



DAC simulation on Matlab/Octave

TELECOM201 - Tutorial lab

Germain Pham - Chadi Jabbour

dpham@telecom-paris.fr

Nov. 2024



Tutorial outline

Previously

Homeworks feedbacks

DAC concept

Matlab framework : Discrete time - Analogue amplitude

Modeling hardware non-idealities

Conclusion

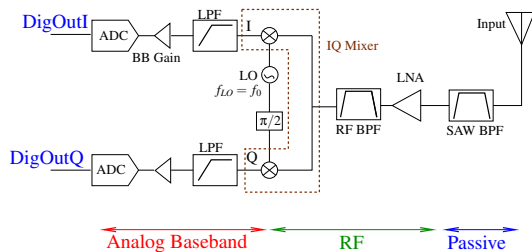


Section outline

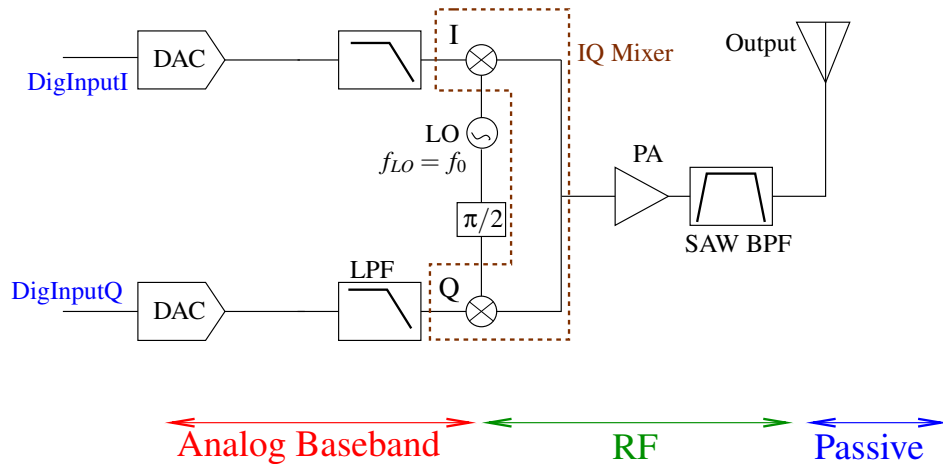
Previously

Previously

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



Today



Code snippets



Pretty display vs raw code

Due to processing for display, the code snippets can not be directly copied and pasted to Matlab/Octave 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

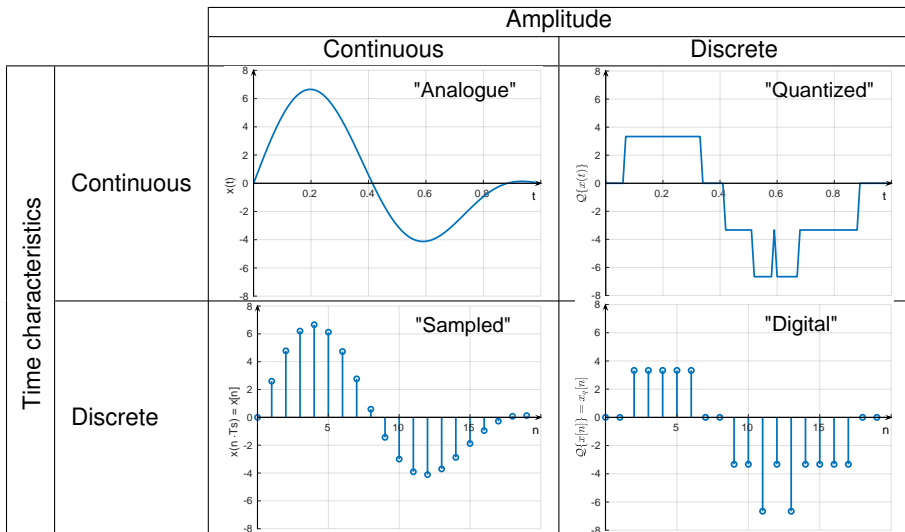
- Recall : Amplitude and/or time continuity

- DAC big picture

- DAC waveforms and ideal reconstruction concept

DAC concept

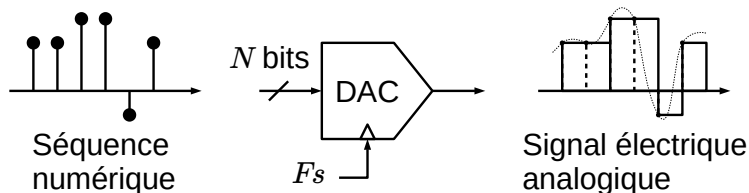
Recall : Amplitude and/or time continuity



DAC concept

DAC big picture

■ Ideal/theoretical model



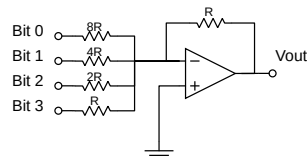
■ Time:

- Input: discrete-time signal
- Output: continuous-time signal

■ Value:

- Input: quantized over N bits
- Output: can be continuous or quantized (depends on the DAC type)

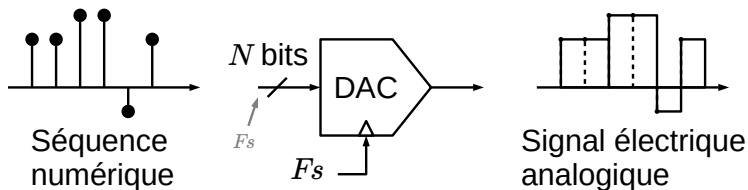
■ A simple implementation example



DAC concept

DAC waveforms and ideal reconstruction concept

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



- Signal equations:

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

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

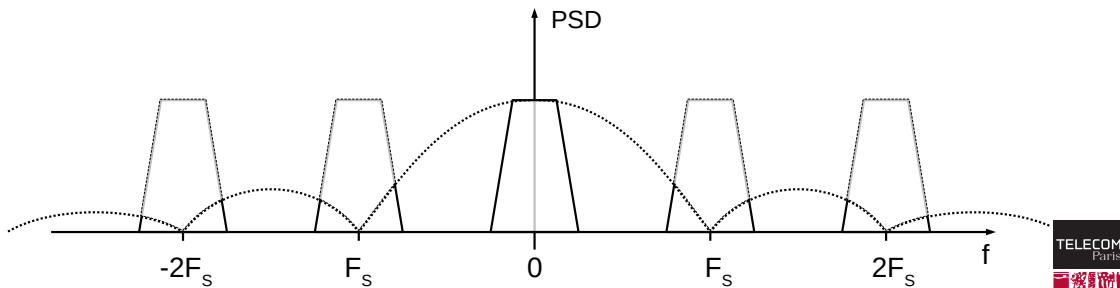
$$\text{with: } h_{\text{ZOH}}(t) = \text{rect}(t)$$

DAC waveforms and ideal reconstruction concept

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

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

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

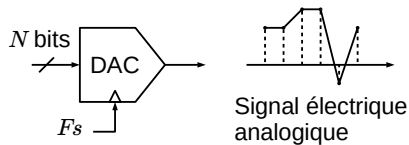


DAC concept

DAC waveforms and ideal reconstruction concept

■ Interpolating DACs= **DAC + filter**

- 1st order interpolation ("FOH" †)



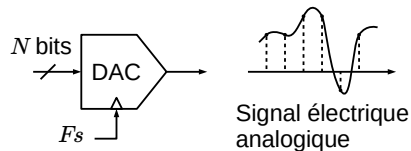
■ Signal equations

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

$$y_{\text{FOH}}(t) = x_D * h_{\text{FOH}}(t) \quad (7)$$

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

- Ideal interpolator (sinc)



■ Signal equations

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

$$y_{\text{sinc}}(t) = x_D * h_{\text{sinc}}(t) \quad (10)$$

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

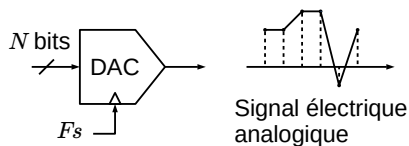
†: Barbarism...!

DAC concept

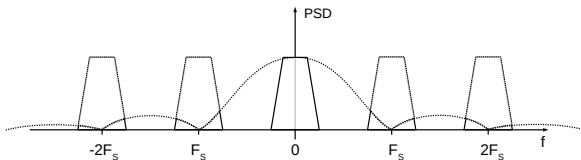
DAC waveforms and ideal reconstruction concept

■ Interpolating DACs

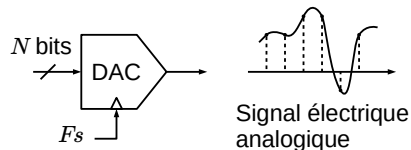
- 1st order interpolation ("FOH" †)



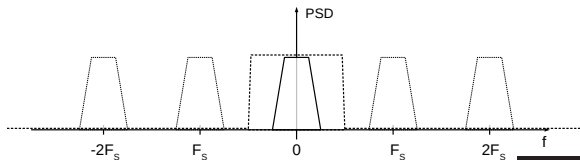
■ Theoretical spectrum



- Ideal interpolator (sinc)



■ 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/Octave. 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.

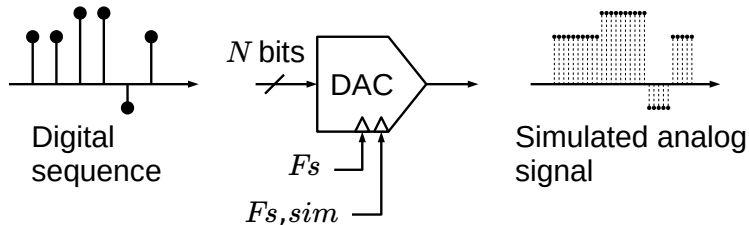
Situations requiring attention

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

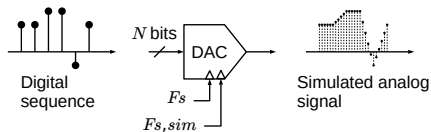
Matlab framework : Discrete time - Analogue amplitude

Practical approach to model continuous time signals

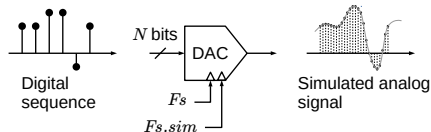
ZOH DAC



FOH DAC



Ideal sinc DAC



Matlab efficient implementations

Please note

- Implementation variability
 - There is no unique way to implement each DAC waveforms on Matlab/Octave.
- 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

■ ZOH

- This implementation is used in TELECOM205 project
 - Compensates for the transient

Using resample

```
% DAC response
h_ZOH = ones(OSR,1);

% Filter
y_ZOH_bis = resample(x,OSR,1,h_ZOH);

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

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)
```

Matlab efficient implementations

■ FOH

Using filtfilt

```
% DAC response
h_ZOH = ones(OSR,1)/sqrt(sum(ones(OSR,1)));
% Filter
y_FOH_ter = filtfilt(h_ZOH,1,y_upsample);

% Plots
figure
% (No delay but transient is ugly !)
stem(t_OSr,y_FOH_ter(1:length(t_OSr))) ;
    hold on
stem(t,x,'k','linewidth',2)
```

Using resample

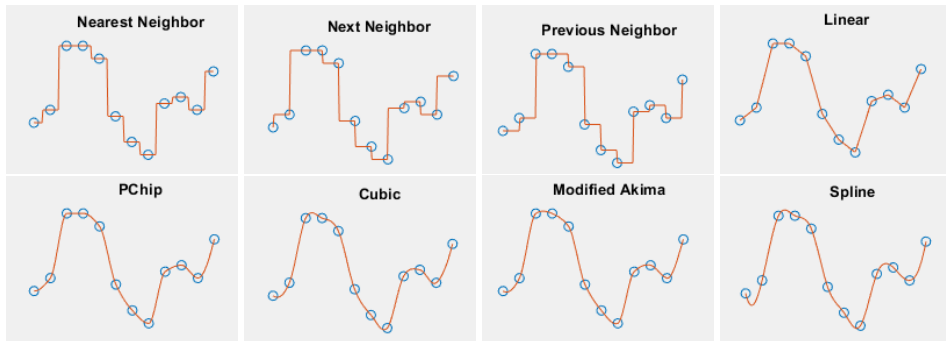
```
y_FOH_quad = resample(x,OSR,1,h_FOH);

% no transient issue, perfect alignment !

% Plots
figure
stem(t_OSr,y_FOH_quad(1:length(t_OSr)))
hold on
stem(t,x,'k','linewidth',2)
```


Additional comments

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



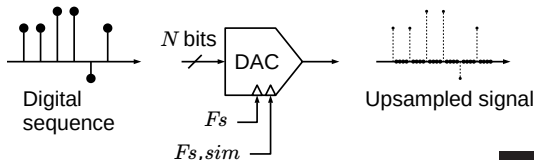
- 1-D data interpolation (table lookup) - MATLAB `interp1` - MathWorks
- Interpolating Gridded Data - MATLAB - MathWorks

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 23

```
% 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_cent] = hist(noise_g,100);
% Averaged histogram
avr_emp_freq = mean(emp_freq,2);

% Plot
figure
bar(bin_cent,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;

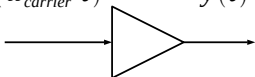
% Visualization
plot(noise_u)

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

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

Nonlinear distortions

- Let's consider the following system:

$$x(t) = \cos(\omega_{\text{carrier}} \cdot t) \quad \longrightarrow \quad y(t) = x(t) - \frac{1}{3}x^3(t) + \frac{1}{5}x^5(t)$$


With

- $F_{\text{carrier}} \approx 1.3 \text{ GHz}$

- $T_{\text{Len},\text{sim}} \approx 50 \times T_{\text{carrier}}$

- $F_{S,\text{sim}} = 15 \text{ GHz}$

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)')
```

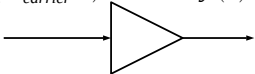
```
% Central bins computations
FS_N      = FSsim/Nlensim
bin_fund   = fix(Fcarr/FS_N)+1
bin_harm3  = fix(3*Fcarr/FS_N)+1
bin_harm5  = fix(5*Fcarr/FS_N)+1

% 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 31/32 **with windowing**.

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

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

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
 $F_{S,sim} \gg 2 \times 7 \times F_{carrier}$
- ...



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 : Fri. 17th Jan. 2025