Digital Signal Processing 2/ Advanced Digital Signal Processing
Lecture 4,
Lloyd-Max Quantizer, LBG

Gerald Schuller, TU Ilmenau

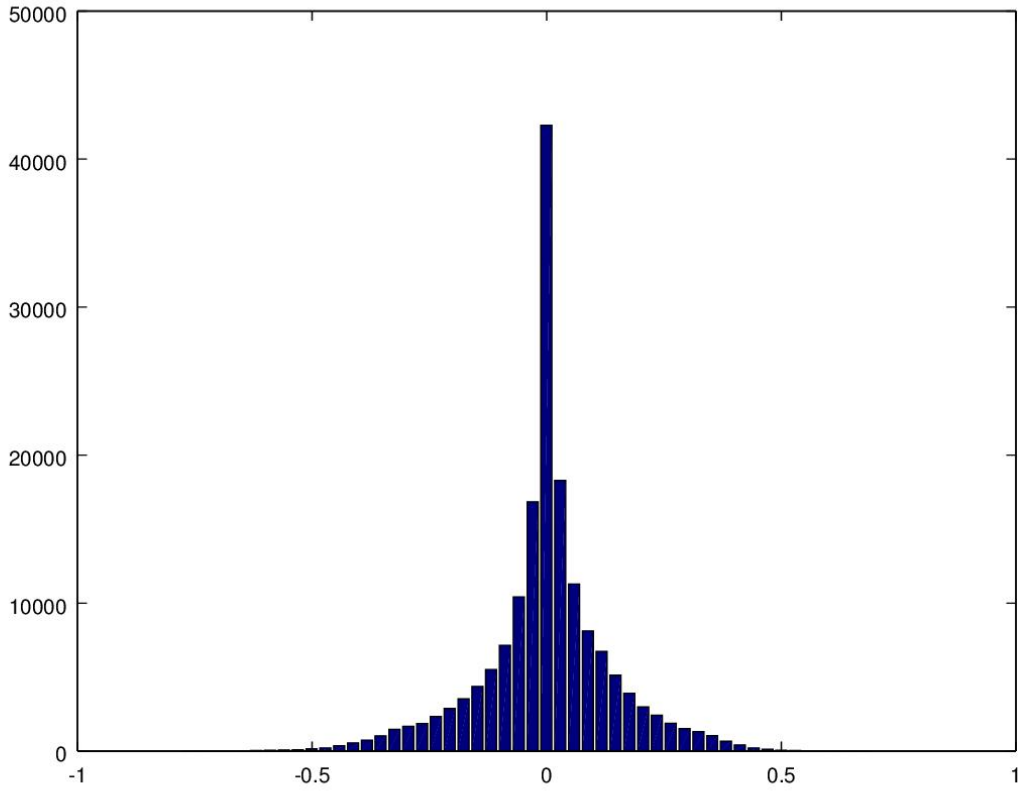
Lloyd-Max Quantizer

Look at the histogram of a typical mixed speech and music file in Python:

ipython --pylab

```
import scipy.io.wavfile as wav
rate, snd = wav.read('mixedshort.wav')
hist(snd/(2.0**15),50)
```

This means we count how often the signal samples in our signal “snd” fall into one of 50 bins between -max and +max amplitude, in this case between -1 and 1 (since we normalized it above). This is the result:



Observe: This distribution is far from being uniform! On the contrary, it is very “peaky”, with most samples being in bin 0, which means most samples have a very small amplitude (magnitude value).

Observe: A histogram becomes a **probability distribution** $p(x)$ (with x a signal value), if we divide it by the total number of samples in it, such that its sum becomes 1.

Idea: Wouldn't it be helpful if we choose our **quantzation steps smaller** where **signal samples appear most often**, to reduce the quantization error there, and make the quantization step size (and also the error) larger, where there are only a few samples?

This is the idea behind the Lloyd-Max quantizer (see also the Book: N.S. Jayant, P. Noll: “Digital coding of waveforms“).

Observe that this is not quite the same as for u-law companding. There, the **small** values get the smallest

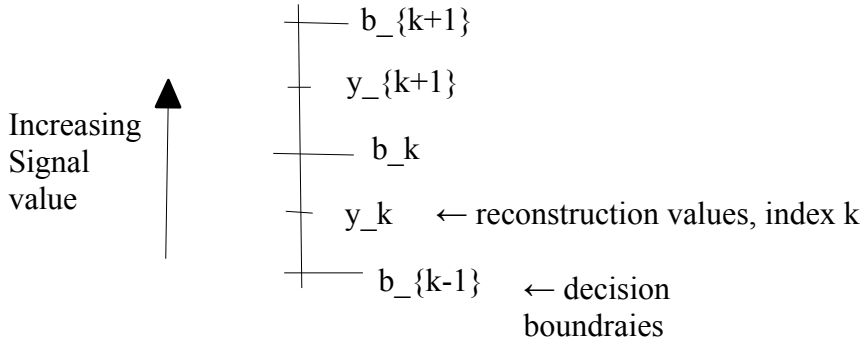
quantization step sizes, here, the **most likely** values get the smallest quantization steps sizes.

This is a type of non-uniform quantizer, which is adapted to the signals pdf. It basically minimizes the expectation of the quanization power (the expectaion of the squared signal, or its second moment), given the pdf of the signal to quantize. Let's call our Quantisation function Q(x) (this is quantization followed by reverse quantization). You can also think of non-uniform quantization as first applying this non-linear function and then to use uniform quantization. Then the expectation of our quantization power is

$$D = E\left(\left(x - Q\left(x\right)\right)^2\right)$$

Observe that we use the square here, and not for instance the magnitude of the error, because the square leads to an easier solution for minimum, which we would like to find.

Our **goal** is to **minimize this expectation** of the quantisation error power D.
Starting with the pdf of our signal, the result should be our quantisation intervals and reconstruction values. Since we now assume non-uniform intervals, we need to give those intervals and their reconstruction values names, which can be see in the following graphic



The encoder knows the b_k, and decides in which interval (what we called Δ before) the sample lies, and assigns the interval index k to it, as before (remember: only the index k is transmitted to the decoder). The decoder takes this index, and assigns the reconstructed value y_k to it, also as before.

We call b_k the decision boundaries, in the A/D converter or encoder (each interval gets an index as before), and on the decoding side we have the y_k as the reconstruction values for each index from the encoding side.
In the multidimensional case, they are also called a "**codeword**".

So using these definitions, and the pdf our the measured **probability distribution** of our signal p(x), we can re-write our equation for the error power or distortion:

$$D = E\left(\left(x - Q(x)\right)^2\right) = \int_{-\infty}^{\infty} \left(x - Q(x)\right)^2 p(x)\, dx =$$

we can now subdivide the integral over the quantisation intervals, assuming we have M quantization intervals, by just adding the quantization error power of all the quantisation intervals (see also: Wikipedia: quantization (signal processing)):

$$D = \sum_{k=1}^{M} \int_{b_{k-1}}^{b_k} \left(x - y_k\right)^2 p(x)\, dx$$

We would now like to have the minimum of this expression for the decision boundaries b_k and the reconstruction values y_k. Hence we need to take the first derivative of the distortion D with respect to those variables and obtain the zero point.

Lets start with the decision boundaries b_k:

$$\frac{\partial D}{\partial b_k}=0$$

To obtain this derivative, we could first solve the integral, over 2 neighbouring quantisation intervals, because each decision interval b_k appears in two intervals (one where it is the upper boundary, and one where it it the lower boundary).

$$D_k=\int_{b_k}^{b_{k+1}}(x-y_{k+1})^2\,p(x)\,dx+\int_{b_{k-1}}^{b_k}(x-y_k)^2\,p(x)\,dx$$

Here we cannot really get a closed form solution for a general probability function p(x). Hence, to simplify matters, we make the **assumption** that p(x) is **approximately constant** over our 2 neighbouring quantisation intervals. This means we assume that our quantisation intervals are small in comparison with the changes of p(x)! We need to keep this assumption in mind, because the derived algorithm is based on this assumption!

Hence we can set:

$$p(x)=p$$

Using this simplification we can now solve this integral:

$$\frac{D_k}{p}=\frac{(b_k-y_k)^3}{3}-\frac{(b_{k-1}-y_k)^3}{3}+\frac{(b_{k+1}-y_{k+1})^3}{3}-\frac{(b_k-y_{k+1})^3}{3}$$

Since we now have a closed form solution, we can easily take the derivative with respect to b_k (which only influences D_k in D, hence we can drop the k in the derivative):

$$\frac{\partial D/p}{\partial b_k}=(b_k-y_k)^2-(b_k-y_{k+1})^2$$

We can set this then to zero, and observing that $y_{k+1} > b_k$ (see above image), we can take the positive square root of both sides:

$$(b_k - y_k)^2 - (b_k - y_{k+1})^2 = 0$$

$$\rightarrow (b_k - y_k) = (y_{k+1} - b_k)$$

$$\rightarrow b_k = \frac{y_{k+1} + y_k}{2}$$

This means that we put our decision boundaries right in the middle of two reconstruction values. But remember, this is only optimal if we assume that the signals pdf is roughly constant over the 2 quantisation intervals! This approach is also called the "**nearest neighbour**", because any signal value or data point is always quantized to the **nearest reconstruction value**. This is one important result of this strategy.

Now we have the decision boundaries, but we still need the reconstruction values y_k . To obtain them, we can again take the derivative of D, and set it to zero. Here we cannot start with an assumption of a uniform pdf, because we would like to have a dependency on a non-uniform pdf. We could make this assumption before, because we only assumed it for the (small) quantisation intervals. This can be true in practice also for non-uniform pdf's, if we have enough quantisation intervals.
But to still have the dependency on the pdf, for the reconstruction values y_k we have to start at the beginning, take the derivative of the original formulation of D.

$$D=\sum_{k=1}^{M}\int_{b_{k-1}}^{b_k}(x-y_k)^2\,p(x)\,dx$$

Here we have the pdf p(x) and the reconstruction values (codewords) y_k. Now we start with taking the derivative with respect to the reconstruction value y_k and set it to 0:

$$0=\frac{\partial D}{\partial y_k}=-\sum_{k=1}^{M}\int_{b_{k-1}}^{b_k}2\cdot(x-y_k)\,p(x)\,dx$$

Since the y_k is only in 1 interval, the sum disappears:

$$0=\frac{\partial D}{\partial y_k}=-\int_{b_{k-1}}^{b_k}2\cdot(x-y_k)\,p(x)\,dx$$

Since we have a sum, we can split this integral in 2 parts (and remove the - sign):

$$0=\int_{b_{k-1}}^{b_k}2\cdot x\,p(x)\,dx-\int_{b_{k-1}}^{b_k}2\cdot y_k\,p(x)\,dx=$$

$$=\int_{b_{k-1}}^{b_k}x\cdot p(x)\,dx-y_k\cdot\int_{b_{k-1}}^{b_k}p(x)\,dx$$

Hence we get the result

$$y_k = \frac{\int_{b_{k-1}}^{b_k} x \cdot p(x)\,dx}{\int_{b_{k-1}}^{b_k} p(x)\,dx}$$

Observe that we now got a result without making any assumptions on p(x).

This can be interpreted as a **conditional expectation** of our signal value over the quantization interval (given the signal is in this interval), or also its "**centroid**" as reconstruction value (codeword). The value in the numerator can be seen as the expectation value of our signal in the interval, and the denominator can be seen as the probability of that signal being in that interval! Hence it can be interpreted as: Given the signal is inside the interval, this is its average or expected value.

Since the decision boundaries b_k depend on the reconstruction values y_k, and the y_k in turn depend on the b_k, we need to come up with a way to compute them. The approach for this is an **iterative algorithm**.

This could look like the following:

1) Decide on M, start (initialize the iteration) with a **random** assignment of M **reconstruction values** (codewords) y_k

2) Using the reconstruction values y_k , **compute** the **boundary values** b_k as mid-points between 2 reconstruction values / codewords (**nearest neighbour rule**)

3) Using the pdf of our signal and the boundary values b_k , update, **compute new reconstruction values (codewords)** y_k as **centroids** or **conditional expectation** over the quantisation areas between b_k and b_k-1
4) Go to 2) until update is sufficiently small (< epsilon)

This algorithm usually converges (it finds an equilibrium and doesn't change anymore), and it results in the minimum distortion D.

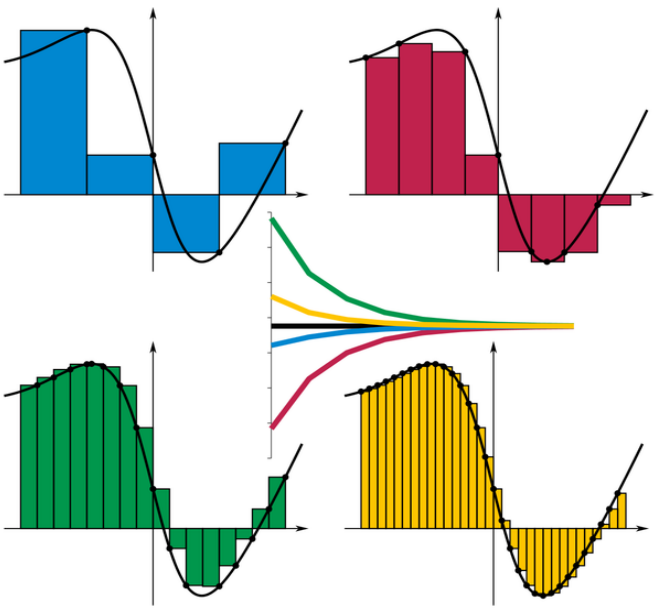
Hint for numerical integration: An integral of the form

$$I(k)=\int_{b_{k-1}}^{b_k} f(x)\,dx$$

can be approximated in Python using a sum. For instance we could divide the integration interval into 100 steps in Python:

dx=(b[k]-b[k-1])/100;

An interval at some point x' can then be seen as a small rectangle with area $f(x')\cdot dx$. The integral is then approximated as the sum of the areas of all these rectangles, as illustrated in following image:



(from: http://en.wikipedia.org/wiki/Integral)

The vector of all function values in the interval b(k-1) to b(k) with spacing dx in Python is:

f(arange(b[k-1],b[k],dx))

for instance:

```
sin(arange(0,3.14,0.1))
```

Hence the integral becomes

I[k]=sum(f(arange(b[k-1]:b[k]:dx))*dx)

example:

```
sum(sin(arange(0,3.14,0.1))*0.1)
Out: 1.9995479597125976
```

In the last example you saw the numerical integration if sin(x) from 0 to pi, which indeed is approximately correct!

Example 1 for Max-Lloyd Iteration

Assume we have a signal x between $0 \le x \le 1$, uniformly distributed (p(x)=1 on this interval) and we want to have 2 reconstruction values/ codewords y_k , and hence 3 boundaries b_k (where $b_0 = 0$ and $b_2 = 1$), we need to find only b_1 .

1) **Random initialization**: $y_1 = 0.3$, $y_2 = 0.8$

2) **Nearest neighbour**: $b_1 = (0.3 + 0.8)/2 = 0.55$

3) **Conditional expectation**:

$$y_k = \frac{\int_{b_{k-1}}^{b_k} x \cdot p(x)\,dx}{\int_{b_{k-1}}^{b_k} p(x)\,dx}$$

now we use that $p(x) = 1$:

$$y_1 = \frac{\int_0^{0.55} x\,dx}{\int_0^{0.55} 1\,dx} = \frac{0.55^2/2}{0.55} = 0.275$$

$$y_2 = \frac{\int_{0.55}^{1} x\,dx}{\int_{0.55}^{1} 1\,dx} = \frac{1/2 - 0.55^2/2}{1 - 0.55} = 0.775$$

4) Go to 2), **nearest neighbour**:

$$b_1 = (0.275 + 0.775)/2 = 0.525$$

3) **Conditional expectation**:

$$y_1 = \frac{0.525^2/2}{0.525} = 0.26250$$

$$y_2 = \frac{1/2 - 0.525^2/2}{1 - 0.525} = 0.76250$$

and so on until it doesn't change much any more. This should converge to $y_1=0.25$, $y_2=0.75$, and $b_1=0.5$.

Example 2: Like above, but now with a **non-uniform**, Laplacian pdf:

$$p(x)=e^{-0.5\cdot|x|}$$

1) **Random initialization**: $y_1=0.3$, $y_2=0.8$

2) **Nearest neighbour**: $b_1=(0.3+0.8)/2=0.55$

3) **Conditional expectation**:

$$y_k=\frac{\int_{b_{k-1}}^{b_k} x\cdot p(x)\,dx}{\int_{b_{k-1}}^{b_k} p(x)\,dx}$$

Now we need Python to compute the numerator integral, for y_1 :

$$\int_0^{b_1} x\cdot p(x)\,dx=\int_0^{0.55} x\cdot e^{-0.5\cdot|(x)|}\,dx$$

In Python we can use the function “scipy.integrate.quad” for integration (type “help(quad)” to get information about its use),

```
ipython -pylab
from scipy.integrate import quad
Num,Nerr=quad(lambda x: x*exp(-0.5*abs(x)),0,0.55)
Num
Out: 0.12618217155266 08
```

For the denominator integral we get,

$\int_0^{0.55} p(x)\,dx$, hence

```
Den,Derr=quad(lambda x: exp(-0.5*abs(x)),0,0.55)
Den
Out: 0.4808557535500631
```

and hence we obtain,

$$y_1 = \frac{Num}{Den} = \frac{0.12618}{0.48086} = 0.26240$$

For y_2 we get,

```
Num,Nerr=quad(lambda x: x*exp(-0.5*abs(x)),0.55,1)
print Num
0.234633870172
Den,Derr=quad(lambda x: exp(-0.5*abs(x)),0.55,1)
print Den
0.306082927025
Num/Den
ans =  0.766569610570364 4
```

Hence $y_2 = 0.76657$.

Go back from here to step 2 until convergence.

Here you can see that it can be worthwhile to encapsulate the iteration in a function. Then you can call the function again and again until it doesn't change anymore.