Matlab Project 2: Sampling and reconstruction

EE-125

This project has two separate parts: one dealing with sampling (and undersampling), and the other with reconstruction.

1 Part 1: Aliasing due to undersampling

In this project you will work with a 'chirp' signal, named for the sound it makes when played through a loudspeaker. The chirp signal is given by

$$x(t) = \sin\left(\Omega_0 t + 1/2\beta t^2\right). \tag{1}$$

where Ω_o is a frequency in radians/sec, so $\Omega_o = 2\pi F$, with F in Hz.

The instantaneous frequency of the chirp is given by the derivative of its phase, i.e. the derivative of the argument of sin():

$$\begin{aligned}\Theta_{inst}(t) &= \frac{d}{dt}\left(\Omega_0 t + 1/2\beta t^2\right) \\ &= \Omega_0 + \beta t.\end{aligned} \tag{2}$$

Chirp signals have interesting autocorrelation properties that make them useful in radar and sonar, and also are found in bat and dolphin biosonar. Here we will explore two aspects of sampling this type of signal.

This lab involves listening to some sounds, so if you will work in a computer lab, you may want to bring earphones!

For this section, assume that the sampling frequency is $f_s = 8192$ Hz, and set up a vector t containing the times you are sampling. There are several ways to do this; you can set up a vector directly (for example, [0:.01:1] will sample times from 0 to 1 at steps of 0.01), or you can use the function 'linspace' (for example, t=linspace(0,T,T*fs) gives the sampled times for the first T seconds). This vector t will help you keep track of the difference between the index of the time sample and the value of the time sample.

- Set $\Omega_0 = 2\pi(1500)$ rad/sec and $\beta = 2\pi(3000)$ rad/sec^2. Calculate the values of x between 0 and 2 sec.
- Use the 'sound' or 'soundsc' commands to listen to the vector x. Can you explain what you just heard?
- For $fs = 8192$ Hz, determine the approximate time at which the chirp signal has its maximum pitch by observing the signal. Create a longer sampled chirp with length 5 sec, and listen to that. How many times does the signal reach its minimum frequency?
- Do a calculation that predicts the times (in seconds) at which the chirp should reach its maximum and minimum frequencies. In doing so, use the formula for the instantaneous frequency, but also use your understanding of how aliasing occurs. Compare those predictions to what you observed in the last step.
- Again for $fs = 8192$ Hz, calculate separately the signals for four different signal lengths: 0.5 sec, 0.8 sec, 1.2 sec, and 2 sec. Use the routine 'ctft.m' (provided on the class website) to calculate the continuous-time Fourier transform for each signal. Make plots of the magnitude of each transformed signal vs. frequency, putting all four plots as subfigures in a single figure. Can you explain the changes in the spectrum you see as the chirp length changes?

<u>Useful Matlab commands:</u> sound, soundsc, linspace, abs, angle, unwrap

Part 2: Reconstruction

In class, we have discussed signal reconstruction, or digital-to-analog conversion (DAC). As we saw, signal reconstruction boils down to using some kind of interpolation filter to interpolate samples at times $x(n) = x(nT)$ to times in between the original samples. For a hardware-based DAC, we end up with a continuous-time signal. In a computer project like this one, we of course can't get a continuous signal $x(t)$. However, we can test different interpolation filters to find the predicted samples at times between our original $x(n)$ samples. This will (hopefully) give insight into how the interpolation approaches work.

Let's say we have a 3-point signal, sampled once per second at $0, 1$, and 2 seconds, so that $x(0) = 2$, $x(1) = 1$, and $x(2) = -1.1$. We can reconstruct different signals from this input, depending on what assumptions we think are true about the signal.

Linear interpolation reconstruction

Let's say we believe the signal is piecewise linear between our samples. We discussed this in class, so you may want to refer to your class notes. Do the following steps:

1. Plot the signals vs time in seconds. Note that by default, Matlab draws straight lines in between the points, so we can see what the linear interpolation result should look like! However, we haven't yet actually computed any interpolated values.

2. Now, let's say we want to compute interpolated values every 0.2 sec. Insert four zeros between each data point, and interpolate that signal with an impulse h(n) that has the values $0.2, 0.4, 0.6, 0.8, 1.0, 0.8, 0.6, 0.4$, and 0.2. Use the `conv` command to convolve the two signals, and plot the result vs time in seconds. Answer the following two questions:

   -linear convolution is a non-causal filter, but `conv` assumes all signals and filters are causal. How do you need to adjust your time axis to reflect this?

   - Does your result match what you expect for linear interpolation, and over what ranges of times? In particular, does this approach seem to be explicitly assuming anything about the values of $x(-1)$ and $x(3)$?

Sinc interpolation

We also reviewed in class the idea that if a signal is assumed to be band-limited, the optimal interpolation is sinc interpolation, which is given by Eq. 6.1.20 in the book

$$y(t) = \sum_{n=-\infty}^{\infty} x(nT) \frac{\sin(\pi(t - nT)/T)}{\pi(t - nT)/T} \tag{3}$$

where our sampled data is $x(n) = x(nT)$, sampled at time intervals T. This equation involves an infinite sum, because $x(n)$ is assumed to be infinitely long. Generally speaking, we can't implement infinite sums in computer code, but we can compute 3 exactly if the number of samples is finite.

Implement sinc-interpolation as described below:

1. Set up a time vector going from -5 to 5 in steps of 0.2 sec. We will get interpolated output on this range.

2. Write a short function called `sincInterp` that implements Eq. 3.

3. Test your code by calling it for a simple 1-sample input signal, $x_{test}(0) = 1$; i.e. a single sample which is '1'. Does the output look like a sinc? Include the figure in your report.

4. Now, interpolate the three-sample function we started with. Overplot the linear interpolation result with your new sinc-interpolated result. Which seems like a more natural-looking signal?

5. For sinc interpolation, we assume that the signal is bandlimited. In this particular case, what frequencies are we assuming the signal is limited to?