A/D and D/A Converter Behavioral Modeling

Abstract. Models for D/A and A/D conversion as well as for quantization are presented. Quality criteria as SINAD, ENOB, SNR, THD, SINAD and SFDR will be described and computed with Matlab.

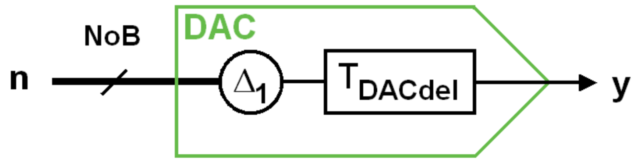
1 Introduction

The organization of this document is as follows:

2 D/A Converter Behavioral Modeling

Goal: This section presents D/A converter modeling. Not respected is delay.

Fig. 2.1: Linear D/A converter (DAC) model with amplification Δ1 and delay T_{DACdel}.



2.1 Nyquist-Sampling DACs

Translation from *NoL* ("number of levels") digital to *NoL* analog levels does not generate quantization noise. Non-linearity is modeled using $\Delta_k \neq 0$ for $k > 1$ in a polynomial form:

$$y = \sum_{k=0}^{NoC_{da}-1} \Delta_k \cdot n^k \quad = \quad \Delta_0 + \Delta_1 \cdot n + \ldots + \Delta_{NoC_{da}-1} \cdot n^{NoC_{da}-1} \tag{2.1}$$

with *NoC* being the number of coefficient, *y* and *n* are the DAC's output (typically voltage) and input, respectively, with *y* being typically a voltage and *n* an integer.

Δ_{DA} the minimum output step of the linear DAC, which is defined as

$$\Delta_k = \begin{cases} \Delta_{DA} & when \quad k = 1 \\ 0 & otherwise \end{cases} \tag{2.3}$$

The *NoC* coefficients can be computed from the *NoC* equations (2.1), that are linear functions of Δ_k and arise from the *NoC* characteristic points ($n_{c\#}$, $y_{c\#}$), #=1…*NoC*. Characteristic points may be outside the minimum / maximum range of the DAC. For example, we may use $x_{c256} = 256$ (corresponding to $y_{c256} = 2.56$V), while the 8-bit input signal has an upper bound of 255. Ansatz

$$\frac{y - y_{c1}}{y_{c2} - y_{c1}} = \frac{n - n_{c1}}{n_{c2} - n_{c1}} \tag{2.4}$$

delivers the linear model

$$\Delta_1 = \frac{y_{c2} - y_{c1}}{n_{c2} - n_{c1}} \quad (2.5), \qquad \Delta_0 = y_{c1} - \Delta_1 \cdot n_{c1} \ . \quad (2.6)$$

Exercise (solutions below)**:** Given are the 2 characteristic points (n_{c1}, y_{c1})=(0, 0) and …

(a) … (n_{c2}, y_{c2})=(256, <u>2.56V</u>). What is its maximum output voltage y_{max} when *NoB*=8 bits?

(b) … (n_{c2}, y_{c2})=(256, <u>3.3V</u>). What is its maximum output voltage y_{max} when *NoB*=8 bits?

Solution (a): Δ_1 = (2.56V-0V) / (256-0) = 1mV, $n_{max} = 2^8\text{-}1=255$. $y_{max} = n_{max}\,\Delta_1 + \Delta_0 = 255 \cdot 1\text{mV} + 0\text{mV} = 2.55\text{V}$. Note that $y_{max} < y_{c2}$!

Solution (b): Δ_1 = (3.3V-0V) / (256-0) = 1.289mV, $n_{max} = 2^8\text{-}1=255$. $y_{max} = n_{max}\,\Delta_1 + \Delta_0 = 255 \cdot 1\text{mV} + 0\text{mV} = 3.287\text{V}$. Again $y_{max} < y_{c2}$!

2.2 Bounding the Output Signal by Clipping

Modeling an upper bound y_{max} of a signal y : $y_{bounded,high} = \min(y_{\max}, y)$ (2.7)

Modeling an lower bound y_{min} of a signal y : $y_{bounded,low} = \max(y_{\min}, y_{bupper})$ (2.8)

Modeling both lower and upper bound to a signal y by combining equations (2.7) and (2.8):

$$y_{bounded} = \max(y_{\min}, \min(y_{\max}, y)) \tag{2.9}$$

2.3 *Matlab* Model without Delay

Listing 2.3: Matlab code of a simple D/A Converter (DAC) behavioral model.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Module : f_dac
% Purpose: polynomial D/A Converter (DAC) model with bounds
% Inputs : n:      vector of input values to be converted
%          delta:  coefficients of DAC: delta_k=delta(k+(1))
%          bounds: =[bmin bmax]: output bounds optional
% Outputs: y, same vector-length as n
% Author : Martin Schubert
% Date   : 23.Jul.2018
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
function y = f_dac(n,delta,bounds)
y = 0;                       % initialize
if exist('delta');           % coefficient vector available?
  n_power_k=1;               % initialize n^k
  for k=1:length(delta);     % evaluate polynomial
    y=y+delta(k)*n_power_k;
    n_power_k = n_power_k.*n;
  end;
end;
if exist('bounds');                       % [ymin,ymax] boundaries available?
  y=max(bounds(1),min(bounds(2),y));      % clip output vector
end;
```

2.4 Modeling DAC Delay

Time domain: $y(t) \rightarrow y(t - T_{DACdel})$

Frequency domain: $Y(s,z) \rightarrow Y(s,z) \cdot e^{-T_{DACdel}}$

Typical assumption: Zero Order Hold (ZOH) using

$$T_{DACdel} = T_s / 2$$

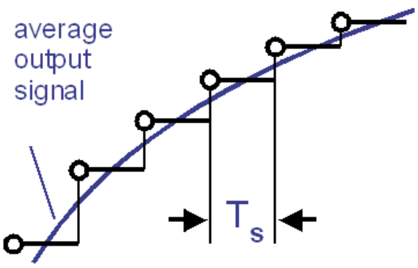


Fig. 2.4: Zero Order Hold (*ZOH*) sampling

2.5 Over-Sampling DACs

2.5.1 Pulse-With-Modulation (PWM) DACs

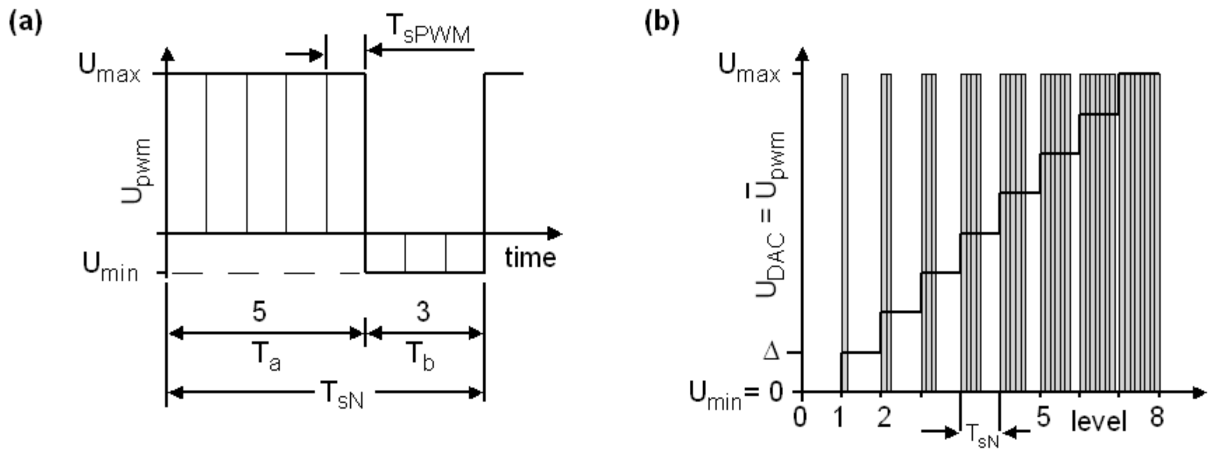


Fig. 2.4: (a) Pulse-width modulated signal, and **(b)** behavioral model using average levels.

For behavioral modeling, use the behavioral model of a *Nyquist*-sampling DAC with *NoL* = *pwm_period*+1 levels as indicated by the staircase line in Fig. 2.2(b), whereas *pwm_period* is the length of the *PWM* bit-sequence. Delay is assumed to be $T_{DACdel} = T_{sN} / 2$.

2.5.2 Delta-Sigma (ΔΣ) DACs

ΔΣ DACs consist of a modulator generating a pseudo-random data stream and a lowpass as demodulator. The output data is "pseudo" random, because the random process is controlled such, that the signal information is coded within the mean value of the data stream. In the technical most important case of so-called switch–mode conversion the modulator outputs a two-level pseudo-random bit-stream.

The ΔΣ modulator is not bound to certain levels as the PWM DAC. Any level can be represented by the average of the pseudo-random output data stream. Consequently:

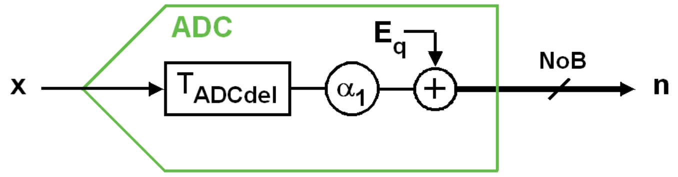
ΔΣ DACs including modulator and demodulator can be modeld without quantization steps.

Delay for ΔΣ modulators depends on the demodulating lowpass and is typically $T_{DACdel} >> T_s / 2$.

3 A/D Converter Behavioral Modeling

Goal: This section presents A/D converter modeling. Not respected is delay.

Fig. 3.1:
Linear A/D converter (ADC) model with delay T_{ADCdel}.



3.1 Value-Discretization (Quantization)

Translation from an infinite number of continuous values to *NoL* digital levels comes with round-off (quantization) noise model by the *round* function as

$$n = round\left(\sum_{k=0}^{NoC_{ad}-1} \alpha_k \cdot x^k\right) = round\left(\alpha_0 + \alpha_1 \cdot x + \ldots + \alpha_{NoC_{ad}-1} \cdot x^{NoC_{ad}-1}\right) \tag{3.1}$$

with *NoC* being the Number of Coefficients and polynomial *order* +1. It turns out that

$$\alpha_1 = \frac{1}{\Delta_{AD}} \tag{3.2}$$

with Δ_{AD} being the minimum input step (resolution) of the ADC. This ideal case is defined as

$$\alpha_k = \begin{cases} 1/\Delta_{AD} & when \quad k = 1 \\ 0 & otherwise \end{cases} \tag{3.3}$$

in the signal processing sense. Mathematical linearity will allow for an offset ($\alpha_0 \neq 0$), too.

The *NoC* coefficients are computed from *NoC* equations (3.1), that are linear in α_k and arise from the *NoC* characteristic points ($n_{c\#}$, $y_{c\#}$), #=1…*NoC*. Characteristic points may be outside the minimum / maximum range of the ADC. For example, output n_{c256} = 256 may correspond to input voltage x_{c256} = 3.3V, while the 8-bit output range has an upper bound of 255. Ansatz:

$$\frac{x - x_{c1}}{x_{c2} - x_{c1}} = \frac{n - n_{c1}}{n_{c2} - n_{c1}} \tag{3.4}$$

delivers the linear model $\alpha_1 = \dfrac{n_{c2} - n_{c1}}{x_{c2} - x_{c1}}$ (3.5), $\alpha_0 = n_{c1} - \alpha_1 \cdot x_{c1}$ (3.6)

Exercise: For sufficiently busy ADC input its quantization noise power is model as $E_q^2 = \Delta_{AD}^2 / 12$ delivering the effective (rms) quantization noise voltage $E_q = \Delta_{AD} / \sqrt{12}$. Equidistribution over f=0…f_s/2 delivers spectral density $E_q^{'2} = \Delta_{AD}^2 / 6 f_s \Leftrightarrow E_q^{'} = \Delta_{AD} / \sqrt{6 f_s}$. Compute E_q and E'_q as function of α_1 instead of Δ_{AD}.

Solution: Replace Δ_{AD} by $1/\alpha_1$: $E_q = 1/(\alpha_1 \cdot \mathrm{sqrt}(12))$, $E'_q = 1/(\alpha_1 \cdot \mathrm{sqrt}(6 \cdot f_s))$.

3.2 Bounding Signals by Clipping

Modeling both lower and upper bound to signal y can be done by

$$n_{bounded} = \max(n_{\min}, \min(n_{\max}, n)) \tag{3.7}$$

3.3 *Matlab* Model without Delay

Listing 3.3: Matlab code of a simple A/D Converter (ADC) behavioral model. Parameter *noquant* prohibits quantization, as computation of error $e_q=y-y_{ref}$ requires an unquantized y_{ref}.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Module : f_adc
% Purpose: polynomial A/D Donverter (ADC) model with bounds
% Inputs : x:      vector of input values to be converted
%          alpha:  coefficients of DAC: alpha_k=alpha(k+(1))
%          bounds: =[bmin bmax]: output bounds, optional
%          noquant: no quantization if this parameter exists
% Outputs: n, same vector-length as x
% Author : Martin Schubert
% Date   : 23.Jul.2018
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
function n = f_adc(x,alpha,bounds,noquant)
n = 0;                 % initialize
x_power_k=1;           % initialize x^k
for k=1:length(alpha); % evaluate polynomial
  n=n+alpha(k)*x_power_k;
  x_power_k = x_power_k.*x;
end;
if not(exist('noquant')); n = round(n); end; % round if not prohibited
if exist('bounds');                     % [ymin,ymax] boundaries available?
  n=max(bounds(1),min(bounds(2),n)); % clip output vector
end;
```

3.4 Modeling ADC Delay

Time domain: $n(t) \rightarrow n(t - T_{ADCdel})$

Frequency domain: $N(s,z) \rightarrow N(s,z) \cdot e^{-T_{ADCdel}}$

Tapcial assumption: $T_{ADCdel} = T_s$

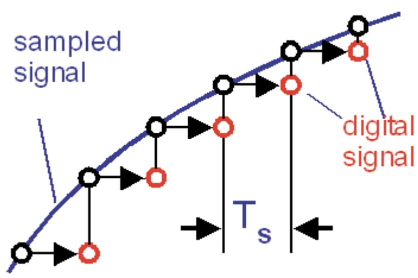


Fig. 3.4: sampled analog input curve

4 Quantization Behavioral Modeling

4.1 Multi-Bit Quantization

The process of value-discretization with a given step Δ_1, also termed quantization, is the representation of a quantity x in terms of an integral multiple of a minimum y step Δ_1. It corresponds to the combination of an ADC followed by a DAC with $\alpha_1 = 1/\Delta_1$.

$$y = \Delta_1 \cdot round\left(x / \Delta_1\right). \tag{4.1}$$

We may add a compensated offset that cancels out using $\alpha_0 = -\Delta_0/\Delta_1 = -\Delta_0 \cdot \alpha_1$:

$$y = \Delta_1 \cdot round\left(\frac{x-\Delta_0}{\Delta_1}\right) + \Delta_0 . \tag{4.2}$$

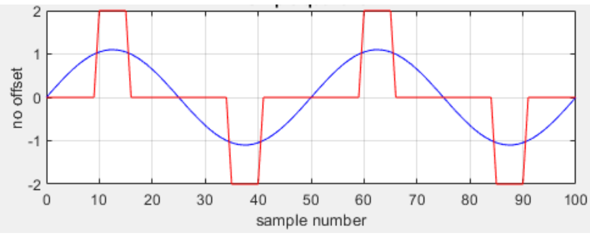
It is seen in Fig. 4.1(b), that using balanced offset cancels out and does not introduce offset after all. Consequently, using this balanced offset is compliant with linearity in a signal processing sense.

Listing 4.1: Quantizer code delivering Figs. 4.2(a) and (b). As Matlab array begin with index 1, we have *delta*(1) = Δ_0 being the offset and *delta*(2) = Δ_1 being the step size in the model.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Module : f_quantize
% Purpose: model linear quantizer y=delta*round(x/delta)
% Inputs : x:      vector of input values to be converted
%          delta:  vector with 2 elements
%                  delta(1)=delta_0: offset.
%                  delta(2)=delta_1: step, when =0 no quantization.
%          bounds: =[bmin bmax]: output bounds, optional
% Outputs: y, same vector-length as x
% Author : Martin Schubert
% Date   : 19.Nov.2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
function y = f_quantize(x,delta,bounds)
if or(delta(2)==0, not(exist('delta')));
  y = x;
else
  y = delta(1) + delta(2)*round((x-delta(1))/delta(2));
end;
if nargin > 2;
  y=max(bounds(1),min(bounds(2),y)); % clip output vector
end;
```

4.2 Single-Bit Quantization

Using compensated offset as in eq. (4.2) is particularly useful in single-bit quantization as illustrated in Fig.2.1 Fig. 2.1(a) uses Δ_1=2, Δ_0=0 delivering unsatisfactory results with 3 levels instead of 2. The desired solution shown in Fig. 2.1(b) is obtained using offset Δ_0 = 1.



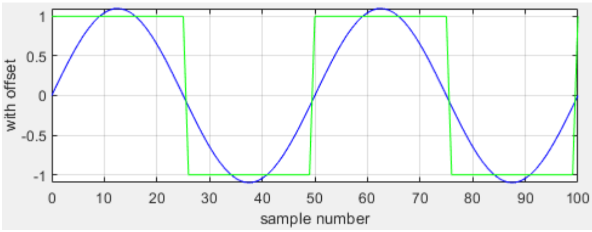


Fig. 4.2: simple 2-level quantization **(a) left:** with $x_0 = 0$ and **(b) right:** with $x_0 = 1$.

Listing 4.2: Matlab testbench generating Fig. 4.2.

```
% tb_quanitze: Testbench for f_quantize
clear all;    % clear workspace
addpath('../../functions');
%
% General specifications for quantization
NoL     = 2;        % Number of quantization Levels
xmin    = -1.2;   xmax =  1.2;
ymin    = -1;     ymax =  1;
bounds = [-0.5  0.5];
%
NoS    = 101;  % Numbe of Samples
t      = 0:NoS-1;
F      = 1/50; % frequency rel to sample -> wavelength=1/F samples
x      = 1.2*sin(2*pi*F*t);
%
% quantization
deltaX   = (ymax-ymin)/(NoL-1);% Quantization deltY: smallest possible step
%          f_quantize(input, [offset, delta])
y1       = f_quantize(  x   , [   0   ,    2   ]); % no   offset adjustment
y2       = f_quantize(  x   , [   1   ,    2   ]); % with offset adjustment
%
% Graphical postprocessing
figure(42);
subplot(211); plot(t,x,'b',t,y1,'r-'); grid on;
title('simple quantizer'); xlabel('sample number'); ylabel('no offset');
subplot(212); plot(t,x,'b',t,y2,'g-'); grid on;
xlabel('sample number'); ylabel('with offset');
```

5 Linear Transmission System Behavioral Modeling

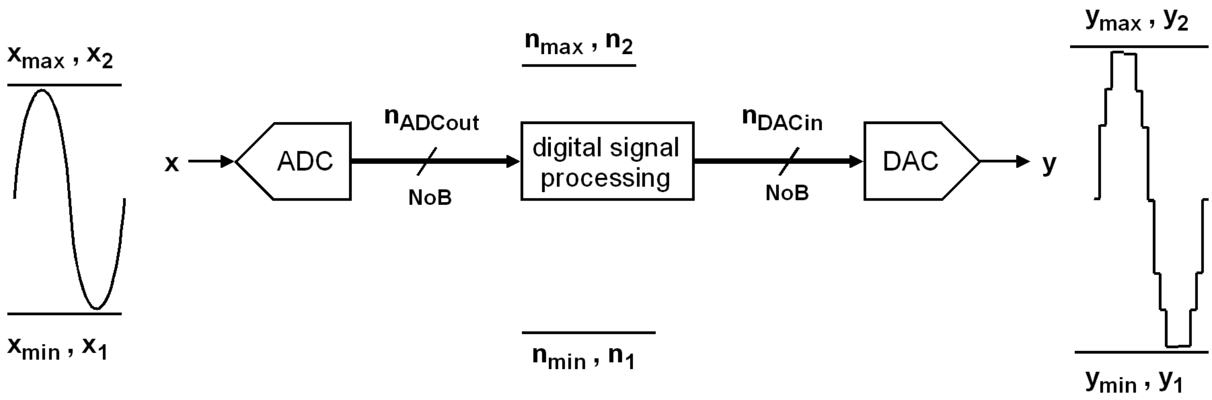


Fig. 5: Top-level view of an A/D and D/A (A/D/A) Conversion system.

Assume any System as shown in Fig. 5, which is linear in a mathematical sense (i.e. it may have offset: $c_0 \neq 0$), so that it can be described as

$$y = c_0 + c_1 x \tag{5.1}$$

If the system is non-inverting, so that input states x_{min}, x_{max} correspond to output states y_{min}, y_{max}, respectively, we write

$$\frac{y - y_{\min}}{y_{\max} - y_{\min}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}\,. \tag{5.2}$$

If the system might be inverting, so that x_{min} corresponds to y_{max}, then it is unambiguous to use input values x_1, x_2 corresponding to output values y_1, y_2, respectively, and write

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1} \tag{5.3}$$

which translates to $y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}\left(x - x_1\right)$ and delivers coefficients

$$c_1 = \frac{y_2 - y_1}{x_2 - x_1} \tag{5.4}$$

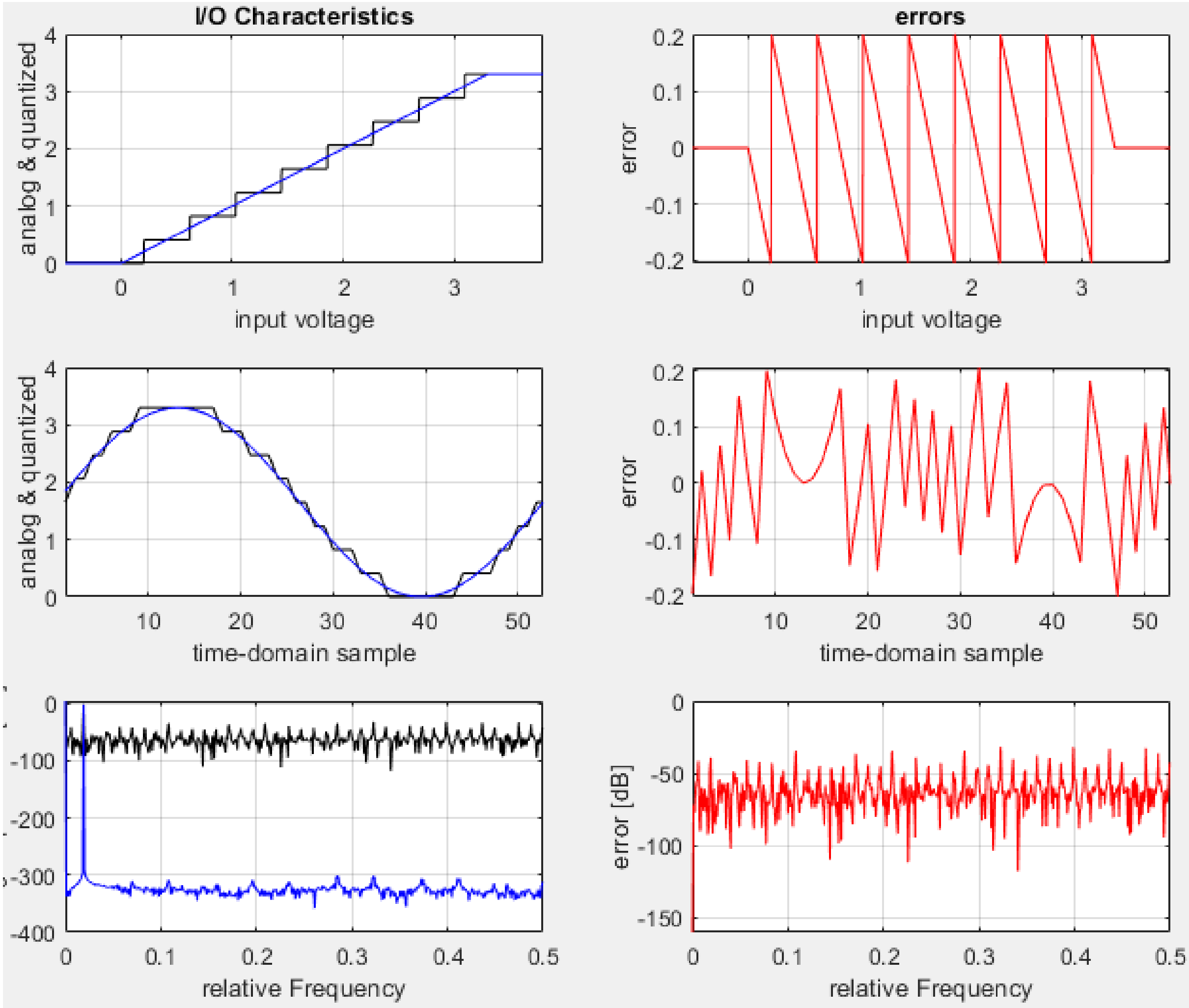
$$c_0 = y_1 - c_1 x_1 \tag{5.5}$$

with respect to equation (5.1).

6 A/D/A Conversion Modeling Using *Matlab*

Goal of this section is to get experience with A/D/A conversion modeling using *Matlab*.

6.1 Getting Started with *Matlab* and Testbench *tb_ada*



Left column: signals, right column error
Domains: top Output vs. Input (DC), middle: time (transient), bottom frequency (AC)

Fig. 6.1: Plot obtained with *f_adc* and *f_dac* and testbench *tb_ada* according to listing 6.1

The *Matlab* script shown in listing 6.1 produces Fig. 6.1. The left hand side shows signals and the right hand side errors. We see:

- Top row: DC mode = static output versus input signal characteristics: $y(x)$, modeled as *y_x*.
- Middle row: Transient mode = time domain: signals over time axis, e.g. $y(t)$ modeled as *y_t*.
- Bottom row: AC mode: signals over absolute frequency $F=f/f_s$, $Y(F)$ modeled as *X_f*.

Get familiar with *table 6.1*. Check for the exercises below.

Listing 6.1: A/D/A modeling

```
% tb_ada: Testbench for A/D and D/A converters
clear all; addpath('../../functions/');
%
% DAC specifications
%nda_c=[0  2      4      6      8   ]; % 5 n-inputs
%yda_c=[0  3.3*1/4 3.3*2/4 3.3*3/4  3.3 ]; % 5 y-outputs
nda_c=[0 1      2     3      4    5      6     7      8  ]; % 9 n-inputs
yda_c=nda_c*3.3/8;                                          % 9 n-outputs
%yda_c=[0 0.4125  0.825 1.2375 1.65 2.0625 2.475 2.8875 3.3]; % 9 n-outputs
% Typical measured DAC characteristics
%yda_c = [0 0.4140 0.8285 1.2430 1.6574 2.0718 2.4864 2.9009 3.3154];
% Untypical measured DAC characteristics needing 8. harmonic for SFDR
%yda_c = [0.0015 0.404 0.803 1.216 1.631 2.013 2.42 2.831 3.242];

bda   = [0.0 3.30];             % output clipping bounds for DAC
delta = f_PolyInit(nda_c,yda_c) % compute coefficints for DAC
%
% ADC specifications
xad_c = [0.5:7.5]*3.3/8;        % characteristic input data points  to ADC
nad_c = [0.5:7.5];              % characteristic output data from ADC
%nad_c(3) = 2.5001;              % add a non-linearity
bad   = [0     8];              % clipping bounds for ADC
alpha = f_PolyInit(xad_c,nad_c) % compute coefficints for ADC
%
% DC: I/O characteristics
NoS_x    = 1001;                % Number of x-Samples
xmargin  = 0.5;                 % abscissa extension over x_c-bounds
xmin   = min(xad_c)-xmargin;    % left end of abscissa x
xmax   = max(xad_c)+xmargin;    % right end of abscissa x
xstep  = (xmax-xmin)/(NoS_x-1); % x step for measurement
x      = xmin:xstep:xmax;       % x for y(x) plot
%
% using the 2 functions: f_adc and f_dac
n_x    = f_adc(x,alpha,bad);           % quantized ADC output
y_x    = f_dac(n_x,delta,bda);         % DAC output from quantized n_x
nref_x = f_adc(x,alpha,bad,'noquant'); % unquantized ADC output nref_x
yref_x = f_dac(nref_x,delta,bda);      % DAC out from unquantized nref_x
eq_x   = y_x - yref_x;                 % quantization error
figure(61); subplot(321); plot(x,y_x,'k',x,yref_x,'b');
title('I/O Characteristics'); xlim([min(x) max(x)]);
xlabel('input voltage'); ylabel('analog & quantized'); grid on;
subplot(322); plot(x,eq_x,'r');  xlim([min(x) max(x)]);
title('errors'); xlabel('input voltage'); ylabel('error'); grid on;
%
% Transient: time domain
NoS_t  = 1001;         % Number of Samples on time axis
t      = 0:NoS_t-1;    % sampled time axis
NoW_t  = 19.0;         % Number of Waves ovr entire time axis
Fsig_t = NoW_t/NoS_t;  % signal frequency relative to sampling rate
xamp_t = 1.65;         % amplitude of sinusoidal input signal
xoff_t = 1.65;         % offset of sinusoidal input signal
x_t    = xamp_t*sin(2*pi*Fsig_t*t) + xoff_t; % sinusoidal test signal
n_t    = f_adc(x_t,alpha,bad);            % quantized ADC output
y_t    = f_dac(n_t,delta,bda);            % DAC output from quantized n_t
nref_t = f_adc(x_t,alpha,bad,'noquant');  % analog ADC output as reference
yref_t = f_dac(nref_t,delta,bda);         % DAC output from  nref_t
eq_t   = y_t - yref_t;                    % quantization error
subplot(323); plot(t,y_t,'k',t,yref_t,'b'); xlim([1 1/Fsig_t]);
xlabel('time-domain sample'); ylabel('analog & quantized'); grid on;
subplot(324); plot(t,eq_t,'r'); xlim([1 1/Fsig_t]);
xlabel('time-domain sample'); ylabel('error'); grid on;

% apply window functions
SideLobeAttenuation_dB = 150; % Relative side-lobe atten., default: 100dB
win=ones(1,NoS_t);                              % rectangular window
%win=chebwin(NoS_t,SideLobeAttenuation_dB)';    % Matlab's chebwin
%win=f_winCheb(NoS_t,SideLobeAttenuation_dB); % selfmade Chebychev window
%win=blackman(NoS_t);                           % Blackman window
%win=blackmanharris(NoS_t);                     % Blackman-Harris window

% frequency domain
NoS_F  = NoS_t;                        % Number of frequency Samples
F      = [0:NoS_F-1]/NoS_F; % rel. Frequency axis, F=f/fs, fs:sampling rate
X_F    = f_dB(fft(x_t.*win)/NoS_F);    % spectral x_t=x(t)
Y_F    = f_dB(fft(y_t.*win)/NoS_F);    % spectral quantized  y_t=y(t)
Yref_F = f_dB(fft(yref_t.*win)/NoS_F); % spectral analog y(t)
Eq_F   = f_dB(fft(eq_t)/NoS_F,1e-8); % spectral error: y(t)-yref(t)
subplot(325); plot(F,Y_F,'k',F,Yref_F,'b'); xlim([0 0.5]);
xlabel('relative Frequency'); ylabel('analog & quantized [dB]'); grid on;
subplot(326); plot(F,Eq_F,'r'); xlim([0 0.5]);
xlabel('relative Frequency'); ylabel('error [dB]'); grid on;
%
%tb_characterize % call characterization script
```

Listing 6.1 uses functions *f_PolyInit* and *f_dB* shown in listings 6.3.1 and 6.3.2, respectively.

Table 6.1: Parameters of listing 6.1 yielding Fig. 6.1

Extension...

_c	denotes characterizing data, e.g. vectors *x_c, n_c, y_c* for defining ADC and DAC characteristic curves.
_x	denotes static (DC) data, e.g. static I/O characteristic curves of ADC and DAC.
_t	denotes time-domain (transient) data, i.e. curves over the time axis.
_F	denotes frequency-domain (AC) data, i.e. curves over the relative frequency axis F, whereas $F=f/f_s$ with f being real frequency in Hz and f_s sampling rate.

General Specifications

NoL	int, Number of possible quantization Levels
NoD	Number of Deltas, *NoD* = *NoL*-1 or *NoL* depending on the situation
x_c	vector of input data points defining the ADC static characteristic curve
n_c	vector of data points, ADC output and DAC input, of static characteristic curve
y_c	vector of output data points defining the DAC static characteristic curve
delta	= [Δ_0 Δ_1 Δ_2 ...] coefficients of the polynomial, same length as *n_c, y_c*, defining a polynomial through points (*n_c*$_k$, *y_c*$_k$), k=1:length(*y_c*).
alpha	= [α_0 α_1 α_2 ...] coefficients of the polynomial, same length as *x_c, n_c*, defining a polynomial through points (*x_c*$_k$, *n_c*$_k$), k=1:length(*x_c*).
f_PolyInit	function computing *NoC* coefficients c_k of a polynomial interpolating data points (x_k,y_k), k=0…*NoC*-1.
bad	output min/max bounds of *f_adc*.
bda	output min/max bounds of *f_dac*.

DC Modeling

x	abscissa of DC plot, *x = xmin : xstep:xmax*.
NoS_x	Number of Samples on *x*
xmargin	extension of DC abscissa x over input signal range *x*.
n_x	= *f_adc(x)* → integral numbers
y_x	= *f_dac(x)*.
nref_x	= *f_adc(n_x)* without quantization → real numbers
yref_x	= *f_dac(nref_x)*. required to compute e_q
eq_x	= *y_x* – *yref_x* , quantization error

Transient (Time-Domain) Modeling

t	abscissa of transient plot, *t = 0:NoS_t-1;*.
x_t	input signal over time axis, same length as *t*
NoS_t	Number of Samples on time axis: *t*
NoW_t	Number of Waves on time axis *t*.
Fsig_t	= *NoW_t/NoS_t*: signal frequency relative to sampling rate (f_s).
xamp_t, xoff_t	= amplitude and offset of signal *x_t* = *xamp_t*$*sin(2\pi F_{sig}\cdot t)+x_{off}$*_t*
n_t	= *f_adc(x_t)* → integral numbers
y_t	= *f_dac(x_t)*.
nref_t	= *f_adc(n_t)* without quantization → real numbers
yref_t	= *f_dac(nref_t)*. required to compute e_q
eq_t	= *y_t* – *yref_t*, quantization error

AC (Frequency-Domain) Modeling

F	abscissa of AC plot, F = [$0{:}NoS\_F\text{-}1$]/NoS_F = [0:1). $F=f/f_s=fT_s$.
X_F	Fourier transformed of x_t in dB
Y_F	Fourier transformed of y_t in dB
Yref_F	Fourier transformed of $yref\_t$ in dB
Eq_F	Fourier transformed of e_q_t in dB
NoS_F	Number of Samples on frequency axis: F , $NoS\_F=NoS\_t$.

6.1.1 Getting started with Matlab model *tb_ada*

Unpack file *ADA_Modeling_Matlab.zip* provided by the author, navigate to subdirectory *ADA_Modeling_Matlab*\testbenches\tb_ada\ and double-click left on script-file *tb_ada.m* . *Matlab* should start. Click on run to get the graphics shown in listing 6.1 related to Fig. 6.1.2, i.e. $\boldsymbol{n}_{ADCout} = \boldsymbol{n} = \boldsymbol{n}_{DACin}$ modeled as *n_x* and *n_t* for DC and transient mode simulation, respectively. Identify *n_x* and *n_t* in the code. We will work with "fractional integers" $\boldsymbol{n}$, which is necessary for error computation and may be useful for oversampling data converters like ΔΣ converters with averaging demodulators. Quality criteria like *effective number of bits* (*ENOB*) are fractional, too.

The second code line

```
clear all; addpath('../../functions/');
```

clears the workspace and adds our directory functions to the *Matlab's* search path, i.e. it makes all our selfmade *Matlab* functions available, that are located in directory ../../functions/. Note that Matlab uses UNIX/Linux notation, where a slash separates directories (not a backslash), and a single dot "." stands for "this directory" and a double-dot ".." for "parent directory".

Frequency domain window:

The lower left widow is the frequency domain window. The blue, unquantized curve as a "signal to noise-floor ratio" of >300…320 dB. How do you explain this range?

Divide them 20dB to get 15…16 decimal places, which is the round-off noise of the double-precision floating point numbers used as default data type by Matlab.

As we have sampled data points only without information of the real time span between them, we compute a relative frequency $F = f / f_s$ with f_s being the sampling frequency, which corresponds to F=1. Consequently, the maximum frequency of our Fourier transform becomes 1. Why do we limit our F-axis to F = 0… ½ instead of F = 0…1?

Sampled signal spectra are periodic in $f=f_s$ and consequently in F=1. They behave symmetric around $f_s/2$ and consequently around F=1/2. Consequently F>1/2 supplies no new information, but only a symmetric copy of range F=0…1/2.

Check it out by changing line

```
subplot(325); plot(F,Y_F,'k',F,Yref_F,'b');  xlim([0 0.5]);
```

to

```
subplot(325); plot(F,Y_F,'k',F,Yref_F,'b');  % xlim([0 0.5]);
```

which out-comments the limitation of the F-axis.

6.1.2 Defining Characteristics of the D/A Converter Model *f_dac*

We supply NoC_{da} points of the DAC's characteristic curve by vectors *nda_c*, *yda_c*. Statement "*delta = f_PolyInit(nda_c,yda_c*" computes the NoC_{da} coefficients of the polynomial

$$y = \sum_{k=0}^{NoC_{da}-1} \Delta_k \cdot n^k \quad = \quad \Delta_0 + \Delta_1 \cdot n + \ldots + \Delta_{NoC_{da}-1} \cdot n^{NoC_{da}-1} \tag{2.1}$$

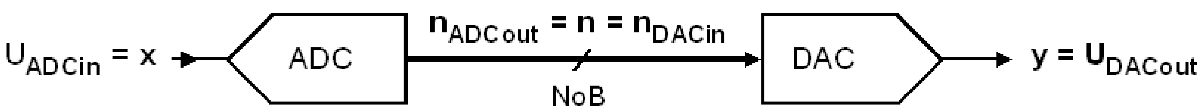
interpolating the NoC_{da} (in the example 9) characteristic points. *Matlab* prints them in its *Command Window*. First of all the DAC is ideal and Δ_1 the only coefficient $\neq 0$. Uncomment other statements defining a vector *yda_c*, knowing that the last writing on a variable overrides all previous assignments. Try some small deviations from the ideal characteristics. Look up *nda_c* in the workspace and observe its impact in the blue curves of the graphics.
How many additional harmonics does the 8th order polynomial generate? **-> 7: 2…8**

Vector *bda* sets the [min max] boundaries, in the example *bda* = [0 3.30]. Use ideal characteristics again, change bounds to *bda* = [0 3.29] and observe the effect of clipping.
How many additional harmonics does a little bit clipping generate? **-> infinite**

Note that characteristic points might be outside the bounds, e.g. point (256⇔3.3V), while the maximum input of an 8-bit word is 255.

After these tests, bring the code back to its initial state (for example with keys *CTRL+z*).

(a) A/D/A system assumed



(b) Dividing 0…3.3V into 8 Δ's using 9 levels.

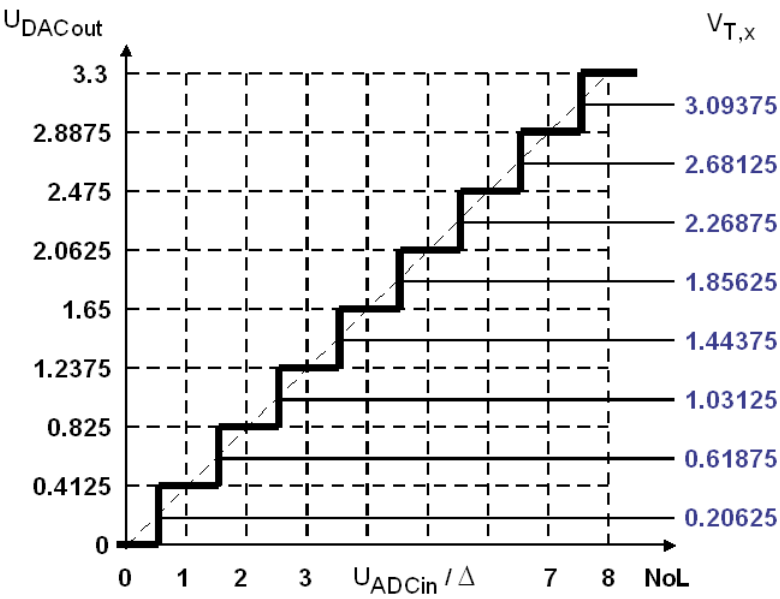


Fig. 6.1.2: Example data

6.1.3 Defining Characteristics of the A/D Converter Model *f_adc*

We supply NoC_{ad} (in the example 8) points of the ADC's characteristic curve by vectors *xad_c*, *nad_c*. Statement "*alpha = f_PolyInit(xad_c,nad_c)*" computes the NoC_{ad} coefficients of the polynomial

$$n = round\left(\sum_{k=0}^{NoC_{ad}-1} \alpha_k \cdot x^k\right) = round\left(\alpha_0 + \alpha_1 \cdot x + \ldots + \alpha_{NoC_{ad}-1} \cdot x^{NoC_{ad}-1}\right) \quad (3.1)$$

interpolating the NoC_{ad} characteristic points, and *Matlab* prints them in its *Command Window*. The DAC is modeled ideal and α_1 the only coefficient $\neq 0$.

Note that the characteristic curve is defined with self-contradictory "fractional integers" like *n.5*. This is because *xad_c* is a vector of thresholds, so that inputs $x = V_{Tn} - \varepsilon \rightarrow n$ while $x = V_{Tn} + \varepsilon \rightarrow n+1$.

Function *f_adc( )* is always used twice, e.g. in lines

```
n_t    = f_adc(x_t,alpha,bad);            % quantized ADC output
y_t    = f_dac(n_t,delta,bda);            % DAC output from quantized n_t
nref_t = f_adc(x_t,alpha,bad,'noquant'); % analog ADC output as reference
yref_t = f_dac(nref_t,delta,bda);         % DAC output from  nref_t
eq_t   = y_t-yref_t;                      % quantization error
```

whereas *n_#* is realistic quantized ADC output while *nref_#* is unquantized theoretically ideal ADC output for infinite resolution, which is required to compute quantization error *eq_#*.

Vector *bad* sets the [min max] boundaries, in the example *bda* = [0 8].

Note that characteristic points might be outside the bounds, e.g. point (3.3V⇔256), while the maximum output of an 8-bit word is 255.

After these tests bring the code back to its initial state (for example with keys *CTRL+z*).

6.1.4 Working with Discrete and Fast *Fourier* Transformation

DFT and FFT

The digital Fourier transformation (DFT) translates *NoS* time domain samples to *NoS* frequency domain samples. It can be greatly accelerated by the *fast Fourier transformation* (FFT), when $NoS = 2^M$ with M being an integral number. *Matlab* offers the functions *fft*( ) and *ifft*( ) for the FFT and its inverse. From the author's experience *Matlab's fft* works excellent even when $NoS = 2^M$ with M being a fractional number, e.g. $NoS = 1001$. Frequency resolution become better with increasing *NoS*.

Both DFT and FFT produce real and imaginary part, allowing for magnitude and phase representation as typical for Bode diagrams. Here, magnitude information is more interesting and is obtained by Matlab function *abs*( ).

Using the *plot*( ) command for FFT

Matlab command $y = fft(x)$ outputs a vector y having the same length as input vector x.

Try *Matlab* command line: (Command "*figure(1)*" creates a plot window marked "Figure 1")

```
figure(1); NoS=101; t=0:NoS-1; x=sin(0.1*t); Y=fft(x); plot(Y);
```

What happens? Explain the result! (Ignore "figure(1)", which causes figure handle to be 1.)

FFT result Y is a vector of complex numbers.
Matlab command plot(Y) plots Re{Y} versus Im{Y}.

Try *Matlab* command line (continues writing in "Figure 1" window)

```
NoS=101; t=0:NoS-1; x=sin(0.1*t); Yabs=abs(fft(x)); plot(Yabs);
```

What happens? Explain the result! Explant abscissa (x-axis) scaling.

Yabs is a vector of real numbers.
Matlab command plot(Yabs) plots Yabs versus index(Yabs).

Try *Matlab* command line using any frequency vector f with same length as t:

```
NoS=101; t=0:NoS-1; x=sin(0.1*t); Yabs=abs(fft(x)); f=628*t; plot(f,Yabs);
```

What happens? Explain the result! Explant abscissa (x-axis) scaling.

Matlab command plot(f,Yabs) plots Yabs versus f.

Frequency scaling for FFT

As we do not know the real-time sampling frequency, we scale it to relative frequency $F = f / f_s$ with correspondence $F = 1 \Leftrightarrow f_s = 1$.

Try *Matlab* command lines with relative frequency vector F with same length as t:

```
NoS=101; t=0:NoS-1; x=sin(0.1*t); Yabs=abs(fft(x));
F=[0:NoS-1]/NoS; plot(F,Yabs);
```

What happens? Explain the result! Explant abscissa (x-axis) scaling.

Matlab command plot(F,Yabs) plots Yabs versus F=0…1.

Sampled signal spectra magnitudes like $Yabs = |Y|$ are symmetric around $f_s / 2$ over frequency f, and consequently around ½ over relative frequency F. So we do not gain new information when we plot the frequency range $F = 0.5\ldots1$ and limit F to $0\ldots0.5$.

Try *Matlab* command lines with abscissa being limited to range $F = 0\ldots0.5$:

```
NoS=101; t=0:NoS-1; x=sin(0.1*t); Yabs=abs(fft(x));
F=[0:NoS-1]/NoS; plot(F,Yabs); xlim([0 0.5]);
```

DFT / FFT Assume Periodic Repetition of the Measurement Window

Run *Matlab* script *tb_ada* again. Line

```
NoW_t   = 19.0;          % Number of Waves over entire time axis
```

sets the number of waves in the transient window, in this case its 19 waves. A discrete *Fourier* transformation has to assume, that the time axis is repeated periodically, so that the curve at the end of the time window must fit to the phase at its beginning without phase jump. In other words: *NoW_t* must be an integral number.

Exercise: What happens if you set *NoW_t* = 19.01? Observe the effect in the lower left frequency-domain window; observe particularly the scaling of the ordinate (dB-axis).

Massive loss of unquantized accuracy from -320dB to -80dB

Explain the -320 dB accuracy. Where does this number -320dB come from?

Round-off noise of 16 decimal places double precision floating point numbers used by Matlab: 16·20dB = 320dB

Is it realistic for real measurements, particularly if we have several frequencies in out signal, which may be sound?

No! -> We have to use window functions

Within *tb_ada* set the Number of Waves over time axis to *NoW_t=20* and Number of Samples to *NoS_t* = 1000. What happens? Explanation?

All the 20 waves are sampled at exactly the same phase. Consequently, we get for every wave exactly the same quantization error. This error is no more a random process but adds up in a few values.

6.1.5 Using Window Functions

Practically we can hardly measure a signal such that its wavelength fits with the required accuracy into our measurement window, and it may be impossible when the signal contains several frequencies. Therefore, we apply so-called window functions. What we have applied so far was the "do-nothing window" consisting of ones only.

Run *tb_ada* with ideal initial conditions. You should yield some 320dB SNR.

For better observability of the man lobe (=main peak) reduce the *F*-axis section to $F = 0\ldots0.2$ within command line

```
subplot(325); plot(F,Y_F,'k',F,Yref_F,'b'); xlim([0 0.2]);
```

Run *tb_ada* with the ideal initial conditions. You should yield some 320dB SNR. Reduce the number of samples from $NoS\_t = 1001$ to 191, while Number of Waves is $NoW\_t = 19.0$. What happens?

We see that more time-domain samples yield a better frequency-domain resolution, but SNR of 320dB still remains.

Set $NoS\_t = 1001$ and reduce the *F*-axis section to $F = 0\ldots0.04$ within command line

```
subplot(325); plot(F,Y_F,'k',F,Yref_F,'b'); xlim([0 0.04]);
```

Run *tb_ada*. You should yield some 320dB SNR again. Then uncomment line

```
win=chebwin(NoS_t, SideLobeAttenuation_dB)';  % Matlab's chebwin
```

What happens?

Loss of SNR -320 to some -SideLobeAttenuation_dB

Try different values for parameter **`SideLobeAttenuation_dB`**, e.g. 50, 100, 150 (whereas 100 is the *Matlab* default value). What do you observe?

SNR is suppressed to "SideLobeAttenuation_dB". The price for better side-lobe attenuation is a wider main lobe. Furthermore, we have to accept some DC value.

Run *tb_ada* with the ideal initial conditions. You should yield some 320dB SNR. Then change the Number of Waves of time-axis from 19.0 to 19.01. What happens?

Strong deterioration of the SNR.

From this situation, uncomment the command

```
win=chebwin(NoS_t, SideLobeAttenuation_dB)';  % Matlab's chebwin
```

again. What happens?

The result in the frequency domain is as good and as bad as with NoW_t = 19.0.

Check for Blackman (*blackman*) or Blackman-Harris (*blackmanharris*) window functions in Matlab. Can we set side-lobe attenuation there, too?

No

Summarize your knowledge with window functions

No window function necessary when a wavelength fits ideally into the measurement window.

If a wavelength does not fit ideally into into the measurement window, results can be greatly improved with window functions.

Better side-lobe attenuation of window functions is paid for with a wider main lobe.

Chebychev window function allows to set a desired side-lobe attenuation, which is not the case for other window functions such as Blackman or Blackman-Harris.

6.2 Computing Quality Criteria with *Matlab*

Goal: Compute quality criteria with Matlab.

Check for *Matlab* commands *sfdr*, type *help sfdr* or check out the *Matlab* homepage.

Definition

- Noise occurs over the complete frequency axis.
- Distortion is caused by harmonics, i.e. non-linearities that reapeat with every wave f the test signal.

6.2.1 *SFDR*: Spurious Free Dynamic Range

The *SFDR* is the distance in power between a sinusoidal test signal and its greatest harmonic:

$$SFDR = \frac{P_{Signal}}{P_{\max.harmonic}}, \qquad THD_{dB} = 10dB \cdot \log\left(\frac{|X(f_1)|^2}{\max\left\{|X(f_k)|^2\right\}} \right) \quad \text{with } k > 1 \text{ and } \quad f_k = k \cdot f_1,$$

To make the SFDR big, the amplitude of the test signal is choosen to be as large as possible.

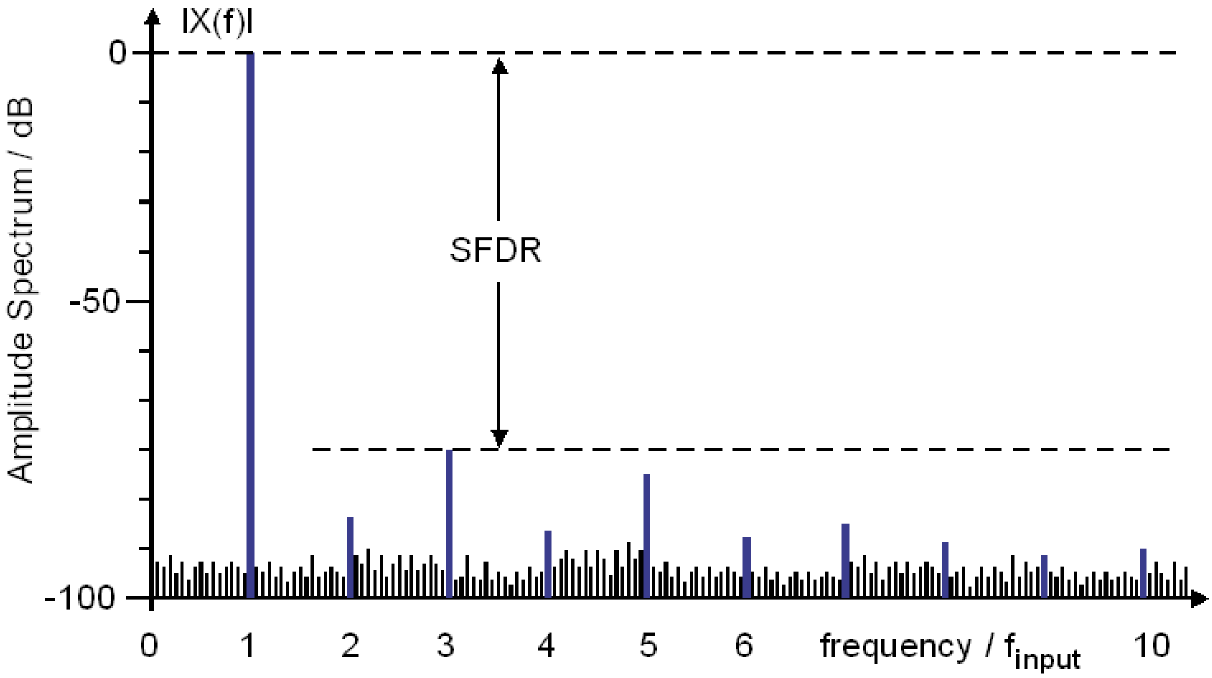


Fig. 6.2.1: Noise power spectrum

6.2.2 *THD*: Total Harmonic Distortion

The Total Harmonic Distortion (*THD,* dt. Klirrfator) is another measure for errors based on non-linearity. Test setup equals tht of *SFDR*, but it computes the energy of the harmonics of the sinusoidal signal with frequency f_1 compared to the energy at f_1 or the total signal energy.

$$THD = \frac{P_{Distortion}}{P_{Signal}}, \qquad THD_{dB} = 10dB \cdot \log\left(\frac{\sum_{k=2}^{N} \left|X(f_k)\right|^2}{\left|X(f_1)\right|^2} \right) \quad \text{with} \quad f_k = k \cdot f_1,$$

Comments:

- *THD* is typically negative, note that distortion is on the numerator, not in the denominator.
- Pricipally, |THD| ≤ SFDR, because $\sum_{k=2}^{N} \left|X(f_k)\right|^2 \geq \max\left\{\left|X(f_k)\right|^2\right\}$

6.2.3 *SINAD*: Signal to Noise and Distortion Ratio

Signal to Noise and Distortion ratio (SINAD) is another measure for errors based on non-linearity. Test setup equals tht of *SFDR*, but it computes the energy of the harmonics of the sinusoidal signal with frequency f_1 compared to the energy at f_1 or the total signal energy.

$$SINAD = \frac{P_{Signal}}{P_{Noise} + P_{Distortion}} \;.$$

Comment:

- As noises is never zero, SINAD < |THD|, and consequently SINAD < |THD| ≤ SFDR.

6.2.4 *SNR*: Signal-to-Noise Ratio

Signal to Noise Ratio: $SNR = \frac{P_{Signal}}{P_{Noise}} = \frac{A^2_{Signal}}{A^2_{Noise}}$ for any curve.

Matlab function *SNR = snr*(*x,e*) assumes *x* to be a vector of signal samples *e* a vector of error (noise) samples. With number of samples being *NoS = length(x) == length(e)* Matlab computes

$$SNR = snr(x,e) \quad \Leftrightarrow \quad SNR = \frac{\sum_{n=1}^{NoS} x_n^2}{\sum_{n=1}^{NoS} e_n^2}$$

If x_n is a sinusoidal test signal with maximum possible amplitude, then shule SNR =SINAD

6.2.5 *ENOB*: Effective Number of Bits:

If an *ADC* offer 16 output pins, but only 12 of them are accurate, the rest is noise, then *ENOB*=12. These supernumerary pins might have several reasons, e.g. inaccuracies in the most significant bits or noise in the input signal of the considered ADC. Offering 16 pins instead of 12 may be reasonable e.g. for pin-compatibility with more expensive *ADCs*.

$$\boldsymbol{ENOB} = \frac{SINAD - 1.76dB}{6.02dB} \quad ,$$

which assumes a sinusoidal test wave and triangular quantization noise waveforms. In case of rectangular quantization noise waveforms replace “-1.76dB” by “+3.01dB”.

6.2.6 *Matlab* modeling

Listing 6.2: ADA model with input/output clipping

```
% tb_characterize: computing A/D/A quality criteria
% run tb_ada before!
figure(62)
xWin=0; % Window function: 0: rectangular, 1: selfmade, 2: Matlab's chebwin
if(xWin==0)       win=ones(1,NoS_t);       % rectangular window
else if(xWin==1) win=f_winCheb(length(NoS_t),150);% selfmade Chebychev win
else              win=chebwin(length(NoS_t),150)'; % Matlab's chebwin
end; end;
% frequency domain, unquantized signal quality
subplot(421); sfdr(yref_t);
subplot(423); thd(yref_t); % thd(yref_t,1,8);
subplot(425); sinad(yref_t);
subplot(427); snr(yref_t);
% frequency domain, unquantized signal quality
subplot(422); sfdr(y_t);
subplot(424); thd(y_t); thd(y_t,1,25);
subplot(426); sinad(y_t);
subplot(428); snr(y_t);
```

Run *tb_ada* first, then run *tb_characterize* to obtain the graphics shown in Fig. 6.2.6 for the ideal, initial *tb_ada* testbench.

Exercise 1: Change line

```
subplot(424);thd(y_t); thd(y_t,1,16);
```

to

```
subplot(424);thd(y_t); % thd(y_t,1,16);
```

What happens to THD in subplot 4 compared to SFDR in subplot 4? Explain!

In this case -THD=35.81dB > SFDR=29.42dB. This is because THD respects by default the first 6 harmonics only.The SFDR shows the maximum peak at F=400m, i.e. harmonic 21!

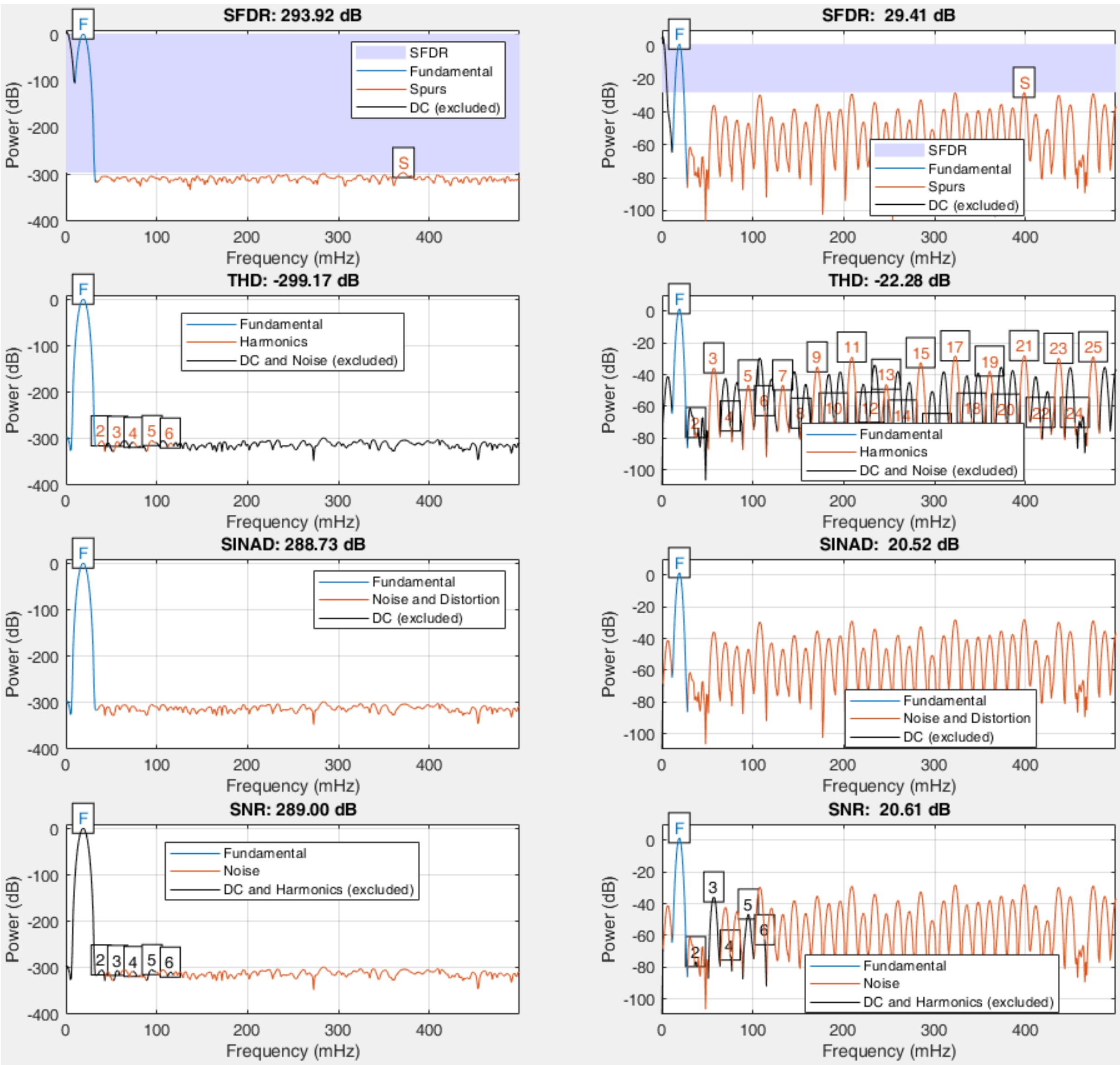


Fig. 6.2.6(a): Computing *SFDR, THD, SINAD, SNR* for the initial testbench *tb_ada*, left-hand side for unquantized (blue) and right-hand side for quantized (black) signal.

Exercise 2: In tb_ada, uncomment line

```
yda_c = [0.0015 0.404 0.803 1.216 1.631 2.013 2.42 2.831 3.242];
```

run *tb_ada* and then run *tb_characterize*. You get the graphics shown below. What happens to *THD* compared to *SFDR* in subplot 3? Explain andcorrect it!

In this case -THD=45.02dB > SFDR = 42.51. This is because THD computation respects by default the first 6 harmonics only, which is good to see at the red color. Change

```
subplot(423);thd(yref_t); % thd(yref_t,1,8);
```

to

```
subplot(423); thd(yref_t,1,8);
```

respecting 8 harmonics yielding THD=-40.28dB.

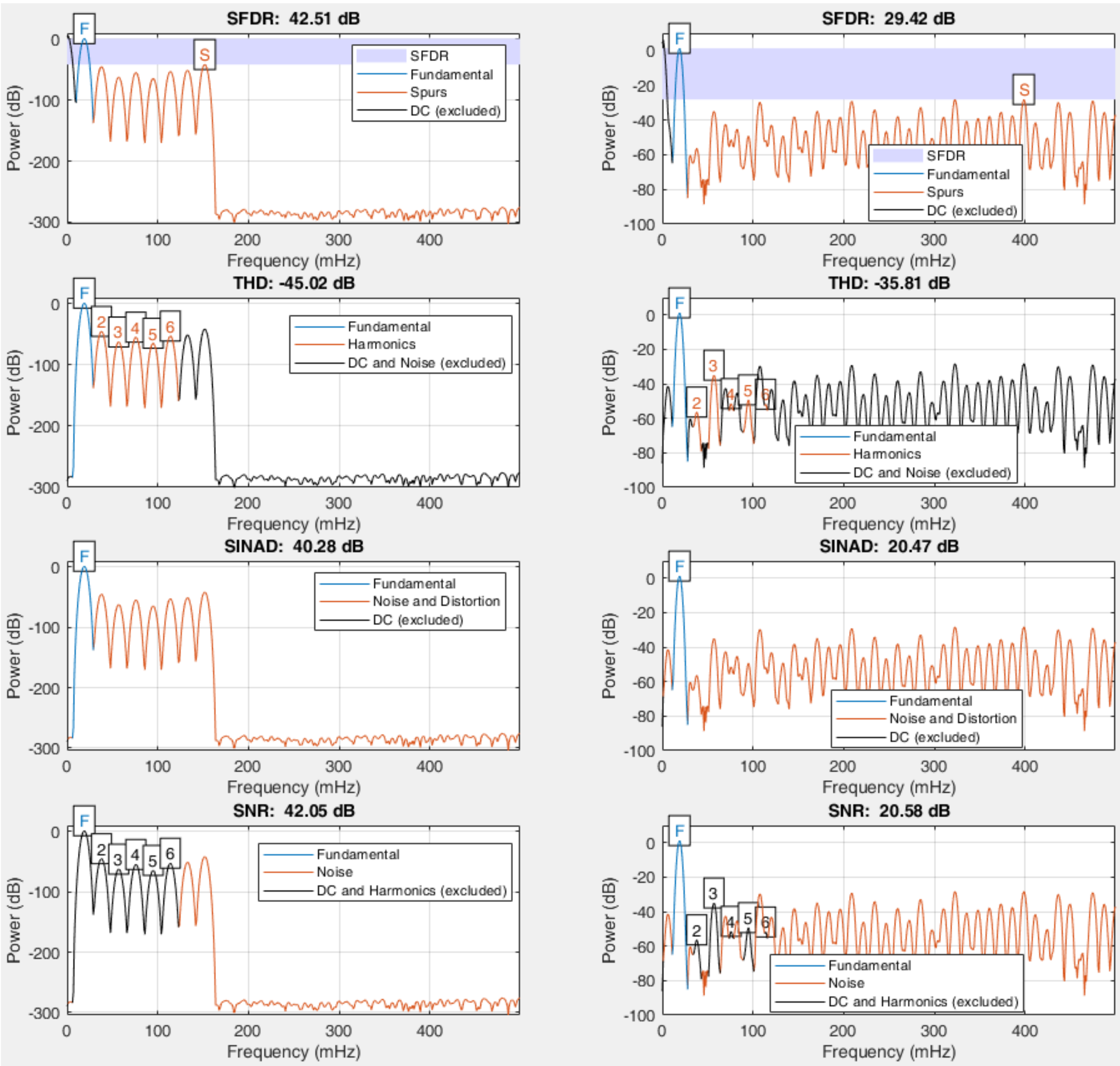


Fig. 6.2.6(b): Computing *SFDR, THD, SINAD, SNR* for the initial testbench *tb_ada*, left-hand side for unquantized (blue) and right-hand side for quantized (black) signal.

6.3 Appendix: Selfmade *Matlab* Functions Used

Listing 6.3.1: Function *f_PolyInit* computes polynomial coefficients interpolating (x_k,y_k)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Purpose: Compute coefs of polynomial y(x) through points (xi,yi)
% Inputs:
%   vx():   x-vector defining polynomial: len(vx)>1 required
%   vy():   y-vector defining polynomial: vy(i) = f(vx(i))
% Outputs:
%   vc():   coefficients: y(x) = vc(i)*x^(i-1)
% Author: Martin Schubert
% Date last modified: 09.Apr.2017 by M. Schubert
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
function vc = f_PolyInit(vx,vy)
NoP = length(vx);
assert(nargin==2,'function requires 2 input vectors');
assert(NoP>1,'input vectors must have at least 2 points');
assert(length(vy)==NoP,'error: vx and vy must have same length');
A = zeros(NoP,NoP);
for row=1:NoP;
  for col=1:NoP;
    A(row,col) = vx(row)^(col-1);
  end;
end;
if size(vy,1)>1;
  vc = A\vy;  % compute coefficients with column vector vy
else
  vc = A\vy'; % the ' brings cy into the upright position
end;
vc=vc';
```

Listing 6.3.2: Function *f_dB* translates an amplification to decibels.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Module:  f_dB
% Purpose: compute amplitude amplification in dB: y_dB = 20*log10(x)
% Inputs:
%   x()  required : |x| is taken as amplitude amplification
%   xmin default 0: x is clipped to |x|>=xmin as log(0)=-infinite
% Outputs:
%   y_dB : 20*log10(max(abs(x),xmin))
% Author: Martin Schubert
% Date last modified: 05.Apr.2017 by M. Schubert
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function y_dB = f_dB(x,xmin);
if nargin==1;
  y_dB = 20*log10(abs(x));
else
  y_dB = 20*log10(max(abs(x),xmin));
end;
```

7 Conclusions

Behavioral models for analog-to-digital and digital-to-analog conversion as well as quantization werde discussed. Non-linear effects like high-order polynomial transfer characteristics and bounding were respected. A *Matlab* model is presented and Matlab exercises are encouraged.

8 References

[1] 1658-2011 - IEEE Standard for Terminology and Test Methods of Digital-to-Analog Converter Devices, available: https://ieeexplore.ieee.org/document/6152113.

9 1241-2010 - EEE Standard for Terminology and Test Methods for Analog-to-Digital Converters, available: https://ieeexplore.ieee.org/document/5692956.

[2] Walt Kester, *Understand SINAD, ENOB, SNR, THD, THD + N, and SFDR so You Don't Get Lost in the Noise Floor*, Analog Devices Tutorial MT003, Available 30.07.2018: http://www.analog.com/media/en/training-seminars/tutorials/MT-003.pdf.

[3] *Matlab*, available Nov. 2017: https://de.mathworks.com/products/matlab.html.

[4] Juan Garcia, Stephen G. LaJeunesse, Douglas Bartow, *Measuring Spurious Free Dynamic Range in a D/A Converter*, Intersil, Technical Brief, TB326, Jan. 1995, Available 30.07.2018: http://www.cse.psu.edu/~chip/course/analog/lecture/SFDR2.pdf.

[5] Wikipedia, *Need for 2 definitions?* and *Definition of the SINAD*, Talk:SINAD, available 30.07.2018: https://en.wikipedia.org/wiki/Talk:SINAD