On Fixed-Point Filter Realizations

Josef Goette
Bern University of Applied Sciences, Biel/Bienne
Institute of Human Centered Engineering - microLab
Josef.Goette@bfh.ch

February 7, 2018

Contents

1 Introduction 1

2 Number Representations 11

3 Coefficient Quantization 34

4 Arithmetic Errors 52
  4.1 Signal Quantization ..................... 53
  4.2 Scaling for Dynamic Range ................. 66
  4.3 Pairing/Ordering in Sos Cascades .......... 87
  4.4 Limit Cycles ............................... 91

Symbols and Notation 97

References 104
1 Introduction

Problem Statement

• floating-point filters: their design solves often only part of design problem

• many real-world DSP systems require:
  – use of minimum power
  – generate minimum heat
  – do not induce computational overload
  – $\Rightarrow$ these demands often mean: use fixed-point filters

• quantizing:
  – convert floating-point filter realization into a fixed-point realization
  – $\Rightarrow$ can loose filter performance and accuracy

• to do
  – respect basic rules of thumb ($\Rightarrow$ the art of 
  – simulate and determine the effects of quantization

The acronym DSP means digital signal processing; sometimes, we likewise use the same acronym to mean digital signal processor.
Our present goal is to find realizations of fixed-point filters in dedicated hardware, which fulfill requirements such as “low-power” and “small chip size.”

Concerning MATLAB and its toolboxes, we note that we can well attain our object—to design filters—with the Signal Processing Toolbox, if the object is to design floating-point filters; if we need to design fixed-point filters, however, the Dsp System Toolbox,\(^1\) accompanied by the Fixed-Point Toolbox, is very useful. If we have additionally available the Filter Design HDL Coder Toolbox, we may even produce VHDL or Verilog code for the filters we design. Additionally there exist, to support design work on the level of SIMULINK, the SIMULINK Fixed Point toolbox and the SIMULINK HDL Coder toolbox.

\(^{1}\)If you use an older MATLAB version, we note that as of release R2011a, the previous Filter Design Toolbox and the Signal Processing Blockset have been merged and renamed to Dsp System Toolbox.
Recall from ’Steps in Digital Filter Design’

1. design specifications
   • frequency domain: magnitude-, phase response
   • time domain: impulse-, step response, . . .

2. approximation of desired characteristics \( H(z) \leadsto h[n] \)

3. here we treat: the realization problem

4. the course on electronics treats: the implementation problem

note: the above four steps are not independent
\( \leadsto \) design iterations might be needed

We have already discussed the above items and their inter-relations in [Goe18, Section 6]; there we had:

**The approximation problem** is to find a causal, linear, time-invariant system described by its transfer function or by its impulse response.

**The realization problem** addresses the selection of a structure that fulfills the design specifications even under finite-wordlength effects of filter coefficients and signals.

**The implementation problem** is the construction of the filter in discrete hardware, in VHDL coding (for integrated-circuit implementations), or in software (to run on a floating-point or on a fixed-point digital signal processor). The goal of the present signal-processing and electronics courses is to obtain VHDL-coded implementations of fixed-point filters.
Illustration of Problem Areas

- consider the following first-order IIR discrete-time filter
- structure

\[ x[n] \quad + \quad y[n] \]

\[ z^{-1} \]

\[ y[n - 1] \]

\[ a_1 \]

- difference equation

\[ y[n] = a_1 y[n - 1] + x[n] \]

- transfer function

\[ H(z) = \frac{1}{1 - a_1 z^{-1}} = \frac{z}{z - a_1} \]
Illustration of Problem Areas (cont’d)

if the filter is implemented on a digital machine

- filter coefficient can have only a certain discrete value $\hat{a}_1$
- $\hat{a}_1$ is, in general, only an approximation of the original design parameter $a_1$ (which is a real number)
- true and desired transfer function

$$\hat{H}(z) = \frac{z}{z - \hat{a}_1} \approx H(z) = \frac{z}{z - a_1}$$

- therefore: the true frequency response is different from the desired frequency response
- $\sim$ coefficient-quantization problem

We note in passing that the coefficient-quantization problem is similar to the sensitivity problem encountered in the design of analog filters; there, the true values of the components do not exactly match the designed values; here, the true coefficient values do not exactly match the designed coefficient values.
Illustration of Problem Areas (cont’d)

coefficient-quantization problem

• only during design process

• perturbation of $h[n]$ \( \rightarrow \) $H(z)$, $\mathcal{H}(\hat{\omega})$
  
  \begin{itemize}
      \item deterministic perturbation
      \item system is still linear
  \end{itemize}

• check the quantized design; if NOK
  
  \begin{itemize}
      \item do a redesign
      \item do a re-structuring
      \item allocate more bits to coefficients
      \item \ldots
  \end{itemize}

• important: the structure of a filter network has a dramatic effect on its sensitivity to coefficient quantization

We use the acronym NOK to mean *not* OK; OK means \ldots, OK you know it.
Illustration of Problem Areas (cont’d)

input sampling error (analog-to-digital (Ad) conversion error)

- given analog (continuous-time) signal \( x(t) \)
- continuous-to-discrete conversion
  \[
  x[n] \doteq x(t)|_{t=nT_s} = x(t = nT_s)
  \]
- amplitude discretization in Ad converter
  \[
  \hat{x}[n] \approx x[n] : \text{ model: } \hat{x}[n] = x[n] + e[n]
  \]
- where the sequence \( e[n] \) is the Ad conversion error generated by the input (amplitude-) quantization process

We denote by \( T_s \) the sampling-time interval of the assumed regular\(^2\) sampling process; correspondingly, \( f_s = 1/T_s \) is the sampling frequency.

We note in passing that beside the presently considered digital implementations there exist also so-called sampled-data implementations, which are implemented in technologies such as switched-capacitor circuits, switched-current circuits, and the like. In these sampled-data systems, the signals are likewise discrete-time signals \( x[n] \); however, these implementations have not discrete amplitude values, but, instead, continuous amplitude values (continuous voltage and charge values in switched-capacitor circuits, and continuous current values in switched-current circuits). Nevertheless, also these continuous amplitude

\(^2\)The samples are spaced by equal time intervals.
values can never be exact because there is always noise in such analog circuits. Hence, the rôle played by the above quantization noise $e[n]$ in digital systems is played by sampled continuous-time noise in sampled-data systems.
Illustration of Problem Areas (cont’d)

arithmetic operations lead to quantization errors

- in our first-order IIR example

\[ x[n] \rightarrow + \rightarrow y[n] \]

\[ v[n] \rightarrow a_1 \rightarrow y[n-1] \]

- we have \( v[n] = a_1 y[n-1] \)

- result \( v[n] \) is quantized to fit the register holding the result (truncation or rounding)

- we can keep only \( \hat{v}[n] \)

\[ \hat{v}[n] \approx v[n] : \text{model: } \hat{v}[n] = v[n] + e_{a_1}[n] \]

- where the sequence \( e_{a_1}[n] \) is the error sequence generated by the product quantization process \( \sim \) roundoff error

- properties of roundoff errors are similar to Ad-conversion errors

- there might also be overflow errors in accumulators

It is clear that any arithmetic operation might cause overflow errors; especially in standard filter realizations, we might have, beside overflow in accumulators, also overflow in multipliers used to realize the filter coefficients. We have, in the above slide, not
mentioned that kind of overflow, because usually in filter designs the complete filter is scaled such that coefficients result which have absolute values that are smaller than unity. Therefore, in the slide of page 9 for example, the signal \( v[n] = a_1y[n-1] \) is smaller than the signal \( y[n-1] \) and will not overflow if \( y[n-1] \) has not overflowed. Clearly, precision might be lost if not sufficiently many bits are assigned to the result of the multiplication, but there is, in the discussed situation, no overflow.

We note that there might be an additional error source in digital filters which are used to emulate analog filters: If the output signal \( y[n] \) has to be converted back to an analog signal by a digital-to-analog (DA) converter that works with fewer bits than \( y[n] \) has, we have an additional re-quantization error. Because of accumulation of round-off errors, we must usually allocated more bits to the internal arithmetic of a filter than we use at the input \( x[n] \), and because often the output DA conversion uses the same number of bits as the input AD conversion does, such re-quantization errors are not unusual.
2 Number Representations

Overview

- generally: fixed-point $\leftrightarrow$ floating-point
- fixed-point:
  - arithmetic quantization in multipliers
  - possibility of overflow in adders
- floating-point:
  - arithmetic quantization in both, multipliers and adders
  - practically no overflow (in stable filters)
- our goal: fixed-point filter realizations
  $\leadsto$ fixed-point number representations

Recall that our goal is to realize fixed-point filters that fulfill requirements such as “low power” and “small chip-size.”
Fixed-Point Number Formats

- signed magnitude
- one’s complement
- two’s complement
- offset binary
- other more special formats such as signed digit
- we only discuss the two’s-complement number format

In the present document, we only discuss the two’s-complement number format—who’s binary addition is modulo addition, see pages 22 ff. below—, because it is the most-commonly used format; also, the Fixed-Point Toolbox of MATLAB presently only has fixed-point utilities for the two’s-complement number format.

Concerning the signed digit format (and other unconventional fixed-point number formats), you might want to consult [Kor93, Sections 2.3 and 2.4 starting on pages 21 and 24, respectively], [Mit06, Section 11.8.5 on page 639], or [MB07, Section 2.2.2 on pages 57 ff.].

---

3If you happen to hold the newer edition of Koren’s book in your hands, you find the referenced topics in [Kor02, Sections 2.3 and 2.4 starting on pages 23 and 27, respectively].
### Examples of Fixed-Point Number Formats

<table>
<thead>
<tr>
<th>decimal equivalent</th>
<th>sign-magnitude</th>
<th>one’s-complement</th>
<th>two’s-complement</th>
<th>offset binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/8</td>
<td>0.111</td>
<td>0.111</td>
<td>0.111</td>
<td>1.111</td>
</tr>
<tr>
<td>6/8</td>
<td>0.110</td>
<td>0.110</td>
<td>0.110</td>
<td>1.110</td>
</tr>
<tr>
<td>5/8</td>
<td>0.101</td>
<td>0.101</td>
<td>0.101</td>
<td>1.101</td>
</tr>
<tr>
<td>4/8</td>
<td>0.100</td>
<td>0.100</td>
<td>0.100</td>
<td>1.100</td>
</tr>
<tr>
<td>3/8</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>1.011</td>
</tr>
<tr>
<td>2/8</td>
<td>0.010</td>
<td>0.010</td>
<td>0.010</td>
<td>1.010</td>
</tr>
<tr>
<td>1/8</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>1.001</td>
</tr>
<tr>
<td>0/8</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>−0/8</td>
<td>1.000</td>
<td>1.111</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>−1/8</td>
<td>1.001</td>
<td>1.110</td>
<td>1.111</td>
<td>0.111</td>
</tr>
<tr>
<td>−2/8</td>
<td>1.010</td>
<td>1.101</td>
<td>1.110</td>
<td>0.110</td>
</tr>
<tr>
<td>−3/8</td>
<td>1.011</td>
<td>1.100</td>
<td>1.101</td>
<td>0.101</td>
</tr>
<tr>
<td>−4/8</td>
<td>1.100</td>
<td>1.011</td>
<td>1.100</td>
<td>0.100</td>
</tr>
<tr>
<td>−5/8</td>
<td>1.101</td>
<td>1.010</td>
<td>1.011</td>
<td>0.011</td>
</tr>
<tr>
<td>−6/8</td>
<td>1.110</td>
<td>1.001</td>
<td>1.010</td>
<td>0.010</td>
</tr>
<tr>
<td>−7/8</td>
<td>1.111</td>
<td>1.000</td>
<td>1.001</td>
<td>0.001</td>
</tr>
<tr>
<td>−8/8</td>
<td>N/A</td>
<td>N/A</td>
<td>1.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

These are [4, 3] bit numbers.

The entries N/A in the above table mean “not available.” The notation “[4, 3] bit numbers” means that the numbers are represented by four bits, three of them being fractional bits, and the forth bit, the left-most bit \( b_3 \), being the sign bit, as indicated by the following sketch:

\[
\begin{array}{cccc}
  b_3 & b_2 & b_1 & b_0 \\
\end{array}
\]

binary point
A sign bit $b_3$ that is zero indicates a positive number, and a sign bit $b_3$ that is one indicates a negative number. Below, on pages 27 and the following, we explain in detail the notation used by MATLAB and its Fixed-Point Toolbox.

We note that in the table on page 13 the formats “sign-magnitude,” “one’s complement,” and “two’s complement” all have the same positive number representation; they only differ in their representation of negative numbers:

**sign-magnitude:** If the sign bit is zero, $b_3 = 0$, the 3-bit fraction is a positive number with magnitude

$$b_2 \cdot 2^{-1} + b_1 \cdot 2^{-2} + b_0 \cdot 2^{-3} = b_2 \cdot \frac{1}{2} + b_1 \cdot \frac{1}{4} + b_0 \cdot \frac{1}{8}.$$

If the sign bit is one, $b_3 = 1$, the 3-bit fraction is a negative number with the same magnitude as for the corresponding positive number:

$$-(b_2 \cdot 2^{-1} + b_1 \cdot 2^{-2} + b_0 \cdot 2^{-3}) = -\left(b_2 \cdot \frac{1}{2} + b_1 \cdot \frac{1}{4} + b_0 \cdot \frac{1}{8}\right).$$

**one’s complement** A positive fraction is represented as in the sign-magnitude form, but a negative fraction is represented by complementing each bit of the binary representation of the corresponding positive fraction. As an example, take from the table on page 13 the number $3/8 \equiv 0.011$ in sign-magnitude as well as in one’s complement and in two’s complement; the negative in one’s complement then is $-3/8 = 1.100$.

**two’s complement** Again, a positive fraction is represented as in the sign-magnitude form; it’s negative is obtained by first complementing each bit of the binary representation, and subsequently adding a 1 to the LSB (least-significant bit). As an example, again take $3/8 \equiv 0.011$
in sign-magnitude as well as in one’s complement and in two’s complement. The complement becomes, first, 1.100 (which is the one’s complement), and, second by adding an LSB, 1.101, which indeed is −3/8 in the two’s complement column of the table on page 13.

**offset binary** The offset-binary representation is most often used in bipolar DA- and AD-conversion. Referring to the table on page 13, the offset-binary representation considers a $b = 3$ bit fraction with an additional sign bit as a $b+1 = 4$ bit number representing the $2^{b+1} = 2^4 = 16$ integer numbers from 0 to 15. Half of these numbers represent the negative fractions, and the other half represent the non-negative fractions (zero and positive fractions). Note that the two’s complement representation is converted to the offset-binary representation, and vice versa, by simply complementing the sign bit $b_3$. 


Quantization: Rounding Versus Truncation

- [3, 2] two’s-complement number example: *rounding*

The symbol $\mathbb{R}$ stands for the set of real numbers. The [3, 2] two’s-complement numbers are numbers from the set

$$\text{set } \hat{\mathbb{Z}} = \left\{-1, -\frac{3}{4}, -\frac{1}{2}, -\frac{1}{4}, 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4} \right\};$$

the notation “$x_q \in [3,2]$-numbers” in the above illustration means that $x_q$ is a member of that set.

The `dfilt` objects of the DSP *System Toolbox* of MATLAB can represent fixed-point filter structures; in this case, there is

*Footnote:* You just set the property `Arithmetic` of the filter object to ’fixed’. Of course, you must additionally have available the *Fixed-Point Toolbox*. 
a property \texttt{RoundMode} specifying how results of arithmetic operations are re-quantized. The above quantization with rounding is obtained by setting \texttt{RoundMode} either to 'round' or to 'convergent'. These two rounding modes just distinguish themselves in how they treat exact midpoint values; whereas 'round' rounds an exact midpoint to the closest representable number in the direction of positive infinity, 'convergent' rounds exact midpoint values up only if the least significant bit (after rounding) is set to 0. To give examples, consider the case where the bits just represent integers (or, likewise, interpret the considered bit-string as an integer). The tie-breaking rule of 'convergent' rounding then means rounding to even integers: 17.5 is rounded to 18, as 18.5 is; -13.5 becomes -14, as does -14.5. The advantage of this rule is that it treats positive and negative values in a symmetric kind, hence does not introduce a sign bias. Although it is deterministic, it has some “random-like” behavior. Other notions for “convergent rounding” are “round-to-nearest-even,” “unbiased rounding,” “statistician’s rounding,” “Gaussian rounding,” “bankers’ rounding,” and many more.
Quantization: Rounding Versus Truncation (2)

- $[3, 2]$ two’s-complement number example: \textit{truncation}

The above quantization by \textit{truncation} is obtained in fixed-point \texttt{dfilt} filter objects by setting \texttt{RoundMode} to ‘floor’; additionally available rounding modes in the Dsp System Toolbox are ‘ceil’ and ‘fix’; also compare pages 92 ff. of the present document.
Quantization: Overflow Versus Saturation

- $[3, 2]$ two’s-complement number example: saturation

Note: The above figure shows saturating overflow together with rounding; obviously, a corresponding characteristic exists for saturating overflow with truncation.

Fixed-point dfilt filter objects have the OverFlowMode property; set this property to 'saturate' to obtain an arithmetic behavior corresponding to the shown saturation characteristics.
Quantization: Overflow Versus Saturation (2)

saturation (also often called clipping)

- normally in AD converters
- sometimes in two’s-complement arithmetic
- size of error does not increase abruptly when overflow occurs
- but disadvantage: does not exploit an interesting and useful property of two’s-complement arithmetic (see below)

We discuss the mentioned “useful property of two’s-complement arithmetic” below on pages 22 ff.
Quantization: Overflow Versus Saturation (3)

- [3, 2] two’s-complement number example: natural overflow

Note: The above figure shows natural overflow together with rounding; obviously, a corresponding characteristic exists for natural overflow with truncation.

Fixed-point dfilt filter objects have the OverflowMode property; set this property to 'wrap' to obtain an arithmetic behavior corresponding to the shown natural overflow characteristics.
Quantization: Overflow Versus Saturation (4)

important property of two’s-complement arithmetic
\(\sim\) *only if used with natural overflow*

- given: \(x_i\) are two’s complement numbers in \([B, L]\)
- given: \(x = \sum_i x_i\) is a two’s complement number that does not overflow
- then: *using natural overflow*, the result of accumulation is correct *even if partial sums overflow*
- because: modulo arithmetic

We just supply two simple examples to illustrate the general truth.

**Example 1:** Consider three \([B, L] = [4, 3]\) numbers \(x_1, x_2, x_3\):

\[
x_1 = \frac{5}{8} \simeq 0.101, \\
x_2 = \frac{3}{4} \simeq 0.110, \\
x_3 = \frac{-1}{2} \simeq 1.100.
\]

Then the sum of these three numbers is \(x_1 + x_2 + x_3 = 7/8\) and clearly is a number that does not overflow in the \([4, 3]\) format. Depending on how we add, however, the partial sums might
overflow; for example, we have $x_1 + x_2 = 11/8 \notin [4, 3]$:

\[
\begin{align*}
  x_1 & \equiv 0101 \\
  x_2 & \equiv 0110
\end{align*}
\]

\[
x_1 + x_2 \equiv 1011 \equiv -5/8 \leftarrow \text{overflow}.
\]

Continuing to add to the overflowed result ($x_1 + x_2 = -5/8$) the term $x_3$, we end up with the correct overall result if we just discard the bit flowing over the sign bit:

\[
\begin{align*}
  x_1 + x_2 & \equiv 1011 \equiv -5/8 \leftarrow \text{overflowed} \\
  x_3 & \equiv 1100 \equiv -1/2
\end{align*}
\]

\[
x_1 + x_2 + x_3 \equiv 0111 \equiv 7/8 \leftarrow \text{correct}.
\]

**Example 2:** Consider all $B = 3$-bits combinations and interpret them in straightforward manner as integers:

\[
\begin{align*}
  x_0 & : 000 \equiv 0 \\
  x_1 & : 001 \equiv 1 \\
  x_2 & : 010 \equiv 2 \\
  x_3 & : 011 \equiv 3 \\
  x_4 & : 100 \equiv 4 \\
  x_5 & : 101 \equiv 5 \\
  x_6 & : 110 \equiv 6 \\
  x_7 & : 111 \equiv 7
\end{align*}
\]

We thus have a set of $2^B = 2^3 = 8$ elements being the integers from zero to seven:

\[
\text{set} \equiv \{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\} = \{0, 1, 2, 3, 4, 5, 6, 7\}.
\]

Now consider counting through the set by starting with zero and always adding one; do the addition of one by straightforwardly
adding 3-bit numbers and discarding an overflow bit if there results such an overflow bit:

\[
\begin{align*}
  x_0 + x_1 &= 0 + 1 = 1 : & 000 \\
             &            & 001 \\
             &            & 001 \\
  x_1 + x_1 &= 1 + 1 = 2 : & 001 \\
             &            & 001 \\
             &            & 010 \\
  x_2 + x_1 &= 2 + 1 = 3 : & 010 \\
             &            & 011 \\
  \vdots \quad 000 &= 0 = 8 \text{ mod } 8 .
\end{align*}
\]

We thus see that the straightforward binary addition with disregarding overflow is—for the considered \( B = 3 \)-bits case—just modulo \( 2^B = 2^3 = 8 \) addition. We may represent the considered set of 8 elements together with modulo 8 addition on a circle:
As far we have interpreted our eight 3-bits strings just as positive integers. But we see that \( x_0 = 000 \) is the identity element for the addition: \( x_0 \) added to any of the other elements just gives back this other element. Therefore, we may ask for inverses: \( x_i + ? = 0 \) mod 8. We then find

\[
\begin{align*}
1 + ? & = 0 : \quad 1 + 7 = 8 \equiv 0 \mod 8 \quad \Rightarrow \quad 7 \triangleq \text{inverse}(1) \triangleq -1, \\
2 + ? & = 0 : \quad 2 + 6 = 8 \equiv 0 \mod 8 \quad \Rightarrow \quad 6 \triangleq \text{inverse}(2) \triangleq -2, \\
3 + ? & = 0 : \quad 3 + 5 = 8 \equiv 0 \mod 8 \quad \Rightarrow \quad 5 \triangleq \text{inverse}(3) \triangleq -3, \\
4 + ? & = 0 : \quad 4 + 4 = 8 \equiv 0 \mod 8 \quad \Rightarrow \quad 4 \triangleq \text{inverse}(4) \triangleq -4.
\end{align*}
\]

The following circle-representation graphically illustrates these findings:

Similarly, we may not only interpret our 3-bits strings as integers, but likewise as fractional numbers; for the format \([B, L] = [3, 2]\), for example, we obtain the following circle:
Note that we have not shown in the above graphic the binary point after the leftmost bit—the sign bit in the considered $[B, L] = [3, 2]$ format—, because this binary point is just an interpretation aid, but does not really exist in the arithmetic.

Finally, we return to our point of departure. If we compute, for example, in the two’s-complement system $[B, L] = [3, 0]$ with the elements

$$\text{set} = \{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\} = \{-4, -3, -2, -1, 0, 1, 2, 3\}$$

we correctly obtain any sum $x = \sum_i x_i$ with $x$ in the given set, independently of whether partial sums are outside the set—have overflow—or not, if we use the natural overflow rule. This is because using natural overflow we indeed do the corresponding modulo arithmetic.

We finally note that the discussed property of two’s-complement arithmetic with natural overflow greatly simplifies the scaling of fixed-point (digital) filters; compare Subsection 4.2, especially see the definition of “critical nodes” on page 67.
Notation in MATLAB

- two quantities specify how numbers are quantized:
  - word-length $B$ in bits
  - fraction-length $L$ in bits

- general representation for a signed two’s-complement fixed-point number

- note: in MATLAB, fraction-length can be larger than word-length, or it can be even a negative integer

- see our examples below
Notation in MATLAB (cont’d)

- decimal value represented by bits in $[B, L]$ format

$$\text{value} = -b_{B-1} \cdot 2^{B-L-1} + b_{B-2} \cdot 2^{B-L-2} + \cdots$$

$$\cdots + b_1 \cdot 2^{-L+1} + b_0 \cdot 2^{-L}$$

- note: $L$ is the negative of the weight of the LSB: $2^{-L}$
  $\Rightarrow$ step size $\equiv$ precision, does not depend on word length

- $B - L - 1$ defines the interval of representable numbers

$$\text{numbers} \in [-2^{B-L-1}, 2^{B-L-1})$$

We denote left-closed, right-open sets by $[x_{\min}, x_{\max})$; thus $x \in [x_{\min}, x_{\max})$ means $x_{\min} \leq x < x_{\max}$. 
**Notation in MATLAB (cont’d)**

- example: \([B, L] = [3, 2] \implies B - L - 1 = 0\)

\[
\begin{align*}
011 &= -0 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} = 3/4 \\
010 &= -0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} = 1/2 \\
001 &= -0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 1/4 \\
000 &= -0 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} = 0 \\
111 &= -1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} = -1/4 \\
110 &= -1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} = -1/2 \\
101 &= -1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = -3/4 \\
100 &= -1 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} = -1
\end{align*}
\]

- represented numbers \(\in [-1, 1)\)

\[
\text{number set} = \left\{-1, -\frac{3}{4}, -\frac{1}{2}, -\frac{1}{4}, 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}\right\}
\]

Note that the interval of representable numbers is given by \(B - L - 1: [-2^{B-L-1}, 2^{B-L-1}] = [-2^0, 2^0] = [-1, 1)\).

The precision (resolution) is \(2^{-L} = 2^{-2} = 1/4\).
Notation in MATLAB (cont’d)

- example: \([B, L] = [3, 1]\) \(\rightarrow B - L - 1 = 1\)

\[
\begin{align*}
011 & = -0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} = 3/2 \\
010 & = -0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} = 2/2 \\
001 & = -0 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} = 1/2 \\
000 & = -0 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} = 0 \\
111 & = -1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} = -1/2 \\
110 & = -1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} = -2/2 \\
101 & = -1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} = -3/2 \\
100 & = -1 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} = -4/2
\end{align*}
\]

- represented numbers \(\in [-2, 2)\)

number set = \(\left\{ -\frac{4}{2}, -\frac{3}{2}, -\frac{2}{2}, -\frac{1}{2}, 0, \frac{1}{2}, \frac{2}{2}, \frac{3}{2} \right\} \)

Note that the interval of representable numbers is given by \(B - L - 1: [-2^{B-L-1}, 2^{B-L-1}] = [-2^{1}, 2^{1}] = [-2, 2)\).

The precision (resolution) is \(2^{-L} = 2^{-1} = 1/2\).
Notation in MATLAB (cont’d)

• example: \([B, L] = [3, 3]\) \(\rightsquigarrow B - L - 1 = -1\)

\[
\begin{align*}
011 &= -0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = 3/8 \\
010 &= -0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} = 2/8 \\
001 &= -0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 1/8 \\
000 &= -0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} = 0 \\
111 &= -1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = -1/8 \\
110 &= -1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} = -2/8 \\
101 &= -1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = -3/8 \\
100 &= -1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} = -4/8
\end{align*}
\]

• represented numbers \(\in \left[-\frac{1}{2}, \frac{1}{2}\right]\)

number set = \(\left\{-\frac{4}{8}, -\frac{3}{8}, -\frac{2}{8}, -\frac{1}{8}, 0, \frac{1}{8}, \frac{2}{8}, \frac{3}{8}\right\}\)

Note that the interval of representable numbers is given by \(B - L - 1: [-2^{B-L-1}, 2^{B-L-1}] = [-2^{-1}, 2^{-1}] = [-1/2, 1/2]\).
The precision (resolution) is \(2^{-L} = 2^{-3} = 1/8\).
Notation in MATLAB (cont’d)

- example: $[B, L] = [3, 4] \implies B - L - 1 = -2$

\[
\begin{align*}
011 &= -0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = \frac{3}{16} \\
010 &= -0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} = \frac{2}{16} \\
001 &= -0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} = \frac{1}{16} \\
000 &= -0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} = 0 \\
111 &= -1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = -\frac{1}{16} \\
110 &= -1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} = -\frac{2}{16} \\
101 &= -1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} = -\frac{3}{16} \\
100 &= -1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} = -\frac{4}{16}
\end{align*}
\]

- represented numbers $\in \left[-\frac{1}{4}, \frac{1}{4}\right)$

number set = \\{ $-\frac{4}{16}, -\frac{3}{16}, -\frac{2}{16}, -\frac{1}{16}, 0, \frac{1}{16}, \frac{2}{16}, \frac{3}{16}$ \}\}

Note that the interval of representable numbers is given by $B - L - 1$: $[-2^{B-L-1}, 2^{B-L-1}) = [-2^{-2}, 2^{-2}) = [-1/4, 1/4)$. The precision (resolution) is $2^{-L} = 2^{-4} = 1/16$. 

4_Fixed-Point Filters 32 2018
Notation in MATLAB (cont’d)

- example: $[B, L] = [3, -2] \implies B - L - 1 = 4$

\[
\begin{align*}
011 & = -0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 = 12 \\
010 & = -0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 = 8 \\
001 & = -0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 = 4 \\
000 & = -0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 = 0 \\
111 & = -1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 = -4 \\
110 & = -1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 = -8 \\
101 & = -1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 = -12 \\
100 & = -1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 = -16
\end{align*}
\]

- represented numbers $\in [-16, 16)$

\[
\text{number set} = \{-16, -12, -8, -4, 0, 4, 8, 12\}
\]

Note that the interval of representable numbers is given by $B - L - 1$: $[-2^{B-L-1}, 2^{B-L-1}] = [-2^4, 2^{-4}] = [-16, 16]$. The precision (resolution) is $2^{-L} = 2^{-(-2)} = 4$. 
3 Coefficient Quantization

Coefficient Quantization: Realizable Poles

- first insight into problem of coefficient quantization
- gained by studying quantization effect on pole (and zero) locations in z-plane
- more general: formal definitions of coefficient sensitivity measures
- here: available real-pole locations for first-order filters
- here: available complex-pole locations for second-order, direct form filters (note: depends on structure!)

We note that the structure of the filter implementation is important for the present discussion; therefore, the qualification “direct form” above is important. Also note that we later discuss an alternative structure, likewise only for second-order filters, the normal form structure on pages 47 ff.; we restrict its discussion to second order, because second-order building blocks are most often used to implement higher-order filters in parallel form, see page 44, or in cascade form, see page 45.
Realizable 1st-Order Real Poles

- direct-form filter structure
- transfer function $H(z) = \frac{1}{1 + az^{-1}} = \frac{z}{z + a}$
- coefficient $a$ quantized as a $[4, 3]$ two’s-complement number
- only “stable” poles (absolute values $< 1$) are shown
Realizable 2nd-Order Complex Poles

- direct-form filter structure
- transfer function \( H(z) = \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2}} \)
- \( a_1, a_2 \) quantized as \([5, 3]\) two’s-complement numbers
- only “stable” complex poles (absolute values \( \leq 1 \)) are shown

If a second-order system has real poles, it can be decomposed into two first-order systems (with real coefficients); therefore, for real poles the discussion of first-order systems applies, and we have omitted the real poles in the above diagram.

We note that in the vicinity of \( z = 1 \) and in the vicinity of \( z = -1 \) the realizable poles are not very dense. This observation has some important consequences on the design of low- and high-pass filters, see our discussion starting on page 39.
Realizable 2nd-Order Complex Poles (cont’d)

- why?

- transfer function

\[
H(z) = \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2}} = \frac{1}{(1 - pz^{-1})(1 - p^* z^{-1})}
\]

- coefficients \(\leftrightarrow\) poles:

\[
(1 - pz^{-1})(1 - p^* z^{-1}) = 1 - (p + p^*) z^{-1} + pp^* z^{-2}
\]

therefore:

\[
a_1 = -(p + p^*) = -2\Re\{p\}
\]

\[
a_2 = pp^* = |p|^2
\]

See the next page for the conclusions from the above relations between coefficients and poles.
Realizable 2nd-Order Complex Poles (cont’d)

- we have from $a_1 = -2\Re\{p\}$: quantization of $a_1$ quantizes the real part of the poles
- we have from $a_2 = |p|^2$: quantization of $a_2$ quantizes the radius of the poles $\sim$ non-uniform spacing

example:

$a_1$ and $a_2$ each quantized to $[5,3]$ two’s complement numbers
Realizable 2nd-Order Complex Poles (cont’d)

simple analysis shows:

- because 2nd-order filters are building blocks of popular
  - cascade forms
  - parallel forms

- we have important conclusions:

  1. narrow-band lowpass (and highpass) filters are most sensitive to coefficient quantization
     $\leadsto$ require high coefficient accuracy
     $\leadsto$ alternatively: look for other structures

  2. oversampling leads to increased coefficient sensitivity because poles are pushed close to $z = 1$ (recall that $z = 1$ corresponds to DC frequency)

To the remark on narrow-band lowpass and highpass filters, we have the following additional remarks: For lowpass filters poles fall near to $z = 1$ in the complex $z$-plane, and for highpass filters the poles are near to $z = -1$. At these locations $z = \pm 1$ the density of poles realizable with a 2nd-order direct-form IIR block is lowest, compare the figure on page 36. There exist alternative 2nd-order structures that reduce sensitivity to coefficient quantization near $z = \pm 1$, see below our discussion starting on page 47.

To the remark on oversampling, consider the situation where we have an oversampling that leads to a sampling frequency that is substantially higher than the minimally needed Nyquist
rate: Such a high oversampling rate pushes the spectrum of the useful signal to the vicinity of $\hat{\omega} = 0$ (vicinity of $Dc$), which corresponds in the $z$-plane to the vicinity of $z = 1$. Obviously, the filters applied to such an oversampled signal then have to perform their work also in the vicinity of $\hat{\omega} = 0$, and, in turn, have poles pushed to the vicinity of $z = 1$. Because poles in the vicinity of $z = 1$ lead to increased sensitivity to coefficient quantization, too much oversampling should be avoided. This advice to reduce oversampling seems to be counter-intuitive at a first glance, because reducing the sampling interval—increasing the sampling frequency—enables the better approximation of the underlying continuous-time signal, and, in turn, to better approximate a corresponding continuous-time filter. However, although we can indeed reduce aliasing by oversampling, we pay with an increased sensitivity of the filter coefficients; additionally, the required speed of the hardware is, of course, also increased—lesser time to process more data. Generally, coefficient sensitivity and hardware speed are more important aspects than aliasing.

The approach to avoid the outlined dilemma is as follows: First, the analog-to-digital conversion is performed with some oversampling—the larger the oversampling rate, the simpler the anti-alias filtering becomes; note that the anti-aliasing filter must precede the analog-to-digital conversion, and note that it is an *analog* filter. Next, a discrete-time low-pass filtering paves the way for a subsequent decimation—a down-sampling in discrete time; such a low-pass filtering will not influence spectral components of the useful signal, because we have done an analog-to-digital conversion with large oversampling. Finally, the planned discrete-time filtering is performed on a signal with low sampling rate, thus relaxing the demands on the the $z$-plane pole locations.
Coefficient Quantization: Higher Order Filters

- higher-order filters in direct form: $\leadsto$ analysis becomes more complicated
- we have seen: coefficient sensitivity increases by going from 1st- to 2nd-order
- guess: even more increased sensitivity for higher-order filters
- well-known result from numerical analysis:
  - sensitivity of roots of a polynomial to accuracy of coefficients increases with order of polynomial
  - stability of direct-form filter might be lost due to coefficient quantization
  - even if filter remains stable: specifications might become violated
  - for fixed-point implementations: generally avoid direct-form IIR structures
  - prefer cascade or parallel forms
Coefficient Quantization: Zeros in FIR Filters

- consider FIR filter in direct form (recall: magnitude response produced entirely by zeros)
- seem to have same sensitivity problems as IIR filters in direct form
- however: important differences to IIR situation:
  - most FIR designs are for linear phase
  - coefficients satisfy symmetry conditions: \( b_k = \pm b_{M-k} \)
  - quantizing both, \( b_k \) and \( b_{M-k} \) to same perturbed value
    - filter is still a linear-phase filter
    - only magnitude response is perturbed
  - zeros on unit circle remain on unit circle
  - symmetry constraint: greatly reduces sensitivity of most FIR filters

- of course: parallel form does not exist
- cascade form: not widely used

We have stated above that zeros designed onto the unit circle remain on the unit circle if the symmetric coefficients are quantized the same. Of course the zeros resulting from quantized coefficients will move along the unit circle as compared to the ideal (designed) locations. If the quantization is so strong that two moving zeros come together, then they will split into reciprocal pairs and, of course, no longer stay on the unit circle.
Concerning the parallel form of FIR filters, it is of course not true that it does not exist, but it just does not make much sense. Consider, as an example, the FIR transfer function $H(z) = b_0 + b_1z^{-1} + b_2z^{-2}$. We might consider a parallel realization with parallel sub-transfer functions $b_0$, and $b_1z^{-1}$, and $b_2z^{-2}$, but this realization is trivial and even not efficient, because we need more memory cells than minimum. In contrast to the FIR filter case, we have in the IIR filter case true parallel-form realizations via the partial fraction decomposition of the complete rational transfer function; see below.

Concerning cascade forms and zeros that do not lie on the unit circle, we just state that a minimum order of 4 for the cascade blocks is needed to satisfy symmetry conditions needed for linear-phase filters, see [Jac96, p. 377].
Parallel Form IIR Filters

- expansion of $H(z)$ into partial fractions
- stopband attenuation: depends mostly on having zeros on unit circle
- in parallel form: zeros are not realized by individual 1st- and 2nd-order terms
  - but zeros are realized by the way how all parallel-section transfer functions add together
- all coefficients (in numerator and in denominator) of all parallel-form sections affect every zero
  - highly sensitive situation
    - zeros are in no way constraint to lie on the unit circle after quantization
- therefore: parallel form is usually avoided for filters with demanding specifications
Cascade Form IIR Filters

- cascade form is more robust under coefficient quantization than parallel form
- poles have same low sensitivity as in parallel form, but zeros are also roots of only 1st- and 2nd-order polynomials
- additionally: in common cases with zeros on the unit circle, the numerator coefficients $b_{2k}$ are all $\pm 1$:
  \[ B_k(z) = 1 + b_{1k}z^{-1} + b_{2k}z^{-2} = 1 + b_{1k}z^{-1} \pm 1 \cdot z^{-2} \]
  \( \sim b_{2k} \) not changed by quantization
- although zeros on unit circle move by quantization of $b_{1k}$, they do not move off the unit circle
  \( \sim \) stopband attenuation specifications are more easily satisfied than in parallel form

We have assumed that the sections in the cascade form are enumerated by $k$, $k = 1, 2, \ldots, \text{last}$.

Note that by stating that the zeros do not move off the unit circle, we of course assume that the zeros on the unit circle do not become real by quantization.

To understand that a complex-conjugate zero pair on the unit circle has the coefficient $b_{2k}$ in the corresponding factor polynomial $B_k(z)$ equal to 1, we assume that we have such a
zero pair at angles $\pm \hat{\omega}_0$ on the unit circle; then $B_k(z)$ becomes

$$B_k(z) = \left(1 - e^{j\hat{\omega}_0}z^{-1}\right) \left(1 - e^{-j\hat{\omega}_0}z^{-1}\right)$$

$$= 1 - \left( e^{j\hat{\omega}_0} + e^{-j\hat{\omega}_0} \right) z^{-1} + e^{j\hat{\omega}_0} e^{-j\hat{\omega}_0} z^{-2},$$

indeed showing that $b_{2k} = 1$. We note in passing that we also see that $b_{1k}$ becomes $-2 \cos(\hat{\omega}_0)$. 
Normal-Form IIR Filters

- how to reduce sensitivity of poles in the vicinity of $z = \pm 1$?
- is a problem only for very narrow-band lowpass/highpass filters
- problem with direct-form IIR biquads: realizable pole locations are intersections of (evenly spaced) vertical lines with *non-evenly* spaced concentric circles
  $\sim$ low pole-density near $z = \pm 1$
- consider the coupled form:
  - from 2nd-order state-space description with input $x[n]$ and output $y[n]$
  \[
  s[n+1] = As[n] + bx[n] \quad s[n] \doteq \text{state vector}
  \]
  \[
  y[n] = c^T s[n] + \delta x[n]
  \]
  - with state-feedback matrix
  \[
  A = \begin{pmatrix}
  \alpha_1 & \alpha_2 \\
  -\alpha_2 & \alpha_1
  \end{pmatrix}
  \]

Don’t panic, see the next pages! For realizable pole locations obtainable with direct-form IIR biquads, you may want to see again the example on pages 36 and 38.

We use bold-face lower-case letters to denote vectors and upper-case bold-face letters for matrices. The system vectors
and/or matrices—\( A, b, \) and \( c^T \)—have elements with Greek letters corresponding to the Latin letter of the vector or the matrix; the vector \( b \), as an example, has, therefore, elements \( \beta_i \). Also note that all vectors are column vectors, and that row vectors come as transposed column vectors, like \( c^T \).

### Normal-Form IIR Filters (cont’d)

![Diagram of IIR filter](image)

- the dynamic equation is

\[
\begin{pmatrix}
  s_1[n+1] \\
  s_2[n+1]
\end{pmatrix} =
\begin{pmatrix}
  \alpha_1 & \alpha_2 \\
  -\alpha_2 & \alpha_1
\end{pmatrix}
\begin{pmatrix}
  s_1[n] \\
  s_2[n]
\end{pmatrix} +
\begin{pmatrix}
  x_1[n] \\
  x_2[n]
\end{pmatrix}
\]

\[= A\]

Note that for a scalar input \( x[n] \) and a scalar output \( y[n] \) we have the inputs in the above diagram to build as

\[
\begin{pmatrix}
  x_1[n] \\
  x_2[n]
\end{pmatrix} =
\begin{pmatrix}
  \beta_1 \\
  \beta_2
\end{pmatrix} x[n],
\]
and the output is computed as

\[ y[n] = \begin{pmatrix} \gamma_1 & \gamma_2 \end{pmatrix} \begin{pmatrix} s_1[n] \\ s_2[n] \end{pmatrix} + \delta x[n]. \]

We now recall that the goal of the presently discussed structure is to reduce the sensitivity of poles in the vicinity of \( z = \pm 1 \); see the next slide! Note, however, that this pole sensitivity is merely a problem for very narrow-band low-pass filters, at \( z = 1 \), or high-pass filters, at \( z = -1 \).

Another question is how to obtain the discussed "normal-form" state-space filter if a certain second-order transfer function \( H(z) \) to be realized is given. First, we observe that the state-space filter from page 47 realizes the transfer function

\[ H(z) = \frac{Y(z)}{X(z)} = c^T(zI - A)^{-1}b + \gamma; \]

here the matrix \( I \) denotes the 2-by-2 identity matrix. Second, we refer to [RM87] that discusses in its Section 9.12 the details of how to obtain the elements \( \{A, b, c^T, \gamma\} \) of the state variable description of the normal-form filter in order to realize a predefined \( H(z) \).

Higher order filters might be built as cascades of the discussed second-order filters. If, thereby, each section is optimal, one calls the complete filter sectional optimal; however, the complete filter is not, in general, optimal, see the discussion and the procedures given by [RM87].
Normal-Form IIR Filters (cont’d)

• for the poles and their quantization we have

\[ \text{poles } \triangleq \text{eigenvalues of } \mathbf{A} \]

\[ = \det (z \mathbf{I} - \mathbf{A}) = 0 \quad : \quad (z - \alpha_1)^2 + \alpha_2^2 = 0 \]

\[ \leadsto z_{p1,p2} = \alpha_1 \pm j\alpha_2 \quad \leadsto \text{both, } \alpha_1 \text{ and } \alpha_2, \text{ are quantized} \]

example:

\[ \alpha_1 \text{ and } \alpha_2 \text{ each quantized to [4,3] two’s complement numbers} \]

The transfer function formula in the previous page contains the matrix inverse of \((z \mathbf{I} - \mathbf{A})\). If we evaluate it using Cramer’s rule, we obtain

\[ (z \mathbf{I} - \mathbf{A})^{-1} = \frac{\text{Adj} (z \mathbf{I} - \mathbf{A})}{\det (z \mathbf{I} - \mathbf{A})} = \frac{\text{Adj} (z \mathbf{I} - \mathbf{A})}{D(z)}, \]

and we find that the denominator polynomial \(D(z)\), which gives the poles of the filter, is likewise the polynomial that determines the eigenvalues of the state-feedback matrix \(\mathbf{A}\), hence the poles of the system equal the eigenvalues of the state-feedback matrix.

4_Fixed-Point Filters 50 2018
Concerning poles and zeros under coefficient quantization, the following general statements hold:

- If we have poles and/or zeros that are highly clustered, small errors in the coefficients—in the present context introduced by quantization—may cause large shifts in the poles and/or the zeros locations. Thereby, the larger the number of polynomial roots (poles of denominator polynomials, zeros of numerator polynomials) is, the greater the sensitivity becomes.

- In cascade-form structures and in parallel-form structures each pair of complex-conjugate poles is realized separately; additionally in cascade-form structures, each pair of complex-conjugate zeros is likewise realized separately. Therefore, the error in a given pole (and zero in a cascade-form structure) is independent of the distance from the other poles (and zeros). Hence, in general, cascade (or possibly parallel) forms are to be preferred over direct forms from a point of view of coefficient quantization. This statement is particularly true for very selective filters that have highly clustered poles. Because cascade-form structures have the advantage not only for poles, but for zeros too, we generally prefer cascade-form structures over parallel-form structures.

---

5 We recall that to realize discrete-time filters there are alternative techniques to digital filters: switched-capacitor or switched-current circuits and the like; in these implementations we have errors in coefficients due to not exactly realizable elements like capacitors, specifically ratios of capacitances.
4 Arithmetic Errors

Overview

• quantization errors
  – in AD conversion
  – in multipliers due to truncation or rounding
  – $\sim$ quantization noise

• overflow errors: in certain internal nodes such as
  – inputs to multipliers
  – outputs of accumulators
  – $\sim$ may lead to large amplitude oscillations
  – $\sim$ probability of overflow minimized by scaling of
    signal levels

• probability of overflow $\leftrightarrow$ signal-to-noise ratio (SNR)
  – more stringent signal scaling
    + probability of overflow $\downarrow$
    – SNR $\downarrow$
  – less stringent signal scaling
    – probability of overflow $\uparrow$
    + SNR $\uparrow$
  – find a compromise
4.1 Signal Quantization

Quantization in AD-Converters

- processing blocks and signals

\[
x(t) \xrightarrow{\text{Sampler}} x[n] \xrightarrow{\text{Quantizer}} x_q[n] \xrightarrow{\text{Coder}} x_{eq}[n]
\]

assumptions:

- \( x(t) \) and \( x[n] \) ∈ \([X_{\min}, X_{\max}]\)
  \( \leadsto \) full-scale range \( X_{\max} - X_{\min} \equiv R_{FS} \)

- quantizer with rounding

- coder: \([B, L] = [B, B - 1]\) two’s-complement numbers
  \( \leadsto x_{eq}[n] \in [-1, 1), \quad \text{resolution} \ \delta = \frac{1}{2^{B-1}} \)

- quantizer: resolution \( \Delta = \frac{R_{FS}}{2^B} \)

Note that the difference between the resolutions \( \delta \) and \( \Delta \) is just that \( \delta \) gives the “abstract” resolution relative to the interval \([-1, 1)\), whereas \( \Delta \) gives the “natural” resolution with respect to the full-scale range; if, for example, \( x(t) \) is a voltage signal, \( R_{FS} \) comes in volts and, in turn, \( \Delta \) comes in volts too.
Quantization in AD-Converters (cont’d)

- example-converter characteristics: $[3, 2]$ two’s complement

\[ \Delta = \frac{R_{FS}}{2^B} = \frac{R_{FS}}{2^3} \]

\[
\begin{align*}
-\frac{9\Delta}{2} & \quad -\frac{7\Delta}{2} & \quad -\frac{5\Delta}{2} & \quad -\frac{3\Delta}{2} & \quad -\frac{\Delta}{2} & \quad 0 & \quad \frac{\Delta}{2} & \quad \frac{3\Delta}{2} & \quad \frac{5\Delta}{2} & \quad \frac{7\Delta}{2} \\
\end{align*}
\]

\[
\begin{align*}
-4\Delta & \quad -3\Delta & \quad -2\Delta & \quad -\Delta & \quad 0 & \quad \Delta & \quad 2\Delta & \quad 3\Delta & \quad 4\Delta \\
\end{align*}
\]

\[
\begin{align*}
000 & \quad 001 & \quad 010 & \quad 011 & \quad 100 & \quad 101 & \quad 110 & \quad 111 \\
\end{align*}
\]

full-scale range $R_{FS}$
Quantization in AD-Converters (cont’d)

- quantization error: signal flow graph

\[ x[n] \xrightarrow{\text{quantization error}} x_q[n] = x[n] + e[n] \]

- quantization error: characteristics for 3-bits example

Note that the signals \( x_q[n], x[n], \) and \( e[n] \) are deterministically interconnected—if two of them are given, then the third follows! This truth is in contrast to the probabilistic models for the quantization noise \( e[n] \) that we introduce below. Because the engineering results that these ill-founded models deliver are useful, it is common practice to nevertheless work with the models.
Quantization in AD-Converters (cont’d)

• common practice: statistical model for quantization error

\[ x[n] + e[n] = x_q[n] \]

• common assumptions:
  - error sequence \( e[n] \) is not deterministically known but described by a statistical model
  - \( e[n] \) is a sample sequence of a stationary random process
  - \( e[n] \) is uncorrelated with the sequence of exact signal samples \( x[n] \)
  - \( e[n] \) is a white-noise process
  - probability distribution of \( e[n] \) is uniform over range of quantization error

We note that the given statistical error model is just motivated by expediency, because it allows a rather simple analysis of quantization effects. Heuristically, the assumptions appear to be valid if the signal \( x[n] \), the signal to be quantized, is sufficiently complex—has sufficient spectral- and amplitude content—, and if the quantization steps are sufficiently small, such that the amplitude of the signal \( x[n] \) traverses—with large likelihood—many quantization levels in going from one sample to the next.
Quantization in AD-Converters (cont’d)

- quantization error model:
  \[ \sim \text{some commonly used probability distributions} \]

\[
\begin{align*}
\text{rounding} & \quad m_e = 0 \\
& \quad \sigma_e^2 = \Delta^2/12 \\
\text{two’s complement truncation} & \quad m_e = -\Delta/2 \\
& \quad \sigma_e^2 = \Delta^2/12 \\
\text{one’s complement truncation} & \quad m_e = 0 \\
& \quad \sigma_e^2 = \Delta^2/3
\end{align*}
\]

- mean values of error samples \( m_e \)
- variances of error samples \( \sigma_e^2 \)

In the above plots, the acronym “pd” stands for probability distribution.

Note that rounding yields the given symmetric probability distribution function for two’s complement as well as for one’s complement.
Arithmetic Quantization: Roundoff Noise

- basic arithmetic operations involved in (linear) digital filter implementations are
  \(\sim\) multiplication and addition

- in fixed-point implementations:
  \(\sim\) multiplication results must be rounded or truncated
  \(\sim\) addition results need not to be rounded or truncated
  (but addition results might overflow \(\sim\) scaling for dynamic range)

- truncation and rounding:
  \(\sim\) product quantizations
  \(\sim\) nonlinear processes

- common practice:
  \(\sim\) use statistical quantization-noise models as in AD conversion

We here consider only linear filters; therefore, the involved operations are only multiplication and addition. Note, however, that many adaptive filters—which are not linear filters—also just need these two basic operations; other adaptive filters use additional computations. For the discussion of finite-precision effects in adaptive filters we refer to [Hay96].

The mentioned statistical quantization-noise model of AD converters is the one discussed on pages 56 and 57.
Roundoff Noise Analysis

• note:
  ~ coefficient multipliers always feed into summation nodes (possibly through delays)

• therefore:
  ~ think of roundoff-noise sources that are associated with multipliers as additional (unwanted) inputs to the summation nodes

• equivalent noise source \( e_j[n] \) at \( j \)-th summation node inside the filter

• define:
  ~ \( G_j(z) \) = transfer function from source \( e_j[n] \) at \( j \)-th summation node inside the filter to the output of the filter

Concerning the equivalent noise sources \( e_j[n] \), consider any of the filter structures in [Goe18], either for FIR filters in [Goe18, Section 4.2] or for IIR filters in [Goe18, Section 5.2]. You observe then that in these structures (internal) signal branches with amplifiers (the multipliers to realize the filter coefficients \( b_k \) for FIR filters or \( b_k \) and \( a_l \) for IIR filters) feed into adders. Since each amplifier (multiplier) has associated with it an additive noise source to describe the round-off noise, the totality of these additive noise sources can be taken (summed) together to one equivalent noise source feeding into the mentioned adder.

The transfer functions \( G_j(z) \) may be termed roundoff-noise transfer functions; concerning the quantization noise of the AD
Roundoff Noise Analysis (cont’d)

- equivalent roundoff-noise system

\[
\begin{align*}
e_1[n] & \quad G_1(z) \\
e_2[n] \\
e_j[n] & \quad G_j(z) \\
e_N[n] & \quad G_N(z) \\
\end{align*}
\]

\[+\] output
Roundoff Noise Analysis (cont’d)

• output-noise spectrum due to roundoff noise

\[ S_{\text{out}}(\hat{\omega}) = \sum_j |G_j(e^{j\hat{\omega}})|^2 \sigma_j^2 \]

where \( \sigma_j^2 \) = variance of equivalent roundoff white-noise source \( e_j[n] \)

• output-noise power due to roundoff noise

\[ \sigma_{\text{out}}^2 = \frac{1}{2\pi} \int_{-\pi}^{\pi} S_{\text{out}}(\hat{\omega}) d\hat{\omega} \]

\( \hat{\sigma} \) = output-noise variance

The DSP System Toolbox of MATLAB supplies the command \texttt{noisepsd()} to compute the noise power spectral density at a filter’s output caused by roundoff noise inside the filter. The average power of the output noise (the integral of the power spectral density) can be computed with the command \texttt{avgpower()}. 
Roundoff Noise Analysis (cont’d)

- variance $\sigma_j^2$ of equivalent noise source $e_j[n]$: 
- if we have
  
  $$\sigma_0^2 \triangleq \text{variance of a single multiplier rounding operation}$$

  - rounding before summation

  $$k_j \triangleq \text{number of multipliers inputting to } j\text{-th summation node}$$

- then the variance of the equivalent noise source $e_j[n]$ is
  
  $$\sigma_j^2 = k_j \sigma_0^2$$

- if rounding is performed after summation: $k_j = 1$
Signal-to-Noise Ratios (SNR)

- noise sources
  - Ad-conversion noise at the filter input
  - roundoff noise inside the filter
- signal-to-noise ratios (SNR)

\[
\text{SNR} \doteq 10 \log_{10} \left( \frac{\sigma_{\text{sig}}^2}{\sigma_{\text{noise}}^2} \right)
\]

where \( \sigma_{\text{sig}}^2 \) = signal variance representing average power of signal

\( \sigma_{\text{noise}}^2 \) = noise variance representing average power of noise

- general goals:
  - as small as possible noise
  - as large as possible SNR
Signal-to-Noise Ratios (cont’d)

• at filter input

\[ \text{SNR} \triangleq 10 \log_{10} \left( \frac{\sigma_x^2}{\sigma_{\text{AD}}^2} \right) \]

where \( x[n] \) = input signal with variance \( \sigma_x^2 \)

\( \sigma_{\text{AD}}^2 \) = variance of Ad-conversion noise

• at filter output

\[ \text{SNR} \triangleq 10 \log_{10} \left( \frac{\sigma_y^2}{\sigma_{\text{noise}}^2} \right) \]

where \( y[n] \) = output signal of filter due to input signal \( x[n] \), variance \( \sigma_y^2 \)

\( \sigma_{\text{noise}}^2 \triangleq \sigma_{\text{out}}^2 + \sigma_{\text{AD, out}}^2 \)

\( \sigma_{\text{out}}^2 \) = noise variance due to roundoff noise

\( \sigma_{\text{AD, out}}^2 \) = variance of Ad-conversion noise transferred to output

The variance \( \sigma_{\text{out}}^2 \) is the variance of the complete roundoff noise as it appears at the filter output; for its computation compare page 61. Correspondingly, \( \sigma_{\text{AD, out}}^2 \) describes the variance of the Ad-conversion quantization noise as it appears at the output of the filter. Because the Ad-conversion noise at the filter input is modeled as a white noise with variance \( \sigma_{\text{AD}}^2 \), the filter
with its transfer function $H(z)$ transforms this white input noise to a colored output noise with spectral density

$$S_{\text{AD, out}}(\hat{\omega}) = \left| H(e^{j\hat{\omega}}) \right|^2 \sigma_{\text{AD}}^2.$$ 

The corresponding variance of the Ad-conversion noise transferred to the filter output—its average power—thus becomes

$$\sigma_{\text{AD, out}}^2 = \frac{1}{2\pi} \int_{-\pi}^{\pi} S_{\text{AD, out}}(\hat{\omega}) d\hat{\omega}$$

$$= \frac{\sigma_{\text{AD}}^2}{2\pi} \int_{-\pi}^{\pi} \left| H(e^{j\hat{\omega}}) \right|^2 d\hat{\omega}.$$ 

Furthermore note that the theory assumes the Ad-conversion noise source and the various roundoff-noise sources to be uncorrelated; therefore, the total noise variance at the filter output, $\sigma_{\text{noise}}^2$, is just the sum of the noise variances of the total roundoff noise and the Ad-conversion noise, both transferred to the output of the filter.
4.2 Scaling for Dynamic Range

Why We Need Scaling

- to increase SNR
  ~ increase signal levels at all nodes inside the filter
- because:
  roundoff-noise level is fixed for a given structure and a given resolution
- but: increasing signal levels too much leads to
  - exceeding of dynamic range of fixed-point arithmetic
  - ~ overflow result in certain computations
  - overflow is a severe nonlinearity
  - ~ preclude overflow
    OR
    ~ make its probability acceptably small and minimize the resulting distortions

To minimize the effect of—low probable—overflow, use saturation at plus/minus full scale instead of wrapping, also see page 96.
Critical Nodes

- it *seems* that we must ensure that signals do not overflow at *any node* inside the filter

- fortunately, *not true* (in two’s complement) because:
  - some nodes separated only by delays
  - summation of more than two numbers:
    - if total sum is small enough not to overflow
    - then the correct sum results regardless of order in which numbers are added
    - independent of overflows that may occur in the partial sums or even if some of the inputs to adder have themselves overflowed as a result of multiplication with a coefficient larger than one

Concerning nodes that are separated only by one or more delays—which are memory cells in a realization—it is clear that if a signal does not overflow at the input, it will not overflow at the output.

The statement that the result of a summation, that lies in the dynamic range of the arithmetic, may have overflows in partial sums without doing harm, is true in any modulo arithmetic such as two’s-complement arithmetic; see, for example, [Jac96] and the references given therein.
**FIR Filters in Direct Form**

- simplest to start with; structure

\[
x[n] \rightarrow b_0 \rightarrow z^{-1} \rightarrow x[n-1] \rightarrow b_1 \rightarrow + \rightarrow z^{-1} \rightarrow x[n-k] \rightarrow b_k \rightarrow + \rightarrow z^{-1} \rightarrow x[n-M] \rightarrow b_M \rightarrow + \rightarrow y[n]
\]

- (finite) impulse response \( h[k] \): \( \{b_0, b_1, \ldots, b_M\} \)
FIR Filters in Direct Form (cont’d)

• assumption: available dynamic range (full scale) is $R$

  input $|x[n]| \leq R$, all $n$

• output

  $$y[n] = \sum_{k} h[k]x[n - k] \doteq \text{convolution sum}$$

• then

  $$|y[n]| = \left| \sum_{k} h[k]x[n - k] \right|$$

  $$\leq \sum_{k} |h[k]| |x[n - k]|$$

  $$\leq R \sum_{k} |h[k]|$$

Concerning the available dynamic range—the full scale—the parameter $R$ is often taken as 1, that is, one often assumes that the input signal values are in the interval $[-1, 1)$.

Note that in the presently considered case of an FIR filter, the convolution summation is over finitely many terms. In the more general case of a filter containing feedback terms, the convolution sum will include infinitely many terms. Also note that the present formulation neglects the additional roundoff-error inputs $e_j[n]$; they are small in any case.
FIR Filters in Direct Form (cont’d)

• then:
  output $y[n]$ is also in available dynamic range $|y[n]| \leq R$

• if
  $$l_1 \triangleq \sum_k |h[k]| \leq 1$$

• $l_1$ is called the $l_1$-norm of the FIR filter

• if $l_1 \not\leq 1$ then scale:

  • scale factor $s \triangleq \frac{1}{l_1}$

  • scale the coefficients $h[k], k = 1, 2, \ldots, M$

  $$\tilde{h}[k] = s \cdot h[k] = \frac{1}{l_1} \cdot h[k]$$

• then: new FIR filter $\tilde{h}[k]$ has no overflows

Note that scaling the coefficients of the original filter $h[k]$ with the scale factor $s$ is equivalent to amplifying the input signal $x[n]$ with a gain $s$. For $s < 1$ this amplification is, of course, an attenuation.
FIR Filters in Direct Form (cont’d)

• by proposed scaling:
  - input signal $x[n]$ is multiplied by $s = 1/l_1$
• for $s < 1$: attenuation
• for $s < 1$: SNR ↓
• $\sim$ of interest: scale factor $s$ as large as possible
• to scale, there are other—less stringent—norms often used
  - $l_2 = L_2$ norm
  - $L_\infty$ norm
• choice of scaling norm: a compromise between
  low probability of overflow $\leftrightarrow$ large SNR

Note that if we use, on one hand, the $l_1$-norm for scaling, as is exemplified on pages 69–70, then we are sure to have no overflow. If we use, on the other hand, some other norms for scaling, then there is a certain—low—probability for overflow.

Concerning other norms used in filter scaling, we give on the pages 72 and 73 the available norms of the DSP System Toolbox of MATLAB.
Norms Used in the DSP System Toolbox

• consider a filter with impulse response $h[k]$:
  
  discrete-time domain $h[k]$ \quad $\mathcal{H}(\hat{\omega})$ frequency domain

• discrete-time domain norms

  $l_1$-norm: $l_1 = \sum_k |h[k]|$

  $l_2$-norm: $l_2 = \sqrt{\sum_k (h[k])^2}$

  $l_\infty$-norm: $l_\infty = \max_k |h[k]|$

We note that the norms defined here apply not only to FIR filters, but, likewise, to IIR filters. Whereas for FIR filters the index $k$ in the above formulae runs over finitely many integers, it runs over infinitely many integers in the more general case of IIR filters.

If we need to preclude overflow with certainty, we must use $l_1$-norm scaling which yields, however, the lowest SNR.

The $l_2$-norm scaling is the same as the $L_2$-norm scaling introduced below, see page 73.
Norms Used in the DSP System Toolbox (cont’d)

- frequency domain norms

\[
L_1\text{-norm} : \quad L_1 = \frac{1}{2\pi} \int_{-\pi}^{\pi} |H(\hat{\omega})| d\hat{\omega}
\]

\[
L_2\text{-norm} : \quad L_2 = \sqrt{\frac{1}{2\pi} \int_{-\pi}^{\pi} |H(\hat{\omega})|^2 d\hat{\omega}}
\]

\[
L_\infty\text{-norm} : \quad L_\infty = \max_{\hat{\omega}} |H(\hat{\omega})|
\]

- relation between these norms

\[
l_1 \geq L_\infty \geq L_2 = l_2 \geq L_1 \geq l_\infty
\]

- identity \( L_2 = l_2 \) is due to Parseval’s theorem

If we use \( L_\infty \)-norm scaling, then we preclude overflow for a pure sinusoidal signal in steady state. The \( L_\infty \)-norm is, however, the most conservative of the frequency-domain constraints, because it assumes that, in the worst case, the input power is concentrated at a single frequency. Overflows are very unlikely with \( L_\infty \)-norm scaling.

For many broadband input signals, \( L_\infty \)-norm scaling is too conservative. In these cases it might be better to use an \( L_2 \)-norm scaling—which is the same as an \( l_2 \)-norm scaling—to obtain a larger SNR and still have a low probability of overflow.
General Case: Critical Nodes

- only certain nodes inside the filter require dynamic range constraints: branch nodes having one or several outputs that feed into multipliers

- critical nodes \( v_i[n] \)

\[
x[n] \quad F_i(z) \quad v_i[n] \quad \alpha \quad + \quad y[n]
\]

- scaling transfer function and corresponding impulse response:

\[
F_i(z) \doteq \frac{V_i(z)}{X(z)} \quad \bullet \quad f_i[n] \quad , \quad v_i[n] \doteq i\text{-th critical node}
\]

- scaling:

\[
\tilde{F}_i(z) \doteq \text{scaled version of } F_i(z) \\
= s_i \cdot F_i(z) \\
\tilde{v}_i[n] \doteq \text{scaled version of } v_i[n]
\]

We recall from page 67 that a summation result lying in the dynamic range of the two’s complement arithmetic is correctly computed even if there are partial sums with overflow. Therefore, we have only to consider as critical nodes those nodes that have one or several outputs that feed into multipliers (if a node only inputs to a summation, it is a partial sum).
General Case: Absolute Bound Scaling

- assumption: available dynamic range (full scale) is $R$
  
  \[ |x[n]| \leq R , \]

- then
  
  \[ v_i[n] = \sum_k f_i[k] x[n - k] \triangleq \text{convolution sum} \]

- then
  
  \[ |v_i[n]| = \left| \sum_k f_i[k] x[n - k] \right| \leq \sum_k |f_i[k]| |x[n - k]| \leq R \sum_k |f_i[k]| \]

Concerning the available dynamic range—the full scale—again note that the parameter $R$ is often taken as 1, that is, one often assumes that the input signal values are in the interval $[-1, 1]$.

Note that in the presently considered case of a general IIR filter, the convolution summation is over infinitely many terms. Furthermore, the given convolution formulation assumes zero initial conditions and neglects small roundoff-error inputs $e_j[n]$. 
General Case: Absolute Bound Scaling (cont’d)

- **if**: scaled version $\tilde{v}_i[n]$ of $v_i[n]$ must satisfy constraint
  \[ |\tilde{v}_i[n]| \leq R , \quad \text{for all critical nodes} \]

- **then**: we must scale the filter such that
  \[ \tilde{l}_{1,i} = \sum_k |\tilde{f}_i[k]| \leq 1 , \quad \text{for all critical nodes} \]

- this is $l_1$ scaling

- simplest scaling: multiply (attenuate/amplify) input signal by
  \[ s = \frac{1}{\max_i l_{1,i}} \]

  where
  \[ l_{1,i} = \sum_k |f_i[k]| , \quad f_i[k] \bullet \bullet F_i(z) \]

  \[ \triangleq l_1\text{-norm of } i\text{-th scaling transfer function } F_i(z) \]

To numerically compute the $l_1$-norm of an FIR filter is simple, because the computation involves only finitely many samples of its impulse response. In the general case of an IIR filter, however, the impulse response has infinitely many terms and we might numerically approximate it as follows: Iteratively compute, for $L = 1, 2, \ldots$, partial sums $l_{1,L} = \sum_{k=0}^{L} |f_i[k]|$, and stop the computation if the difference $l_{1,L} - l_{1,(L-1)}$ becomes smaller than a pre-specified limit; a typical value of the limit might be $10^{-6}$. 
General Case: Alternative Scaling Rules

- above scaling rule
  - based on worst case
  - does not fully utilize dynamic range of all adder outputs
  - ∼ reduces SNR significantly

- more practical scaling rules
  - use another norm
    \[ l_1 \geq L_\infty \geq L_2 = l_2 \geq L_1 \geq l_\infty \]
  - for a selected norm to work with, say \( \ell \), let
    \[ \ell_i \hat{=} \ell \text{-norm of } F_i(z) \bullet f_i[k] , \text{ unscaled} \]
    \[ \tilde{\ell}_i \hat{=} \ell \text{-norm of } \tilde{F}_i(z) \bullet \tilde{f}_i[k] , \text{ scaled} \]
  - then, after scaling, obtain
    \[ \tilde{\ell}_i \leq 1 , \text{ for all critical nodes} \]
  - with
    \[ \tilde{F}_i(z) = s_i F_i(z) \implies s_i \leq \frac{1}{\ell_i} \]

The above scaling constants \( s_i \) denote the cumulative scalings that are applied to nodes \( v_i[n] \).

For the norms given in the above slide we refer to pages 72 and 73.
General Case: Alternative Scaling Rules (cont’d)

more practical scaling rules (cont’d)

- exploit special structures

- most important structures: Sos cascades
  \[ \sim \text{scale individual blocks} \]

- *Signal Processing Toolbox* of MATLAB: command `scale()`
  \[ \sim \text{automatic scaling of SOS cascades with one of the norms} \]
  \[ l_1 \geq L_{\infty} \geq L_2 = l_2 \geq L_1 \geq l_{\infty} \]

- also see the following example

Again we refer to pages 72 and 73 for the definitions of the norms enumerated in the slide above.

We say above that you should “see” the example on pages 79–86, and you might find the exemplified scaling to involve quite tedious computations. But note that the computations are fully deterministic (algorithmic), and that in a practical application the DSP System Toolbox command `scale()` will free you from doing the scaling by hand.
Example: Direct-Form II Sections in Cascade

- given:

\[
H(z) = G \cdot \frac{1 + b_{11}z^{-1} + b_{21}z^{-2}}{1 - a_{11}z^{-1} - a_{21}z^{-2}} \cdot \frac{1 + b_{12}z^{-1} + b_{22}z^{-2}}{1 - a_{12}z^{-1} - a_{22}z^{-2}}
\]

Note that using the gain \( G \) allows to have the leading FIR coefficients \( b_{0j}, j = 1, 2 \), to be unity.

Also note that the nodes with the signals \( v_1[] \) and \( v_2[] \) are the critical nodes, because they feed—here only via delays \( z^{-1} \) and \( z^{-1} \cdot z^{-1} = z^{-2} \), but these delays do not matter—into branches with gains.
Example (cont’d): Scaling Transfer Functions

- recall
  \[
  H(z) = G \cdot \frac{1 + b_{11}z^{-1} + b_{21}z^{-2}}{1 - a_{11}z^{-1} - a_{21}z^{-2}} \cdot \frac{1 + b_{12}z^{-1} + b_{22}z^{-2}}{1 - a_{12}z^{-1} - a_{22}z^{-2}} = H_1(z) \]
  \[
  \simeq H_2(z)
  \]

- critical nodes: \( v_1[n] \) and \( v_2[n] \)

- scaling transfer functions (from input to critical nodes)
  \[
  F_1(z) = G \cdot \frac{1 + b_{11}z^{-1} + b_{21}z^{-2}}{1 - a_{11}z^{-1} - a_{21}z^{-2}}
  \]
  \[
  F_2(z) = H_1(z) \cdot G \cdot \frac{1 + b_{12}z^{-1} + b_{22}z^{-2}}{1 - a_{12}z^{-1} - a_{22}z^{-2}}
  \]

where
  \[
  H_1(z) = \frac{1 + b_{11}z^{-1} + b_{21}z^{-2}}{1 - a_{11}z^{-1} - a_{21}z^{-2}}
  \]
Example (cont’d): Norms

- select a norm to work with, say $\sim \ell$

- example: if we would use the $L_\infty$ norm, then $\ell \cong L_\infty$

- then we have the following $\ell$-norms of the relevant transfer functions

$$\ell\text{-norm of } F_1(z) : \ell_{F_1}$$
$$\ell\text{-norm of } F_2(z) : \ell_{F_2}$$
$$\ell\text{-norm of } H(z) : \ell_{H}$$
Example (cont’d): Scaled Version

- note: the gains $g_1$ and $g_2$ can be absorbed into existing feed-forward multipliers of preceding section

Note that if we absorb the gain $g_j$ into the preceding section, the FIR coefficient $\tilde{b}_{0j}$ will no longer be unity; also see the next page 83.
Example (cont’d): Scaled Version (cont’d)

• in detail, we obtain for the scaled structure

Note that the above scaled structure has new values for the feed-forward multipliers \( \tilde{b}_{ij}, i = 0, 1, 2, j = 1 \) for the first and \( j = 2 \) for the second section; also note that new multipliers \( \tilde{b}_{0j} \) have been introduced.

If the zeros of the transfer function are on the unit circle, as is usually the case in practical filters, these (complex-conjugate) zeros \( z = e^{j\phi} \) and \( z^* = e^{-j\phi} \) yield numerators of the form \( (1 - z z^{-1}) (1 - z^*_z z^{-1}) = 1 - (z + z^*_z) z^{-1} + z^{-2} \), that is, values for \( b_{2j} \) that are unity. In such cases the choice \( \tilde{b}_{0j} = \tilde{b}_{2j} = 2^{-\beta} \), which is an approximation, might be useful to reduce the total number of true multipliers (recall that multiplying by \( 2^{-\beta} \) is just a shift); of course, we pay for this simplification with a slight decrease in the SNR.
Example (cont’d): Scaling

- after scaling we have

\[ \tilde{F}_1(z) = \frac{\tilde{G}}{1 - a_{11}z^{-1} - a_{21}z^{-2}} \]

\[ \tilde{F}_2(z) = \tilde{H}_1(z) \cdot \frac{\tilde{G}}{1 - a_{12}z^{-1} - a_{22}z^{-2}} \]

where \[ \tilde{H}_1(z) = \frac{\tilde{b}_{01} + \tilde{b}_{11}z^{-1} + \tilde{b}_{21}z^{-2}}{1 - a_{11}z^{-1} - a_{21}z^{-2}} \]

- for scaling, we use the following scaling constants

\[ s_0 : \quad \tilde{G} = s_0 \cdot G \, , \]
\[ s_1 : \quad \tilde{b}_{01} = s_1 \cdot 1 \, , \quad \tilde{b}_{11} = s_1 \cdot b_{11} \, , \quad \tilde{b}_{21} = s_1 \cdot b_{21} \]
\[ s_2 : \quad \tilde{b}_{02} = s_2 \cdot 1 \, , \quad \tilde{b}_{12} = s_2 \cdot b_{12} \, , \quad \tilde{b}_{22} = s_2 \cdot b_{22} \]

Note that the scaling constants \( s_0 \), \( s_1 \), and \( s_2 \) are explicitly determined only later, see page 86.
Example (cont’d): Scaling (cont’d)

- relations between scaled and unscaled transfer functions

\[ \tilde{F}_1(z) = \frac{\tilde{G}}{1 - a_{11}z^{-1} - a_{21}z^{-2}} = \frac{s_0 \cdot G}{1 - a_{11}z^{-1} - a_{21}z^{-2}} = s_0 \cdot F_1(z) \]

\[ \tilde{F}_2(z) = \tilde{H}_1(z) \cdot \frac{\tilde{G}}{1 - a_{12}z^{-1} - a_{22}z^{-2}} = (s_1 \cdot H_1(z)) \left( \frac{s_0 \cdot G}{1 - a_{12}z^{-1} - a_{22}z^{-2}} \right) = (s_0 \cdot s_1) F_2(z) \]

\[ \tilde{H}(z) = \tilde{G} \tilde{H}_1(z) \tilde{H}_2(z) = (s_0 G)(s_1 H_1(z))(s_2 H_2(z)) = (s_0 s_1 s_2) G H_1(z) H_2(z) = (s_0 s_1 s_2) H(z) \]
Example (cont’d): Scaling (cont’d)

- notation: for some transfer function $G(z)$, we use
  \[ \ell\)-norm of $G(z) = \ell(G(z)) = \ell_G \]

- after scaling, we require
  \[ \ell\left(\tilde{F}_1(z)\right) = \ell(s_0 \cdot F_1(z)) = s_0 \ell(F_1(z)) = s_0 \ell_{F_1} \quad \text{must} = 1 \]
  \[ \ell\left(\tilde{F}_2(z)\right) = \ell(s_0 s_1 \cdot F_2(z)) = s_0 s_1 \ell(F_2(z)) = s_0 s_1 \ell_{F_2} \quad \text{must} = 1 \]
  \[ \ell\left(\tilde{H}(z)\right) = \ell(s_0 s_1 s_2 \cdot H(z)) = s_0 s_1 s_2 \ell(H(z)) = s_0 s_1 s_2 \ell_{H} \quad \text{must} = 1 \]

- solving these equations, we obtain the scaling constants
  \[ s_0 = \frac{1}{\ell_{F_1}}, \quad s_1 = \frac{1}{s_0 \ell_{F_2}} = \frac{\ell_{F_1}}{\ell_{F_2}}, \quad s_2 = \frac{1}{s_0 s_1 \ell_{H}} = \frac{\ell_{F_2}}{\ell_{H}} \]
4.3 Pairing/Ordering in Sos Cascades

An important question to answer when designing digital IIR filters in form of second-order-section (SOS) cascades is, which poles should be paired together with which zeros, and, how these pole-zero pairs, which each make up a second-order section, have finally to be ordered inside of the complete SOS cascade.

**Problem Formulation**

- IIR filter in SOS-cascade structure
- question 1: how to order individual sections?
- question 2: how to pair poles and zeros?
- goals
  - minimize probability of overflow
  - maximize SNR
- only rule of thumb solutions
  ~ depends on norm used for scaling

There are many ways to represent a filter in second-order section form. Through careful pairing of the poles and zeros, careful ordering of the sections in the cascade, and multiplicative scaling of the sections, it is possible to reduce quantization noise gain and avoid overflow in some fixed-point filter implementations. The functions `zp2sos()` and `ss2sos()` of the *Signal...*
Processing Toolbox of MATLAB perform pole-zero pairing, section scaling, and section ordering; we refer to the Linear System Transformations available in the Signal Processing Toolbox.\textsuperscript{6} The function \texttt{zp2sos()} transforms a filter specified by its zeros and poles to a cascade of second-order sections; correspondingly, the function \texttt{ss2sos()} transforms a filter specified by its state-space description to a cascade of second-order sections. The function \texttt{tf2sos()}, which transforms a filter specified by its transfer function to a cascade of second-order sections, does also ordering, but it does it by calling \texttt{zp2sos()}. 

\textsuperscript{6}
With $L_\infty$-Norm Scaling

- select ordering of sections
- select pairing of poles and zeros
- according to following rules:

1. start with last section in (remaining) cascade
   \[\sim\] it must obtain
   - complex-conjugate pole-pair closest to the unit circle
   - combine that pole-pair with the zero-pair that is closest to it

2. for remaining sections
   - continue with rule (1)
   - until all sections are defined

If we use the command\(^7\) `scale()` of the DSP System Toolbox of MATLAB, we may, as an option, indicate the desired ordering; if we do not indicate the ordering, `scale()` does an automatic ordering which adapts to the given situation. The command `reorder()` rearranges the order of the sections in a second-order section filter. See also our hints on page 87.

\(^7\)Note that `scale()` works only for the special but practically most important class of second-order-section (Sos) filters.
With $L_2$-Norm Scaling

- *ordering* rule *opposite* to rule for $L_\infty$-norm scaling
- *pairing* rule *same* as rule for $L_\infty$-norm scaling
- select ordering of sections
- select pairing of poles and zeros
- according to following rules:

1. start with *first* section in (remaining) cascade
   \sim it must obtain
   - complex-conjugate pole-pair closest to the unit circle
   - combine that pole-pair with the zero-pair that is closest to it

2. for remaining sections
   - continue with rule (1)
   - until all sections are defined

See our hints on pages 87 and 89.
4.4 Limit Cycles

What are Limit Cycles

- as far: roundoff-error analysis based on assumption of uncorrelated errors $\sim$ statistical noise model
- is OK for most signals
  - signals with sufficient spectral content
  - signals with sufficient amplitude content
- however
  - problems for low-level inputs
  - problems for periodic input signals that are synchronized with sampling rate
- then: roundoff errors cannot be assumed to be white noise
- then: linear roundoff-noise model cannot explain behavior
- extreme and important case: zero or constant input signal
  - ideally: output of stable $Dt$ filter would asymptotically approach zero or a constant
  - but: consider example below

We abbreviate discrete time by $Dt$.

The mentioned example starts on page 92 and ends on page 94.
What are Limit Cycles: Example

- given: first-order recursive filter

\[ y[n] = -0.5y[n-1] + x[n] \]
\[ x[n] = 0, \text{ zero input signal} \]
\[ y[0] = 7/8, \text{ initial state} \]

- given:
  - state \( y[n] \) is represented as a \([B, L] = [4, 3]\) two’s-complement number
  - coefficient \(-0.5\) is represented as a \([B, L] = [2, 1]\) two’s-complement number
  - arithmetic with rounding (see comments in text)

It is intuitively clear, that the nonlinear behavior introduced by quantization strongly depends on how the quantization is done. For the example-results shown on the page 94 we have used “rounding to the nearest” with the exact midpoint rounded to the closest representable number in the direction of larger magnitude. This corresponds to the MATLAB integer rounding command \texttt{round()}, which produces, for example, \texttt{round(-3.5)} = -4; so we have computed, for example, \( y[1] = \texttt{round((7/8) \times (-1/2))} \) = \texttt{round(-7/16)} = \texttt{round(-3.5/8)} = -4/8. We note, however, that the \textit{Fixed Point Toolbox} of MATLAB, on which the

\[8\]The \textit{Fixed-Point Toolbox} of MATLAB has a demo on limit cycles, which is called \texttt{filimitcycledemo.m}. You find it in the \texttt{Help} window via a search to ‘‘limit cycles’’; see in Demo Search Results.
Dsp System Toolbox bases for fixed-point filter implementations, does it slightly different: Setting the rounding mode to ‘round’ will round to the closest representable number, with the exact midpoint rounded to the closest representable number in the direction of positive infinity;\(^9\) if we set the rounding mode to ‘convergent’, it also rounds to the nearest allowable quantized value, but exact midpoint values are rounded up only if the least significant bit (after rounding) would be set to 0.

Other available rounding modes in the Dsp System Toolbox are ‘ceil’, ‘fix’, and ‘floor’; ‘ceil’ rounds to the closest representable number in the direction of positive infinity; ‘fix’ rounds to the closest representable number in the direction of zero; and ‘floor’, which is equivalent to truncation, rounds to the closest representable number in the direction of negative infinity. For details consult the manuals.

\(^9\)In our example we would then obtain \(y[1] = \text{round}((7/8) \times (-1/2)) = \text{round}(-7/16) = \text{round}(-3.5/8) = -3/8\).
What are Limit Cycles: Example (cont’d)

- then: we obtain

<table>
<thead>
<tr>
<th>$n$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>ideal $y[n]$</td>
<td>$\frac{7}{8}$</td>
<td>$-\frac{7}{16}$</td>
<td>$\frac{7}{32}$</td>
<td>$-\frac{7}{64}$</td>
<td>$\frac{7}{128}$</td>
<td>$-\frac{7}{256}$</td>
<td>$\frac{7}{512}$</td>
</tr>
<tr>
<td>real $y[n]$</td>
<td>$\frac{7}{8}$</td>
<td>$-\frac{4}{8}$</td>
<td>$\frac{2}{8}$</td>
<td>$-\frac{1}{8}$</td>
<td>$\frac{1}{8}$</td>
<td>$-\frac{1}{8}$</td>
<td>$\frac{1}{8}$</td>
</tr>
</tbody>
</table>

- real $y[n]$
  - does not converge to zero
  - oscillates $\Rightarrow$ periodic output $\Rightarrow$ limit cycles

- for the [4, 3] two’s-complement represented state with rounding computations we have

$$\text{quant}\left(\frac{2^{-3}/2}{1-|0.5|}\right) = \text{quant}\left(\frac{1/16}{1/2}\right) = \frac{1}{8}$$

$\Rightarrow$ dead band = $\left(-\frac{1}{8}, \frac{1}{8}\right)$

Concerning dead-band computations, it can be shown that for a general first-order system

$$y[n] = a \cdot y[n-1] + x[n],$$
the dead-band \((-y_d, y_d)\) is given for \([B, L]\) two’s-complement number computations with rounding as

\[ y_d = \text{quant} \left( \frac{2^{-L/2}}{1 - |a|} \right). \]

Note that for \(a > 0\) the resulting limit cycles are constants (a DC signal), whereas for \(a < 0\) the signs alternate, giving thus a oscillatory signal with (maximum) radian frequency \(\hat{\omega} = \pi\).

For second-order systems the analysis becomes more complicated but is still feasible. For higher-order systems with parallel realizations of second-order blocks, the outputs of the individual second-order blocks are independent when the input is zero, thus the second-order system analysis can directly be applied. In the case of cascade realizations, however, only the first section has a zero input, and succeeding sections may exhibit other characteristic limit cycles or may just filter the limit cycles generated in the first section. In any case, the limit cycle behavior of higher-order systems becomes much more complex, and, correspondingly, the analysis becomes much more complicated. To learn more about limit cycles and their analysis, you might want to consult one of \([OS89], [Jac96], [RM87]\), or \([Mit98]\), and the literature cited therein.

Roberts and Mullis \([RM87]\) discuss in Section 9.17 constant-input limit cycles and indicate measures to reduce limit-cycle effects. Such countermeasures include the addition of a small amount of noise (“dither”) in the computation, or to use magnitude truncation instead of rounding. Mitra discusses limit cycles in \([Mit98, Section 9.2]\). He indicates, with references to additional literature, that random rounding is a technique to suppress zero-input limit cycles. Furthermore, he introduces limit-cycle free structures and indicates that most general approaches base on state-space structures. A recent paper \([YAK08]\) discussing limit-cycle free structures indicates that limit cycles and their suppression are still open research problems.
Overflow Oscillations

- above, there are \textit{rounding} limit cycles:
  - can always be reduced to an acceptable level
  - by increasing number of bits used to represent signal
- another form of autonomous oscillations:
  - produced by overflow characteristics of two’s-complement arithmetic
  - oscillations consume entire dynamic range of filter
  - \textit{must be avoided}
  - usual scaling: does minimize probability of overflow, but does not preclude it entirely
  - solution: modify accumulator to implement a saturation arithmetic
    \textit{↗} but \textit{not} saturating in partial sum computations, \textit{only} saturating the resulting total

The above statements are likewise true for one’s-complement computations.

We finally note that the following notions are also often used:

- \textit{small-scale limit cycles}, which are due to quantization;
- \textit{large-scale limit cycles}, which are due to overflow.
Notation and Symbols

$a_i$ Feedback coefficients (in an IIR filter); depending on the context, the feedback coefficients might also appear with double indices as $a_{ij}$.

$A(z)$ Denotes the denominator polynomial of a discrete-time transfer function.

$A$ State-feedback matrix of a system in state-space description; see page 47.

$b_i$ Feed-forward coefficients (FIR filter or FIR-part of an IIR filter); depending on the context, the feed-forward coefficients might also appear with double indices as $b_{ij}$. But note that $b_i$ might also denote bit number $i$.

$b$ Input weight vector of a system in state-space description; see page 47.

$B$ Usually denotes the number of bits in a fixed-point number.

$B(z)$ Denotes the numerator polynomial of a discrete-time transfer function.

$[B, L]$ MATLAB notation for a fixed-point $B$-bits two’s complement number having $L$ fraction bits; see page 27.

$c$ Output vector of a system in state-space description; see page 47.

$e[:]$ Usually denotes a discrete-time error signal.

$e_j$ Equivalent noise source at $j$-th summation node; see the model for the round-off noise analysis, page 59.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$</td>
<td>Natural frequency of continuous-time signals in Hz.</td>
</tr>
<tr>
<td>$\hat{f}$</td>
<td>Natural frequency of discrete-time signals. Note that $\hat{f}$ is a dimensionless quantity: $\hat{f} = f/f_s$ where $f_s$ is the sampling frequency.</td>
</tr>
<tr>
<td>$f_s$</td>
<td>Sampling frequency; $f_s = 1/T_s$; see page 7.</td>
</tr>
<tr>
<td>$f_i[k]$</td>
<td>Impulse response corresponding to scaling transfer function $F_i(z)$: $f_i[k] \rightarrow F_i(z)$; see page 74.</td>
</tr>
<tr>
<td>$F_i(z)$</td>
<td>Scaling transfer function number $i$; see page 74.</td>
</tr>
<tr>
<td>$g_i$</td>
<td>Scaling gains in an Sos cascade; see page 82.</td>
</tr>
<tr>
<td>$G$</td>
<td>As a scalar it denotes a gain; see page 79.</td>
</tr>
<tr>
<td>$G_j(z)$</td>
<td>Transfer function from $j$-th summation node inside of the filter to its output; see the model for the round-off noise analysis, page 59.</td>
</tr>
<tr>
<td>$h[k]$</td>
<td>Impulse response of a discrete-time linear system. The impulse response and the transfer function of the system are a $z$-transformation pair: $h[k] \rightarrow H(z)$.</td>
</tr>
<tr>
<td>$\tilde{h}[k]$</td>
<td>Impulse response of scaled filter that had before scaling the impulse response $h[k]$; see page 70.</td>
</tr>
<tr>
<td>$H(z)$</td>
<td>Transfer function of a discrete-time linear system. The transfer function and the impulse response of the system are a $z$-transformation pair: $h[k] \rightarrow H(z)$.</td>
</tr>
<tr>
<td>$I$</td>
<td>Identity matrix of appropriate dimension; see page 50.</td>
</tr>
<tr>
<td>$j$</td>
<td>Usually is the complex unit, $j = \sqrt{-1}$; we use $j$ sometimes also as an index variable, but the context makes clear what is the correct interpretation.</td>
</tr>
</tbody>
</table>
$l_1$ Denotes the $l_1$ norm of a filter, which is defined in the time domain; for an FIR filter see page 70; for the more general case see page 72.

$l_2$ Denotes the $l_2$ norm of a filter, which is defined in the time domain; see page 72.

$l_\infty$ Denotes the $l_\infty$ norm of a filter, which is defined in the time domain; see page 72.

$L$ Usually denotes the number of fractional bits in a fixed-point number.

$L_1$ Denotes the $L_1$ norm of a filter, which is defined in the frequency domain; see page 73.

$L_2$ Denotes the $L_2$ norm of a filter, which is defined in the frequency domain; see page 73.

$L_\infty$ Denotes the $L_\infty$ norm of a filter, which is defined in the frequency domain; see page 73.

$m_e$ Mean of the random variable $e$; see page 57.

$n$ Variable to denote discrete-time.

$p$ Often used to denote a pole of a system.

$pd(\cdot)$ Probability distribution; see page 57.

$quant(\cdot)$ Quantized value of number.

$R_{FS}$ Full-scale range; see page 53. We note that sometimes we abbreviate $R_{FS}$ to just $R$.

$s$ Scaling factor; see page 70. Often, scaling factors appear with subscripts, as in $s_i$.

$s[\cdot]$ State vector of a discrete-time system in state-space description; see page 47.
\( S_{out}(\hat{\omega}) \) Noise spectral density at the output of a filter; see page 61.

\( t \) Variable to denote continuous time.

\( T \) As superscript indicates vector- and matrix transpose.

\( T_s \) Sample time interval; \( T_s = 1/f_s \); see page 7.

\( v_i[\cdot] \) Signal at critical node number \( i \); see page 74. Critical nodes are a concept in the realm of scaling for dynamic range.

\( \tilde{v}_i[\cdot] \) Signal at critical node number \( i \) after scaling; see page 74.

\( x[\cdot] \) Discrete-time signal; see page 7.

\( x_q[\cdot] \) Discrete-time signal which is quantized in amplitude; see page 53.

\( x_{eq}[\cdot] \) Discrete-time signal which is quantized in amplitude, and which is described by a certain code word in the interval \([-1, 1)\); see page 53.

\( x(\cdot) \) Continuous-time signal; see page 7.

\([x_{\text{min}}, x_{\text{max}}]\) Left-closed, right-open set: \( x \in [x_{\text{min}}, x_{\text{max}}] \) thus means \( x_{\text{min}} \leq x < x_{\text{max}} \); compare page 28.

\( y[\cdot] \) Discrete-time signal; usually an output signal of a system.

\( z \) Complex variable of the \( z \) transformation.

\( \alpha_i \) The scalar elements of state-feedback matrix \( A \) of a state variable description of a filter; see page 47.
\[ \beta_i \] The scalar elements of the input vector \( b \) of a state variable description of a filter; see page 47 and the following.

\[ \gamma_i \] The scalar elements of the output vector \( c^T \) of a state variable description of a filter; see page 47 and the following.

\[ \delta \] “Abstract” resolution; see page 53. Also used for a scalar feed-through term in a state variable description of a filter; see page 47 and the following.

\[ \Delta \] “Natural” resolution; see page 53.

\[ \sigma^2_e \] Variance of the random variable \( e \).

\[ \sigma^2_{out} \] Noise power at the output of a filter; see page 61.

\[ \sigma^2_{sig} \] Average power of a signal; see page 63.

\[ \sigma^2_{noise} \] Average noise power; see page 63.

\[ \sigma^2_{AD} \] Average of AD-conversion noise; see page 64.

\[ \sigma^2_{AD, out} \] Average of AD-conversion noise transferred to the output of a considered system; see page 64.

\[ \sigma^2_x \] Average power of signal \( x[\cdot] \); see page 64.

\( \omega \) Radian frequency of continuous-time signals, \( \omega = 2\pi f \) where \( f \) denotes the natural frequency in Hz.

\( \hat{\omega} \) Radian frequency of discrete-time signals. As \( \hat{f} \), the discrete-time radian frequency is likewise dimensionless: \( \hat{\omega} = \omega T_s = 2\pi f / f_s \), where \( \omega \), \( T_s \), and \( f_s \) denote the radian frequency of continuous-time signals, the sampling interval, and the sampling frequency, respectively.
A generic norm selected to be used for a specific scaling; see page 77. With subscript, $\ell_H$ is the $\ell$-norm of the (partial) filter with transfer function $H$.

$\mathcal{H}(\hat{\omega})$ Frequency response of a filter with transfer function $H(z)$: $\mathcal{H}(\hat{\omega}) = H(z = e^{j\hat{\omega}})$.

$\mathbb{R}\{\cdot\}$ $\mathbb{R}\{z\}$ gives the real part of the complex number $z$.

$\tilde{\cdot}$ We use the tilde applied to a quantity to mean that the quantity is scaled. As an example, take an impulse response $h[k]$ of some filter before scaling; the filter after scaling will realize the scaled impulse response $\tilde{h}[k]$.

$|\cdot|$ Denotes the absolute value.

$[\cdot)$ Left-closed, right-open sets: $x \in [x_{\text{min}}, x_{\text{max}})$ means that $x_{\text{min}} \leq x < x_{\text{max}}$; see page 28.
We next list (some of) the used abbreviations. The list is to be understood as “work in progress;” we do not yet have all abbreviations, but we are willing to constantly improve this list.

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD</td>
<td>Analog-to-digital (converter); see page 7.</td>
</tr>
<tr>
<td>CT</td>
<td>Continuous time.</td>
</tr>
<tr>
<td>DA</td>
<td>Digital-to-analog (converter); see page 10.</td>
</tr>
<tr>
<td>DfII</td>
<td>Direct-form II (IIR filter).</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital signal processing, or digital signal processor; see page 1.</td>
</tr>
<tr>
<td>DT</td>
<td>Discrete time, see page 91.</td>
</tr>
<tr>
<td>FIR</td>
<td>Finite impulse response (filter).</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware description language; denotes any computer or programming language for the formal description of (mainly digital) electronic circuits.</td>
</tr>
<tr>
<td>IIR</td>
<td>Infinite impulse response (filter).</td>
</tr>
<tr>
<td>LSB</td>
<td>Least-significant bit.</td>
</tr>
<tr>
<td>MSB</td>
<td>Most-significant bit.</td>
</tr>
<tr>
<td>N/A</td>
<td>Not available; see page 13.</td>
</tr>
<tr>
<td>NOK</td>
<td>Not OK.</td>
</tr>
<tr>
<td>OK</td>
<td>All correct.</td>
</tr>
<tr>
<td>pd(·)</td>
<td>Probability distribution; see page 57.</td>
</tr>
<tr>
<td>SNR</td>
<td>Signal-to-noise ratio, see page 52.</td>
</tr>
<tr>
<td>SOS</td>
<td>Second-order section, see page 78.</td>
</tr>
<tr>
<td>VHDL</td>
<td>Very high speed integrated circuit hardware description language.</td>
</tr>
</tbody>
</table>
References


