DAC simulation on Matlab

TELECOM201 - Tutorial lab

Germain Pham - Chadi Jabbour
dpham@telecom-paris.fr

Jan. 2024

Tutorial outline

Previously

Homeworks feedbacks

DAC concept

Matlab framework : Discrete time - Analogue amplitude

Modeling hardware non-idealities

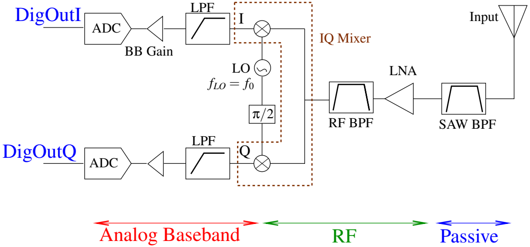
Conclusion

Section outline

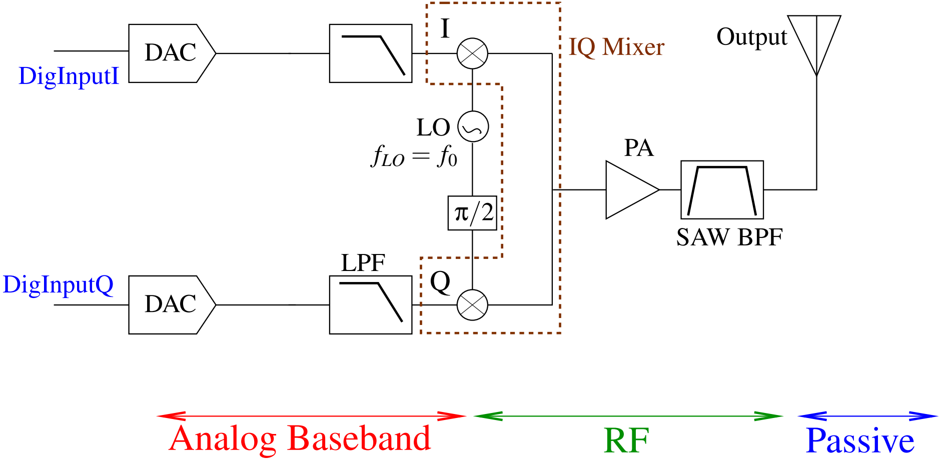
Previously

Previously

- Matlab basics (`plots`, vectors,. . .)
- ADC operations
  - Sampling (inherent)
  - Quantization (`floor`, `fix`, `round`, . . .)
- Spectrum visualization (`fft`, bins, PSD, `fftshift`,. . .)
  - SNR computation



Today


DigInputI
DAC
I
IQ Mixer
Output
LO
$f_{LO} = f_0$
PA
$\pi/2$
SAW BPF
LPF
DigInputQ
DAC
Q
Analog Baseband
RF
Passive

Code snippets

⚠

Pretty display vs raw code

Due to processing for display, the code snippets can not be directly copied and pasted to Matlab terminal. You must copy/paste into a text editor and just remove the extra `\n` before using the code.
Snippets are available at: `https://gitlab.telecom-paris.fr/-/snippets/191`

Section outline

Homeworks feedbacks

Homeworks feedbacks

Warning ⚠

- Homeworks are mandatory !

- Dumb mandatory rules:
  - debug code when theoretical plot does not match empirical plot
    - **it is useless to make the DAC homework if the ADC homework is not working**
  - check code executability before uploading (why not send to friend before?)
  - write a README file when you have more than 3 files
- Advices for future works:
  - generate signals outside from ADC/DAC
  - define a PSD function (and a possibly PSDdB)
  - superimpose plot lines when you compare theoretical with empirical

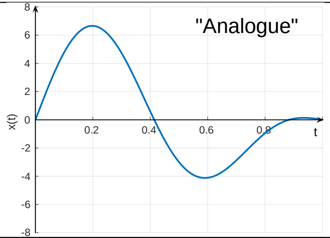
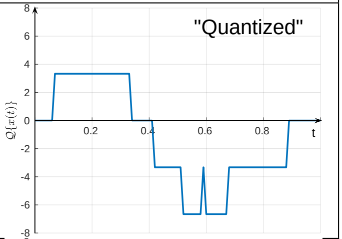
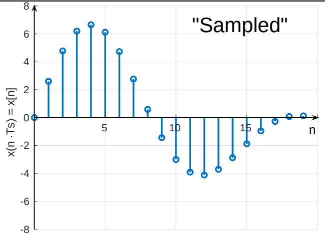
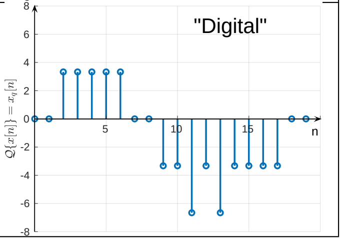
Section outline

DAC concept

- Recall : Amplitude and/or time continuity
- DAC big picture
- DAC waveforms and ideal reconstruction concept

DAC concept

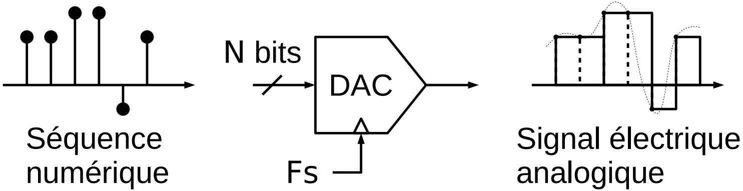
Recall : Amplitude and/or time continuity

		Amplitude	
		Continuous	Discrete
Time characteristics	Continuous		
	Discrete		

DAC concept

DAC big picture

- Ideal/theoretical model



Séquence numérique

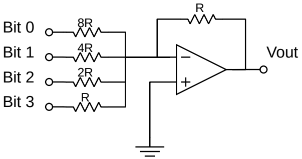
N bits

DAC

Fs

Signal électrique analogique

- A simple implementation example



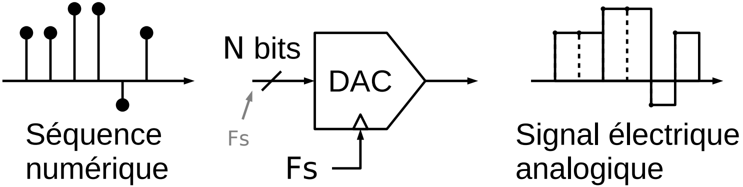
- Ensure quantization
- "**unsampling**" † : make continuous-time
- "unquantization" †
  - not always, depends on what you include in the DAC

†: Barbarism...!

DAC concept

DAC waveforms and ideal reconstruction concept

- Basic DAC model: ZOH(rectangular pulse/boxcar function)



- Signal equations:

$$x[n] \longrightarrow x_D(t) = \sum_n x[n]\delta(t - nT_s) \tag{1}$$

$$y_{ZOH}(t) = \sum_n x[n]h_{ZOH}(t - nT_s) = x_D * h_{ZOH}(t) \tag{2}$$

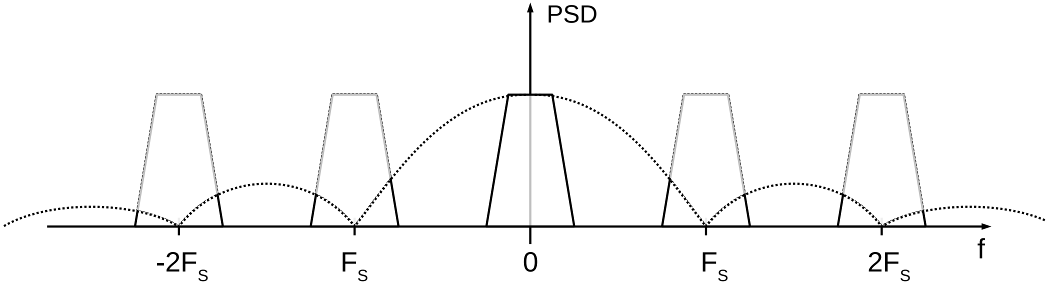
$$\text{with: } h_{ZOH}(t) = \mathrm{rect}(t) \tag{3}$$

DAC waveforms and ideal reconstruction concept

- Basic DAC model: ZOH (rectangular pulse/boxcar function)
- Theoretical spectrum

$$Y_{\text{ZOH}}(f) = X_D(f) \times H_{\text{ZOH}}(f) \tag{4}$$

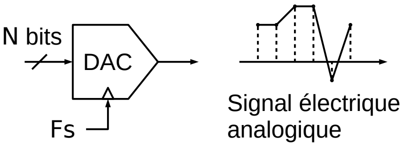
$$\text{with: } \begin{cases} X_D(f) & = \sum_k X(f - kF_s) \\ H_{\text{ZOH}}(f) & = \operatorname{sinc}(\pi f) \end{cases} \tag{5}$$



DAC concept

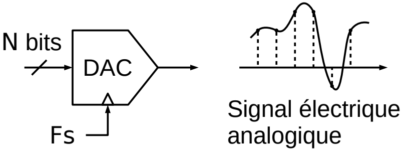
DAC waveforms and ideal reconstruction concept

- Interpolating DACs= **DAC + filter**
  - 1st order interpolation ("FOH" [†])



Signal électrique analogique

  - Ideal interpolator (sinc)



Signal électrique analogique

- Signal equations

$$y_{\mathsf{FOH}}(t) = \sum_n x[n]h_{\mathsf{FOH}}(t - nT_s) \tag{6}$$

$$y_{\mathsf{FOH}}(t) = x_D * h_{\mathsf{FOH}}(t) \tag{7}$$

$$\text{with: } h_{\mathsf{FOH}}(t) = \mathrm{rect} * \mathrm{rect}(t) \tag{8}$$

- Signal equations

$$y_{\mathsf{sinc}}(t) = \sum_n x[n]h_{\mathsf{sinc}}(t - nT_s) \tag{9}$$

$$y_{\mathsf{sinc}}(t) = x_D * h_{\mathsf{sinc}}(t) \tag{10}$$

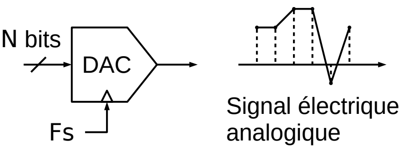
$$\text{with: } h_{\mathsf{sinc}}(t) = \mathrm{sinc}_{T_s}(t) \tag{11}$$

[†]: Barbarism...!

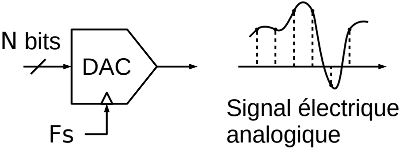
DAC concept

DAC waveforms and ideal reconstruction concept

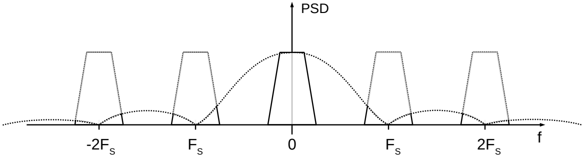
- Interpolating DACs
  - 1st order interpolation ("FOH" [†])



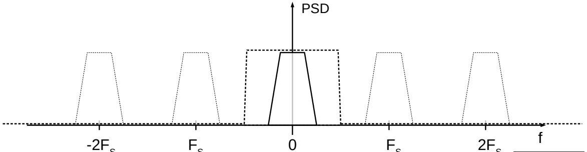
  - Ideal interpolator (sinc)



- Theoretical spectrum



- Theoretical spectrum



[†]: Barbarism...!

Section outline

Matlab framework : Discrete time - Analogue amplitude

- Practical approach to model continuous time signals
- Matlab efficient implementations
- Additional comments
- Spectral analysis

Matlab framework : Discrete time - Analogue amplitude

Practical approach to model continuous time signals

Discrete time - Analogue amplitude

It is impossible to generate continuous-time signal on Matlab. Only discrete-time signals can be produced.

Solution : Huge (over-)sampling

Simulation sampling frequency

Various notations : $F_{S,sim}$, `continuousTimeSamplingRate`, `1/Ts_Cont`,...

Oversampling

Confusion ⚠

It is important not to confuse the notion of oversampling in this case which is just a trick to emulate continuous time operation with the notion of oversampling in analog-to-digital converters which reflects a real hardware choice.

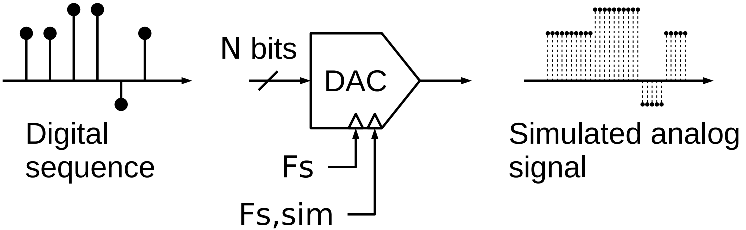
Situations requiring attention

- Be careful with aliasing of high order harmonics
- Works only for smooth signals (no PWM-like signals)

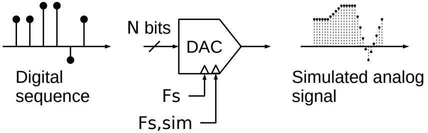
Matlab framework : Discrete time - Analogue amplitude

Practical approach to model continuous time signals

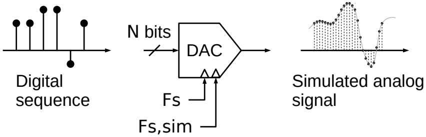
- ZOH DAC



- FOH DAC



- Ideal sinc DAC



Matlab efficient implementations

Please note

- Implementation variability
  - There is no unique way to implement each DAC waveforms on Matlab.
- For sake of simplicity, we ignore quantization here

Matlab efficient implementations

- ZOH

Using kron

```
% Time parameters
Ts    = 0.1; OSR = 5;
t     = 0:Ts:3; t=t(:);
% DAC input
x     = cos(2*pi*t);
% DAC output
y_ZOH = kron(x,ones(OSR,1));
% (Caution: y_ZOH is longer than t_OSR...!)
% New time parameters
Tsim  = Ts/OSR;
t_OSR = t(1):Tsim:t(end); t_OSR=t_OSR(:);
% Plots
stem(t_OSR,y_ZOH(1:length(t_OSR))) ; hold on
stem(t,x,'k','linewidth',2)
```

Using filter

```
% Upsample (zero padding)
y_upsample = zeros(length(x)*OSR,1);
y_upsample(1:OSR:end) = x;
% (Caution: y_upsample is longer than t_OSR)

% DAC response
h_ZOH = ones(OSR,1);
% Filter
y_ZOH_bis = filter(h_ZOH,1,y_upsample);

% Plots
stem(t_OSR,y_ZOH_bis(1:length(t_OSR)))
hold on
stem(t,x,'k','linewidth',2)
```

You can also use `repmat` (but you have to handle a `reshape`); and also `interp1`

Matlab efficient implementations

- FOH

Using filter

```
% Upsample (zero padding)
y_upsample = zeros(length(x)*OSR,1);
y_upsample(1:OSR:end) = x;
% (Caution: y_upsample is longer than t_OSR)

% DAC response
h_FOH = conv(ones(OSR,1),ones(OSR,1));
h_FOH = h_FOH/max(abs(h_FOH));
% Filter
y_FOH = filter(h_FOH,1,y_upsample);

% Plots
figure
% (Caution, there is a transient !)
stem(t_OSR,y_FOH(OSR:end)) ; hold on
stem(t,x,'k','linewidth',2)
```

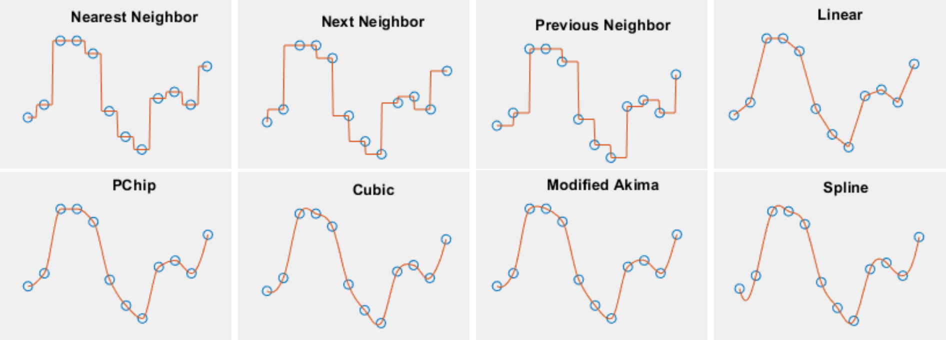
Using interp1

```
y_FOH_bis = interp1(t,x,t_OSR);

% Plots
figure
stem(t_OSR,y_FOH_bis)
hold on
stem(t,x,'k','linewidth',2)
```

You can also use `filtfilt` (but you have to use `h_ZOH`)

Additional comments

- `interp1` can be used for many different interpolation methods



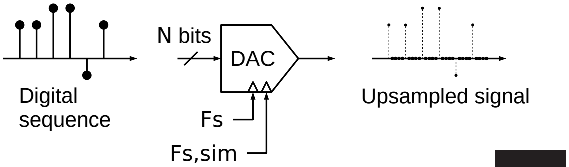
- 1-D data interpolation (table lookup) - MATLAB interp1 - MathWorks
- Interpolating Gridded Data - MATLAB - MathWorks

Additional comments

- Implementing the ideal (low-pass) `sinc` interpolator is impossible.
  - Only time-limited versions can be realized on Matlab
- Sometimes, interpolation behaves badly from the spectral point of view (particularly for sine waveforms)
  - It is then preferable to regenerate the samples from the ideal sinewave if possible.
- We can implement with Matlab an *impulse* DAC (or *zero-padding* DAC, or *simple upsampling*)

Cf. Slides 21 and 22

```
% Upsample (zero padding)
y_upsample = zeros(length(x)*OSR,1);
y_upsample(1:OSR:end) = x;
```



Spectral analysis

Multi-rate signals ⚠

The rate of the DAC input sequence is different from the output rate. Display frequency vectors must be appropriately computed.

Raw FFT

```
% Simulation parameters
fsig     = 8.5e6;
Fs       = 30.72e6;
Fs_sim   = 16*Fs;
Tstop    = 5e-6;
t_in     = 0:1/Fs:Tstop; t_in = t_in(:);
t_sim    = 0:1/Fs_sim:t_in(end);
t_sim    = t_sim(:);
% Signal generation
x        = sin(2*pi*fsig*t_in);
Xlen     = length(x);
% Spectrum computation
X        = fft(x.*blackman(Xlen));
PSDx     = abs(X).^2;
```

```
% DAC processing
y        = interp1(t_in,x,t_sim);  Ylen = length(y);
% Spectrum computation
Y        = fft(y.*blackman(Ylen)); PSDy = abs(Y).^2;
% Display frequency vectors
freq_disp_in  = (0:(Xlen-1))/Xlen*Fs/1e6;
freq_disp_out = (0:(Ylen-1))/Ylen*Fs_sim/1e6;
% Plots
plot(freq_disp_in,10*log10(PSDx));xlim([0 Fs/2]/1e6)
xlabel('Freq (MHz)');ylabel('PSD (dB)')
figure
plot(freq_disp_out,10*log10(PSDy))
xlim([0 Fs_sim/2]/1e6)
xlabel('Freq (MHz)');ylabel('PSD (dB)')
```

Spectral analysis

FFT bins

- Bins must be computed for each sampling rate.
- It is better to place signal on a bin for spectral visualization.

Bin computation

```
% Signal bin for the FFT of the input
sig_bin_in = fix(fsig/Fs*Xlen)+1;
sig_bins_in = sig_bin_in + [-2:2]

% Visual check
plot(10*log10(PSDx))
xlim(sig_bin_in + [-10 10])
xticks(sig_bin_in + [-10:10])
grid on
```

```
% Signal bin for the FFT of the output
sig_bin_out = fix(fsig/Fs_sim*Ylen)+1;
sig_bins_out = sig_bin_out + [-2:2]

% Visual check
plot(10*log10(PSDy))
xlim(sig_bin_out + [-10 10])
xticks(sig_bin_out + [-10:10])
grid on
```

Section outline

Modeling hardware non-idealities

- Noise
- Nonlinear distortions

Noise

- Gaussian noise

Implementation of Gaussian noise

```
Nsamples = 1e4; Ndraws = 15;
noise_g  = 0.4*randn(Nsamples,Ndraws)+1;

% Visualization
plot(noise_g)

% Empirical distribution
[emp_freq,bin_centr]  = hist(noise_g,100);
% Averaged histogram
avr_emp_freq          = mean(emp_freq,2);

% Plot
figure
bar(bin_centr,avr_emp_freq)
xlabel('Value')
ylabel('Empirical frequency')
```

- Uniform noise

Implementation of Uniform noise

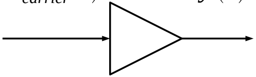
```
Nsamples = 1e4; Ndraws = 15;
noise_u  = 0.4*rand(Nsamples,Ndraws)+1;

% Visualization
plot(noise_u)

% Empirical distribution
[emp_freq,bin_centr]  = hist(noise_u,100);
% Averaged histogram
avr_emp_freq          = mean(emp_freq,2);

% Plot
figure
bar(bin_centr,avr_emp_freq)
xlabel('Value') ; xlim(1.2+0.6*[-1 1])
ylabel('Empirical frequency')
```

Nonlinear distortions

- Let’s consider the following system:

$$x(t)=\cos(\omega_{carrier}\cdot t)$$

$$y(t)=x(t)-\frac{1}{3}x^{3}(t)+\frac{1}{5}x^{5}(t)$$



With

- $F_{carrier} \approx 1.3\,\text{GHz}$
- $T_{Len,sim} \approx 50 \times T_{carrier}$
- $F_{S,sim} = 15\,\text{GHz}$

Implementation and bins computation

Illustration code

```
Fcarr_orig     = 1.3e9;
Tlensim_orig   = 50*1/Fcarr_orig;
FSsim          = 15e9;

% Compute final values
Tlensim = round(Tlensim_orig*FSsim)/FSsim;
Nlensim = Tlensim*FSsim;
Fcarr   = round(Fcarr_orig/FSsim*Nlensim)/Nlensim*FSsim;

t = 0:1/FSsim:(Nlensim-1)/FSsim; t = t(:);
x = cos(2*pi*Fcarr*t);
y = x -1/3*x.^3 + 1/5*x.^5;

Ypsd                = abs(fft(y)).^2; % WINDOWING IS USELESS
bin_freq_val_shift = -(length(Ypsd)-1)/2:(length(Ypsd)-1)/2;
freq_val_shift     = bin_freq_val_shift/length(Ypsd)*FSsim;

plot(freq_val_shift/1e6,fftshift(10*log10(Ypsd)))
xlim([0 FSsim/2]/1e6)
xlabel('Freq (MHz)') ; ylabel('PSD (dB)')
```

```
% Central bins computations
FS_N        = FSsim/Nlensim
bin_fund    = fix(Fcarr/FS_N)+1
bin_harm3 = fix(3*Fcarr/FS_N)+1
bin_harm5 = fix(5*Fcarr/FS_N)+1

% Visual check
plot(10*log10(Ypsd))
```

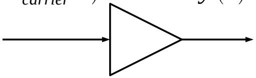
Try by yourself

Five minute trial:
Single-tone in nonlinear system

Plot the output spectrum of the nonlinear system of Slide 29/30 **with windowing**.

$$x(t)=\cos(\omega_{carrier}\cdot t)$$

$$y(t)=x(t)-\frac{1}{3}x^{3}(t)+\frac{1}{5}x^{5}(t)$$



Solution

Illustration code

```
Fcarr_orig     = 1.3e9;
Tlensim_orig  = 50*1/Fcarr_orig;
FSsim          = 15e9;

% Compute final values
Tlensim = round(Tlensim_orig*FSsim)/FSsim;
Nlensim = Tlensim*FSsim;
Fcarr   = round(Fcarr_orig/FSsim*Nlensim)/Nlensim*FSsim;

t = 0:1/FSsim:(Nlensim-1)/FSsim; t = t(:);
x = cos(2*pi*Fcarr*t);
y = x -1/3*x.^3 + 1/5*x.^5;

Ypsd                = abs(fft(y.*blackman(Nlensim))).^2;
bin_freq_val_shift = -(length(Ypsd)-1)/2:(length(Ypsd)-1)/2;
freq_val_shift     = bin_freq_val_shift/length(Ypsd)*FSsim;

plot(freq_val_shift/1e6,fftshift(10*log10(Ypsd)))
xlim([0 FSsim/2]/1e6)
xlabel('Freq (MHz)') ; ylabel('PSD (dB)')
```

```
% Central bins computations
FS_N        = FSsim/Nlensim
bin_fund   = fix(Fcarr/FS_N)+1
bin_harm3 = fix(3*Fcarr/FS_N)+1
bin_harm5 = fix(5*Fcarr/FS_N)+1

% Bins
bins_fund   = bin_fund  + [-2:2]
bins_harm3 = bin_harm3 + [-2:2]
bins_harm5 = bin_harm5 + [-2:2]

% Visual check
plot(10*log10(Ypsd))
```

Practical issue: simulation sampling and nonlinear systems

In order to analyze a nonlinear system modeled as a

- 5th order polynomial, you should sample your system at $F_{S,sim} \gg 2 \times \ 5 \times F_{carrier}$
- 7th order polynomial, you should sample your system at $F_{S,sim} \gg 2 \times \ 7 \times F_{carrier}$
- $\ldots$

Section outline

Conclusion

Conclusion

- DAC simulation
  - (Quantization)
  - Upsampling
  - Interpolation (filtering)
- Homework :
  - Homeworks text on (C2S) TELECOM201b website
    - ⚠: please fix ADC code and implement DAC.
    - Deadline : 19th Jan. 2024