ADC simulation on Matlab/Octave

TELECOM201 - Tutorial lab

Germain Pham - Chadi Jabbour
dpham@telecom-paris.fr
Nov. 2024

Tutorial outline

Section outline

Why Matlab

- Main features
- Other softwares

Why Matlab

Main features

- *Built-in* interactive prompt
- *Built-in* interactive vizualization
- Easy to debug
- Fast computation with fast development cycle
- Everything can be easily done programmatically (no need for GUI actions) = fast + *reproducible*

Simulink will not be covered here

Simulink uses a different modeling paradigm :

- *kinda* Object-Oriented
- GUI environment : block diagrams
- hybrid-time systems (partly discrete & continuous) are actually discrete-time
- Adresses a different work stage

Other softwares

What about Python ?

- Completely equivalent to some extent.
- Requires additionnal software interfaces for interactive prompt and vizualization.

What about SystemVue, ADS, Cadence softwares ?

- GUI oriented
- More difficult to acquire on your personal computers
- Very powerful features for hardware analysis (*only* valuable for latter design phase)

How to use softwares at school from your home
without installing it on your personal computer

Remote Desktop

Guide Télécomien-ne à distance | Eole : La DSI a rendu possible la connexion à des machines de salles de TP (page EOLE condition d'utilisation).
Vous trouverez sur ce lien[1], `https://supervision.enst.fr/tp/`, un état indiquant, en temps réel

- L'état de chaque machine
- Le nombre de sessions RDP, SSH et X de chaque machine

Cet état est accessible bien sûr depuis le réseau de l'école, mais surtout **en VPN**.
Télécharger documentation-rdp_0.pdf.

- Windows et Linux : cf. documentation-rdp_0.pdf
- MacOS : Microsoft Remote Desktop

[1] ou bien `https://tp.telecom-paris.fr/`

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. All the snippets are available in `ADC_DM_tuto.m` in the following archive : TELECOM201>Labs and Tutorials>Tutorial : ADC Simulation (Slides + Code)

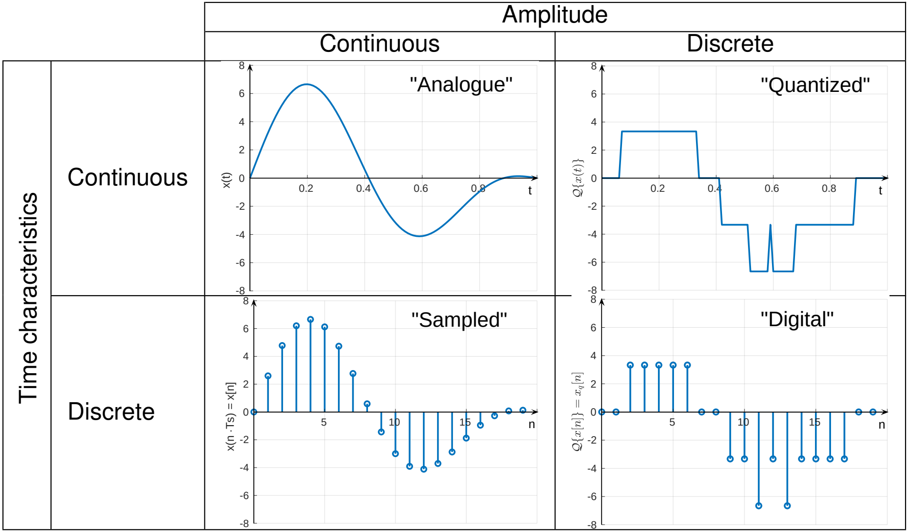
Section outline

Signal types

- Amplitude and/or time continuity
- Matlab framework : Discrete time - Analogue amplitude
- Signal transformation

Signal types

Amplitude and/or time continuity



Signal types

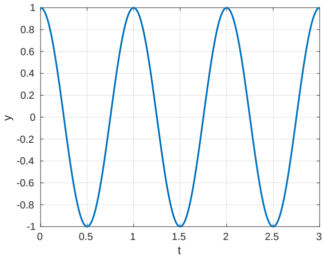
Matlab framework : Discrete time - Analogue amplitude

Discrete time - Analogue amplitude

It is impossible to generate continuous-time signal on Matlab/Octave [2].

Sinus generation

```
t_sim = 0:0.01:10  ;      % Time is inherently sampled
y     = cos(2*pi*t_sim);  % Cosinus is also sampled
plot(t_sim,y)
```



However, the signal appears continuous on the plot !

[2] Formal waveforms maybe considered with `symbolic` processing but this approach is very restrictive

Appropriate plot function to visualize sampled nature

`stem(x,y)`

If we want to show the discrete time nature of a signal it is best to use `stem` or additionnal plot parameters.

Sinus generation - discrete

```
t_sim = 0:0.1:3;
y     = cos(2*pi*t_sim);
% Prepare figure with two plots
subplot(211)
% Use stem to display the sampled sequence
stem(t_sim,y,'linewidth',2)
xlabel('time')
ylabel('y')
```

```
% Enable second plot
subplot(212)
% Use plot to display the sampled sequence
plot(t_sim,y,':o')
xlabel('time')
ylabel('y')
```

Try by yourself

One minute trial

Plot the cosine function of Slide 10 using `stem` and `plot` and smaller sampling steps (e.g. `0.01`) ; comment on the readability of the result.

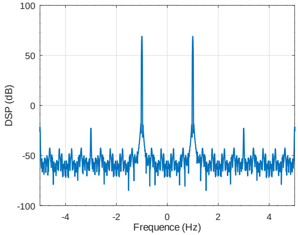
Signal transformation

Quantization

Quantization code

```
t_sim = 0:0.01:2;
Vref  = 1.5; Nbits = 4;
x     = 1.35*cos(2*pi*1*t_sim);
quantizedInput = floor((x+Vref)*2^(Nbits-1)/Vref);     % Quantizing the sampled data
quantizedInput(quantizedInput<0)         = 0;          % Clipping Down
quantizedInput(quantizedInput>2^Nbits-1) = 2^Nbits-1;  % Clipping Up
DigOutput = (quantizedInput-2^(Nbits-1))/2^(Nbits-1)*Vref+Vref/2^Nbits;
stem(t_sim,DigOutput) ; xlabel('Temps (s)') ; ylabel('Sortie Quantifiee')
```

Resulting spectrum :



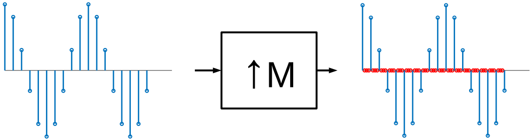
Signal transformation

Raw (Re)Sampling : integer factor

Signal example

```
fs   = 10;    tstop = 1.75;    t = 0:1/fs:tstop;    f = 1;   y = cos(2*pi*f*t);
```

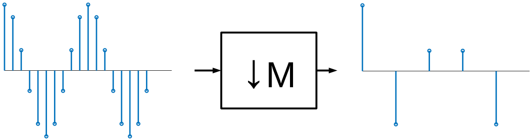
Up-sampling : zero insertion



Code

```
USR          = 4; % upsampling ratio
fs_up        = fs*USR;
y_up         = zeros(1,(length(y)-1)*USR+1);
y_up(1:USR:end) = y;
t_up         = 0:1/fs_up:t(end);
```

Down-sampling (a.k.a. decimation)



Code

```
DSR          = 4; % downsampling ratio
fs_down      = fs/DSR;
y_down       = y(1:DSR:end);
t_down       = 0:1/fs_down:tstop;
```

Section outline

Spectral analysis

- The different Fourier transforms
- Matlab framework

Spectral analysis

The different Fourier transforms

		Nature of the exponential variable	
		$p = j\omega$	$p = \sigma + j\omega$
Time characteristics	Continuous	Fourier Transf. $\mathcal{F}(\omega) = \int f(t)\,\mathrm{e}^{-j\omega t}\,\mathrm{d}t$	Laplace Transf. $\mathrm{F}(p) = \int f(t)\,\mathrm{e}^{-pt}\,\mathrm{d}t$
	Discrete $f(t) \cdot \sum_n \delta(t - nT)$	DT Fourier Transf. $\mathcal{F}_{TD}(\nu) = \sum^N f[n]\,\mathrm{e}^{-j2\pi n\nu}$ $\downarrow$ **Discrete Fourier Transf.** $\mathcal{F}_D[k] = \sum_{n=0}^{N-1} f[n]\,\mathrm{e}^{-j2\pi nk/N}$	Z-Transf. $\mathrm{F}_Z(z) = \sum^\infty f[n] z^{-n}$

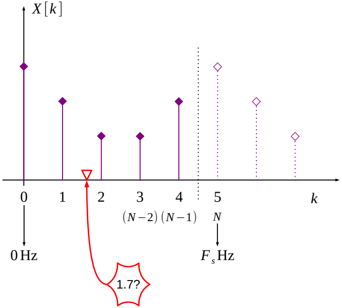
Spectral analysis
Matlab framework: DFT only

Recall

Matlab uses only discrete sequences

Practical consequences:

- the spectrum is a (frequency) sampled sequence. Its most accurate representation is by `stem(...)` .
- the FFT *bin* concept.



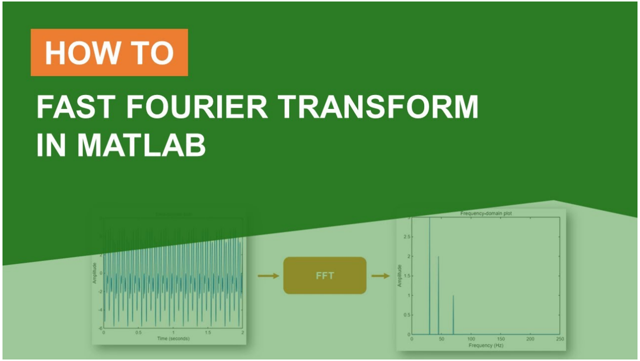
Discrete spectrum signals !

Code for multi-tone signals must be adequately designed !!!

- Spectral leakage

A tutorial video for the FFT in Matlab

Please watch this video to get a quick understanding of the FFT in Matlab :
Youtube MATLAB Channel : How to Do FFT in MATLAB



Spectral analysis

Matlab framework: spectrum plot

Random signal spectrum

```
Fs           = 1;
x            = rand(51,1) + 1i*rand(51,1); % Complex signal
Xpsd         = abs(fft(x)).^2;             % Note the dot ! Note the square !
Nx           = length(Xpsd);               % length of the FFT, also length of x
bin_freq_val = [0:Nx-1];
subplot(2,1,1); stem(0:Nx-1,real(x))
xlabel('Time index');    ylabel('Magnitude')
subplot(2,1,2); plot(bin_freq_val,10*log10(Xpsd)) % Note the 10xlog10 !
xlabel('Frequency bin'); ylabel('Power spectral density (dB)')
```

Visualization improvements : DC centering, frequency values

```
bin_freq_val_shift = -(Nx-1)/2 : (Nx-1)/2;
freq_val_shift     = bin_freq_val_shift/Nx*Fs;
plot(freq_val_shift,fftshift(10*log10(Xpsd))) ; xlim(0.5*[-1 1])
xlabel('Frequency (Hz)'); ylabel('Power spectral density (dB)')
```

Try by yourself

Five minute trial: Single-tone visualization

Plot the spectrum of a simple sinus wave:

$$x(t) = \cos(\omega_{carrier} \cdot t) \quad (1)$$

with

- $F_{carrier} = 1.3\,\text{GHz}$
- $T_{Len,sim} = 50 \times T_{carrier}$
- $F_{S,sim} = 7\,\text{GHz}$

Spectral analysis

Matlab framework: Frequency planning and windowing

Frequency planning: (**for tones only**)

- Set the frequency of your tone so that it is exactly a bin frequency.

$$bin_{sig} = \left\lfloor \frac{f_{sig}}{F_S} \times N \right\rfloor \qquad (2)$$

Code snippet

```
fsig_bin = round(fsig/Fs*Nsim)/Nsim*Fs;
x        = sin(2*pi*fsig_bin*t_sim);
```

Windowing: (**for all cases**)

- Use a (time-domain) windowing function
  - hann
  - hamming
  - blackman
  - ...

Code snippet

```
x          = sin(2*pi*fsig_random*t_sim);
win        = blackman(length(x),'periodic');
x_windowed = x(:).*win(:); % Time domain mult
Xpsd       = abs(fft(x_windowed)).^2;
```

https://en.wikipedia.org/wiki/Window_function

Try by yourself

Same as Slide 20

Five minute trial: Single-tone visualization

Plot the spectrum of a simple sinus wave, **set the frequency to be in an FFT bin**:

$$x(t) = \cos(\omega_{carrier} \cdot t) \tag{3}$$

with

- $F_{carrier} \approx 1.3\,\text{GHz}$
- $F_{S,sim} = 7\,\text{GHz}$
- $T_{Len,sim} \approx 50 \times T_{carrier}$

Section outline

Power analysis

- (Useful) Signal power analysis
- Error/noise/distorsion power analysis

Power analysis

(Useful) Signal power analysis

Time domain

- Instantaneous power: $|x|^2[n]$

Code snippet

```
sig_pow_inst = abs(x).^2
```

- Average power: $\frac{1}{N}\sum^{N}|x|^2[n]$

Code snippet

```
sig_pow_avg = mean(sig_pow_inst)
```

Spectral approach: Parseval

$$\sum_{n=0}^{N-1}|x[n]|^2 = \frac{1}{N}\sum_{k=0}^{N-1}|X[k]|^2 \quad (4)$$

Code snippet

```
x = sin(2*pi*fsig_random*t_sim);
X = fft(x)/sqrt(length(x));
sig_pow_avg = mean(abs(X).^2)
```

⚠Warnings

- Be careful about windowing !
- Absolute power measurements require careful implementation in Matlab

Power analysis

Error/noise/distorsion power analysis

Time domain

$$e[n] = x_{actual}[n] - x_{ideal}[n] \qquad (5)$$

Code snippet

```
x     = sin(2*pi*fsig*t_sim);
x_noi = x+0.2*randn(size(t_sim));
y     = 1.5*x_noi -0.3*x_noi.^3;
error = y - 1.5*x;
```

⚠Warnings

- Delays (filters)
- Scaling (gains/nonlinearities)

Spectral domain (windowing!)

Code snippet

```
Ny   = length(y);
win  = blackman(Ny,'periodic');
yPSD = abs(fft(y(:).*win(:))).^2;
yPSD = yPSD/Ny; % Parseval
sig_bin     = fix(fsig/FSsim * Ny)+1;
% +1 is due to Matlab array indexing style
sig_bin_win = sig_bin + [-2:2];
err_bin     = setdiff(1:round(Ny/2),sig_bin_win);
```

More details

Matlab Doc: Measure Power of Deterministic Periodic Signals

Section outline

Common practices for systems modelling and signal analysis in Matlab

- Useful Commands in Matlab
- Matlab Pro tips
- Vectorization
- Randomization
- Resampling
- Delay compensation

Common practices for systems modelling and signal analysis in Matlab

Useful Commands in Matlab

- `size(x)` to obtain the dimensions `(n;m)` a matrix `x`
- `length(x)` to obtain the length of the longest dimension of a matrix `x`
- `[maxvalue,maxIndex]=max(x)` gives the maximum value of `x` and its index
- `min(x)`, `mean(x)`, `max(x)` and `rms(x)` to obtain the minimum, average, max and rms value of `x`
- `x(x<0)=0` sets all negative terms of `x` to `0`
- `x(10:100)` is a *slice* of `x` from the 10th to 100th position
- `sum(x(10:100))` to sum the elements of `x` from the 10th to 100th position
- `plot(x)` to plot `x` in *time domain*
- `plot(20*log10(abs(fft(x))))`, to plot `x` in *frequency domain*
- `hist(x)` to plot the histogram of `x`
- `finddelay(x,y)` to find the (integer) delay between `x` and `y`, very useful for synchronisation

Place your favorite script/functions in the default MATLAB userpath folder:

- Windows® platforms — `%USERPROFILE\%/Documents/MATLAB`.
- Mac platforms — `$HOME/Documents/MATLAB`.
- Linux® platforms — `$HOME/Documents/MATLAB`.

Organize your MATLAB folder and use a MATLAB `startup.m` file :

```
germain@tp:~/Documents/MATLAB$ tree -L 1
.
|-- ccc.m
|-- Examples
|-- FileExchange
|-- meanErr.m
|-- meanErrMat.m
|-- meanSqErr.m
|-- meanSqErrMat.m
|-- saveaspdfcrop.m
|-- Spectral-Analysis
|-- startup.m
|-- SupportPackages
`-- upsample_zoh_foh.m
```

startup.m

```
set(groot,'DefaultLineLinewidth',2)
set(groot,'DefaultAxesFontSize',12)
set(groot,'DefaultAxesXGrid','on')
set(groot,'DefaultAxesYGrid','on')
addpath(genpath('/home/germain/Documents/MATLAB/'))
```

`startup.m` documentations:

- User-defined startup script for MATLAB
- GNU Octave: Startup Files

Common practices for systems modelling and signal analysis in Matlab

Vectorization

Store your sequences as columns

Plots

Code snippet

```
x     = sin(2*pi*fsig*t_sim);
x_noi = x+0.2*randn(size(t_sim));
Xmat  = [x(:) x_noi(:)];
plot(t_sim,Xmat)
legend('Ideal','Noisy')
```

Built-in vectorized functions

Code snippet

```
for nx = 1:10
    Xmat(:,nx) = sin(2*pi*fsig*t_sim + rand(1)*2*pi);
end
Xmat_noi = Xmat + 0.1*randn(size(Xmat));
Xpsd     = abs(fft(Xmat_noi)).^2;  % Vectorized processing
Xpsd_avg = mean(Xpsd,2);            % Average spectrum
Xfilt    = filter([1 1 1],1,Xmat); % Vectorized processing
```

Loop vectorization

```
t_sim_mat = repmat(t_sim(:),1,10);                   % Horizontal repetition
theta_mat = repmat(rand(1,10)*2*pi,length(t_sim),1); % Vertical repetition
Xmat      = sin(2*pi*fsig*t_sim_mat + theta_mat);
```

Common practices for systems modelling and signal analysis in Matlab

Randomization

Average spectrum measurements/visualizations

SNR computation

```
fsig      = 1.5; FSsim =100; t_sim = 0:1/FSsim:7;
fsig      = round(fsig/FSsim*length(t_sim))*FSsim/length(t_sim); % Frequency planning
t_sim_mat = repmat(t_sim(:),1,10);                    % Horizontal repetition
theta_mat = repmat(rand(1,10)*2*pi,length(t_sim),1);  % Vertical repetition
Xmat      = sin(2*pi*fsig*t_sim_mat + theta_mat);     % Each column is a realization
Xpsd      = abs(fft(Xmat)).^2;                        % Vectorized processing
Xpsd_avg  = mean(Xpsd,2);                             % Average spectrum
% Bin computation
Nx          = length(Xpsd_avg);
sig_bin     = fix(fsig/FSsim * Nx)+1;                 % Frequency planning (no leakage)
err_bin     = setdiff(1:round(Nx/2),sig_bin);
% Power integration
sig_pow = sum(Xpsd_avg(sig_bin));
err_pow = sum(Xpsd_avg(err_bin));
SNR_avg = 10*log10(sig_pow/err_pow);
```

Common practices for systems modelling and signal analysis in Matlab

Resampling

Deterministic signals (single-tone, multi-tones)

- Change the sampling rate at the signal generation

Code snippet

```
x = sin(2*pi*fsig*t_sim); % Change t_sim
```

Common practices for systems modelling and signal analysis in Matlab

Resampling

Random signals (telecom signals)

- Built-in interpolation: `interp,interp1,resample,...`
- Custom (upsample + filter)

Code snippet

```
% Generate LTE signal 1.1 5MHz (Doc : Generate a Test Model)
tm = '1.1'; bw = '5MHz';
[timeDomainSig,grid,testdata] = lteTestModelTool(tm,bw);
Fs = testdata.SamplingRate;
% Resampling : upsample + filter
OvSampleRatio   = 15;
signal_upsample = upsample(timeDomainSig,OvSampleRatio);
% Design filter (Doc : LTE Downlink ACLR Measurement)
firFilter = dsp.LowpassFilter();
firFilter.SampleRate = info.SamplingRate;
firFilter.PassbandFrequency = 2.5e6;
firFilter.StopbandFrequency = info.SamplingRate/2;
% Apply filter
waveform = firFilter(signal_upsample);
```

Warning

Be careful with delays due to filtering ! You may want to use `resample`.

This code snippet is only valid for Matlab

Common practices for systems modelling and signal analysis in Matlab

Delay compensation

Theoretical approach:

- `grpdelay(...)`
- (this approach is **only precise for FIR filters**)

Code snippet

```
Fs = 500; N = 500;
rng default
xn = ecg(N)+0.1*randn([1 N]);
tn = (0:N-1)/Fs;
% Filter example
Nfir = 70; Fst = 75;
firf = designfilt('lowpassfir','FilterOrder',Nfir, 'CutoffFrequency',Fst,'SampleRate',Fs);
delay = mean(grpdelay(firf))
```

(Example source: Matlab Doc: Compensate for Delay and Distortion Introduced by Filters)

Common practices for systems modelling and signal analysis in Matlab

Delay compensation

Correlation approach:

- `xcorr`
- `alignsignals` only on Matlab
- `finddelay` only on Matlab

Code snippet

```
x = triang(20);
y = [zeros(3,1);x]+0.3*randn(length(x)+3,1);
[xc,lags] = xcorr(y,x);
[~,delay] = max(abs(xc));
% Signal truncations
y_trunc = y(lags(delay)+1:end);
x_trunc = x;
```

(Example source: Matlab Doc: Cross-Correlation of Delayed Signal in Noise)

Going further

Matlab documentation

- Find a Signal in a Measurement
  - You receive some data and would like to know if it matches a longer stream you have measured.
- Measuring Signal Similarities
  - How do I compare signals with different sampling rates? and other topics...

Section outline

Homework

Homeworks

Download the following archive : TELECOM201>Homeworks>Homeworks

HOMEWORKS DEADLINES

- ADC must be handed in on Fri. 20th Dec. 2024
- DAC must be handed in on Fri. 17th Jan. 2025

Warning ⚠

- **Homeworks are mandatory !**
  - **No homework = malus**
  - **Good homework = bonus**

Each homework is expected to be done in less than 2 hours each.

Homeworks guidelines

- Dumb mandatory rules:
  - debug code when theoretical plot does not match empirical plot
  - check code executability before uploading (why not send to friend before?)
  - write a README file when you have more than 3 files
- Advices for home works:
  - generate signals outside from ADC/DAC
  - define a PSD function (and a possibly PSDdB)
  - superimpose plot lines when you compare theoretical with empirical
  - Come to see us as soon as you have an unexpected result that you cannot solve by yourself (syntax error is your duty).
  - Avoid using AI tools, they will not help you in the long run and if we spot, you will be **penalized**