Endpoint Discontinuities and Spectral Leakage

2018-02-10 - By Robert Elder

Introduction

This article is effectively an appendix to the article The Fast Meme Transform: Convert Audio Into Linux Commands. In this article, we will review some of the problems that occur due to endpoint discontinuities that exist when performing the Fourier transform on audio samples. Mitigation techniques for spectral leakage are explored. A review of the effects of FFT zero padding is also discussed.

Interpreting Diagrams In This Article

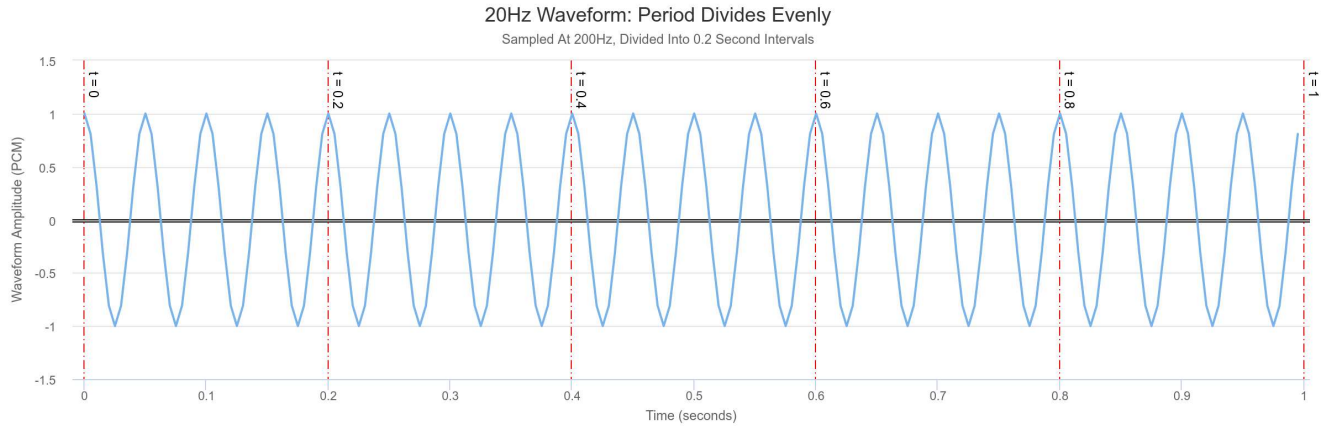
There are a number of diagrams in this article that depict both the input and output of a discrete Fourier transform. The meaning of the input time domain graph should be self evident, but the meaning of the frequency domain output needs a bit more explanation. For starters, the Fourier transform used to transform the input into the output as described in this article is the Discrete Fourier transform, not to be confused with Continuous Time Fourier transform. The Discrete Fourier transform takes as input a sequence of N complex numbers in the time domain and transforms them into N complex numbers in the frequency domain. Since this article only deals with real-valued signals, the imaginary part of all signals has always been set to zero in the time domain. Furthermore, since the output of the Discrete Fourier transform is a sequence of complex numbers, you would need two graphs to represent both the real and imaginary parts of the output, but in this article, I've opted to only show the absolute value of the complex number.

Finally, the red dashed vertical lines on the time domain graphs represent the places where the signal has been chopped up in order to perform the Fourier transform. The x axis of the time domain graphs is simply the time of the data point, but for the frequency domain graphs the x axis is less intuitive. For clarity, I opted to depict the same dashed red lines on the frequency domain output to show the correspondence between the sample windows in the time domain, and the frequency domain output that corresponds to these same windows. In the frequency domain graphs, the power spectrum is shown multiple times, once for each sample window. This depiction is convenient because the size of the output is the same size as the input.

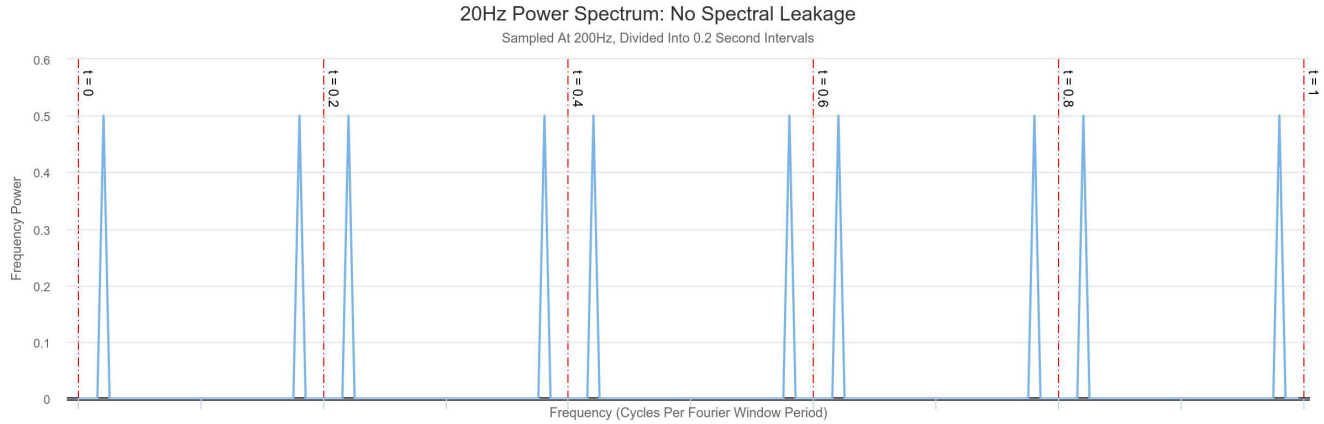
Mitigating Spectral Leakage

In order to keep the audio playback as high quality as possible, one important consideration is the effects of spectral leakage. This occurs when the endpoints of your sample interval don't exactly align with the period of a given frequency you may be solving for with the Fourier Transform. For example, first consider a simple signal that consists of a cos wave oscillating at a frequency of 20 hz and divide it into intervals of 0.2 seconds. Note that each interval contains exactly 4 complete cycles of this cos wave. There is also a tricky off-by-one error to be made here if you want to consider the points in the intervals disjoint: In

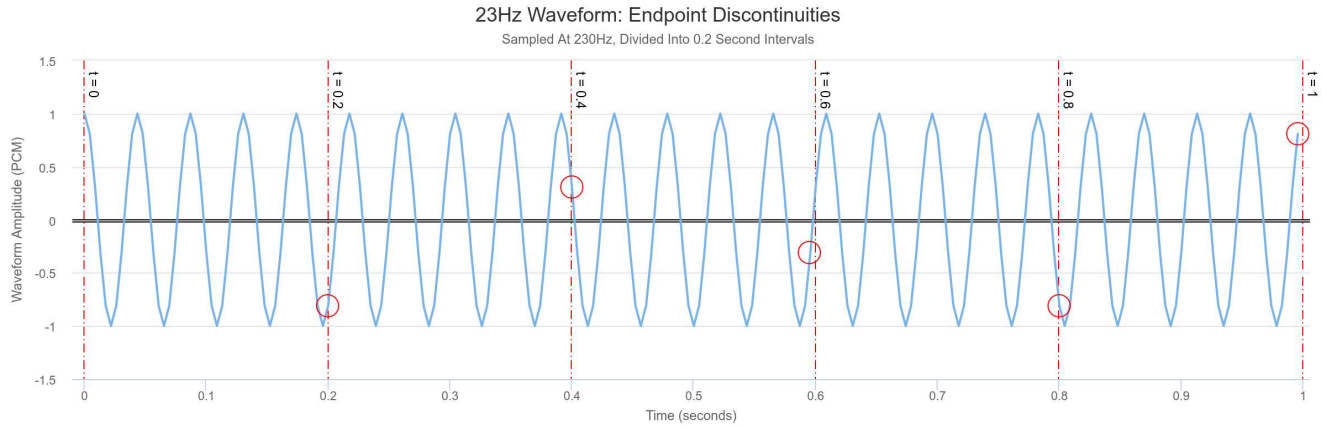
this example, we will assume that the last point of each interval (the one under the red dotted line) is present in both in both the interval before and after the line.



In this case, the Fourier transform will be able to exactly represent the signal with only two sharp peaks as shown in the frequency spectrum below:

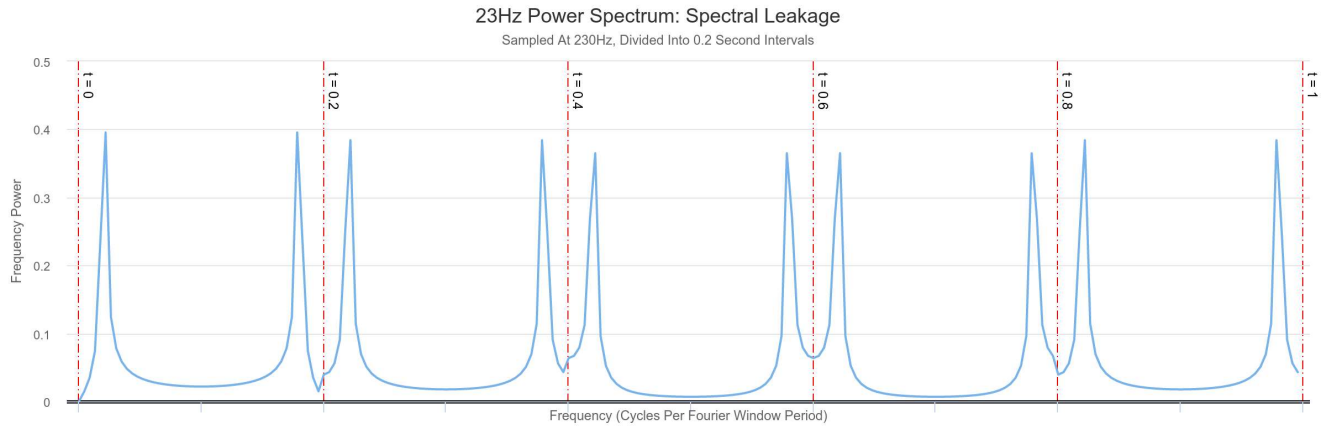


However, if we consider a signal with a frequency of 23hz that gets divided into intervals of 0.2 seconds, we end up only being able to represent 4.6 cycles of the cos wave per interval. This means that our intervals will end or start with a discontinuity that will be difficult to approximate for the Fourier transform, since the endpoints of the window upon which the Fourier transform is applied are expected to be easily approximated by the end of a cos or sin wave cycle:



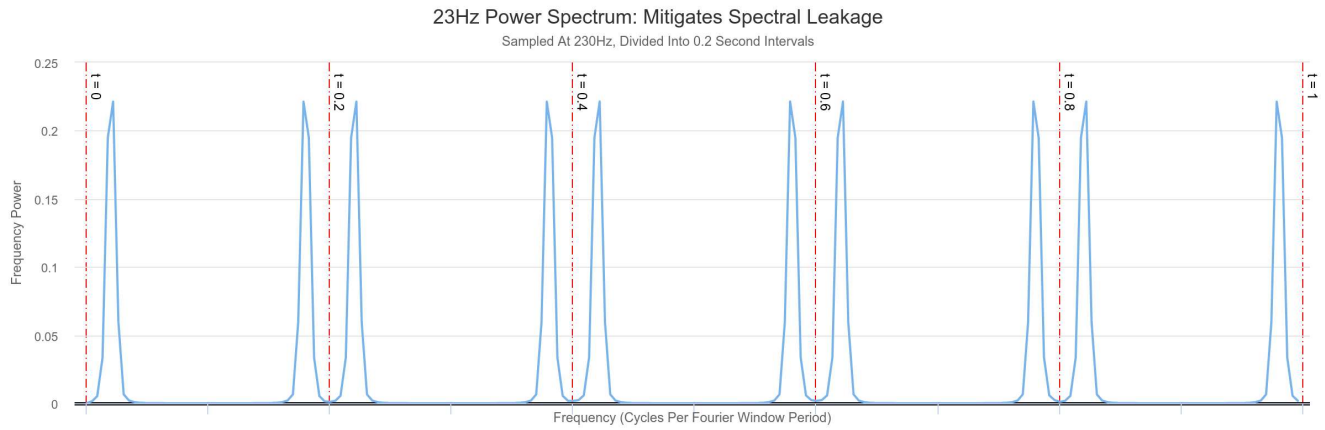
The result is spectral leakage, where the 'power' in our frequencies will 'leak' into a number of different frequency bins as the Fourier transform uses a large number of different sin or cos signals in an effort to reach these strange endpoint values that don't correspond to exactly the endpoint of a cycle in a sin or cos wave. This is a problem because it means the information needed to re-construct the signal will be spread out among a large number of

floating point values, each of which will come with their own rounding errors. It is also a problem because now we need to store more information when we remember which frequencies are the 'important' ones. Shown in the next image is what spectral leakage looks like:



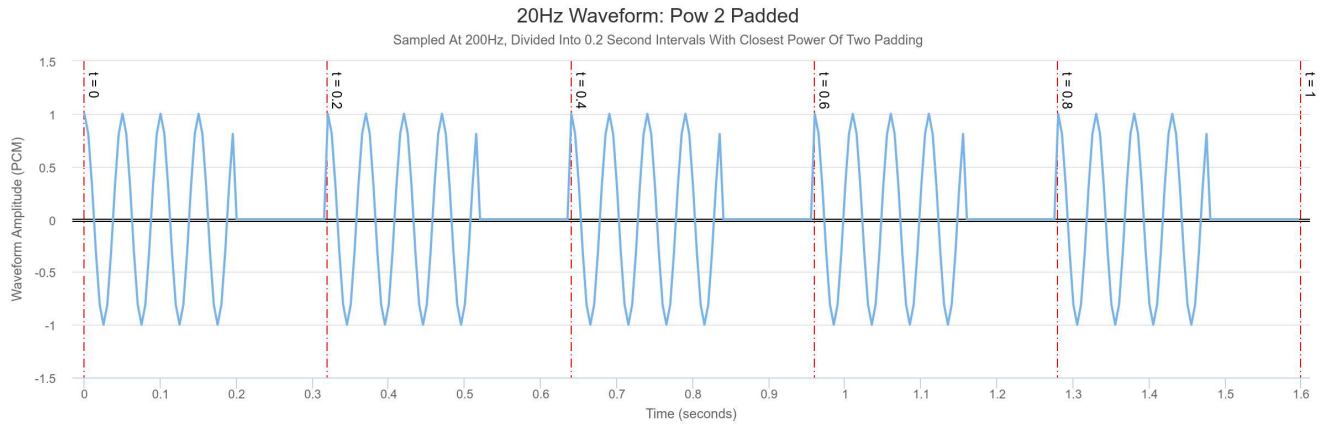
One way to mitigate this problem is by applying a windowing function that attenuates the endpoints toward zero so they can be more easily represented by a sin or cos wave. Note that the windowing function can possibly destroy important information at the endpoints, but it may provide us with a better representation of the signal after we perform a Fourier transform. The windowing function below is called a Hann window:

Observe below that the spectral leakage has been mitigated. The peaks in the frequency spectrum that you see in the diagram below are still a bit wider than with the 20hz signal we saw above, but they don't get smeared over the entire spectrum like in the 23hz signal with no window function applied. The fact that the peaks are not as wide means that we need to keep track of less information if we want to re-construct the original signal with high accuracy (the 'original' signal meaning the signal after the Hann window function was applied). This is true because we can just ignore parts of the frequency spectrum that are zero (or very close to zero).



FFT Zero Padding

The Cooley-Tukey Fast Fourier Transform algorithm is much faster than the trivial Discrete Fourier Transform algorithm. In fact, it is so much faster that it isn't really practical to use the trivial one for many applications. Unfortunately, the Cooley-Tukey algorithm requires that the input time domain series length be exactly a power of two. This means that if our sample time period doesn't contain exactly an even power of two number of data points, we'll need to force our input sequence length to be a perfect power of two before we can run the FFT on it.



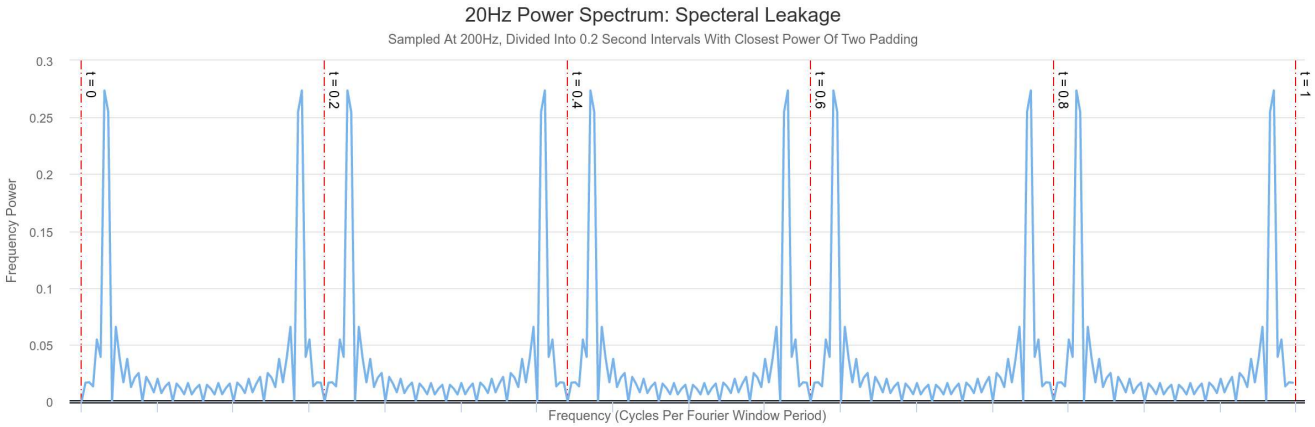
Zero padding the input introduces a couple problems: One is the issue discussed previously with spectral leakage due to endpoint discontinuities, but another is the fact that when the signal is re-constructed, we'll may need to (depending on the re-construction method) do a mathematical adjustment to correct for the frequency shift that occurs from adding extra zero data points to our series. The discrete Fourier transform transforms a sequence of N complex numbers in the time domain into another sequence of N complex numbers in the frequency domain. Each complex number in the frequency domain (let's call them {F_0, F_1, F_2, ... F_(N-1)}) represents the weighting of a signal that oscillates with a frequency of 'x cycles over the course of the entire input time domain sequence'.

For example, F_0, oscillates with a frequency of '0 cycles over the course of the input sequence', in other words it is a constant bias applied over the entire duration of the time domain signal. In many contexts, this F_0 is often called 'DC' as a reference to Direct Current since Fourier and Laplace transforms are an excellent way to generalize the analysis of electrical circuits beyond the simple AC/DC circuits into complex hybrids that may exhibit properties of both or neither.

F_1, oscillates with a frequency of '1 cycle over the course of the input sequence', and F_2 does so '2 cycles over the course of the input sequence', and so on. This obviously means that we are limited to only being able to represent signals in terms of sin/cos waves that are in the frequency range of 0 to N-1. Furthermore, if we zero pad the input before taking the Fourier transform, we end up artificially increasing the measured frequency

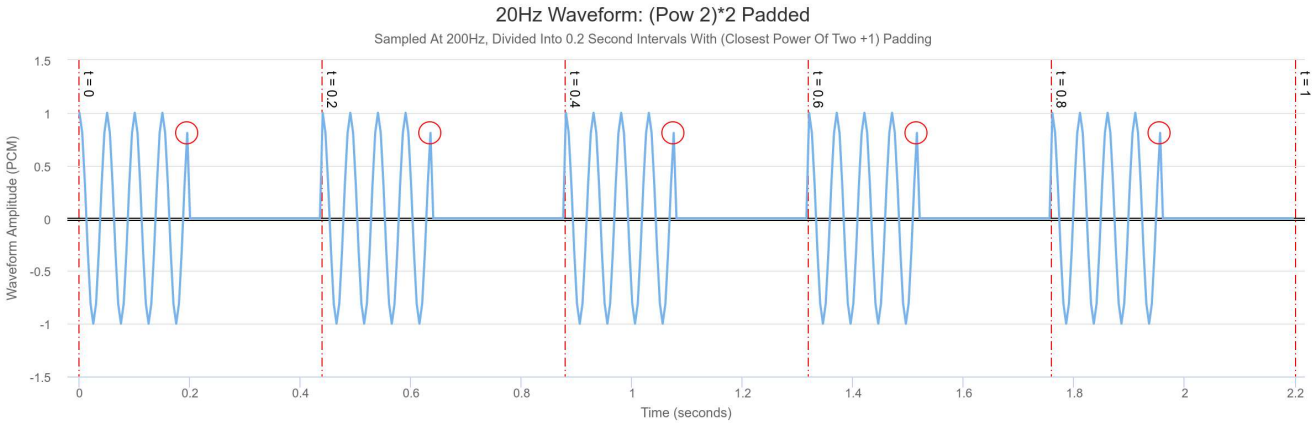
components of the data in the sample we wanted to analyze. For example, if you doubled the length of a signal that oscillated at 1hz by zero padding, that signal is now going to look like a signal would oscillate '2 cycles over the course of the input sequence' if it kept going at the same rate through the zero padded region. Therefore, when zero padding, we can correct for the original frequency by multiplying the artificially increased frequency by ((number of samples without zero padding -1) / (number of samples with zero padding -1)).

Observe how the location of the peaks in the power spectrum have shifted, and there is spectral leakage resulting from the zero padded input of the fast Fourier transform:

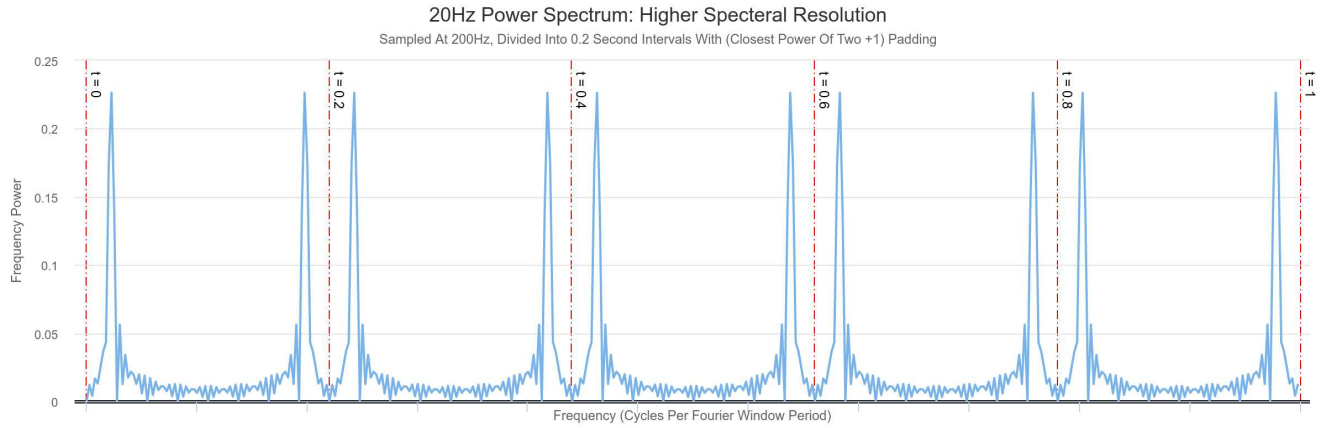


The act of zero padding the input before performing the fast Fourier transform has another affect which could actually be considered a benefit, and that is to increase the spectral resolution. This doesn't just occur with zero padding, but it is an effect of simply have more input data points. This should be obvious since the length of the output from a Fourier transform is the same as the length of the input, and increasing the length of the input sequence increases the number of bins available to represent unique frequencies.

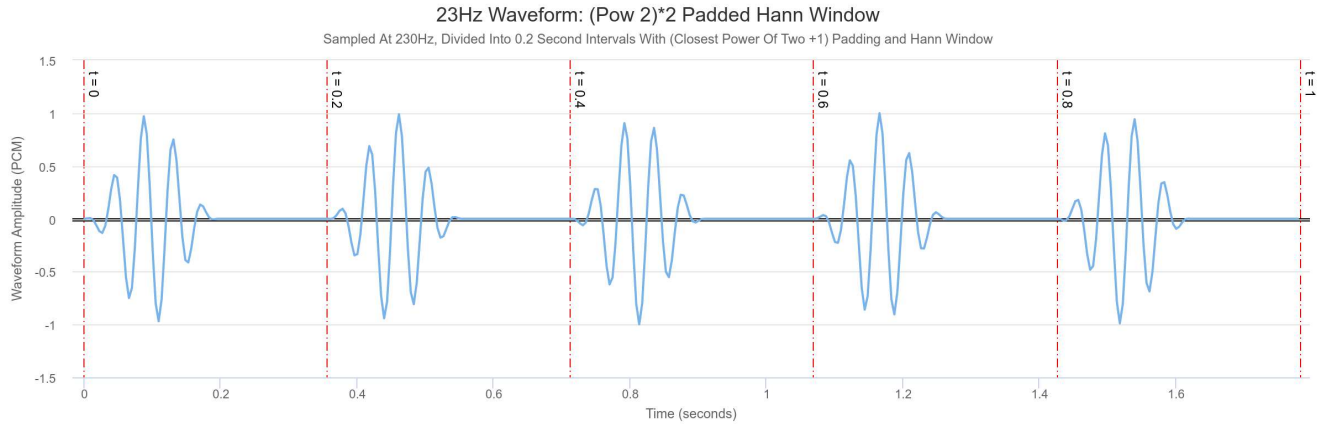
Here is the same 20hz signal seen above with more padding:



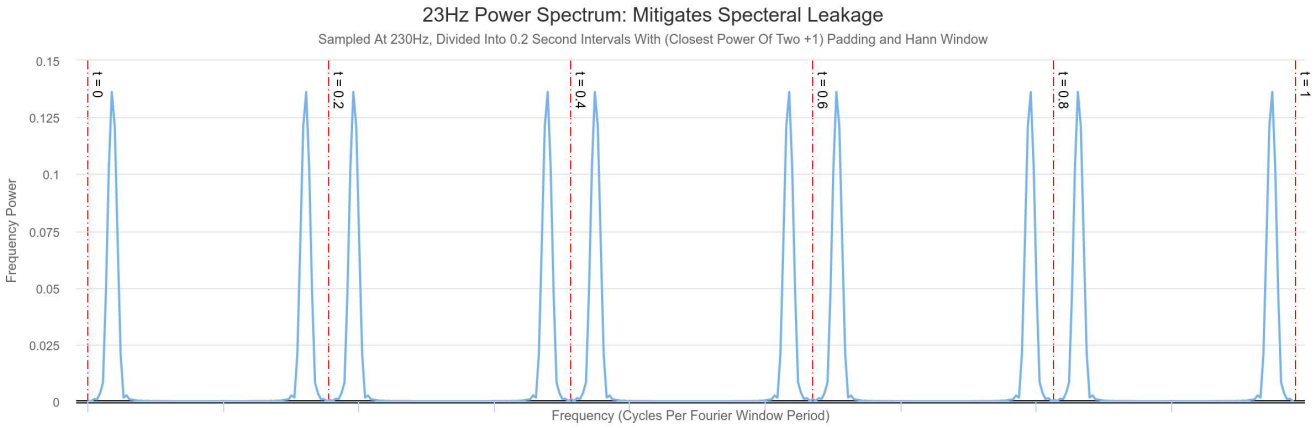
And the corresponding power spectrum:



Increasing the spectral resolution isn't necessarily always a universally good thing since you could easily end up with a situation where none of your frequency bins precisely represent your frequency. For example, consider a situation where you want to do frequency analysis of a 5hz signal sampled at a rate of 1000 samples per second with 1001 samples per Fourier transform input window. The extra 1 is included because the amount of time under consideration in the interval will be dt * (#samples -1) and we want the last point to be on the end of the sin/cos cycle. In this case, each bin can represent frequencies of 0, 1, 2, ..., 1000 cycles per Fourier input window. Since in this case the input window describes the waveform over a duration of 1 second, the frequencies are 0hz, 1hz, 2hz, ..., 1000 and 5hz will fall exactly into bin 5 (consider zero based indexing). If you add one extra data point to the input with the same sample rate (even if the new sample comes from the 5hz signal and not just zero padding), then there is no longer any frequency bin which exactly represents 5hz. Because of the extra time added to the input by the new data point, the new frequency bins will represents frequencies of 0, 1, 2, ..., 1001 cycles per Fourier input window, but since an input window now considers 1002 samples, or 1.001 seconds of audio, each frequency bin corresponds to 0hz, 0.9990hz, 1.9980hz, ... 999.0009hz. This is obvious when you consider the earlier issue related to endpoint discontinuities. Here is the same 23hz signal from before with the extra padding and a Hann window applied:



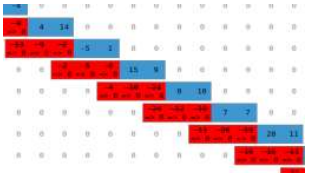
And below is the corresponding power spectrum. Note that the top of the peak is now sharper in the power spectrum since we can resolve the most dominant frequency more precisely, but there are also more points that surround the peak since the output sequence has more points in general and the dominant frequency gets smeared over a range of bins:



Conclusion

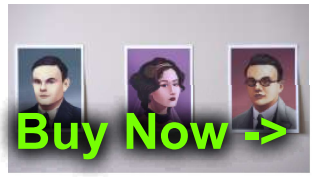
In conclusion, it has been shown that endpoint discontinuities can occur in a signal of one specific frequency if the time period under consideration in the interval is not compatible with the frequency of the true signal. These endpoint discontinuities can cause the true frequency present to be smeared out into a large number of frequency bins which is problematic since it magnifies imprecision due to rounding errors. This effect can be mitigated, although with some drawbacks, by using a windowing function such as the Hann window. The use of more or less zero padding can have a significant influence on the amount of spectral leakage. More frequency bins give more opportunity to accurately capture the exact frequency present, although when attempting to minimize spectral leakage, the number of frequency bins is not nearly as important as the relative divisibility of the number of frequency bins and the true frequencies present in the signal.

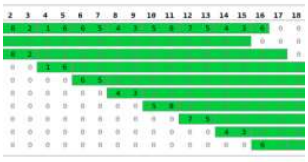
Finally, I should note that the details included in this article are at the limits of my knowledge on this subject. If you believe you have found an error above, please let me know so I can correct it at info@robertelder.org.



[Overlap Add, Overlap Save Visual Explanation]

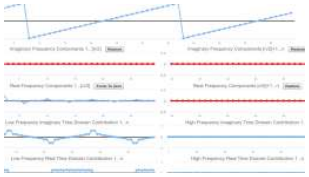
Published 2018-02-10





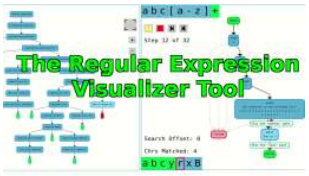
[Using Fourier Transforms To Multiply Numbers - Interactive Examples]

Published 2019-01-10



[Fourier Transform Coefficients Of Real Valued Audio Signals]

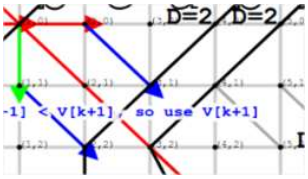
Published 2018-02-10



[The Regular Expression Visualizer, Simulator & Cross-Compiler Tool]

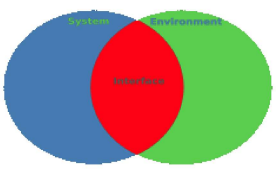
[Why Is It so Hard to Detect Keyup Event on Linux?]

Published 2019-01-10



[Myers Diff Algorithm - Code & Interactive Visualization]

Published 2017-06-07



[Interfaces - The Most Important Software Engineering Concept]

Published 2016-02-01

Published 2020-07-09

Join My Mailing List

Email: Submit

Privacy Policy

Why Bother Subscribing?

- **Free Software/Engineering Content.** I publish all of my educational content publicly for free so everybody can make use of it. Why bother signing up for a paid 'course', when you can just sign up for this email list?
- **Read about cool new products that I'm building.** How do I make money? Glad you asked! You'll get some emails with examples of things that I sell. You might even get some business ideas of your own :)
- **People actually like this email list.** I know that sounds crazy, because who *actually* subscribes to email lists these days, right? Well, some do, and if you end up not liking it, I give you permission to unsubscribe and mark it as spam.