Summary using Examples

01 - 04: Quantization:

Original live microphone signal in time domain:

`pyrecplotanimation.py`

Uniform quantization:

`pyrecplay_quantizationblock.py`

Non-uniform quantization (mu-Law):

`pyrecplay_mulawquantizationblock.py`

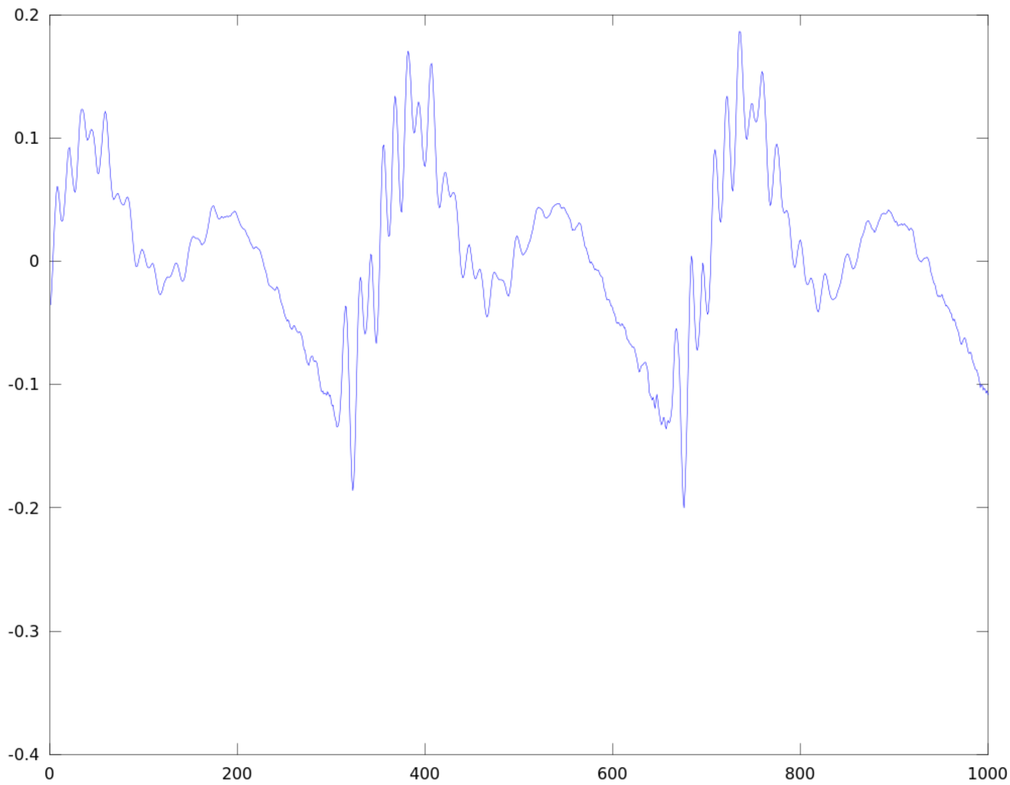
Log compression function:

`pyrecplay_loglimiterblock.py`

05: Vector Quantization:

2-dimensional vectors:

```
ipython --pylab
import scipy.io.wavfile as wav
rate, snd = wav.read('mspeech.wav')
#Take an excerpt of 1000 samples,
#starting at sample 2001 and plot it:
spex=snd[2000+np.arange(1,1000)]
plot(spex)
```



#Now plot the 2 dimensional vectors resulting from groups of
#even and odd samples:

```
plot(spex[2::2], spex[1::2], '+')
```

06: Sampling:

```
pyrecplay_samplingblock.py
```

```
python pyrecspecwaterfallsampling.py
```

07, 08: z-Transform, Filtering:

```
pyrecplay_filterblock.py
```

09,10: Allpass, Warping:

Python function:

warpingphase(w,a)

Minimum Phase Filters:

$$H(z) = H_{min}(z) \cdot H_{ap}(z)$$

11: Hilbert Transform

```
# construct a delayed impulse to implement the
# delay for the real part:
delt = np.zeros(20)
delt[9] = 1
#delt =
#0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0
#Then we need to add our imaginary part as our
#Hilbert transform to obtain our complex filter:
h = np.zeros(20);
n = np.arange(-9, 10+1, 2);
h[(n-1)+10] = 2./(np.pi*n);
hone = delt+h*1j
```

```
hone
array([ 0.-0.07073553j,  0.+0.j        ,  0.-0.09094568j,  0.+0.j
,
        0.-0.12732395j,  0.+0.j        ,  0.-0.21220659j,  0.+0.j
,
        0.-0.63661977j,  1.+0.j        ,  0.+0.63661977j,  0.+0.j
,
        0.+0.21220659j,  0.+0.j        ,  0.+0.12732395j,  0.+0.j
,
        0.+0.09094568j,  0.+0.j        ,  0.+0.07073553j,  0.+0.j
])
```

```
freqz(hone,1, whole=True, axisFreqz=[0, 6.28,-40,10]);
```

12: Wiener Filter, Matched Filter

Python Example for denoising speech:

```
ipython -pylab

from sound import *

from scipy import signal as sp

x, fs = wavread('fspeech.wav')

#make x a matrix and transpose it into a
column:

x=matrix(x).T

sound(array(x), fs)


#additive zero mean white noise (for
-2**15<x<+2**15):

y=x+0.1*(random.random(shape(x))-0.5)*2**15

sound(array(y), fs)
```

#we assume 10 coefficients for our Wiener filter.
#10 to 12 is a good number for speech signals.

```
A = matrix(zeros((100000, 10)))

for m in range(100000):

    A[m,:] = y[m+arange(10)].T
```

#Our matrix has 100000 rows and 10 colums:

```
print A.shape
# (100000, 10)
```

#Compute Wiener Filter:

#Trick: allow (flipped) filter delay of 5
#samples to get better working denoising.
#This corresponds to the center of our Wiener filter.
#The desired signal hence is x[5:100005].
#Observe: Since we have the matrix type, operator
„*“ is matrix multiplication!

```
h=inv(A.T*A)*A.T*x[5:100005]

xw = sp.lfilter(array(flipud(h).T)[0],
[1],array(y.T)[0])
```

#and listen to it:

```
sound(xw, fs)
```

13 Matched Filter:

Python Example:

Construct a signal sig (length 11):

```
ipython -pylab

sig = arange(0, 1.1, 0.1)

sig
```

```
#Out: array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,
#0.6,  0.7,  0.8,  0.9,  1. ])

plot(sig)

xlabel('Sample')

ylabel('Value')

title('An Example Signal')

signoise = random.rand(20)-
0.5+hstack([zeros(4),sig,zeros(5)])

plot(signoise)

xlabel('Sample')

ylabel('Value')

title('The Example Signal in Noise')
h = sig[::-1] # fliplr

signoisemf = sp.lfilter(h, 1, signoise)

plot(signoisemf)

xlabel('Sample')

ylabel('Value')

title('The Example Signal in Noise after
Macthed Filtering')
```

14 Prediction:

LMS with Quantizer Python Example

Encoder:

```
import numpy as np
from sound import *
```

```
import matplotlib.pyplot as plt

x, fs = wavread('fspeech.wav')
#normalized float, -1<x<1
x = np.array(x,dtype=float)/2**15
print np.size(x)
e = np.zeros(np.size(x))

h = np.zeros(10)
xrek=np.zeros(np.size(x));
P=0;
#have same 0 starting values as in decoder:
x[0:10]=0.0
quantstep=0.05;
for n in range(10, len(x)):
    if n> 4000 and n< 4010:
      print "encoder h: ", h, "e=", e
    #prediction error and filter, using the vector of the time reversed
IR:
    #predicted value from past reconstructed values:
    P=np.dot(xrek[n-10+np.arange(0,10)], np.flipud(h))
    #quantize and de-quantize e to step-size 0.05 (mid tread):
    e[n]=np.round((x[n]-P)/quantstep)*quantstep;
    #Decoder in encoder:
    #new reconstructed value:
    xrek[n]=e[n]+P;
    #LMS update rule:
    h = h + 1.0* e[n]*np.flipud(xrek[n-10+np.arange(0,10)])

print "Mean squared prediction error:", np.dot(e, e) /np.max(np.size(e))
#without quant.: 0.000215852452838
#with quant.: 0.000532170587442

#listen to it:
sound(2**15*e, fs)

plt.figure()
plt.plot(x)
#plt.hold(True)
plt.plot(e,'r')
plt.xlabel('Sample')
plt.ylabel('Normalized Sample')
plt.title('Least Mean Squares (LMS) Online Adaptation')
plt.legend(('Original','Prediction Error'))
plt.show()
```

Decoder:

```
# Decoder
h = np.zeros(10);
xrek = np.zeros(np.size(x));
for n in range(10, len(x)):
    if n> 4000 and n< 4010:
       print "decoder h: ", h
    P=np.dot(xrek[n-10+np.arange(10)], np.flipud(h))
    xrek[n] = e[n] + P
    #LMS update:
    h = h + 1.0 * e[n]*np.flipud(xrek[n-10+np.arange(10)]);

plt.plot(xrek)
plt.show()
#Listen to the reconstructed signal:
sound(2**15*xrek,fs)
```

Execute it with:

```
python lmsquantexample.py
```