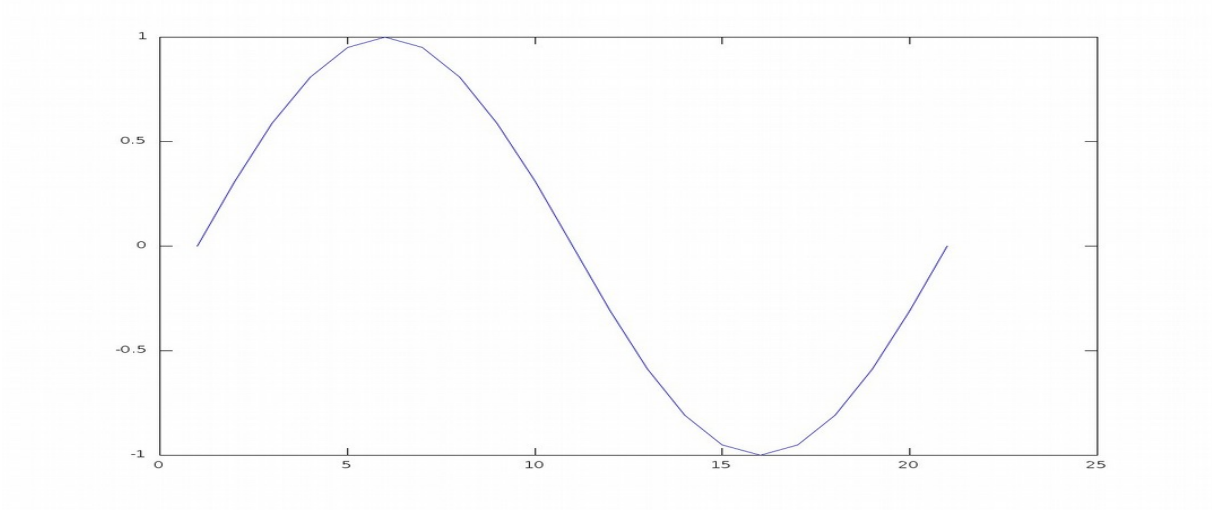
Digital Signal Processing 2/ Advanced Digital Signal Processing
Lecture 3,
SNR, non-uniform Quantisation

Gerald Schuller, TU Ilmenau

What is our SNR if we have a sinusoidal signal? What is its pdf? Basically it is its normalized histogram, such that its integral becomes 1, to obtain a probability distribution.



If we look at the signal, and try to see how probable it is for the signal to be in a certain small interval on the y axis, we see that the signal stays longest around +1 and -1, because there the signal slowly turns around. Hence we would expect a pdf, which has peaks at +1 and -1.
If you calculate the pdf of a sine wave, x=sin(t), with t being continous and with a range larger than 2pi, then the result is

$$p(x)=\frac{1}{\pi\cdot\sqrt{1-x^2}}$$

This results from the derivative of the inverse sine function (arcsin). This derivation can be found for instance on Wikipedia. For our pdf we need to know how fast a signal x passes through a given bin in x. This is what we obtain if we compute the inverse function $x = f^{-1}(y)$, and then its derivative $d f^{-1}(x)/dy$.
Here we can see that p(x) indeed becomes infinite at x=+/-1!
We could now use the same approach as before to obtain the expectation of the power, multiplying it with x^2 and integrating it. But this seems to be somewhat tedious. But since we now have a deterministic signal, we can also try an **alternative** solution, since the sine function is not a probabilistic function, but a deterministic function.
We can simply directly compute the power of our sine signal over t, and then take the average over at least one period of the sine function.

$$E(x^2) = \frac{1}{2\pi} \int_{t=0}^{2\pi} \sin^2(t)\,dt = \frac{1}{2\pi} \int_{t=0}^{2\pi} (1-\cos(2t))/2\,dt$$

the cosine integrated over complete periods becomes 0, hence we get

$$= \frac{1}{2\pi} \int_{t=0}^{2\pi} 1/2\,dt = \frac{1}{2\pi} \cdot \pi = \frac{1}{2}$$

What do we get for a sinusoid with a different amplitude, say $A/2 \cdot \sin(t)$?
The expected power is

$$E(x^2) = \frac{A^2}{8}$$

So this leads to an SNR of

$$SNR = \frac{A^2/8}{\Delta^2/12} = \frac{3 \cdot A^2}{2 \cdot \Delta^2}$$

Now assume again we have a A/D converter with N bits, and the sinusoid is at full range for this converter. Then

$$A = 2^N \cdot \Delta$$

We can plug in this result into the above equation, and get

$$SNR = \frac{3 \cdot 2^{2N} \cdot \Delta^2}{2 \cdot \Delta^2} = 1.5 \cdot 2^{2N}$$

In dB this will now be

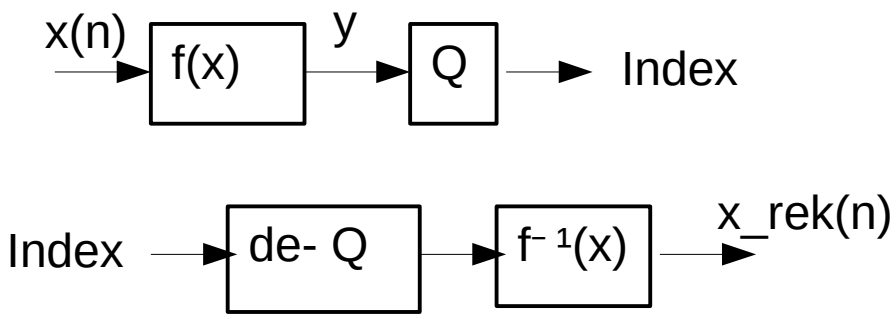
$$10 \cdot \log_{10}(SNR) = 10 \cdot \log_{10}(1.5) + N \cdot 20 \cdot \log_{10}(2) =$$
$$= 1.76\,dB + N \cdot 6.02\,dB$$

Here we can see now, that using a sinusoidal signal instead of a uniformly distributed signal gives us a **boost of 1.76 dB** in SNR. This is because it is more likely to have larger values!
We see that our rule of 6dB more SNR for each bit still holds!

Companding

This is a scheme to make the SNR less dependent on the signal size.
This is a synonym for compression and expanding. Uniform quantization can be seen as a quantization value which is constant on the absolute scale. Non-uniform quantization, using companding, can be seen as having step sizes which stay constant relative to the amplitude, their step size grows with the amplitude.
We obtain this non-uniform quantization by first applying a non-linear function to the signal (to boost small values), and

then apply a uniform quantizer. On the decoding side we first apply the reverse quantizer, and then the inverse non-linear function (the expansion, we reduce small values again to restore their original size).



The range of (index) values is compressed, smaller values become larger, large values don't grow as fast. The following functions are standardized as “ μ -Law” and “A-Law”:

EQUATION 22-1
Mu law companding. This equation provides the nonlinearity for μ255 law companding. The constant, μ, has a value of 255, accounting for the name of this standard.

$$y = \frac{\ln(1+\mu x)}{\ln(1+\mu)} \qquad \text{for } 0 \le x \le 1$$

EQUATION 22-2
"A" law companding. The constant, A, has a value of 87.6.

$$y = \frac{1+\ln(Ax)}{1+\ln(A)} \qquad \text{for } 1/A \le x \le 1$$

$$y = \frac{Ax}{1+\ln(A)} \qquad \text{for } 0 \le x \le 1/A$$

(From: http://www.dspguide.com/ch22/5.htm, also below)

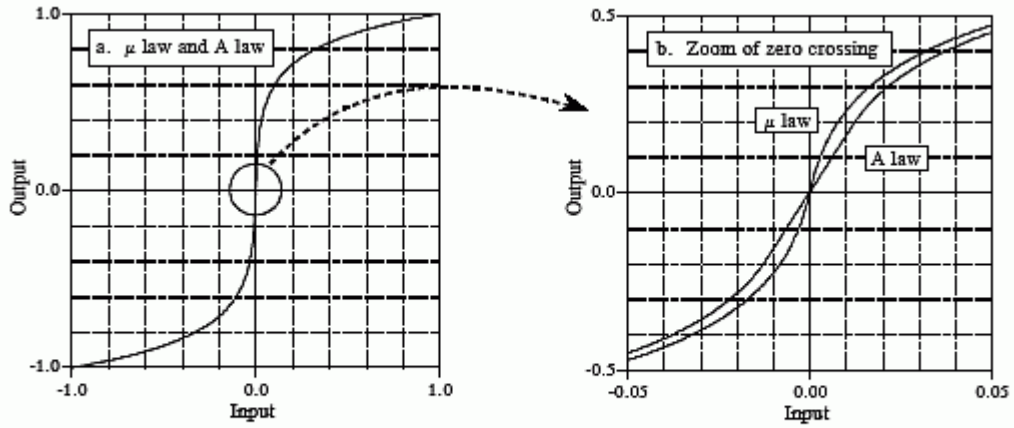


FIGURE 22-7
Companding curves. The μ255 law and "A" law companding curves are nearly identical, differing only near the origin. Companding increases the amplitude when the signal is small, and decreases it when it is large.

See also:
https://en.wikipedia.org/wiki/%CE%9C-law_algorithm

This "**compression**" function is applied **before a uniform quantizer** in the **encoder**. In the decoder, after uniform reverse quantisation, the inverse function is applied, turning y back into x.
Observe that these equations assume that we first normalize our signal and keep its sign separate.
Example: For $\mu = 255$ (which is used in the standard) and an x with a maximum amplitude of A, hence $-A \le x \le A$ we obtain:

$$y = sign(x) \cdot \frac{\ln(1+255 \cdot |(x/A)|)}{\ln(1+255)}$$

In the example of 8-bit mu-law PCM the quantization **index** is then (for a Mid-Tread quantizer following this compression function):

$$index = round(y/\Delta)$$

Here, y has the range of -1,…,+1. Hence the quantization step size for 8 bits is

$$\Delta=(1-(-1))/2^8=2/2^8=1/2^7$$

The index is then encoded as an 8 bit **codeword**.

In the **decoder** we compute the de-quantized y from a Mid-Tread de-quantizer, including its sign from the index,

$$y_{rek}=index\cdot q$$

and we compute the inverse compression function, the "**expanding**" function (hence the name "**companding**")
We obtain the inverse through the following steps,

$$|(y)|=\frac{\ln(1+255\cdot|(x/A)|)}{\ln(1+255)}$$

$$|y|\cdot\ln(256)=\ln(1+255\cdot|x/A|)$$

$$\rightarrow e^{\ln(256)\cdot|y|}=1+255\cdot|x|/A$$

$$\rightarrow\frac{(256^{|y|}-1)}{255}\cdot A=|x|$$

$$\rightarrow x=sign(y)\frac{(256^{|y|}-1)}{255}\cdot A$$

(in the decoder we replace y by y_{rek} and x by x_{rek}).
This x is now our **de-quantized value** or signal.
Observe that with this companding, the effective quantisation step size remains approximately constant **relative** to the **signal amplitude**. Large signal components have large effective step sizes and hence larger quantisation errors, small signals have smaller effective quantisation step sizes and hence smaller quantisation errors. In this way we get a

more or less **constant SNR** over a wide **range of signal amplitudes**.
Important point to remember: this approach is **identical** to having **non-uniform quantization** step sizes, smaller step sizes at smaller signal values, and larger step sizes at larger signal values. The compression and expanding of the signal makes the uniform step sizes “look” relatively smaller to the signal, it has more quantization steps to cover. And this has the same effect as a smaller signal with smaller quantization steps.

Python example to mu-law quantizer:
First listen to the uniform mid-tread quantizer using our python script, and make sure the part for the mid-tread quantizer is un-commented:
`python pyrecplay_quantizationblock.py`
Observe that the voice dissapear, is rounded to zero, if the speaker is at some distance from the microphone.
Now use mu-law quantization with:
`python pyrecplay_mulawquantizationblock.py`
Observe: At the distance at which the speech disappeared in the uniform case, the speech is now there. Here, the smaller quantisation steps help with the smaller amplitudes of the speech signal.
At closer distance, where the signal amplitudes are bigger, there is not much difference.

The programs use 16 bit audio samples from the sound card, 1 bit is for the sign, 15 bits remain for the magnitude, hence the factor $2^{15}=32678$ for

the normalization to the range of the formula of 0...1.
The mu-law program simulates 4 bit quatization (16 quantization intervals).