Digital Signals - Sampling and Quantization

A signal is defined as some variable which changes subject to some other independent variable. We will assume here, that the independent variable is time, denoted by t and the dependent variable could be any physical measurement-variable which changes over time - think for example of a time varying electric voltage. We will denote the generic measurement variable with x or $x(t)$ to make its time-dependence explicit. An elementary example of such a signal is a sinosoid. When we want to represent such a sinosoid in the digital domain, we have to do two things: sampling and quantization which are described in turn.

Sampling

The first thing we have to do, is to obtain signal values from the continuous signal at regular time-intervals. This process is known as sampling. The sampling interval is denoted as T_s and its reciprocal, the sampling-frequency or sample-rate is denoted as f_s, where $f_s = 1/T_s$. The result of this process is just a sequence of numbers. We will use n as an index into this sequence and our discrete time signal is denoted as $x[n]$ - it is customary in DSP literature to use parentheses for continuous variables such as the time t and brackets for discrete variables such as our sample index n. Having defined our sampling interval T_s, sampling just extracts the signals value at all integer multiples of T_s such that our discrete time sequence becomes:

$$x[n] = x(n \cdot T_s) \tag{1}$$

Note that at this point (after sampling), our signal is not yet completely digital because the values $x[n]$ can still take on any number from a continuous range - that's why we use the terms discrete-time signal here and not digital signal. Figure 1 illustrates the process of sampling a continuous sinosoid. Although it

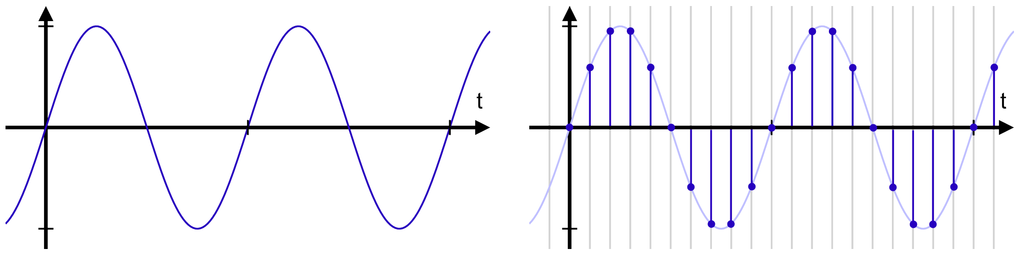

Figure 1: Sampling a sinosoid - we measure the value of the signal at regular time-intervals

has been drawn in the right plot, the underlying continuous signal is lost in this process - all we have left after the sampling is a sequence of numbers. Those numbers themselves are termed samples in the DSP community - each such number is a sample in this terminology. This is different from what a musician usually means when talking about samples - musicians refer to a short recording of an acoustic event as sample. In this article we will use the DSP terminology. The question is: can we reconstruct the original continuous signal from its samples?

Interpolation

Whereas the continuous signal $x(t)$ is defined for all values of t, our discrete-time signal is only defined for times which are integer multiples of T_s. To reconstruct a continuous signal from the samples, we must somehow 'guess', what value the signal could probably take on in between our samples. Interpolation is the process of 'guessing' signal values at arbitrary instants of time, which fall - in general - in between the actual samples. Thereby interpolation creates a continuous time signal and can be seen as an inverse process to sampling. Ideally, we would want our interpolation algorithm to 'guess right' - that is: the continuous signal obtained from interpolation should be equal to the original continuous signal. The crudest of all interpolation schemes is piecewise constant interpolation - we just take the value of one of the neighbouring samples as guessed signal value at any instant of time in between. The reconstructed interpolated function will have a stairstep-like shape. The next better and very popular interpolation method is linear interpolation - to reconstruct a signal value we simply connect the values at our sampling instants with straight lines. Figure 2 shows the reconstructed continuous signals for piecewise constant

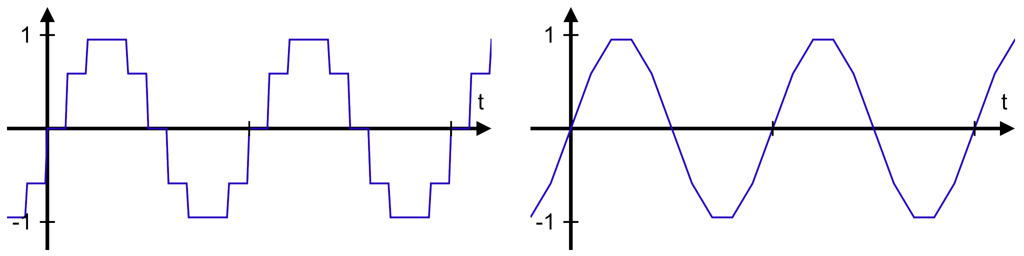

Figure 2: Reconstructing the sinusoid from its samples via piecewise constant and linear interpolation

and linear interpolation. More sophisticated interpolation methods such as higher order polynomial interpolation (quadratic, cubic, quartic, pentic, etc.), (windowed) sinc-interpolation exist as well - they will yield smoother and more faithful reconstructions than the two simple ones described above.

The Sampling Theorem

Sampling and interpolation take us back and forth between discrete and continuous time and vice versa. However - our reconstructed (interpolated) continuous time signal is by no means guaranteed to be even close to the original continuous time signal. A major breakthrough for doing this sampling and interpolation business 'right' was achieved by Claude Shannon in 1948 with his famous Sampling Theorem. The sampling theorem states conditions under which a continuous time signal can be reconstructed *exactly* from its samples and also defines the interpolation algorithm which should be used to achieve this exact reconstruction. In loose terms, the sampling theorem states, that the original continuous time signal can be reconstructed from its samples exactly, when the highest frequency (denoted as f_h) present in the signal (seen as composition of sinosoids) is lower than a half of the sampling frequency:

$$f_h < \frac{f_s}{2} \tag{2}$$

Half of the sample-rate is also often called the Nyquist frequency, in honor to Harry Nyquist who also contributed a lot to sampling theory. When this condition is met, we can reconstruct the underlying continuous time signal exactly by a process known as sinc-interpolation (the details of which are beyond the scope of this article). A subtle side-note: in some books, the above inequation is stated with a $\leq$ rather than a $<$ - this is wrong: f_h has to be strictly less than half the sample-rate because when you try to sample a sinusoid with frequency of exactly half the sampling frequency, you *may* be able to capture it faithfully (when the sample-instants happen to coincide with the maxima of the sinusoid), but when the sample-instants happen to coincide with the zero-crossings, you will capture nothing - for intermediate cases, you will capture the sinusoid with a wrong amplitude. Thus, perfect reconstruction is not guaranteed for $f_h = f_s/2$. When the above condition is not met, but we sample and reconstruct the signal anyway, an odd phenomenon occurs, widely known and quite often misunderstood - it's called...

Aliasing

According to the sampling theorem, the signal in figure 1 is sampled properly. As this particular signal is just a single sinosoid (as opposed to a composition of sinosoids), the highest frequency present in this signal is just the frequency of this sinosoid. The sampling frequency is 10 times the sinosoids frequency, that is: $f_h = \frac{f_s}{10} < \frac{f_s}{2}$. To see it the other way around: 10 samples are taken from each period of the signal and this is more than enough to exactly reconstruct the underlying continuous time sinosoid. To make it more concrete, imagine the sinusoid has a frequency of $1Hz$ and the sample-rate was $10Hz$ (not really practical values for audio signals, but that doesn't matter for a general discussion). Now let's see what happens when we try to sample an $8Hz$ sinusoid at a samplerate of $10Hz$. This is depicted in figure 3. The high frequency sinusoid at $8Hz$ produces a certain sequence of samples as usual. However,

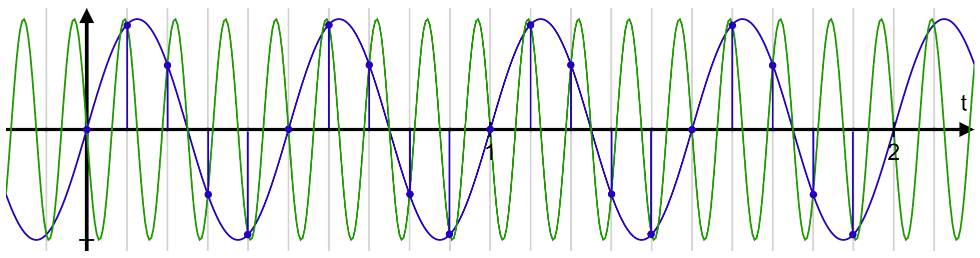

Figure 3: Aliasing: the sinusoid at $f = 8Hz$ is aliased into another sinusoid at $f = 2Hz$, at a samplerate $f_s = 10Hz$ because both produce the same sequence of samples

there exists a sinusoid of frequency $2Hz$ which would give rise to the exact same sequence of samples - and it is this lower frequency sinusoid which will be reconstructed by our interpolation algorithm. The general rule is: whenever we sample a sinusoid of a frequency f above half the sample-rate but below the samplerate ($f_s > f > \frac{f_s}{2}$), then its alias will appear at a frequency of $f' = f_s - f$ or $f' = \frac{f_s}{2} - (f - f_s/2)$ where the second form makes it clear that we must reflect the excess of the signals frequency over the Nyquist frequency at the Nyquist frequency to obtain its alias. Figure 4 illustrates the process of reflecting at the Nyquist frequency for a signal with a spectrum that violates the condition of inequation 2 - this reflection is as sometimes termed as foldover. When the sinusoid is even above the samplerate, we will see it spectrally shifted into the band below half the sample-rate (which is called baseband). For even higher frequencies the whole thing repeats itself over and over - this explanation is probably a bit quirky, but figure 5 should give a fairly good intuition of which goes where and when we have to reflect. The fact that higher frequencies intrude into the spectrum which we want to capture calls for lowpass-filtering the continuous signal before we sample it. This is, what we need the so called Anti-Alias filters for. Without those filters, we would have our spectrum completely messed up with aliasing products of frequencies above the Nyquist frequency. Moreover, these alias products will in general not have any harmonic relationship to our signal to be captured, which makes them sonically much more annoying than harmonic distortion as we know them from analog equipment. Aliasing does not only occur when we want to capture analog

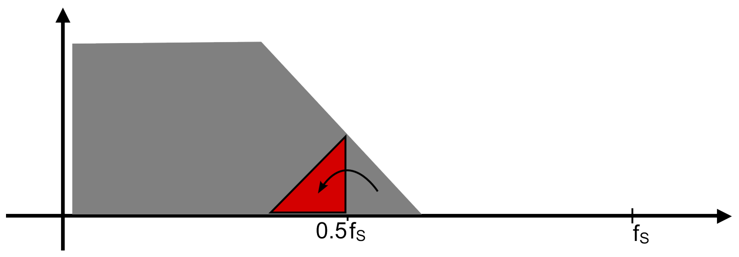

Figure 4: Aliasing from a spectral point of view: parts of the spectrum above the Nyquist frequency are reflected back into the baseband

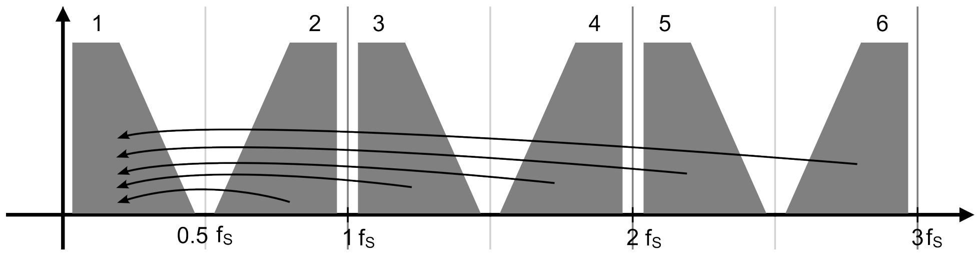

Figure 5: If the original spectrum would have looked either one of the gray blocks, the resulting reconstructed spectrum would always look like the first gray block

signals which are not properly anti-alias filtered, but also when we generate signals inside the computer in the first place.

Quantization

After the sampling we have a sequence of numbers which can theoretically still take on any value on a continuous range of values. Because this range in continuous, there are infinitely many possible values for each number, in fact even uncountably infinitely many. In order to be able to represent each number from such a continuous range, we would need an infinite number of digits - something we don't have. Instead, we must represent our numbers with a finite number of digits, that is: after discretizing the time-variable, we now have to discretize the amplitude-variable as well. This discretization of the amplitude values is called quantization. Assume, our sequence takes on values in the range between $-1... + 1$. Now assume that we must represent each number from this range with just two decimal digits: one before and one after the point. Our possible amplitude values are therefore: $-1.0, -0.9, \dots, -0.1, 0.0, 0.1, \dots, 0.9, 1.0$. These are exactly 21 distinct levels for the amplitude and we will denote this number of quantization levels with N_q. Each level is a step of 0.1 higher than its predecessor and we will denote this quantization stepsize as q. Now we assign to each number from our continuous range that quantization level which is closest to our actual amplitude: the range $-0.05... + 0.05$ maps to quantization level 0.0, the range 0.05...0.15 maps to 0.1 and so on. That mapping can be viewed as a piecewise constant function acting on our continuous amplitude variable x. This is depicted in figure 6. Note, that this mapping also includes a clipping operation at ± 1: values larger than 0.95 are mapped to quantization level 1, no matter how large, and analogous for negative values.

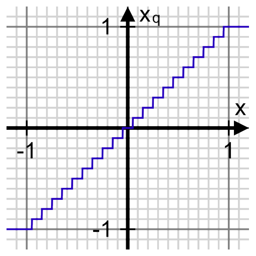

Figure 6: Characteristic line of a quantizer - inputs from a continuous range x are mapped to discrete levels x_q

Quantization Noise

When forcing an arbitrary signal value x to its closest quantization level x_q, this x_q value can be seen as x plus some error. We will denote that error as e_q (for quantization error) and so we have:

$$x_q = x + e_q \qquad \Leftrightarrow \qquad e_q = x_q - x \tag{3}$$

The quantization error is restricted to the range $-q/2... + q/2$ - we will never make an error larger than half of the quantization step size. When the signal to be sampled has no specific relationship to the sampling process, we can assume, that this quantization error (now treated as a discrete time signal $e_q[n]$) will manifest itself as additive white noise with equal probability for all error values in the range $-q/2... + q/2$. Mathematically, we can view the error-signal as a random signal with a uniform probability density function between $-q/2$ and $+q/2$, that is:

$$p(e) = \begin{cases} \frac{1}{q} & \text{for } \frac{-q}{2} \leq e < \frac{q}{2} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

For this reason, the quantization error is often also called quantization noise. A more rigorous mathematical justification for the treatment of the quantization error as uniformly distributed white noise is provided by Widrow's Quantization Theorem, but we won't pursue that any further here. We define the signal-to-noise ratio of a system as the ratio between the RMS-value of the wanted signal and the RMS value of the noise expressed in decibels:

$$SNR = 20 \log_{10} \frac{x_{rms}}{e_{rms}} \tag{5}$$

where the RMS value is the square root of the (average) power of the signal. Denoting the power of the signal as P_x and the power of the error as P_e and using an elementary rule for logarithms, we can rewrite this as:

$$SNR = 10 \log_{10} \frac{P_x}{P_e} \tag{6}$$

The power of the quantization error is given by the variance of the associated continuous random variable and comes out as:

$$P_e = \int_{\frac{-q}{2}}^{\frac{q}{2}} \frac{1}{q} e^2 \, de = \frac{q^2}{12} \tag{7}$$

We take a sinusoid at unit amplitude as the wanted signal, this signal has a power of:

$$P_x = \frac{1}{2} \tag{8}$$

Putting these expressions into equation 6, we obtain:

$$SNR = 10 \log_{10} \frac{6}{q^2} \tag{9}$$

To reiterate, this equation can be used to calculate the SNR between an unit amplitude sinusoid and the quantization error that is made when we quantize the sinusoid with a quantization step size of q. In the discussion above, we used a step size of 0.1 - this was only for easy presentation of the principles. Computers do not represent numbers using decimal digits, instead they use binary digits - better known as bits. With b bits, we can represent 2^b distinct numbers. Because we want those numbers to represent the range between -1 and 1, we could represent 2^{b-1} quantization levels below zero and 2^{b-1} above zero - however, we have not yet represented the zero level itself. To do so, we 'steal' one binary pattern from the positive range - now we can only represent $2^{b-1} - 1$ strictly positive levels. The step size is given by:

$$q = \frac{1}{2^{b-1}} \tag{10}$$

Having all this machinery in place, we are now in the position to calculate the SNR for a quantization of the signal with b bits as:

$$SNR = 10 \log_{10}(6 \cdot 2^{2b-2}) \tag{11}$$

and as an easy rule of thumb:

$$SNR \approx 6 \cdot b \tag{12}$$

This says: each additional bit increases the signal-to-noise ratio by $6dB$. The exact formula gives an SNR of roughly $98dB$ for 16 bits and an SNR of $146dB$ for 24 bits (as opposed to 96 and 144 for the thumb-rule). Taking a sinusoid at full amplitude as reference signal is of course a rather optimistic assumption, because this is a signal with a quite high power - its power is $\frac{1}{2}$ and its RMS value therefore $\frac{1}{\sqrt{2}} = 0.707$..... You are probably familiar with the notion of RMS-levels in decibels: the unit amplitude sine has an RMS level of $-3.01dB$ in this notation - such a high RMS level will occur rarely for practical signals. Thus, we would expect a somewhat lower SNR for practical signals. A practical implication of the above analysis is, that when recording audio signals, we have to make a trade off between headroom and SNR. Specifically: for each $6dB$ of headroom we effectively relinquish one bit of resolution.