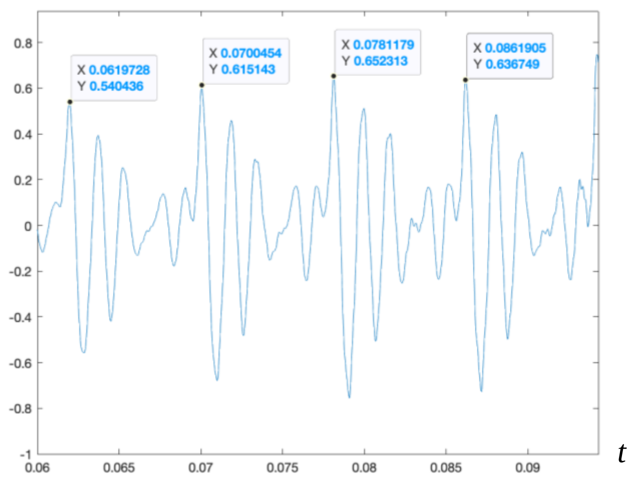
   ii. How does your estimate of the principal frequency compare with the principal frequency of a ‘C4’ note?

d) What is the sampling rate of the original recording in the The McGill University Master Samples collection? What is the sampling rate of recording in the file ‘violin-C4.wav’? What’s the ratio between these sampling rates?

Estimating the Fundamental Period

Here's an example of determining the fundamental period using a speech utterance. On the right, I've highlighted the starting times of four fundamental periods by using the "Data Tips" tool available in the "Tools" menu in a plot window.

The plot on the right has three fundamental periods that last from the first point to the fourth point, which is a total of 24.218ms for the three fundamental periods or 8.0727ms for each period on average. For T_0 = 8.0727ms, f_0 = 123.9 Hz.



2.3 Frequency and Time-Frequency Analysis

We can now analyze the recorded instrument in the frequency and time-frequency domains. For the frequency domain analysis, we'll use the Fast Fourier Transform (FFT) algorithm to compute the Fourier series. See Appendix A for the connection between the FFT and the Fourier series.

Please copy and paste the following code into a new script window in MATLAB and run it.

```
% Read the contents of the audio file
waveFilename = 'violin-C4.wav';
[instrumentSound, fs] = audioread(waveFilename);

% Plot the magnitude of the frequency content
% using a discrete-time version of the Fourier series
fourierSeriesCoeffs = fft(instrumentSound);
N = length(instrumentSound);
freqResolution = fs / N;
ff = (-fs/2) : freqResolution : (fs/2)-freqResolution;
figure;
plot(ff, abs(fftshift(fourierSeriesCoeffs)));
xlabel('f');

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

Please submit the following:

a) Frequency-domain plot of the recording.
b) From the frequency-domain plot, give the gain, frequency, and phase corresponding to the highest peak in positive frequencies. You'll likely need to modify the above code. Explain how you determined the gain, frequency, and phase values, including any MATLAB code you've written.
c) Spectrogram of the recording.

d) From the spectrogram, for what range of time is the principal frequency present?

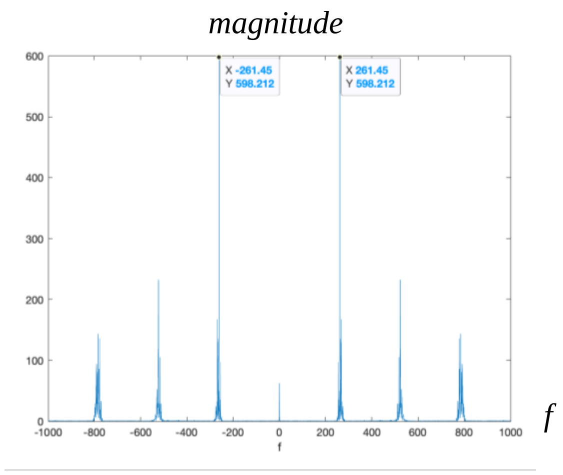
3.0 Synthesizing the Sound

Based on your analysis of the audio recording in the previous section, you'll next try to synthesize the sound using a finite sum of sinusoids given on the right. One approach is to identify the peaks in the spectrum.

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

(a) We can eyeball the peaks in the spectrum and use the Data Tips tool to find out which frequencies to use. On the right, the magnitude spectrum is zoomed to -1000 Hz < f < 1000 Hz and the Data Tips tool highlights the pair of largest peaks as a starting point. We would then need to find where the frequency of 261.45 Hz and -261.45 Hz fall in `fourierSeriesCoeffs` vector. From Appendix A, a frequency f_k = 261.45 Hz would correspond to $f_k = k\frac{f_s}{N}$ which means $k = N\frac{f_k}{f_s} = 38500\ \frac{261.45\text{ Hz}}{11025\text{ Hz}} = 913$ and f_k = -261.45 Hz would correspond to $f_k = f_s\left(\frac{k}{N} - 1\right)$ which means

$$k = N\left(\frac{f_k}{f_s} + 1\right) = 38500\ \left(\frac{-261.45\text{ Hz}}{11025\text{ Hz}} + 1\right) = 37587$$



Please keep in mind that MATLAB vectors start indexing at 1 instead of 0. So, k = 913 means an index of 914 into the `fourierSeriesCoeffs` vector.

i. Manually synthesize the audio signal by using the two largest peaks and then the four largest peaks.
ii. Describe what each synthesized signal sounds like.

(b) We can automate how many of the largest peaks to keep by using the code below which keeps the `Nkeep` strongest frequency components.

```
% Needs fourierSeriesCoeffs vector computed in Section 2.3 above
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);
```

i. Keep 2, 4, 6, and 8 of the largest peaks and describe what you hear.
ii. Give the peak frequencies of the peaks for each case of keeping 2, 4, and 6 largest peaks.

(c) The algorithm in part (b) will keep the peaks at -261.45 Hz and 261.45 Hz, and then keep the second highesv peaks at -261.736 Hz and 261.736 Hz. The frequency components at -261.736 Hz and 261.736 Hz will be “masked” (i.e. not audible) by the stronger nearby frequency components at -261.45 Hz and 261.45 Hz, respectively. [2] To prevent masked frequencies from being kept, change

```
fourierSeriesCoeffsAbs(k) = 0;
```

in the above code to zero out all the frequencies from the peak frequency minus 7.5 Hz to the peak frequency plus 7.5 Hz

```
fourierSeriesCoeffsAbs(kmin:kmax) = 0;
```

The frequency masking range is chosen arbitrarily to match the bandwidth of a C4 note.

i. Give the code to compute `kmin` and `kmax`. In finding `kmin` and `kmax`, please note that the frequency resolution for the Fast Fourier Transform is f_s/N.
ii. Use the algorithm to keep 2, 4, 6, and 8 of the largest peaks and describe what you hear.
iii. Give the frequencies of the peaks when keeping 2, 4, 6, and 8 of the largest peaks.

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] “Auditory Masking”, Wikipedia, Accessed Sep. 12, 2023.

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 $k = 1, \ldots, N-1$.

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

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

In the Fourier series coefficients $X[k]$, index k corresponds to continuous-time frequency $f_k = k\frac{f_s}{N}$ in Hz for $k = 0, 1, \ldots, \frac{N}{2} - 1$ and $f_k = f_s\left(\frac{k}{N} - 1\right)$ for $k = \frac{N}{2}, \frac{N}{2} + 1, \ldots, N - 1$, assuming N is even.

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 $f_s > 0$.

The Fast Fourier Transform (FFT) is 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. The direct matrix-vector implementation to compute $X[k]$ would create a complex-valued N x N matrix of the term $\exp(-j\ (2\pi/N)\ kn)$ for $k = 0, 1, \ldots, N-1$ in one dimension and $n = 0, 1, \ldots, N-1$ in the other, form a column vector of $x[0], x[1], \ldots, x[N-1]$, and multiply the matrix and vector to find the column vector of $X[0], X[1], \ldots, X[N-1]$.

Appendix B: Recording from a Microphone

If you would like to record your own sounds, please copy and paste the MATLAB code below into a new script window and run it to record three seconds from your microphone.

```matlab
% Record from the microphone
fs = 48000;
numBits = 16;
numChannels = 1;
recordingTime = 3;
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 );
```

Appendix C: Homework and Mini-Project Guidelines

Here are some things you should follow for all assignments.

Amount of work to show:

1. An explanation should be given for every single answer. Answers written without explanation will lose two-thirds of the points allotted for that part.
2. Only "standard" formulas (like Euler's formula, trigonometric formulas, etc.) can be used without a reference. If you're using something non-standard, then please put a reference to the formula number in the book, or whatever source you got it from. Just using the final result of a similar problem done in the class, and omitting the intermediate steps, is not okay. You have to show show your work.
3. There shouldn't be big jumps in logic from one step to the next.
4. For everything, expect to show at least one intermediate step between the first line and the answer. Even if it seems unnecessary to you, please err on the side of caution. Things that seem obvious to you when you're writing the solution are not quite so obvious for someone reading it.
5. If you're in any doubt about how much work to show, please ask the instructor or the teaching assistant.

MATLAB source code guidelines:

1. Put a comment before the solution of each part, telling the question number of the solution.
2. If you're using complicated logic, leave a comment telling what that block of code is supposed to do.
3. Use variable names that related to their meaning/use.
4. Avoid using two different variables for the same thing.
5. Try to avoid using "magic numbers" in the code. If you're using a number, write a comment telling me how you derived it.
6. Make sure that your code will compile & run in a clean workspace; i.e., one without any variables present. Use a clear all; at least once before submitting it.
7. No marks will be deducted based on the efficiency of the code unless the problem asks you to write efficient code.

Technical points:

1. Merge all the files together into one PDF file.
2. Please adjust the contrast, exposure etc., to get a good scan quality so that the TA can easily read what you write. Take extra care to get a good scan for parts written in pencil.
3. For the MATLAB code you write for an assignment, please copy the code into Word or include a screenshot showing the code. Do not submit handwritten code.

Other things:

1. All plots must have axis labels, with units.
2. Final answers must be boxed, or underlined or otherwise differentiated from the rest of the solution.
3. All final answers must have units, if they exist.
4. Read the questions carefully.
5. Try to answer all parts of a question together. If the solution to some parts of a question is written elsewhere, then leave a note telling the reader where to find it.

Organization of a mini-project report:

Please write a self-contained narrative report. The audience is someone who has taken the equivalent of this 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 when a sum of sinusoids can be used to analyze and synthesize a signal. Build on your experiences so far in the class. You can also use ideas from the

Introduction section in the mini-project assignment. Use appropriate references. Probably half of a page for this section.

2. Overview -- explain in your own words and with appropriate references the general approach for representing a signal as a sum of sinsuoids in the mini-project, including mathematical formulas. You can also use ideas from other sections in the mini-project #1 assignment. About a page for this section.
3. Analyzing an audio signal -- answer the questions in Section 2 of the assignment and use information from Section 2 in your writeup.
4. Synthesizing an audio signal -- answer the questions in Section 3 of the assignment. and use information from Section 3 in your writeup.
5. Conclusion -- draw conclusions from your work and explanations in the earlier sections. Probably half of a page for this section.

This mini-project report is something that you could bring with you on interviews to show as an example of your work. Here are example mini-project #1 reports written by the instructors on "FM Synthesis for Musical Instruments" (2018) and "Sinusoidal Speech Synthesis" (2021).