FINITE PRECISION EFFECTS

1. FLOATING POINT VERSUS FIXED POINT
2. WHEN IS FIXED POINT NEEDED?
3. TYPES OF FINITE PRECISION EFFECTS
4. FACTORS INFLUENCING FINITE PRECISION EFFECTS
5. FINITE PRECISION EFFECTS: FIR VERSUS IIR FILTERS
6. DIRECT FIR FILTER STRUCTURE
7. CASCADE FIR FILTER STRUCTURE
8. STRUCTURES WITH QUANTIZERS
9. THE OVERFLOW PROBLEM
10. TIME-DOMAIN SCALING
11. FREQUENCY-DOMAIN SCALING
12. OVERFLOW OF INTERNAL SIGNALS
13. QUANTIZATION OF FILTER COEFFICIENTS
14. COEFFICIENT QUANTIZATION: EXAMPLE
15. SIGNAL QUANTIZATION
16. QUANTIZATION ERROR POWER
17. EXPRESSING σ_e^2 IN TERMS OF THE NUMBER OF BITS
18. SIGNAL TO QUANTIZATION NOISE RATIO (SNR)
19. SIGNAL QUANTIZATION

FLOATING POINT VERSUS FIXED POINT

- Signals can not be represented exactly on a computer, but only with finite precision.
- In floating point arithmetic, the finite precision errors are generally not a problem.
- However, with fixed point arithmetic, the finite word length causes several problems.
- For this reason, a floating point implementation is preferred when it is feasible.
- Because of finite precision effects, fixed point implementation is used only when

  1. speed,
  2. power,
  3. size, and
  4. cost

  are very important.

WHEN IS FIXED POINT NEEDED?

From Matlab documentation: "When you have enough power to run a floating point DSP, such as on a desktop PC or in your car, fixed-point processing and filtering is unnecessary. But, when your filter needs to run in a cellular phone, or you want to run a hearing aid for hours instead of seconds, fixed-point processing can be essential to ensure long battery life and small size.

Many real-world DSP systems require that filters use minimum power, generate minimum heat, etc. Meeting these requirements often means using a fixed-point implementation."

TYPES OF FINITE PRECISION EFFECTS

- Overflow
- Quantization of filter coefficients
- Signal quantization
  1. Analog/Digital conversion (A/D converter)
  2. Round-off noise
  3. Limit cycles (recursive filters only)

FACTORS INFLUENCING FINITE PRECISION EFFECTS

1. The structure used for implementation (Direct, transpose, etc)
2. The word-length and data-type (fixed-point, floating-point, etc) (1-s complement, 2-s complement, etc)
3. The multiplication-type (rounding, truncation, etc)

FINITE PRECISION EFFECT: FIR VERSUS IIR FILTERS

Remarks on FIR filters:

1. Filter coefficient quantization: coefficient quantization is quite serious for FIR filters due to the typically large dynamic range of coefficients of FIR filters.
2. Round-off noise: Round-off noise is not as serious for FIR filters as it is for IIR filters.
3. Limit cycles: Non-recursive (FIR) filters do not have limit cycles.
4. For FIR filters, the direct form is generally preferred.

Remarks on IIR filters:

1. Filter coefficient quantization: coefficient quantization can make a stable IIR filter unstable! (The implementation of an IIR filters using a cascade of second order sections prevents that.)
2. Round-off noise: For a cascade of second order sections the round-off noise depends on
   (a) the poles-zero pairing,
   (b) the ordering of the sections.

DIRECT FIR FILTER STRUCTURE

To clarify the forgoing discussion on the effects of fixed-point arithmetic, it will be convenient to refer a couple of different structures for the implementation of FIR filters.

Fig 1

This structure implements an FIR filter of length 4. The output $y(n)$ is given by

$$y(n) = h(0)\,x(n) + h(1)\,x(n-1) + h(2)\,x(n-2) + h(3)\,x(n-3)$$

and so the impulse response is

$$h(n) = h(0)\,\delta(n) + h(1)\,\delta(n-1) + h(2)\,\delta(n-2) + h(3)\,\delta(n-3)$$

and the transfer function is given by

$$H(z) = h(0) + h(1)\,z^{-1} + h(2)\,z^{-2} + h(3)\,z^{-3}.$$

This is called a *direct* structure because the transfer function parameters appear as multipliers in the structure itself.

CASCADE FIR FILTER STRUCTURE

If the FIR transfer function $H(z)$ is factored as

$$H(z) = H_1(z)\,H_2(z)$$

then the FIR filter can be implemented as a *cascade* of two shorter FIR filters. If each of the shorter filters are implemented with the direct form, then the total flowgraph for the cascade structure is as shown in the following figure.

Fig 2

If both structures are implemented with infinite precision, then they will implemented exactly the same transfer function. However, when they are implemented with finite-precision arithmetic, they will have different properties. Some structures are more immune than other structures when fixed-point precision is used.

STRUCTURES WITH QUANTIZERS

In fixed-point implementations

1. The multipliers are quantized.
2. The input, output, and internal signals are quantized.

Both types of quantization introduce some error.

If the binary numbers X and Y are each B bits long, then the product $Z = X \cdot Y$ is $2B - 1$ bits long.
Therefore, the multiplications occurring in a digital filter with fixed point arithmetic must be truncated down to the word length used. (This is represented by the Q block in the following diagrams.)

On the other hand, the *addition* of two-fixed point numbers does not lead to a loss of precision.

In some (old) processors, signal quantization occurs after every multiplier. For example, the FIR direct structure can be illustrated as shown.

Fig 3

In most processors today, the quantizer Q comes *after* the multiply-sum operation. This is the multiply-accumulate, or MAC, command. The following figure illustrates this.

Fig 4

Later, we will see how to analyze the degradation of accuracy caused by the quantizer.
In processors where the quantizer comes after every multiplier, it can be seen that there are many more quantizations than when the quantizer comes after the multiply-sum operation.
We will assume from now that the quantizer comes *after* the sum.

OVERFLOW

The range of values of a B-bit binary number is limited. If a digital filter is not properly implemented, overflow may occur when the output or internal signals are larger than the maximum value possible. To avoid overflow, it may be necessary to scale down the input signal.

- If the signal is not scaled appropriately, many overflows will occur.
- If the signal is scaled down too much, then the the full available dynamic range is not utilized and the ratio of the signal power to the round-off noise (SNR) will be degraded.

Scaling is usually

1. absorbed in the other multipliers (so it does not introduce further round-off noise), or
2. by 2^{-k} so it can be implemented by a bit shift (to minimize complexity)

TIME-DOMAIN SCALING

Suppose the signals $x(n)$ and $y(n)$ can only be represented in the range

$$|x(n)| \le x_{max}$$
$$|y(n)| \le y_{max}$$

We want to implement a scaled version of the filter $H(z)$ and we want to choose the scaling constant c so as to maximize the dynamic range of the output, while at the same time avoiding $y(n)$ from overflowing (exceeding y_{max}).

$$\begin{tikzcd} x(n) \arrow[r] & \boxed{c \cdot H(z)} \arrow[r] & y(n) \end{tikzcd}$$

Problem: find the maximum value of c such that $|y(n)| \le y_{max}$.
You can assume that $|x(n)| \le x_{max}$.
The solution begins with the convolutional sum.

$$\begin{aligned} y(n) &= \sum_k c\, h(k)\, x(n-k) \\ |y(n)| &\le c \sum_k |h(k)|\, |x(n-k)| \\ &\le c \sum_k |h(k)|\, x_{max} \\ &\le c\, x_{max} \sum_k |h(k)| \\ &\le c\, x_{max}\, ||h(n)||_1 \end{aligned}$$

where the *1-norm* of $h(n)$ is defined as

$$||h(n)||_1 := \sum_k |h(k)|.$$

So

$$|y(n)| \leq c\, x_{max}\, ||h(n)||_1 \leq y_{max}$$

gives

$$\boxed{c \leq \frac{y_{max}}{x_{max}} \cdot \frac{1}{||h(n)||_1}}$$

This is the *time-domain 1-norm* scaling rule.
In practice, c is often

1. rounded downward to $1/2^l$ (so it can be implemented with a bit shift), or
2. absorbed into existing multipliers in the filter structure.

The value

$$c = \frac{y_{max}}{x_{max}} \cdot \frac{1}{||h(n)||_1}$$

is often overly conservative. It guarantees that overflow will never occur, however, it many cases, it is acceptable to allow overflow to occur as long as it is rare, and to use a larger value of c so that the full dynamic range is better utilized.

FREQUENCY-DOMAIN SCALING

Another approach to scaling for overflow control is derived in the frequency domain.

$$\begin{aligned} y(n) &= \text{IDTFT}\left\{Y^f(\omega)\right\} \\ &= \text{IDTFT}\left\{c\, H^f(\omega)\, X^f(\omega)\right\} \\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi} c\, H^f(\omega)\, X^f(\omega)\, e^{-jn\omega}\, d\omega \end{aligned}$$

Because the 'absolute value of an integral is smaller than the integral of the absolute value', we can write

$$\begin{aligned} |y(n)| &\le \frac{c}{2\pi}\int_{-\pi}^{\pi} |H^f(\omega)|\, |X^f(\omega)|\, d\omega \\ &\le \frac{c}{2\pi}\, ||X^f||_\infty \int_{-\pi}^{\pi} |H^f(\omega)|\, d\omega \end{aligned}$$

where the frequency-domain infinity-norm is defined as

$$||X^f||_\infty := \max_\omega |X^f(\omega)|.$$

Defining the frequency-domain 1-norm as

$$||H^f||_1 := \frac{1}{2\pi}\int_{-\pi}^{\pi} |H^f(\omega)|\, d\omega,$$

we can write

$$|y(n)| \le c\, ||X^f||_\infty\, ||H^f||_1$$

To guarantee that no overflows occur, we have the condition

$$|y(n)| \le c\, ||X^f||_\infty\, ||H^f||_1 \le y_{max}$$

or

$$\boxed{c \le \frac{y_{max}}{||X^f||_\infty} \cdot \frac{1}{||H^f||_1}}$$

This is the *frequency-domain 1-norm* scaling rule.
Note that it requires an estimate of $|||X^f|_\infty$.

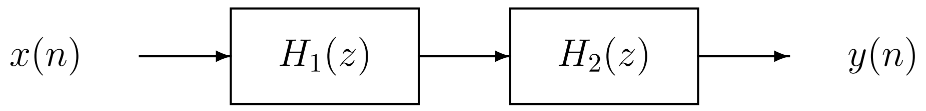
In the development of this scaling rule, we could have swapped the role of $X^f(\omega)$ and $H^f(\omega)$. Then we would obtain the following scaling rule.

$$\boxed{c \leq \frac{y_{max}}{||X^f||_1} \cdot \frac{1}{||H^f||_\infty}}$$

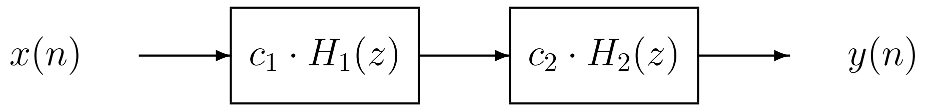
This is the *frequency-domain infinity-norm* scaling rule.

OVERFLOW OF INTERNAL SIGNALS

In addition to ensuring that the output signal rarely overflows, it is also necessary to ensure that the internal signals rarely overflow as well.
For example, consider the cascade of two FIR filters.



Both $H_1(z)$ and $H_2(z)$ must be scaled. We then have the following diagram.



It is necessary to choose the two scaling constants c_1 and c_2 so that overflow of both the intermediate signal *and* the output signal $y(n)$ are rare.
If we call the intermediate (internal) signal $v(n)$, then

1. Choose c_1 first, so that $v(n)$ does not overflow.
2. Choose c_2 second, so that $y(n)$ does not overflow.

If there are more than one internal signal which may overflow, then begin with the first internal signal and proceed through the filter structure.

QUANTIZATION OF FILTER COEFFICIENTS

For a fixed point implementation of a digital filter the filter coefficients must be quantized to binary numbers.

This changes the actual system to be implemented and usually results in a degradation of the frequency response.

The quantization of floating point values can be performed in Matlab with the `round` command:

```
    y = q*round(x/q);
```

where `q` is the quantization step.

COEFFICIENT QUANTIZATION: EXAMPLE

If floating-point (or infinite precision) is used for $h(n)$, then the minimal length TYPE-I FIR filter satisfying

$$1 - \delta_p \leq A(\omega) \leq 1 + \delta_p \qquad |\omega| \leq 0.32\pi$$

$$|A(\omega)| \leq \delta_p \qquad 0.37\pi \leq |\omega| \leq \pi$$

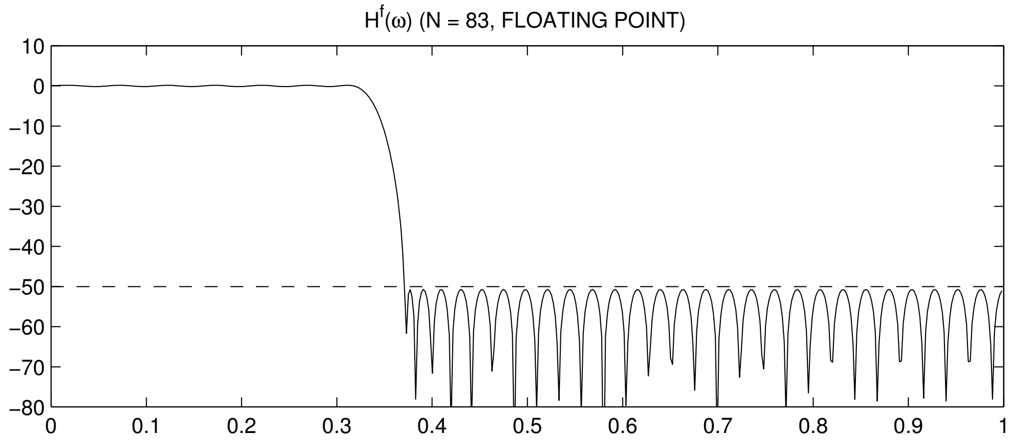
with

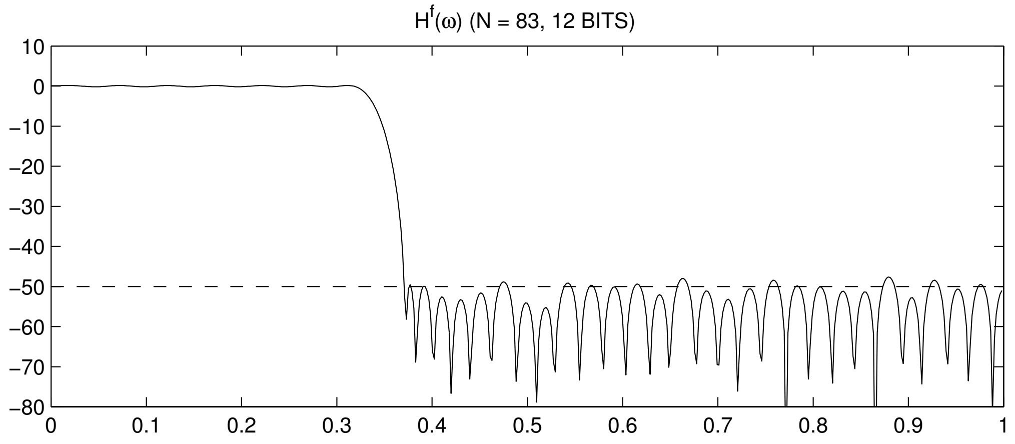
$$\delta_p = 0.02, \qquad \delta_s = -50 \text{ dB}$$

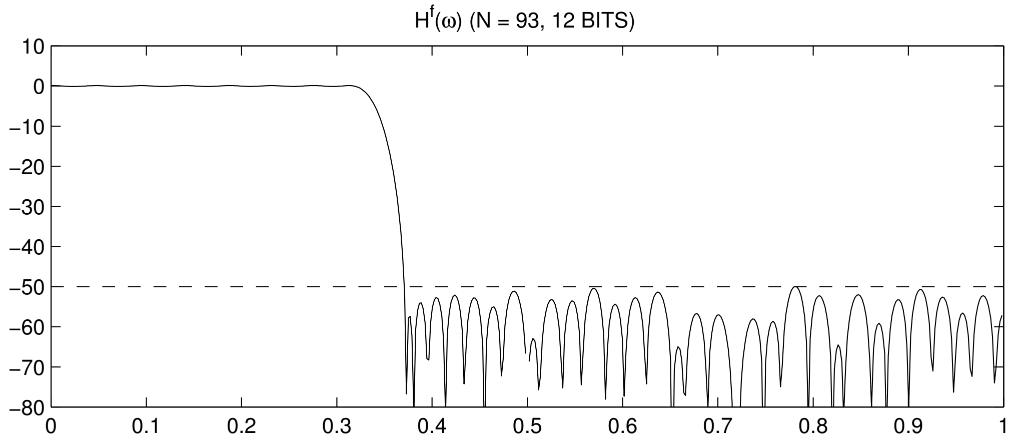
is of length $N = 83$. (See the figures on the next page.)

However, if a *fixed-point* filter $h(n)$ is used by quantizing $h(n)$, then the degraded frequency response does not meet the specifications. A longer impulse response is needed. (See the next page.)

COEFFICIENT QUANTIZATION: EXAMPLE





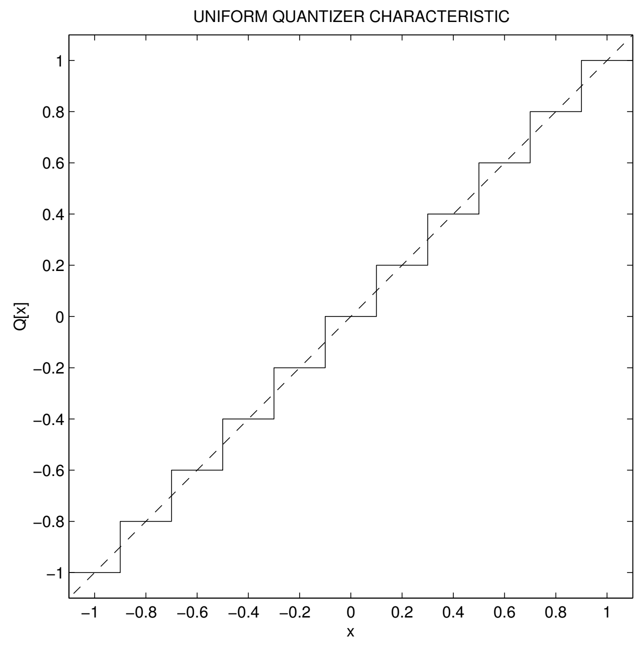


This example was produced with the following Matlab code.

```
N = 83;
Dp = 0.02;           % desired passband error
Ds = 10^(-50/20);    % desired stopband error (50dB)
Kp = 1/Dp;
Ks = 1/Ds;
wp = 0.32*pi;
ws = 0.37*pi;
wo = (wp+ws)/2;
L = 1000;
w = [0:L]*pi/L;
W = Kp*(w<=wp) + Ks*(w>=ws);
D = (w<=wo);
[h,del] = fircheb(N,D,W);
dp = del/Kp;         % actual passband error
ds = del/Ks;         % actual stopband error

[H,w] = freqz(h);
subplot(3,1,1)
plot(w/pi,20*log10(abs(H)),[0 1],20*log10(Ds)*[1 1],'--')
axis([0 1 -80 10])
title('H^f(\omega) (N = 83, FLOATING POINT) ')

b = 12;              % number of bits used for fixed-point word-length
Q = 1/2^b;           % quantization step
hq = Q*round(h/Q);   % quantized coefficients
[Hq,w] = freqz(hq);  % frequency response of quantized coefficients
subplot(3,1,2)
plot(w/pi,20*log10(abs(Hq)),[0 1],20*log10(Ds)*[1 1],'--')
axis([0 1 -80 10])
title('H^f(\omega) (N = 83, 12 BITS) ')

N = 93;                         % increase the filter length
[h,del] = fircheb(N,D,W);       % redesign with longer filter length
hq = Q*round(h/Q);              % quantize longer impulse response
[Hq,w] = freqz(hq);
subplot(3,1,3)
plot(w/pi,20*log10(abs(Hq)),[0 1],20*log10(Ds)*[1 1],'--')
axis([0 1 -80 10])
title('H^f(\omega) (N = 93, 12 BITS) ')
```

SIGNAL QUANTIZATION

Quantizing a signal is a memoryless nonlinear operation.



SIGNAL QUANTIZATION

If $Q[x(n)]$ is the quantized version of $x(n)$, then we can define the quantization error signal $e(n)$ as

$$e(n) := Q[x(n)] - x(n)$$

Then we can write

$$Q[x(n)] = x(n) + e(n)$$

Under many realistic operating conditions, the quantization error signal $e(n)$ will appear to fluctuate randomly from sample to sample. So even though the quantization is totally deterministic, it can be modeled as a random variable.
Specifically, under the following conditions, or assumptions, the quantization error signal can be accurately modeled as random signal which is (1) uncorrelated from sample to sample and (2) uncorrelated from the true signal $x(n)$.

1. The quantization step is small compared to the signal amplitude.
2. Overflow is negligible.
3. The signal changes rapidly enough from sample to sample.

We will assume that a uniform quantizer is used (all quantization steps are equal). If the quantization step is Q, then the quantization error signal will vary between $-Q/2$ and $Q/2$,

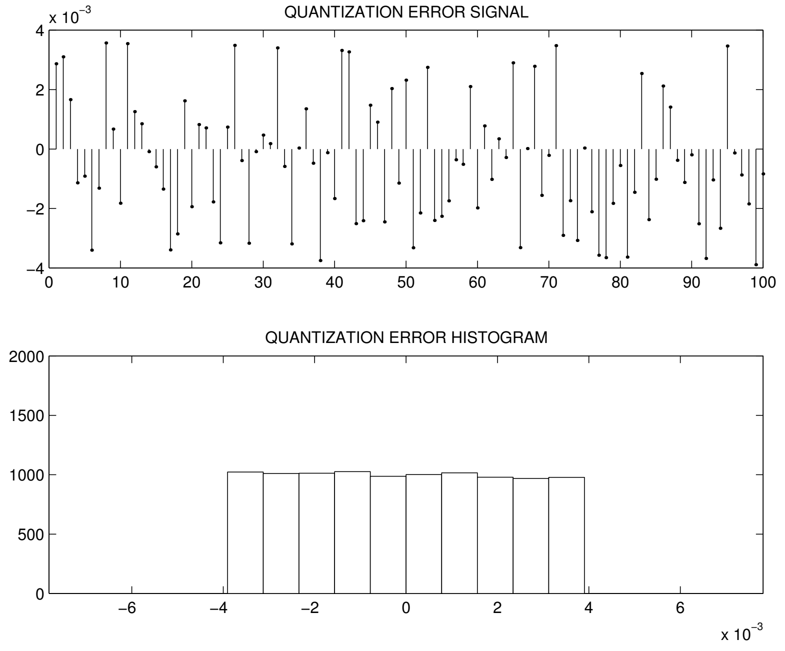
$$-\frac{Q}{2} \le e(n) \le \frac{Q}{2}$$

In addition, if all values between $-Q/2$ and $Q/2$ are equally likely, then the probability distribution function (PDF) $p(e)$ that describes $e(n)$ is uniform between $-Q/2$ and $Q/2$.

Fig 5: Uniform PDF

QUANTIZATION ERROR: MATLAB EXAMPLE

```
L = 10000;
x = 2*rand(L,1)-1;      % original signal x(n), with -1 < x(n) < 1

Rfs = 2;                % Rfs: full scale range
b = 8;                  % number of bits
Q = Rfs/2^b;            % quantization step
y = Q*round(x/Q);       % quantized signal y(n)

e = x - y;              % quantization error signal e(n)

figure(1)
subplot(2,1,1)
stem(e(1:100),'.')
title('QUANTIZATION ERROR SIGNAL')

subplot(2,1,2)
hist(e,10)              % compute histogram (with 10 bins)
colormap([1 1 1]*1)
title('QUANTIZATION ERROR HISTOGRAM')
axis([-Q Q 0 2*L/10])

print -deps sig_quant
```



QUANTIZATION ERROR POWER

The variance of $e(n)$ is used to quantify how bad the quantization error is. As $e(n)$ is zero mean, the variance of $e(n)$ is just given by the average value of $e^2(n)$.

$$\sigma_e^2 := \mathsf{VAR}\{e(n)\} = \mathsf{E}\{e^2(n)\}$$

$\mathsf{E}\{e^2(n)\}$ is the called the *power* of the random signal $e(n)$. We can compute the average value of $e^2(n)$ by the formula

$$\begin{aligned}
\mathsf{E}\{e^2(n)\} &= \int_{-Q/2}^{Q/2} e^2\, p(e)\, de \\
&= \int_{-Q/2}^{Q/2} e^2\, \frac{1}{Q}\, de \\
&= \frac{1}{Q} \cdot \frac{e^3}{3}\bigg|_{e=-Q/2}^{e=Q/2} \\
&= \frac{1}{3\,Q}\left[\left(\frac{Q}{2}\right)^3 - \left(-\frac{Q}{2}\right)^3\right] \\
&= \frac{1}{3\,Q}\left[\frac{Q^3}{8} + \frac{Q^3}{8}\right] \\
&= \frac{Q^2}{3}\left[\frac{2}{8}\right] \\
&= \frac{Q^2}{12}
\end{aligned}$$

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

QUANTIZATION ERROR POWER

The following Matlab program numerically checks the correctness of the expression $\sigma_e^2 = \frac{Q^2}{12}$.

```
L = 10000;
x = 2*rand(L,1)-1;          % original signal x(n)
                            %  with -1 < x(n) < 1

Rfs = 2;                    % Rfs: full scale range
b = 8;                      % number of bits
Q = Rfs/2^b;                % quantization step
y = Q*round(x/Q);           % quantized signal y(n)

e = x - y;                  % quantization error signal e(n)

errvar_meas = mean(e.^2)    % E{e(n)^2} : measured value

errvar_model = Q^2/12       % E{e(n)^2} : value from model
```

We got the following result when we ran this program, which verifies the formula for σ_e^2.

```
errvar_meas =

   5.1035e-06

errvar_model =

   5.0863e-06
```

Note that each time we run the program, we will get a different value for `errvar_meas` because the simulation begins with a randomly genegerated signal $x(n)$.

EXPRESSING σ_e^2 IN TERMS OF THE NUMBER OF BITS

Lets find and expression for σ_e^2 in terms of the number of bits used.

Let R_{fs} denote the *full-scale range* of the quantizer. That is the range from the least value to the greatest value.

If b bits are used to represent $x(n)$, then there are 2^b quantization levels. Therefore

$$2^b \cdot Q = R_{fs}$$

and

$$Q = \frac{R_{fs}}{2^b} = 2^{-b} \cdot R_{fs}$$

and

$$\sigma_e^2 = \frac{Q^2}{12} = \frac{2^{-2b}\, R_{fs}^2}{12}$$

$$\boxed{\sigma_e^2 = \frac{2^{-2b}\, R_{fs}^2}{12}}$$

SIGNAL TO QUANTIZATION NOISE RATIO (SNR)

The signal to quantization noise ratio (SNR OR SQRN) is defined as

$$\text{SNR} := 10\,\log_{10}\frac{\sigma_x^2}{\sigma_e^2} = 20\,\log_{10}\frac{\sigma_x}{\sigma_e} \qquad \text{in dB}$$

where

$$\sigma_x^2 := \text{VAR}\{x(n)\}$$
$$\sigma_e^2 := \text{VAR}\{e(n)\}$$

It will be interesting to examine how the SNR depends on the number of bits b. First, we can write

$$\text{SNR} = 10\,\log_{10}\sigma_x^2 - 10\,\log_{10}\sigma_e^2.$$

Using $\sigma_e^2 = \dfrac{2^{-2b}\,R_{fs}^2}{12}$ we have

$$\begin{aligned}\text{SNR} &= 10\,\log_{10}\sigma_x^2 - 10\,\log_{10}\left(\frac{2^{-2b}\,R_{fs}^2}{12}\right)\\ &= 10\,\log_{10}\sigma_x^2 - 10\,\log_{10}2^{-2b} - 10\,\log_{10}\left(\frac{R_{fs}^2}{12}\right)\\ &= 10\,\log_{10}\sigma_x^2 + 20\,b\,\log_{10}2 - 10\,\log_{10}\left(\frac{R_{fs}^2}{12}\right)\end{aligned}$$

or

$$\text{SNR} = C + 20\,b\,\log_{10}2$$

where C does not depend on b. As $20\,\log_{10}2 = 6.02$, we have

$$\boxed{\text{SNR} = C + 6.02\,b}$$

That is the well known result: *for each additional bit, the SNR is improved by 6 dB.*