Advanced Signal Processing

(Course materials)

Master 1 : Telecommunications & Networks

Faculty of Electrical Engineering

Houari Boumediene University of Science and Technology

Personal website: http://perso.usthb.dz/~akourgli/

e-mails: akourgli@usthb.dz , kourgliassia@gmail.com

Course content

Introduction: Miscellaneous Reminders

Chapter 1. Reminders on digital filters (FIR and IIR) (3 Weeks)

1. Transformed in Z
2. Structures , transfer functions , stability and implementation of digital filters (FIR and IIR)
3. Digital minimum phase filter
4. Synthesis methods of FIR filters and IIR filters
5. Multirate digital filters

Practical work n°1: Synthesis of FIR and IIR filters

Practical work n°2: Multi -rate filtering

Chapter 2. Random Signals and Stochastic Processes (4 Weeks)

1. Reminder about Random Processes
2. Notions of stochastic processes
3. Stationarities in the broad and strict sense and Ergodicity
4. Examples of stochastic processes (Poisson process, Gaussian and Markovian process)
5. Higher order statistics (Moments and cumulants , Polyspectra , non-Gaussian processes, nonlinear treatments)
6. Power Spectral Density
7. Matched filter , Wiener filter
8. Periodogram , correlogram, averaged periodogram , smoothed periodogram
9. Introduction to Particle Filtering

Practical work n°3: Random Processes, periodogram and variants

Practical work n°4: Matched filtering and Wiener filtering

Adaptive Digital Filtering (4 Weeks)

1. Parametric Methods
2. AR model (Lévinson , Yulewalker , Burg, Pisarenko , Music …)- ARMA model
3. LMS Stochastic Gradient Algorithm
4. RLS Recursive Least Squares Algorithm

Practical work n°5: AR modeling and adaptive filtering (LMS)

Chapter 4. Time-Frequency and Time-Scale Analysis (4 Weeks)

1. Time-frequency duality
2. Short term Fourier transform
3. Continuous, discrete and dyadic wavelets
4. Multi-resolution analysis and wavelet bases
5. Wigner-Ville transform
6. Time-Scale Analysis ,

Practical work n°6: Time- frequency /Scale transforms

Practical work n°1: Synthesis of FIR and IIR filters

Purpose of the lab: In this lab, we test two different methods to synthesize a digital filter: an FIR filter by the window method and an IIR filter by the bilinear transformation method.

1. Reminders

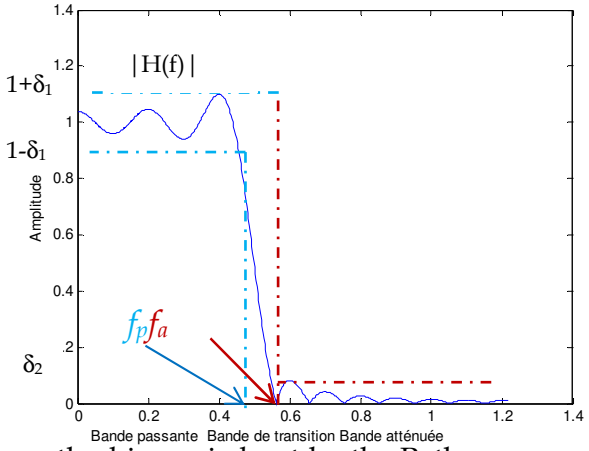
A finite impulse response (FIR) filter has a polynomial transfer function. It cannot be obtained by transposing a continuous filter, as is done for IIR filters. FIR filters have the disadvantage of requiring a large number of coefficients to obtain the same frequency characteristics. But on the other hand, they are unconditionally stable. It is possible to synthesize FIR filters with linear phase, that is to say with constant group propagation time.

The advantage of recursive filters (RII) is their low computational cost. With very few poles and zeros we can provide most of the frequency responses that we may need in the different applications. The disadvantages of recursive filters are: their non-linearity in linear phase (linear phase: constant propagation time for any frequency), and their numerical instability. Indeed, with the filter being feedback-enabled, errors in numerical precision become a matter of importance, as they can amplify and become out of control, first in the form of noise, but eventually in the form of jitter. Note that IIR filters can be designed by methods similar to those used for analog filters.

Template of a filter

The template of a filter is none other than the set of characteristics of the filter, namely:

- The gain of the filter in the passband.
- The attenuation of the notched band filter *fa* .
- The cutoff frequency *fc* is often expressed in normalized form with respect to the sampling frequency.
- The desired transition bandwidth Δf which must be as small as possible.
- Any oscillations in bandwidth and/or attenuated.



- The determination of the coefficients of an FIR filter by the window method is carried out by the Python function firwin of the scipy.signal library.

2. Demo

```
# -*- coding: utf-8 -*-

import numpy as np; import scipy. signal as sp; import matplotlib.pyplot as plt
from plot_zplane import zplane
#Design of a low-pass FIR filter fc=200, fe=1000, Deltaf=100, Aa>20
plt.figure(1)
fc=200;fe=1000; Deltaf=100
fcn=2*fc/fe; Deltafn=Deltaf/(fe/2)
Ncoef = int(np.ceil(1.8/Deltafn))
n = np.arange(-(Ncoef//2),Ncoef//2+1)
b = fcn*np.sinc(n*fcn);
#Visualization of impulse response and frequency response
L=256
f,Hf= sp.freqz(b,1,L,fs=fe);
plt.subplot(211); plt. stem(b);
plt.subplot(212); plt.plot(f, np.abs(Hf))
plt.title('Filter Module'); plt.xlabel('Frequency (Hz)');plt.ylabel('Amplitude')
plt.legend()
```

```
# Determination and plotting of poles and zeros
plt.figure(2)
a=np.array([1])
z,p = zplane(b,a)

# Effect of increasing the number of coefficients
fc=0.2;fe=1; fcn=2*fc/fe
A=np.array([21, 61, 101]);i=1;
a = np.array([1]);
L = 256;
plt.figure(3)
for N in A:
n = np.arange(-(N//2),N//2+1)
b = fcn*np.sinc(n*fcn);
f,Hf= sp.freqz(b,a,L,fs=fe);
plt.subplot(2,3,i); plt.stem(b);
plt.subplot(212); plt.plot(f, np.abs(Hf),label='N=%d'%N)
plt.title('Filter Module'); plt.grid(True); plt.xlabel('Frequency (Hz)');plt.ylabel('Amplitude')
   plt.legend()
i+=1

#Effect of windowing
from scipy.signal import windows
plt.figure(4)
N=41;
A=np.array([np.ones(N), windows.hann(N),windows.hamming(N) ]);
i=1;
for Fen in A:
n1 = np.arange(-(N//2),N//2+1)
b1 = fcn*np.sinc(n1*fcn)*Fen;
f,Hf1= sp.freqz(b1,a,L,fs=fe);
plt.subplot(2,3,i); plt. stem(b1); plt.title('Filter * Fen %d'%i);
plt.subplot(212); plt.plot(f, np.abs(Hf1),label='Win %d'%i)
plt.title('Filter Module'); plt.grid(True); plt.xlabel('Frequency (Hz)'); plt.ylabel('Amplitude')
   plt.legend()
i+=1
#Effect of windowing (display in db)
plt.figure(5)
fc=0.2;fe=2; fcn=2*fc/fe; N=41;
a = np. array([1]);
L=256;
A=np.array([np.ones(N), windows.hann(N),windows.hamming(N) ]);i=1;
for Fen in A:
n2 = np.arange(-(N//2),N//2+1)
b2= fcn*np.sinc(n2*fcn)*Fen;
f,Hf1= sp.freqz(b2,a,L,fs=fe);
plt.subplot(2,3,i); plt. stem(b2); plt.title('Filter * Fen %d'%i);
plt.subplot(212); plt.plot(f, 20*np.log10(np.abs(Hf1)),label='Win %d'%i)
plt.title('Filter Module'); plt.grid(True); plt.xlabel('Frequency (Hz)'); plt.ylabel('Amplitude')
plt.legend()
i+=1

# Bilinear transformation
fp=195;fa=205; fe=1000;
att_p=1; att_a=40;
```

```
wpn = fp*2*np.pi;
wan = fa*2*np.pi;
wpa = 2*fe*np.tan(np.pi*fp/fe) #Heartbeat compensation

N, WA = sp.buttord(wpn,wan, att_p,att_a, fs=fe)
z,p,k = sp.buttap(N); """Analog filter hn(p) in the form of poles, zeros and gain"""
# # sp.cheb2ap(N,att_a); sp.buttap(N); sp.ellipap(N,att_p,att_a)
Bpn,Apn = sp.zpk2tf(z,p,k); """Analog filter hn(p) in num, den form """
Bp, Ap = sp.lp2lp (Bpn,Apn,wpa); """Denormalization"""
# # sp.lp2hp sp.lp2bp sp.lp2bs
poles_anal = np.roots(Ap);
Bz1, Az1 = sp.bilinear(Bp,Ap,fe); """Transition from H(p) to H(z) by Bilinear Transformation"""

# Comparison of analog Ha(f) and digital Hz(f) filters
L = 256;
fz,Hz= sp.freqz(Bz1,Az1,L,fs=fe);
wa,Ha = sp.freqs(Bp,Ap, worN=np.arange(1,np.pi*fe)); fa=wa*0.5/np.pi;
plt.figure(6);
plt.plot(fz, np.abs(Hz),label='H(f) of the Digital filter');
plt.plot(fa, np.abs(Ha), label='H(f) of Analog filter'); plt.grid(); plt.legend()

plt.figure(7);
z,p=zplane(Bz1,Az1);
```

3. To do in TP

```
# import numpy as np; import scipy. signal as sp; import matplotlib.pyplot as plt
# from plot_zplane import zplane
# import sounddevice as snd
# import scipy.io.wavfile as wav
# fe,x = wav.read('youhavemail waiting.wav')
# snd. play(x, fe)

# Te=1/fe; N=len(x); t = np.arange(0,N)*Te;
# plt. figure(1); plt. subplot(211); plt. plot(t,x); plt.grid(True);
# plt.xlabel('time'); plt.ylabel('Amp'); plt.title('Phrase');

# # t = np.arange(0,N)*Te;
# # f0 = 1500; b = 20*np.cos(2*np.pi*f0*t);
# # Xb =x +b ;
# # snd.play(np.int8(Xb), fe)

"""
----------To do---------- ----------
```

Calculate and View your DFT in db in figure (1)
Add noise (3 sinusoids with a frequency of 1500, 2000 and 3000)
Listen to the noisy signal
View the DFT of the noisy signal in db
Design a filter by the window method with Aa>50 and Deltaf <fe/10, fc to be determined using firwin
View Impulse Response and Filter Transfer Function
Filter the noisy signal Xb (y = sp.lfilter(b,a,x)) with the filter created (b,a)
Listen and view the DFT of the filtered signal and compare with the original'

Design Filter by the bilinear transformation method with Aa>50 Ap=2 with N to be determined ,
you have to try several analog filters (Butterworth, Cheby 1, Cheby 2), which one seems adequate to you?
View Impulse Response and Filter Transfer Function for each method
Filter the noisy signal Xb with the filter created
Listen and view the DFT of the filtered signal
Comment
Compare between the 2 filter synthesis methods. For this example, which do you think is preferable ?
"""

Practical work no . 2: Multirate filtering

1. Interpolation and Decimation: Demo

```
# -*- coding: utf-8 -*-
import numpy as np; import matplotlib.pyplot as plt; import scipy.signal as sp;

N=30; f0=100; f1=300; fe=2000; Te=1/fe; NF=1024;
t = np.arange(0.0, N*Te, Te); x = np.sin(2.0*np.pi*f0*t) + np.sin(2.0*np.pi*f1*t);
TFx = np.fft.fft(x,NF); TFx = np.fft.fftshift(TFx); freq = np.arange(-NF/2,NF/2)*fe/NF;
plt.figure(1)
plt.subplot(321); plt. stem(t, x); plt.subplot(322); plt. plot(freq, np. abs(TFx));
x1 = np.zeros(2*N); x1[::2]=x[::] ;t1 = np.arange(0.0, N*Te, Te/2); TFx1 = np.fft.fft(x1,NF);
TFx1 = np.fft.fftshift(TFx1); freq = np.arange(-NF/2,NF/2)*2*fe/NF;
plt.subplot(323); plt. stem(t1, x1); plt. subplot(324); plt. plot(freq, np. abs(TFx1));
x2 = np.zeros(4*N); x2[::4]=x[::] ;t2 = np.arange(0.0, N*Te, Te/4); TFx2 = np.fft.fft(x2,NF);
TFx2 = np.fft.fftshift(TFx2); freq = np.arange(-NF/2,NF/2)*4*fe/NF;
plt.subplot(325); plt. stem(t2, x2); plt. subplot(326); plt. plot(freq, np. abs(TFx2));

plt.figure(2)
freq = np.arange(-NF/2,NF/2)*fe/NF;
plt.subplot(321); plt. stem(t, x); plt.subplot(322); plt. plot(freq, np. abs(TFx));
x1 = sp.resample(x,2*N); t1 = np.arange(0.0, N*Te, Te/2); TFx1 = np.fft.fft(x1,NF);
TFx1 = np.fft.fftshift(TFx1); freq = np.arange(-NF/2,NF/2)*2*fe/NF;
plt.subplot(323); plt. stem(t1, x1); plt. subplot(324); plt. plot(freq, np. abs(TFx1));
x2 = sp.resample(x,4*N); t2 = np.arange(0.0, N*Te, Te/4); TFx2 = np.fft.fft(x2,NF);
TFx2 = np.fft.fftshift(TFx2); freq = np.arange(-NF/2,NF/2)*4*fe/NF;
plt.subplot(325); plt. stem(t2, x2); plt. subplot(326); plt. plot(freq, np. abs(TFx2));

N=100; f0=100; f1=300; fe=2000; Te=1/fe; NF=1024;
t = np.arange(0.0, N*Te, Te); x = np.sin(2.0*np.pi*f0*t) + np.sin(2.0*np.pi*f1*t);
TFx = np.fft.fft(x,NF); TFx = np.fft.fftshift(TFx); freq = np.arange(-NF/2,NF/2)*fe/NF;
plt.figure(3)
plt.subplot(321); plt. stem(t, x); plt.subplot(322); plt. plot(freq, np. abs(TFx));
x1 = x[::2]; t1 = t[::2]; TFx1 = np.fft.fft(x1,NF); TFx1 = np.fft.fftshift(TFx1);
freq = np.arange(-NF/2,NF/2)*fe/(2*NF);
plt.subplot(323); plt. stem(t1, x1); plt. subplot(324); plt. plot(freq, np. abs(TFx1));
x2 = x[::6]; t2 = t[::6]; TFx2 = np.fft.fft(x2,NF); TFx2 = np.fft.fftshift(TFx2);
freq = np.arange(-NF/2,NF/2)*fe/(6*NF);
plt.subplot(325); plt. stem(t2, x2); plt.subplot(326); plt. plot(freq, np. abs(TFx2));

plt.figure(4)
freq = np.arange(-NF/2,NF/2)*fe/NF;
plt.subplot(321); plt.stem(t, x); plt.subplot(322); plt.plot(freq, np.abs(TFx));
x1 = sp.decimate(x,2); t1 = t[::2]; TFx1 = np.fft.fft(x1,NF); TFx1 = np.fft.fftshift(TFx1);
freq = np.arange(-NF/2,NF/2)*fe/(2*NF);
plt.subplot(323); plt. stem(t1, x1); plt. subplot(324); plt. plot(freq, np. abs(TFx1));
x2 = sp.decimate(x,6); t2 = t[::6]; TFx2 = np.fft.fft(x2,NF); TFx2 = np.fft.fftshift(TFx2);
freq = np.arange(-NF/2,NF/2)*fe/(6*NF);
plt.subplot(325); plt. stem(t2, x2); plt. subplot(326); plt. plot(freq, np. abs(TFx2));
```

2. Interpolation and Decimation: To do

```
# -*- coding: utf-8 -*-

import numpy as np; import matplotlib.pyplot as plt
#from zp_plot import zplane
import scipy. signal as sp;

from scipy.io import wavfile as wf; import winsound;
fname = 'youhavependingmail.wav';
winsound.PlaySound(fname, winsound.SND_FILENAME)
fe, x = wf.read(fname);
Te=1/fe; N=len(x); t = np.arange(0,N)*Te;
```

```
# plt.figure(1);plt.subplot(211); plt.plot(t,x); plt.grid(True);
# plt.xlabel('time'); plt.ylabel('Amp'); plt.title('Phrase');
# #Write to an audio file
# fnameb='Copy signal.wav';
#Xb=X
#Xb=np.int8(Xb); wf.write(fnameb, fe, Xb);
# winsound.PlaySound(fnameb,winsound.SND_FILENAME);
```

#----------To do---------- ----------

#1

Test downsampling with and without low-pass filtering

Display the signal and its TFD in each case and zoom in on the same area for a duration of 50 ms and comment

Listen to the obtained sign each time

Is there any distortion of the sound? What is it due to?

Up to what decimation factor can we go without deformation

What happens if we keep the same (original) fe playing?

#2

Test oversampling with and without lowpass filtering

Display the signal and its TFD in each case and zoom in on the same area for a duration of 50 ms or 20 ms and comment

Listen to the signa obtained each time (wf.wite with the new fe by multiplying by the Interp factor)

Do we perceive a distortion of the sound? Why?

What happens if you keep listening to the same fe?

#3

We want to move to a signal whose sampling frequency is 1.5*fe

Propose 2 solutions

Which one allows you to preserve the original values the most?

Practical work no. 3 : Random processes and spectral estimation

Goals: Manipulate random signals, characterize them using their moments of order 1 (mean, variance) and order 2 (autocorrelation, covariance). Approach and acquire the notions of stationarity and ergodicity. Spectral estimation. Address the notion of Forming Filter.

1. Demo

```
# -*- coding: utf-8 -*-
import numpy as np; import matplotlib.pyplot as plt;
import scipy.signal as sp
N_R=500; N_va=1000; A=1; f0=.002; X=[];

t = np.linspace(0,N_va-1,N_va)
phi=np.random.uniform(0,2*np.pi,N_R);

for i in range (N_R):
X=np.append(X,A*np.cos(2*np.pi*f0*t+phi[i]))
X= np.resize(X,new_shape=(N_R,N_va))

plt.figure(1);plt.plot(t,X[0,:]);plt.plot(t,X[1,:]);plt.plot(t,X[2,:]);plt .plot(t,X[3,:]);
"""Sationarity of order 1"""
avg=np.mean(X,0); var=np.var(X,0);
plt. figure(2); plt.subplot(211); plt.plot(avg); plt.subplot(212); plt. plot(var);
"""Sationarity of order 2"""
Cx=np.cov(X, rowvar=False);
plt. figure(3); plt. imshow(Cx);
plt. figure(4); plt. plot(Cx[0,:]); plt. plot(Cx[20,:]); plt. plot(Cx[40,:]);
"""Ergodism of order 1"""
X1=X[1,:];
avg_t=np.mean(X1); var_t=np.var(X1);
"""Ergodism of order 2"""
Cx_t=np.correlate(X1-avg_t,X1-avg_t,'full')/N_va
Rx_t=np.correlate(X1,X1,'full')/N_va
plt.figure(5)
plt.plot(Cx_t,label='Cov temp'); plt.plot(Rx_t,label='Cor temp');
plt.legend()

# A = np.random.uniform(-1,1,N_R);
# for i in range (N_R):
# X=np.append(X,A[i]*np.cos(2*np.pi*f0*t))

#X=np.random.randn(N_R,N_va)

# mux=0; varx=1;
# X=mux+np.sqrt(varx)*np.random.randn(N_R,N_va)

""" 1 realization of Gaussian white noise """
""" Ergodic + stationary hypothesis """
mux=0; varx=1; N=1000;
bb=mux+np.sqrt(varx)*np.random.randn(N)
fe=22050; Te=1/fe;
t = np.linspace(0, N-1, N)*Te;
Moy_s=Moy_t=np.mean(bb);
```

```
Var_s=Var_t=np.var(bb);
Rb_s=Rb_t = np.correlate(bb,bb,mode='same')/N;
tt = np.linspace(1-N/2, N/2-1, N)*Te;
plt.figure(6);plt.subplot(311); plt.plot(t,bb); plt.grid(True);
plt.subplot(312); plt.plot(tt,Rb_s); plt.grid(True);
plt.xlabel('time'); plt.ylabel('Amp'); plt.title('White noise statistical autocor');
NF=2048;
TFb = np.fft.fft(bb,NF); TFb=np.fft.fftshift(TFb);
Sbf = np.abs(TFb)**2/N; ff = np.linspace(-NF/2, NF/2-1, NF)*fe/NF;
s=np.mean(Sbf)
plt.subplot(313); plt.plot(ff,Sbf); plt.grid(True);plt.title("White Noise DSP");


""" Calculation of DSP: Spectral estimation"""
f0=5000; X = bb + 0.5*np.cos(2*np.pi*f0*t)
#f1=5050; X = X + 0.5*np.cos(2*np.pi*f1*t)
NF=2048;
TFx1 = np.fft.fft(X,NF)/N; TFx1=np.fft.fftshift(TFx1);
Sxf1 = np.abs(TFx1)**2; ff = np.linspace(-NF/2, NF/2-1, NF)*fe/NF;
Rx_s=Rx_t = np.correlate(X,X,mode='same')/N; Sxf2 = np.fft.fft(Rx_s,NF)/N; Sxf2=np.fft.fftshift(Sxf2); Sxf2 = np.abs(Sxf2)
ff3,Sxf3 = sp.periodogram(X,fs=fe,window='boxcar',nfft=NF,return_onesided=False,scaling='spectrum')
ff4,Sxf4 = sp.periodogram(X,fs=fe,window='hann',nfft=NF,return_onesided=False,scaling='spectrum')
ff5, Sxf5 = sp.welch(X,fs=fe,window='hann',nperseg=N//4, nfft=NF,return_onesided=False,scaling='spectrum')
var_ss=np.mean(Sxf1)
ff = np.linspace(-NF/2, NF/2-1, NF)*fe/NF;
plt.figure(7)
plt.subplot(221);plt.plot(ff,Sxf1,ff3,Sxf3); plt.title('DSP=Rectangular Window Periodogram');
plt.subplot(222);plt.plot(ff,Sxf2); plt.title('Correlogram');
plt.subplot(223);plt.plot(ff4,Sxf4); plt.title('Hanning Window Periodogram');
plt.subplot(224);plt.plot(ff5,Sxf5); plt.title('Average Hanning Window Periodogram');

""" Concept of Shaper filter """
a = np.array([1,-0.839,-0.015,-0.320,0.197,0.055,-0.285,0.067,0.044,0.003,0.178])
b = np.array([1,0]);
L=500; f,H= sp.freqz(b,a,L,fs=fe);
Y = sp.lfilter(b,a,bb);
TFb = TFb[0:L];
plt. figure(8); plt.subplot(211);plt.plot(f, 20*np.log10(abs(TFb/TFb.max())));
plt.title('Input Signal: White Noise'); plt.xlabel('Freq  ( Hz)'); plt.ylabel('Amp');
plt.subplot(212); plt.plot(f, 20*np.log10(abs(H/H.max())));
plt.title('Filter Module (blue)+ Filter Output (orange)'); plt.xlabel('Freq  ( Hz)'); plt.ylabel('Amp');
TFy = np.fft.fft(Y); TFy = TFy[0:L];
plt.plot(f, 20*np.log10(abs(TFy/TFy.max())));
```

2. To do

Download the 'you have mail waiting.wav' file and place it in the same directory as your program starting as follows:

```
import t numpy as np; import matplotlib.pyplot as plt;
from scipy.io import wavfile as wf; import winsound;
fname = 'youhavependingmail.wav';
winsound.PlaySound(fname, winsound.SND_FILENAME)
fe, X = wf.read(fname);
Te=1/fe; N=len(X); t = np.linspace(0, N-1, N)*Te;
plt.figure(1);plt.subplot(211); plt.plot(t,X); plt.grid(True);
```

```
plt.xlabel('time'); plt.ylabel('Amp'); plt.title('Phrase');
N1=21000;N2=21500; Y=X[N1:N2] ; ty=np.linspace(N1, N2-1, N2-N1)*Te;
plt.subplot(212); plt.plot(ty,Y); plt.grid(True);
plt.xlabel('time'); plt.ylabel('Amp'); plt.title('piece of sentence');
#zz=np.int8(Y); wf.write("z.wav", fe, zz);
#winsound.PlaySound("z.wav",winsound.SND_FILENAME);
```

1. Is the signal studied a random process or a realization of a random signal?
2. Calculate the PSD. Is it of statistical or temporal origin?
3. Calculate and visualize the temporal mean and variance for different parts of the signal that overlap for durations of N=1000,500, 250, 125. For what value of N can we consider the process stationary?

B. Consider the following lines of a python code permitting comparison of 4 spectral estimation techniques

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import welch
# Parameters
N = 1024; Te = 1 / N; Fe = N; f1, f2 = 100, 150; A1, A2, A3 = 2, 1.2, 4
t = np.arange(N) * Te
# Signal
x = A1 * np.cos(2 * np.pi * f1 * t) + A2 * np.cos(2 * np.pi * f2 * t) + A3 * np.random.randn(N)
plt.figure()
plt.plot(t, x); plt.title("Signal x(t)"); plt.xlabel("Time (s)"); plt.ylabel("Amplitude")
```

1. Complete the code code to estimate the PSD with the 4 spectral estimation approaches
2. Compare the 4 techniques by varying A1, A2 and A3 and comment.
3. Bring the frequencies closer and observe.
4. Comment on the four techniques for estimating PSD by varying N.
5. What is the advantage and disadvantage of averaging?

Practical work n° 4 : Matched filter and Wiener filter

Goals: Optimal filtering

- Detection of a known deterministic signal drowned in white noise →Matched filter
- Filtering a random WSS signal drowned in WSS noise →Wiener filter

1. Demo

```
# -*- coding: utf-8 -*-
import numpy as np; import matplotlib.pyplot as plt
"Matched filtering"
N=500; var = 0.16; mu = 0;
x=np.ones(10); Delay = 100
y = np.sqrt(var) * np.random.randn(N) + mu
y[Delay:Delay+len(x)]= y[Delay:Delay+len(x)] + x
#Delay=300; y[Delay:Delay+len(x)]= y[Delay:Delay+len(x)] + x
Ryx = np.correlate(y,x,"full"); Ryx=Ryx[len(x)-1:]
plt.figure(1)
plt.subplot(311); plt. plot(x); plt.title('sent signal')
plt.subplot(312); plt. plot(y); plt.title('signal received')
plt.subplot(313); plt. plot(Ryx); plt.title('Intercorrelation between transmitted signal and received signal')
import scipy.linalg as la; import scipy. signal as sp;
fecg=[]; meg=[];

""" Loading Files """
with open('fecg.txt') as f:
for i in f:
fecg.append(float(i))
fecg = np.array(fecg)
with open('mecg.txt') as f:
for i in f:
mecg.append(float(i))
mecg = np.array(mecg)
""" Added noise """
N=len(mecg)
var = 0.01; bb=(var**0.5)*np.random.randn(N); obs=fecg+mecg+bb;
"""Determination of the coefficients of the Wiener Filter"""
P=10;
obs=obs-np.mean(obs); mecg=mecg-np.mean(mecg);
Rxx = np.correlate(obs,obs,mode='full')[len(obs)-1:];
Rmecg = np.correlate(mecg,mecg,mode='full')[len(mecg)-1:];
Ryx = Rxx-Rmecg;
C = la.toeplitz(Rxx[0:P]);
B = Ryx[0:P];
b = la.inv(C).dot(B)
"""Filtering"""
a = np.array([1,0]);
y = sp.lfilter(b,a,obs);
plt.figure(2);
plt.subplot(311); plt.plot(fecg); plt.grid(True); plt.title('ecg baby');
plt.subplot(312); plt.plot(obs); plt.grid(True);plt.title('Observation');
plt.subplot(313); plt.plot(fecg); plt.plot(y); plt.grid(True);plt.title('estimated baby ecg');
```

2. To do

"To do Matched filtering"

1. Create a signal composed of a random sequence of A and -A symbols with a duration of 10 each separated by 20s

```
import numpy as np
import matplotlib.pyplot as plt
W = np.random.randint(0,2,10); A=5;
```

```
W = 2*W-1
W = A*W
x = []; break = np.zeros(20)
for i in W:
symb = i*np.ones(10)
x = np.append(x,symb)
x = np.append(x, pause)
```

2. Add white noise
3. Use suitable filter
4. Visualize the 3 signals

"To Do Wiener Filtering"

```
import numpy as np; import scipy. signal as sp; import matplotlib.pyplot as plt
import sounddevice as snd
import scipy.io.wavfile as wav
import scipy.linalg as la; import scipy. signal as sp;
fe,x = wav.read('vousavezducourrierenattente.wav')
```

1. Create a signal composed of the audio signal + white noise
2. Listen to check for the presence of the echo
3. Determine the coefficients of the Wiener filter
4. Filter
5. Listen filtered signal
6. View the 3 signals: original audio signal, with echo and the filtered one
7. View the 3 signal periodograms: original, noisy audio signal and the one filtered on a small portion

Practical work n° 5: Parametric modeling and adaptive digital filtering

This lab aims to:
- Model a random signal by AR and MA models (by AR approximation) and extract useful information.
- Test an adaptive filtering method (LMS)

Exercise 1: Download the file 'you have mail waiting.wav' and place it in the same directory as your program then select a part between 21000 and 21500

I. Demo

```
# -*- coding: utf-8 -*-

"""AR Modeling"""
# import numpy as np; import matplotlib.pyplot as plt;
# from scipy.io import wavfile as wf;
# import scipy. linalg as la; import scipy. signal as sp;

# fname = 'youhavemail waiting.wav';
# #winsound.PlaySound(fname, winsound.SND_FILENAME)
# fe, S = wf.read(fname);
# Te=1/fe; N=len(S); t = np.linspace(0, N-1, N)*Te;
# plt. figure(1); plt. subplot(211); plt.plot(t,S); plt.grid(True);
# plt.xlabel('time'); plt.ylabel('Amp'); plt.title('Phrase');
# L=500;N1=21000;N2=N1+L; y=S[N1:N2]; ty=np.linspace(N1, N2-1, N2-N1)*Te;
# plt. subplot(212); plt. plot(ty,y); plt.grid(True);
# plt.xlabel('time'); plt.ylabel('Amp'); plt.title('part of the sentence');


# """ AR Modeling"""
# P=20; "P Number of coefficients of the AR model>2  "
# y=y-np.mean(y)
# R = np. correlate(y,y,mode='full')[len(y)-1:];
# #R=R/R.max(); "Autocor"
# C = la.toeplitz(R[0:P]);
# B = -R[1:P+1];
# a = la.inv(C).dot(B)
# a = np.append(1,a)
# sigma_square = sum(a*R[0:P+1]);
# """ Visualization of the frequency response of the filter found and the starting y signal"""
# b = np. array([1,0]);
# f,H= sp.freqz(b,a,L//2,fs=fe);
# TFy = np.fft.fft(y); TFy = TFy[1:L//2+1];
# plt.figure(2); plt.subplot(121);plt.plot(f, 20*np.log10(abs(TFy/TFy.max())));
# plt.plot(f, 20*np.log10(abs(H/H.max())));
# plt.title('Filter module found (orange)+ Original signal modeled (blue)');
# plt.xlabel('Freq  ( Hz)'); plt.ylabel('Amp');
# # """ Synthesis from found filter"""
# bb = (sigma_square**0.5)*np.random.randn(L);
# #bb=bb-np.mean(bb);
# y_synt = sp.lfilter(b,a,bb);
# TFys = np.fft.fft(y_synt); TFys = TFys[0:L//2];
# plt.figure(2); plt.subplot(122);plt.plot(f, 20*np.log10(abs(TFys/TFys.max())));
# plt.plot(f, 20*np.log10(abs(TFy/TFy.max())));
# plt.title('synthesized signal (orange)+ synthesized signal (blue)');
# plt.xlabel('Freq  ( Hz)'); plt.ylabel('Amp');
"""---------------------------------------------------------- ----------------------"""
```

```
""" LMS: Stochastic Gradient Algorithm: Restoration  " ""
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as sp

fecg=[]; meg=[];
""" Reading files """
with open('fecg.txt') as f:
for i in f: fecg.append(float(i))
fecg = np.array(fecg)
with open('mecg.txt') as f:
for i in f: mecg.append(float(i))
mecg = np.array(mecg)
""" Added noise """
N=len(mecg); var = 0.01;
bb=(var**0.5)*np.random.randn(N);
x=fecg+bb+mecg; #x=x-np.mean(x)
y=fecg; #y=y-np.mean(y)

""" Filter determination by LMS """
Ncoeff=7; mu=0.1;
h = np.zeros(Ncoef) # Initialization of the filter coefficients
L = len(x) # Signal length
yest = np.zeros(N) # y(n) estimated
En = np.zeros(N) # Error
for i in range(Ncoef,L//10):
   X= np. flipud(x[i-Ncoef+1:i+1]); #Take X=[x(n),x(n-1),...,x(n-Ncoef+1)))]
   yest[i] = np.dot(h, X)
En[i] = x[i] - yest[i]
h = h + mu * En[i] * X
h = h/sum(abs(h))

""" Viewing results"""
plt.figure(1)
y_est =sp.lfilter(h,1,x)
plt.subplot(311); plt.plot(x,'r', label= 'observation'); plt.plot(y,'b', label= 'fecg'); plt.grid(); plt.legend()
plt.subplot(313);plt.plot(En,'g',label='error'); plt.grid(); plt.legend()
plt.subplot(312); plt. plot(y,'b', label= 'fecg'); plt.plot(y_est,'r', label= 'ecg restored'); plt.legend()

"""---------------------------------------------- ----------------------"""
# """ Prediction by LMS """
# mux=0; varx=0.01; N=1000
# bb=mux+np.sqrt(varx)*np.random.randn(N)

# fe=10000; Te=1/fe;
# f0= 100;
# t = np.arange(0,N)*Te
# y = 0.5*np.cos(2*np.pi*f0*t)
# x = y + bb

# Ncoef=5; mu=0.2;
# hp = np.zeros(Ncoef) # Initialization of filter coefficients
# L = len(x) # Signal length
# yest = np.zeros(N) # y(n) estimated
# En = np.zeros(N) # Error
# for i in range(Ncoef,L-1):
# X= np. flipud(x[i-Ncoef+1:i+1]); #Take X=[x(n),x(n-1),...,x(n-Ncoef+1)))]
# yest[i] = np.dot(hp, X)
# En[i] = x[i] - yest[i]
# hp = hp + mu * En[i] * X

# plt.figure(2); plt.plot(x,'b', label= 'observation'); plt.plot(yest,'r', label= 'estimated')
# plt.plot(En,'g', label= 'Instant error'); plt.grid(); plt.legend()
"""---------------------------------------------- ----------------------"""
```

```
""" System identification by LMS """
# mux=0; varx=0.1; N=500
# x = mux+np.sqrt(varx)*np.random.randn(N)
# b = np.array([0.65, -0.35, 0.1])
# y = sp.lfilter(b,[1,0],x)

#Ncoef=3; mu=0.6;
# hs = np.zeros(Ncoef) # Initialization of filter coefficients
# yest = np.zeros(N) # y(n) estimated
# En = np.zeros(N) # Error
# for i in range(Ncoef,N-1):
# X= np. flipud(x[i-Ncoef+1:i+1]); #Take X=[x(n),x(n-1),...,x(n-Ncoef+1)))]
# yest[i] = np.dot(hs, X)
# In[i] = y[i] - yest[i]
# hs = hs + mu * En[i] * X

# print(hs)
# plt. figure(3); plt.plot(y,'b', label= 'Output'); plt.plot(yest,'r', label= 'Estimated output')
# plt.plot(En,'g', label= 'Instant error'); plt.grid(); plt.legend()


"""To do AR"""

# import numpy as np; import scipy. signal as sp; import matplotlib.pyplot as plt
# import sounddevice as snd
# import scipy.io.wavfile as wav
# import scipy.linalg as la;

# def ModelAR(y,P):
# y=y-np.mean(y)
# R = np.correlate(y,y,mode='full')[len(y)-1:]
# C = la.toeplitz(R[0:P])
#B = -R[1:P+1]
# a = la.inv(C).dot(B)
# a = np.append(1,a)
# sigma_carre = sum(a*R[0:P+1]);
# return a,sigma_carre

# fe,x = wav.read('vowels.wav')
# snd. play(x, fe)

#1. Read, view and listen to the vowels.wav file
# 2. Identify the 3 parts corresponding to the 3 “A”
#NA=7000
#A1 = x[162400:162400+NA];
#A2 = x[253200:253200+NA]
#A3 = x[340000:340000+NA]
#3. Show the 3 "A's"
# 4. Calculate and display their periodogram in db
# 5. Determine the shaper filter of each "A" using the ModelAR function
# 6. Calculate the mean squared error (MSE) between the coefficients of each A
# from sklearn.metrics import mean_squared_error
# mseA12=mean_squared_error(a1,a2)
# 7. Repeat steps 2 to 6 for the letter "E"
# 8. Calculate the MSE between the 3 "A's" and the 3 "E's"

"""To do Restoration by LMS """
"""
# 1. Load the audio vousavezducourrierenattente.wav in a variable y
# 2. Create a signal x composed of audio signal (y) + white noise
```

3. Listen to check for the presence of noise
4. Center the y and x variables
5. Determine by LMS the filter h by taking the part corresponding to silence (mu=0.01 and for i in range(Ncoef,100))
#6. Invert and normalize h-filter
7. Filter x by the filter
#8. Visualize Results
9. Listen to restored audio
ss=input()
snd.play(np.int8(y_est), fe)
"""

Practical work n°6: From TFCT, Wigner-Ville to wavelets

This lab aims to:
- Become familiar with time-frequency and time-scale representations
- Approach the DWT, the notions of filter banks through the decomposition and synthesis filters
- Apply DWT to denoising

1. Demo

```
# -*- coding: utf-8 -*-
import numpy as np; import matplotlib.pyplot as plt; import scipy.signal as sp;
NF=1024; Te=0.005; fe=1/Te;N=round(2/Te);
t = np.linspace(0, 2, N);
x1= sp.chirp(t, f0=50, t1=2, f1=1, method='linear');
x2= sp.chirp(t, f0=1, t1=2, f1=50, method='linear');

""" TFD"""
TFx1 = np.fft.fft(x1,NF); TFx1 = np.fft.fftshift(TFx1);
TFx2 = np.fft.fft(x2,NF); TFx2 = np.fft.fftshift(TFx2); freq = np.arange(-NF/2,NF/2)*fe/NF;
plt.figure(1)
plt.subplot(221); plt. plot(t, x1); plt.title('Chirp from 50 to 1Hz'); plt.subplot(222); plt. plot(freq, np. abs(TFx1));
plt.subplot(223); plt. plot(t, x2); plt.title('Chirp from 1 to 50 Hz'); plt. subplot(224); plt. plot(freq, np. abs(TFx2));

"""Spectrogram: TFCT"""
f, tt, Sxx1 = sp.spectrogram(x1, fe, nperseg=100, noverlap=20);
f, tt, Sxx2 = sp.spectrogram(x2, fe, nperseg=100, noverlap=20);
plt.figure(2)
plt.subplot(221); plt. plot(t, x1); plt.subplot(222); plt.pcolormesh(tt, f, Sxx1, shading='auto')
plt.ylabel('Frequency [Hz]'); plt.xlabel('Time [sec]'); plt.title('Spectrogram Tfen=100, TRec=20');
plt.subplot(223); plt.plot(t, x2); plt.subplot(224); plt.pcolormesh(tt, f, Sxx2, shading='auto')
plt.ylabel('Frequency [Hz]'); plt.xlabel('Time [sec]'); plt.title('Spectrogram Tfen=100, TRec=20');
f, tt, Sxx1 = sp.spectrogram(x1, fe, nperseg=20, noverlap=4);
f, tt, Sxx2 = sp.spectrogram(x2, fe, nperseg=20, noverlap=4);
plt.figure(3)
plt.subplot(221); plt.plot(t, x1); plt.subplot(222); plt.pcolormesh(tt, f, Sxx1,shading='auto')
plt.ylabel('Frequency [Hz]'); plt.xlabel('Time [sec]'); plt.title('Spectrogram Tfen=20, TRec=4');
plt.subplot(223); plt. plot(t, x2); plt. subplot(224); plt.pcolormesh(tt, f, Sxx2, shading='auto')
plt.ylabel('Frequency [Hz]'); plt.xlabel('Time [sec]'); plt.title('Spectrogram Tfen=20, TRec=4');

"""Wigner-Ville """
from tftb.processing import WignerCityDistribution
WVx1=WignerCityDistribution(x1,timestamps=t)
WVx2=WignerCityDistribution(x2,timestamps=t)
tfr_x1, t_x1, f_x1 = WVx1.run()
tfr_x2, t_x2, f_x2 = WVx2.run()
plt.figure(4)
plt.subplot(221); plt. plot(t, x1); plt.subplot(222); plt.pcolormesh(t_x1, f_x1*fe, np.abs(tfr_x1),shading='auto')
plt.ylabel('Frequency [Hz]'); plt.xlabel('Time [sec]'); plt.title('Wigner-City');
plt.subplot(223); plt.plot(t, x2); plt.subplot(224); plt.pcolormesh(t_x2, f_x2*fe, np.abs(tfr_x2),shading='auto')
plt.ylabel('Frequency [Hz]'); plt.xlabel('Time [sec]'); plt.title('Wigner-City');

"""Continuous wavelets"""
import pywt
Width = np.arange(1, 31);
cwtmatr,fs = pywt.cwt(x2, Width, 'mexh')
plt.figure(5)
plt.imshow(cwtmatr, aspect='auto', vmax=abs(cwtmatr).max(), vmin=-abs(cwtmatr).max())

""" Discrete Wavelets: Analysis Filters and Synthesis """
wavelet = pywt.Wavelet('haar')
plt.figure(6)
plt.subplot(221); plt.stem(wavelets.dec_lo); plt.title('Low Pass Decomposition');
plt.subplot(222); plt.stem(wavelets.dec_hi); plt.title('High Pass Decomposition');
```

```
plt.subplot(223); plt.stem(wavelets.rec_lo); plt.title('Reconstruction Low Pass');
plt.subplot(224); plt.stem(wavelets.rec_hi); plt.title('Reconstruction Low Pass');
L=512;

""" Application of the DWT: Decomposition and Reconstruction """
x = np.genfromtxt('ecg.dat');
coeffs = pywt.wavedec(x, 'db5', level=3)
cA3, cD3, cD2, cD1 = coeffs
plt.figure(7)
plt.subplot(321); plt. plot(x); plt.title('Original signal');
plt.subplot(322); plt.plot(cD1); plt.title('Level 1 detail');
plt.subplot(323); plt.plot(cD2); plt.title('Level 2 detail');
plt. subplot(324); plt.plot(cD3); plt.title('Level 3 detail');
plt.subplot(325); plt.plot(cA3); plt.title('Level 3 approximation');

x_reconst=pywt.waverec([cA3, cD3, cD2,], 'db5');
plt.subplot(326); plt.plot(x_reconst); plt.title('Signal Reconstructed');
```

2. To do

1. Set some level details to 0 and rebuild (remove on rebuild)
2. Choose one and set the detail values below the threshold to zero.
3. Test other wavelets and observe the quality of the reconstruction

1.1. Applying to an image

Using pywt.dwt2, decompose and reconstruct an image on several levels, set details to zeros and comment.