AA 321 — Aerospace Laboratory
Lab Eight: Beam Vibrations

University of Washington

February 28, 2021

1 A quick introduction to spectral analysis

A ubiquitously utilized tool in vibration theory is spectral analysis. The main character is the Fourier series, whose central idea is that nearly any periodic waveform[1] $f(x)$ can be decomposed into a series of sine and cosine functions, with convergence in the limit of an infinite number of terms. If the period of $f(x)$ is L (that is, $f(x+L) = f(x)$), then

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(k_n x) + b_n \sin(k_n x) \tag{1}$$

is its Fourier series, where the coefficients a_n, b_n depend on the particular function $f(x)$ and the frequencies $k_n = \frac{2\pi}{L} n$ are called "wavenumbers" (because they tell you the number of wave periods in a length L). This works because $\sin(x)$ and $\cos(x)$ form a complete set of orthogonal eigenfunctions for one-dimensional intervals. Let's now go further into this point.

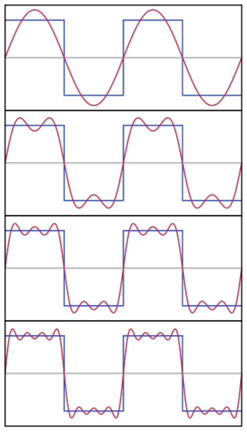


Figure 1: First four partial sums of the Fourier series for the rectangular wave [Wiki].

[1] That is, one satisfying minimally restrictive requirements called the Dirichlet conditions.

The space that a function lives in is characterized by certain eigenfunctions (*eigen* means "own" or "inherent", so read as "characteristic functions" of the space). The eigenfunctions are a *basis* for the space. In the same way that vector components are found by projection onto basis vectors, a function's components are found by projecting onto the eigenfunctions (Fig. 2). Just like basis vectors there are many possible choices of function basis. Of course, the most convenient ones are *orthogonal* and the sines and cosines have just this property.

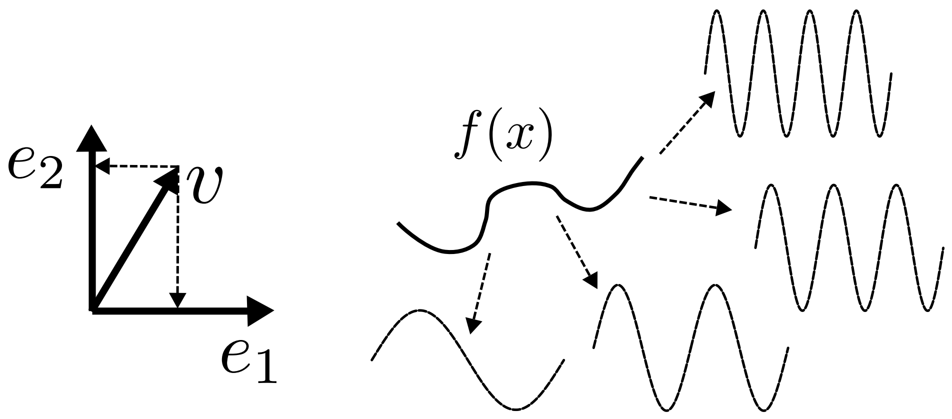


Figure 2: The Fourier coefficients a_n, b_n of $f(x)$ are projections to an orthogonal basis, in the very same way that vector components of v are projections onto basis vectors e_1 and e_2.

What do we mean by orthogonal functions? With vectors, orthogonality means that the measure of one against the other (using the inner product) is zero. For functions the notion of "measuring one against the other" is integration over the interval $x \in [0, L]$,

$$\langle \sin(k_n x) | \sin(k_m x) \rangle = \int_0^L \sin\left(\frac{2\pi}{L} n x\right) \sin\left(\frac{2\pi}{L} m x\right) dx = \begin{cases} \frac{L}{2} & n = m \\ 0 & n \neq m \end{cases} \tag{2}$$

making $\sin(k_n x)$ a family of orthogonal functions. This means that measuring one member against another gives zero, except for itself (try it for $L = 2\pi$ and $n = 1$ to convince yourself). Using orthogonality, the coefficients in Eqn. 1 are found by the integrals (the projections)

$$a_n = \frac{2}{L} \int_0^L f(x) \cos(k_n x) dx, \tag{3}$$

$$b_n = \frac{2}{L} \int_0^L f(x) \sin(k_n x) dx. \tag{4}$$

If the math seems heavy, know that its generally more important to understand what these equations *mean* than to actually do the calculations! There are many tools for calculations.

Now, the set of Fourier coefficients a_n, b_n is called the *spectrum* of the function and gives a measure of "how much" of the signal is composed of the frequencies k_n. For example, if you record yourself playing a note F# on a guitar (370 Hz) and then calculate the coefficients a_n, b_n of the waveform in the music file then there will be a huge peak around 370 Hz. Let's now go into some background on how Matlab will calculate these Fourier coefficients.

1.1 Complex form of the Fourier series

Matlab does not calculate the integrals of Eqns. 3-4, instead it approximates it using a little fancier theory. One can rewrite the Fourier series in the form

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{ik_n x}, \tag{5}$$

$$c_n = \frac{1}{L}\int_0^L f(x) e^{-ik_n x} dx \tag{6}$$

by using Euler's identity $e^{ix} = \cos(x) + i\sin(x)$. The notation $\sum_{n=-\infty}^{\infty}$ means to add all terms in the pairs $n = 0, \pm 1, \pm 2, \pm 3, \cdots$. For example,

$$f(x) = c_0 + c_{-1}e^{-ik_1 x} + c_1 e^{ik_1 x} + c_{-2}e^{-ik_2 x} + c_2 e^{ik_2 x} + \cdots . \tag{7}$$

The relationship of these coefficients c_n to the sine and cosine ones is that $c_n = \frac{1}{2}(a_n - ib_n)$. This form of the Fourier series is much more economical and the Fourier integral Eqn. 6 is generally easier to calculate than Eqns. 3-4. If $f(x)$ is a real function, then all the imaginary parts will cancel out in the summation. Again, understanding what this equation means is more important than doing calculations with it. One is representing $f(x)$ in terms of its orthogonal components, the "wiggles" $b_n \sin(k_n x)$ and $a_n \cos(k_n x)$.

1.2 The "discrete Fourier transform"

Suppose you are analyzing a music clip (or beam vibrations) and want to know which frequencies are present. Now the "Fourier integral" Eqn. 6 can't be done because the function $f(x)$ is not known. Instead, one has a *time series* of data points, $f = [f_1, f_2, f_3, \cdots, f_N]$ with some sampling rate Δx. Using this time series, one can approximate Eqn. 6 using the simplest possible integral approximation, just adding up all of the integrand values,

$$c_n = \frac{1}{L}\int_0^L f(x)e^{-ik_n x}dx \approx \frac{1}{L}\sum_{j=1}^{N} f_j e^{-ik_n x_j}\Delta x = \frac{\Delta x}{L}\sum_{j=1}^{N} f_j e^{-ik_n x_j} \tag{8}$$

assuming a constant sampling rate. Because $L = N\Delta x$ and the phase factor simplifies like

$$-ik_n x_j = -i\frac{2\pi}{L}n\Delta x j = -i\frac{2\pi}{N\Delta x}n\Delta x j = -i2\pi n\frac{j}{N} \tag{9}$$

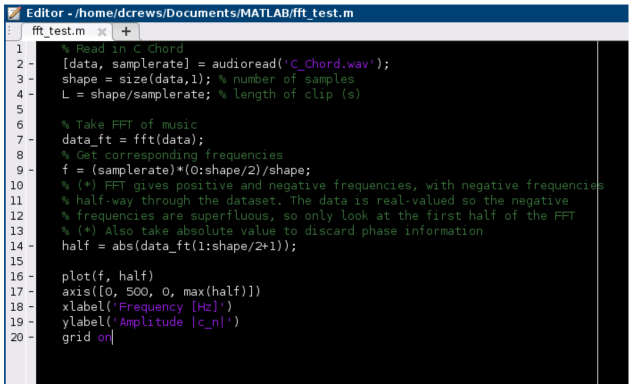
the result, called the discrete Fourier series coefficients, is usually written as

$$c_n = \frac{1}{N}\sum_{j=1}^{N} f_j e^{-i2\pi nj/N} \tag{10}$$

and is simply a summation over the data points with corresponding oscillator functions e^{ix}. It is a projection onto the orthogonal basis in the same way as before, just using a discrete set of data points! Now even a summation is pretty slow if the number of terms N is large (like many millions for typical data files). Matlab calculates Eqn. 10 using a clever trick called the *fast Fourier transform* or FFT, which breaks the summation down into smaller chunks. Fortunately, all you have to do is call `fft(X)` for `X` your data array!

1.3 A quick Matlab FFT tutorial

Here is a quick "how to" take an FFT in Matlab. Probably the most confusing part is assembling the correct vector of frequencies (note: there is a function in Python that does this for you...!). This example (Fig. 3) reads in a music file (a 9-second clip of a single C chord on the guitar) and analyzes its frequencies. To find the sample frequencies, we need



```matlab
% Read in C Chord
[data, samplerate] = audioread('C_Chord.wav');
shape = size(data,1); % number of samples
L = shape/samplerate; % length of clip (s)

% Take FFT of music
data_ft = fft(data);
% Get corresponding frequencies
f = (samplerate)*(0:shape/2)/shape;
% (*) FFT gives positive and negative frequencies, with negative frequencies
% half-way through the dataset. The data is real-valued so the negative
% frequencies are superfluous, so only look at the first half of the FFT
% (*) Also take absolute value to discard phase information
half = abs(data_ft(1:shape/2+1));

plot(f, half)
axis([0, 500, 0, max(half)])
xlabel('Frequency [Hz]')
ylabel('Amplitude |c_n|')
grid on
```

Figure 3: A quick test script to calculate the FFT a music clip.

to form the wavenumbers $k_n = \frac{2\pi}{L} n$. The FFT calculates the wavenumbers up to a certain cut-off rate called the Nyquist frequency to avoid aliasing effects (which cause the result to look "pixelly", so that many video applications use special filters called "anti-aliasing" techniques). The Nyquist frequency is half of the sampling rate, so wavenumbers are built up to half of the total samples of the signal. Zooming in on the data with the `axis` command, Fig. 4 is the resulting spectrum of the C-chord. Now try this on your beam vibration data!

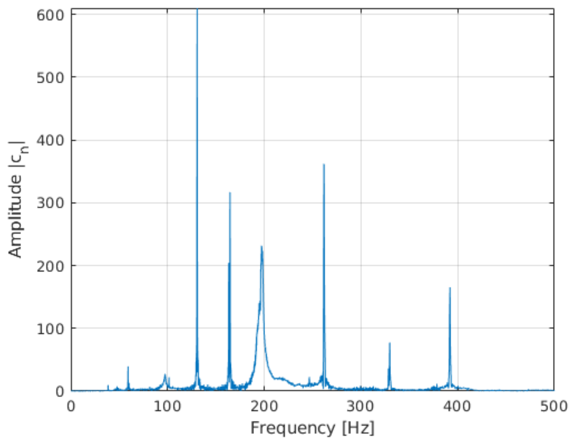


Figure 4: Component frequencies of the C-chord clip. The four notes played were the chord tonic C (131 Hz), then E (165 Hz), G (196 Hz), and finally the octave tonic C (262 Hz). Note that the octave is exactly twice the frequency of the original C! The spike at 393 Hz (G) is a *harmonic* and was not actually played.