# LAB 4
# FOURIER SERIES AND THE GIBBS PHENOMENON

EE 235: Continuous-Time Linear Systems
Department of Electrical Engineering
University of Washington

This work[1] was written by Amittai Axelrod, Jayson Bowen, and Maya Gupta, and is licensed under the Creative Commons Attribution License.[2]

## 1 Introduction

In this lab you will examine the Fourier series representation of periodic signals using MATLAB. In particular, you'll study the truncated Fourier series reconstruction of a periodic function.

## 2 Useful MATLAB Commands

In addition to what you already know about MATLAB, you may need to use the following commands. You've seen them in Lab 1, but use `help` to remind yourself of their syntax.

- `abs` – Computes the magnitude of a complex number.
- `angle` – Computes the phase angle of a complex number.
- `stem` – Draws discrete plots (as opposed to `plot`, which draws continuous plots).

## 3 Signal Synthesis

Later on in this lab you will use a Fourier series to approximate a square wave. The Fourier series represents the sum of smooth sinusoids, but a square wave is full of sharp edges, so this is a counter-intuitive result! Let's start with something a little easier: synthesizing the sound of a trumpet. Many musical instruments produce very periodic waveforms. In particular, the pitch of certain notes is determined by specific frequencies, such as 440Hz (which is Concert A, for tuning strings).

[1] Last revision: Mon May 3 02:48:54 EDT 2010
[2] http://www.creativecommons.org/licenses/by/2.0

## 3.1 Trumpet Synthesizer

First, download the trumpet sound sample called `trumpet.mat` from the lab webpage[3]. This trumpet sound was sampled at a rate of `Fs = 11,025 Hz`.

**Exercise 1:**
• Play the `trumpet` sound to verify your setup.
• Plot a small section of the trumpet sound (100 samples).
This will show about three periods of the sampled signal. Is the signal periodic?
• Look at the frequency spectrum of the trumpet sound by doing the following:

```
>> Fs = 11025;              % the sample rate is 11025 Hz
>> Y = fft(trumpet, 512);   % take the FFT of trumpet
>> Ymag = abs(Y);           % take the magnitude of Y
>> f = Fs * (0:256)/512;    % scale the axis to be meaningful
>> plot(f, Ymag(1:257));    % plot Ymag (only half the points are needed)
>> xlabel('Frequency (Hz)')
>> ylabel('Magnitude')
```

The series of peaks that you see are the harmonics of the instrument.
• Use the "data cursor" tool in MATLAB's figure window to read the graph data.
Write down the frequency and magnitude (the strength of the harmonic) for five to ten of the strongest peaks.

You'll now synthesize the trumpet using only the information from these peaks. Each peak represents a scaled cosine at a particular frequency. The trumpet sound is the sum of these (possibly infinitely many) scaled cosines.

**Exercise 2:**
• Create a function called `addcosines` that will add several cosines together and normalize the output.
`addcosines` should take in three vectors: a time vector `time`, a vector of frequencies `freqs`, and a vector of magnitudes `mags`.
The output should have the same dimensions as `time`. Furthermore, the output should only take on the range of values [-1,1] because you will want to play it like a sound vector.
• Verify your code by passing the inputs `time = 0:1/Fs:1`, `freqs = [100 150]`, and `mags = [1 2]` to `addcosines`. Plot the inputs and output. Are they what you expect?

[3]Or from http://cnx.org/content/m13854/latest/trumpet.mat

Here are some hints for the above:

- Assume the `time` vector will have the form `0:1/Fs:duration`, where `duration` is in seconds.
- Each cosine function should look like `mag(i)*cos(2*pi*freq(i)*t)`.
- Use a `for` loop to add together the cosines. You'll have one cosine for each frequency/magnitude pair in the `freqs` and `mags` vectors.
- To scale the output, recall that you did an identical thing in your mixer from Lab 2. Alternatively, you may use `soundsc` this time.

**Exercise 3:**
• Use your `addcosines` function to sum the cosines specified by the peak data you collected from the `trumpet` sound in the previous exercise.
• Play `trumpet` and your new synthesized sound. Do they sound the same?
• Use `subplot` to plot a small section of your new synthesized sound along with `trumpet`. Do they look the same?
• Try synthesizing `trumpet` with fewer frequencies, then try again with more frequencies.
• How does this affect the sound of your synthesized `trumpet`?

## 3.2 That Funny Phase

Although the waveforms of your synthesized trumpet sounds look different from the original, they ought to sound pretty similar. This should convince you that the Fourier series coefficients are a valid approximation of the original signal! Let's look at another example:

**Exercise 4:**
• Pick two harmonic frequencies and create two cosines, `cos1` and `cos2`, at these frequencies.
Use the time vector `time = 0:1/Fs:1` with `Fs = 8000`, but remember that you can only reproduce frequencies that are less than Fs/2.
• Add `cos1` and `cos2` to produce a signal `sig1`.
• Generate `cos2b` by time-delaying `cos2` by half a cycle.
• Now create a second signal `sig2` that is the sum of `cos1` and `cos2b`.
• Use `subplot` to show a few cycles of `sig1` and `sig2`. How do they compare?
What did the time delay do to the phase of `sig2`?
• Play both `sig1` and `sig2` with `soundsc`. How do they compare?
• Make a couple other variants of `sig2` (call them `sig2b`, `sig2c`, etc) with different delays on `cos2`, and compare the outputs to the original `sig1`.

**Exercise 5:**
- Create a third cosine `cos3` at an arbitrary frequency.
- Add `cos3` with a quarter-cycle-delayed version of itself to generate `sig3`.
- Use `subplot` to show a few cycles of `cos3` and `sig3`. How do they compare?
- Play both `cos3` and `sig3` with `soundsc`. How do they compare?
- What can you conclude about human hearing capabilities?

## 4 Truncated Fourier Series

In this exercise you'll reconstruct the periodic function `x(t)`, shown below in Figure 1, by synthesizing a periodic signal from a number of Fourier Series coefficients. Along the way, you'll observe some similarities and differences between your reconstruction and the original signal.

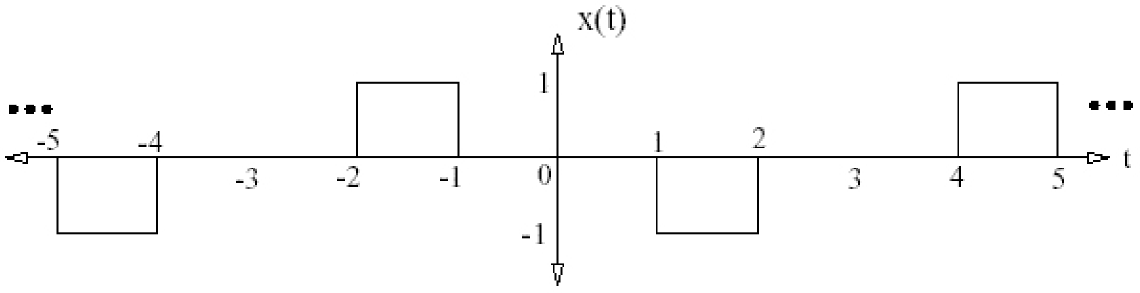

**Figure 1:** Periodic Signal

Figure 1: A periodic signal.

### 4.1 Gibbs Phenomena

Download the `Ck.m` function from the lab webpage[4]. This function takes one argument `k` , and computes the `k`$^{th}$ Fourier series coefficient for the square wave in Figure 1. The formula for the coefficient is:

$$C_k = \begin{cases} 0 & \text{if } k = 0, \text{ or } k \text{ is even;} \\ \frac{1}{jk\pi}[\cos\frac{2k\pi}{3} - \cos\frac{k\pi}{3}] & \text{if } k \text{ is odd.} \end{cases}$$

[4]Or from http://cnx.org/content/m13854/latest/Ck.m

For example, $C_k(1) = \frac{-1}{j\pi} = 0 + j0.3183$.

**Exercise 6:**
• Compute the coefficients $C_k$ for $k \in \{-10, -9, \ldots, 9, 10\}$.
• Use `subplot` to show the magnitude and phase (versus frequency $\omega$) for each coefficient. The coefficients are a function of $k$, which is integer-valued, so use `stem` to draw the plots.

You've seen that a signal `x(t)` can be approximated with a truncated Fourier series. The actual calculation, for a given coefficient range $k \in \{-K_{max}, -K_{max}+1, \ldots, K_{max}-1, K_{max}\}$ is as follows:

$$\begin{aligned} x(t) &= \sum_{k=-Kmax}^{Kmax} C_k e^{jk\omega_0 t} \\ &= \sum_{k=0}^{Kmax} 2|C_k| \cos\left(k\omega_0 t + \angle C_k\right) \end{aligned}$$

The reason for using the cosine form is to avoid numerical problems and ensure a real answer. For this example, $\omega_0 = 1$.

**Exercise 7:**
• Implement a function that approximates `x(t)` with a truncated Fourier series, as described above. Show the TA your (commented) code.
• Compute `x(t)` over the time vector `t = -6:.01:6` for each of the following three cases: $K_{max} = 5$, 15, and 30.
• Use `subplot` to plot all three approximations in the same figure. How do the vertical and horizontal edges compare?

As you add more cosines to the approximation, you do get closer to the square wave (in terms of squared error). However, at the edges there is some undershoot and overshoot that becomes shorter in time, but the magnitude of the undershoot and overshoot stay large. This persistent undershoot and overshoot at edges is called the Gibbs Phenomenon.

In general, this kind of "ringing" occurs at discontinuities – the sharp vertical jumps in the square wave – when you try to synthesize a sharp edge out of too few low frequencies. Another view is that low-pass filtering will trigger the ringing or Gibbs Phenomenon. If you start with a real signal and filter out its higher frequencies, then this is equivalent to going the other way and synthesizing the signal from low frequencies– which is what you just did in the last exercise!

As an example, higher frequencies are usually removed when compressing an audio signal (that is, the audio signal is low-pass filtered). Then, if there is an impulse edge or "attack" in the music, ringing will occur. However, the ringing (called "pre-echo" in audio) can be heard only before the attack, because the attack masks the ringing that comes after it (this masking effect is psychoacoustic; it happens in your head). High-quality MP3 systems put a lot of effort into detecting attacks and processing the signals to avoid pre-echo.

## 5 Links

This applet[5], from the National Taiwan University, provides a great interface for listening to sinusoids and their harmonics.

EOF

[5]http://www.phy.ntnu.edu.tw/ntnujava/viewtopic.php?t=33