1 Quantization noise

The process of quantisation is happening all around us all the time. Whenever we use the in-built microphone or camera in our phone or laptop, for example, we are utilizing quantisation to bring real-world, continuously-varying sounds and images in to the discrete-time, discrete-amplitude domain. The measurement transducers in the closed-loop feedback systems that are the mainstay of the process industry, similarly, capture continuously-varying signals using a sample+hold and quantising process.

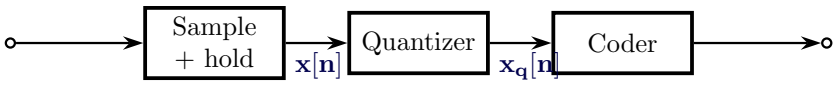


Figure 1: Analog-to-digital converter (including quantizer)

Denote the input to our quantiser as $x[n]$.

Figure 2 offers an intuitive picture of what happens during the quantisation process.

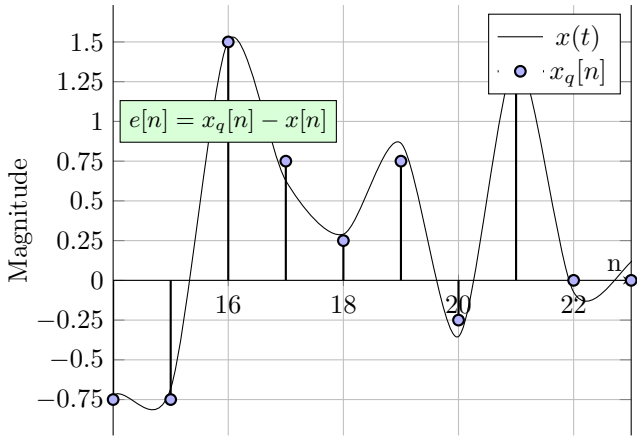


Figure 2: Quantization of continuous-varying signal

Two quantities are important in our analysis here. The first is the quantisation level, Q, given by

$$\boxed{Q = \frac{R}{2^b}}. \tag{1}$$

The second is the variance, or power, in the quantisation error signal. If the original signal meets certain criteria, it is reasonable to assume that the quantisation error is uniformly distributed between $-Q$ and Q. This leads to Equation 2.

$$\sigma_e^2 = \frac{Q^2}{12} \tag{2}$$

where σ_e, according to standard notation, denotes the standard deviation in the error signal. Qualitatively, Equation 1 simply indicates that the quantisation level is proportional to the range, R, and inversely proportional to the number of discrete levels.

We don't need to take this statement on faith, of course. We can compute the actual variance of our error signal with MATLAB. In this case, we will use a normally-distributed random signal with a mean of zero and variance of one.

```
% Generate a normally-distributed (pseudo-)random signal
% with zero mean and variance one.
x = randn(size(n));
```

This is an approximation of white noise, and so meets the criteria necessary for Equation 2 to be valid. We then quantise this signal and subtract to get the quantisation error.

```
xq = round(x*2^(b-1)) / 2^(b-1);
% What is the actual error?
e = xq - x;
var(e)
```

The result is $\sigma_e^2 = .0052117$, as expected.

Sinusoidal inputs Let us try again, but using a sinusoidal input signal.

```
f = 7;
x = cos(2*pi*f*n/N);
```

In this case, the quantisation error is *not* uniformly distributed between $-\frac{Q}{2}$ and $\frac{Q}{2}$, as the histogram in Figure 3. Equation 2 rests on the assumption that the quantisation error *is* uniformly distributed across this range, so we can not expect the equation to be valid in this case. Indeed, it is not. The variance, in this case, is

$$\sigma_e^2 = .004594$$

.

```
% Title: Distribution and variance of quantisation noise
% Author: David Collins

N = 2^14; % no. of samples
n = 0:N-1;
% No. of bits
b = 3;

% Generate a normally-distributed (pseudo-)random signal
% with zero mean and variance one.
x = randn(size(n));

% Quantise these values.
% Note that we are rounding _down_ in this case.
%xq = floor(x*2^(b-1)) / 2^(b-1);
% We could, alternatively, round it to the nearest number.
```

```
xq = round(x*2^(b-1)) / 2^(b-1);

% What is the actual error?
e = xq - x;

% We expect the variance to be approximately equal to Q^2 / 12
% (where Q is the quantisation level).
Q = 1 / 2^(b-1)
Q^2 / 12

var(e)

%plotAnnotation1 = sprintf('$\sigma_e^2$ = %.2f', var(e));

% Compare the quantised signal with the original (over a certain
% range of samples at least)
rng = 128:196;
subplot(2,1,1);
plot(x(rng),'r');
hold on;
stem(xq(rng), 'bo');
grid();
xlabel('Sample number');
ylabel('Magnitude');
title('Normally-distributed random signal - original and quantised
    versions');
% Let's plot the error itself - to get a sense of its
% time-domain characteristics.
subplot(2,1,2);
plot(e);
grid();
xlabel('Sample number');
ylabel('Magnitude');
title('Quantisation error');
pause();

print -dpdf 'quantNoise1.pdf';

% Lets plot a histogram of the error also.
clf();
hist(e);

print -dpdf 'quantNoise1-histogram.pdf';

A = [n' x' xq' e'];
rows = 13:100;
csvwrite('quantNoise.csv', A(rows,:));

save('quantisedInput.mat', 'xq', 'x');
```

1.1 Variance of filter output

What happens if we pass this quantised signal through a filter? Lets examine what happens by constructing a low-pass filter and passing our quantised signal through it. We will determine the error in the output signal (Figure 5) according

to

$$e_o[n] = y_q[n] - y[n].$$

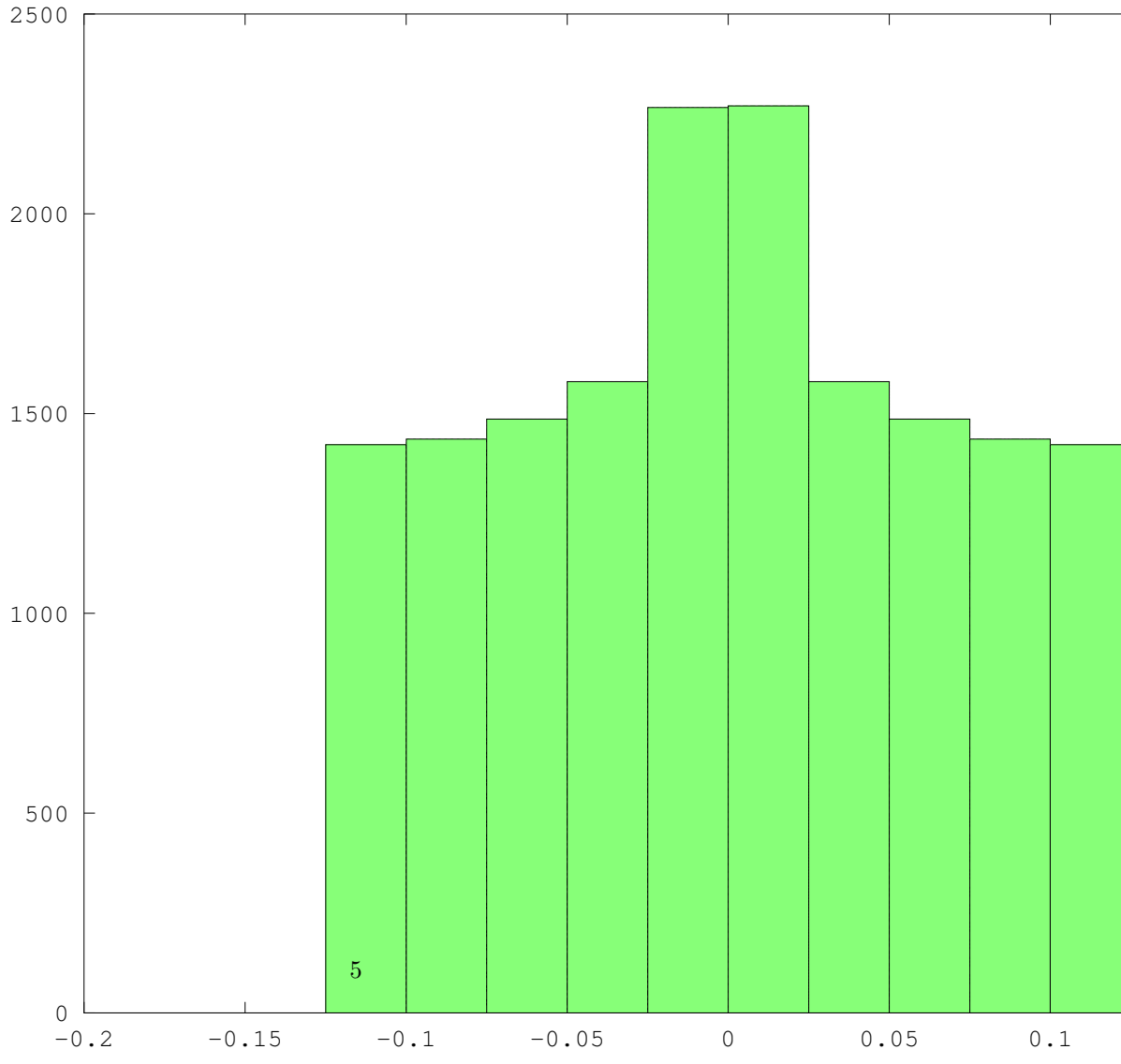
It can be shown, then, that the variance of the error in the output is given by

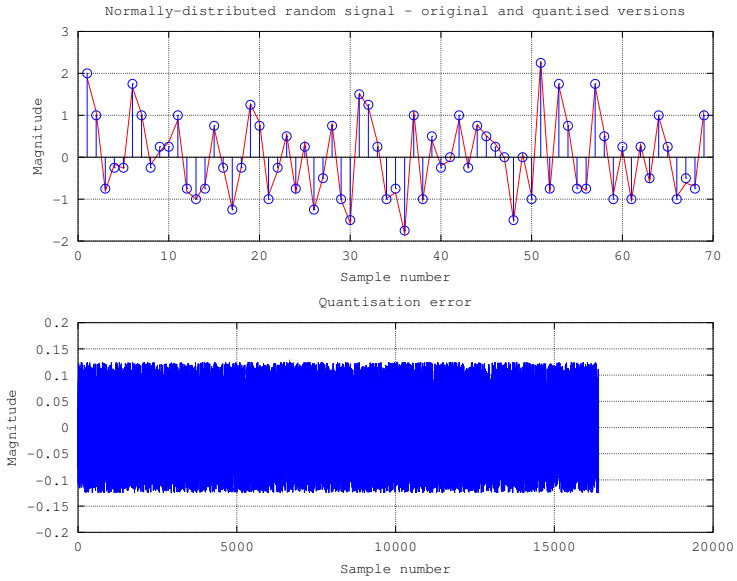
$$\sigma_o^2 = \sigma_e^2 \sum_{n=0}^{\infty} |h[n]|^2 . \tag{3}$$

We already know σ_e^2, but $\sum_{n=0}^{\infty} |h[n]|^2$ depends on the particular filter. When we design a filter we may be working with the b and a coefficients rather than the impulse response, h. We can determine the impulse response using MATLAB's impz(b,a) function, however. This is demonstrated below (Listing 1).

Listing 1: Quantised signals and filtering. What is the variance of the error in the output?

```
% Filter to examine the effects of quantization noise
% in discrete systems.
% Author: David Collins

% Load quantised input signal
load('quantisedInput.mat');

% Low-pass filter with cut-off frequency at 2/5 pi radians / sample
wp = 2/5; % pass-band edge frequency
ws = 3/5; % stop-band edge frequency
Rp = .1; % amount of ripple in pass-band
Rs = 30; % degree of attenuation in stop-band

[order,wp] = ellipord(wp, ws, Rp, Rs);

[b,a] = ellip(order, Rp, Rs, wp);

printf('Filter order is %d\n', order);

% What is the impulse response of our filter?
h = impz(b,a);

% Sum of the square of the elements of the impulse response
h*h'

% What is the output of the filter when we pass the quantised input
    through it?
yq = filter(b, a, xq);
y = filter(b, a, x);

e = yq - y;

% Determine the variance of the signal
var(e)
```


2500
2000
1500
1000
500
0
-0.2
-0.15
-0.1
-0.05
0
0.05
0.1
5



;

Figure 4: Quantization error analysis in MATLAB

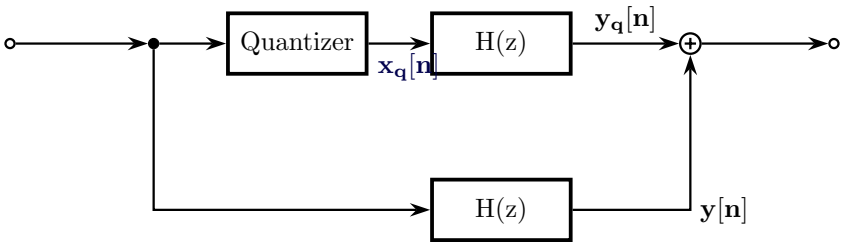


Figure 5: Passing the quantised signal through a filter