Digital Signal Processing 2/ Advanced Digital Signal Processing/
Audio-Video Signalverarbeitung
Lecture 1,
Organisation,
A/D conversion, Quantization

Gerald Schuller, TU Ilmenau

Gerald Schuller
gerald.schuller@tu-ilmenau.de

1. Organisation

-Lecture each week, 2SWS,
-Seminar every other week, 1 SWS
-Homework assignments, they will receive points over the course of the semester, and the sum of points will count towards the final grade. The homework is divided into programming assignments and quizzes in the online learning platform “Moodle 2” (moodle2.tu-ilmenau.de), in which you need to register.
There will also be a final written exam at the end of the semester.
-Homework will count for 30%, and final exam for 70% of the final grade. For the homework, quizzes will count for 25%.
You still need to pass the exam to pass the course.
-Web site: There is a Moodle 2 web site for the lecture, which contains the most current lecture slides, and the quizzes for them, and the seminar slides and assignemnts, and current announcements, under moolde2.tu-ilmenau.de- Fakultaet EI-Medientechnik- FG Angewandte Mediensysteme.

We also have an external Website, which will contain last years slides
https://www.tu-ilmenau.de/mt/lehrveranstaltungen/lehre-fuer-master-mt/adsp-digital-signal-procesing-2/

or clicking through in http://www.tu-ilmenau.de/mt
-Lehrveranstaltungen Master/Bacelor
-DSP2/ADSP.

We have 3-5 LP ("Leistungspunkte") for lecture and seminar, 1 LP means about 2-3 hours per week, together 8-12 hour/week.
ADSP will get an extra Homework assignment in the end.

Will will use **Python** for programming. The Open Source system **Python Pylab** is quite similar to Matlab/Octave and can be faster and closer to the hardware, so we will mostly use it. Octave and Python Pylab can be downloaded freely from the Internet for all operating systems. We assume some familiarity with Matlab, Octave or Python Pylab for this course.
For instance, from a Terminal, you start Python Pylab with the command: "**ipython –pylab**".
We recommend using the operating system **Linux**, for instance in a **dual-boot** configuration on your laptop or on an inexpensive **Raspberry Pi** one-board computer, because Python is pre-installed in it, libraries can be easily installed, and it is all together more flawless, because Linux is made for programming. For instance, for installing the pylab library, you open a **terminal window** (e.g. using the shortcut alt-ctl-T) and type:

```
sudo apt install python.pylab
```

(in MacOS, pkgsrc might work for apt).
Since we use Python, we will also allow **programmable scientific pocket calculators** in the final exams.

Observe: The lecture slides are not a replacement of **books** on the subject, but something like a help to explain a topic, and something like a protocol for topics and questions treated. References to books are given throughout the lecture.
The slides are kept in **editing mode** on purpose. In this way they work like a **blackboard**, and answers to questions can be included immediately.
The main **purpose** of the lecture is to "**tailor**" the lecture content to your background and to **answer questions.** I will read parts of the slides, so that you can then **ask questions** about it, or such that I can further explain a topic according to my guess of your background.
Hence it is best to read the **slides before** the lecture (the previous version is online) and to think about possible questions for the lecture, whose answer I then include in the updated version.

2. Introduction

Definitions:

(Advanced) Digital Signal Processing 2:

-**Digital**: Quantization, sampling, encoding, digital filters, Nyquist
-**Signal**: Discrete Time, continuous time, TDMA, CDMA, OFDM, Voice, electrical current, white noise, colored noise,

Sound, audio signals, electromagnetic waves, video signals, X-Ray, EEG (brain waves), Heart beat waves (ECG), Seismic waves.
Processing: Fourier Transform, Fourier Series, Cosine/Sine Transform, Filters, z-Transform, Discrete Fourier Transform (DFT), Fast Fourier Transform (FFT), upsampling, downsampling, encoding/decoding, quantization, modulation, denoising,

Part 2/Advanced: Previously learned? Which to start from? Your background? Media Technology: Multrate Signal Processing, CSP: Statistics, DFT, sampling quatization, Fourier Series, analog signal processing.

Application Example

ITU G.711 speech coding.
This is the first standard for ISDN speech coding, for speech transmission at **64 kb/s**, with telephone speech quality. It uses a **sampling rate of 8000 samples/second or 8 kHz**, and **8 bits/sample** (hence we get 8*8=64 kbits/s). To obtain the 8 bits/sample, it uses companding, A-law or mu-law **companding**, to obtain 8 bits/sample from originally 16 bits/sample, with still acceptable speech quality.
(ITU means International Telecommunications Unions, it standardizes speech communications)

The procedure of G.711 can be seen as the following

-Microphone input
-Analog-to-Digital converter (A/D converter), including sampling at 8 kHz sampling rate. It usually generates 14-16 bits/sample. Before the sampling there is an (analog) low pass filter with cut-off frequency of 3.4 kHz.
-Companding (compressing) with an A-Law or u-Law function before 8-bit quantization, like

$$y=\frac{\ln(1+\mu x)}{\ln(1+\mu)}$$ with $\mu=255$ for the North American standard, which generates 8 bits/sample from the original 16 bits/sample (see eg. http://www.dspguide.com/ch22/5.htm or the **Book: Jayant, Noll: "Digital Coding of Waveforms"**, Prentice Hall)
-quantization with uniform quantization step size (for 8 bit), turn a level or value into an 8 bit codeword.
-Transmission to the receiver
In the receiver:
-De-quantization, turn an 8-bit codeword into a level or value
-Expanding the signal using the inverse A-Law or u-Law function after inverse quantization, to obtain the original 16 bits/sample range
-Digital to Analog conversion, including low pass filtering (about 3.4 kHz).

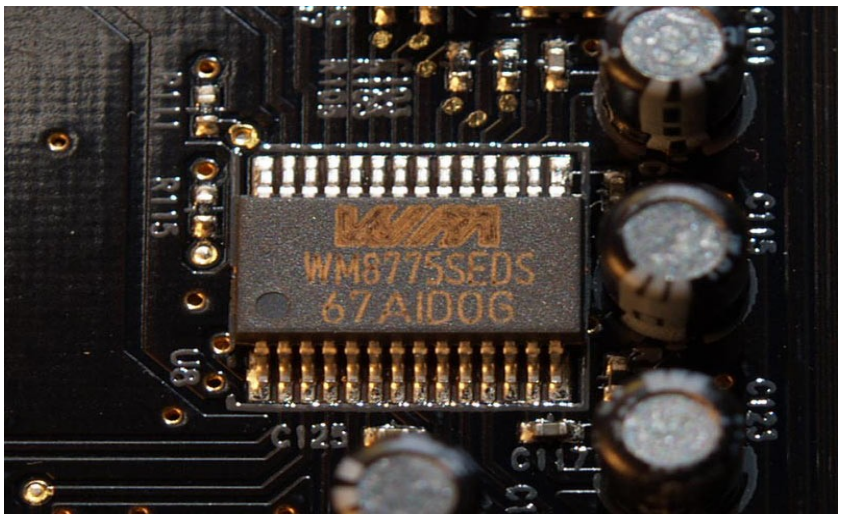
This system allow us to have a **telephone conversation** over a digital ISDN line at **64 kb/s**.
This system was standardized in the 1970's, and at that time the hardware did not allow for much more than companding. Today we have much more powerful hardware (Moore's

Law), and hence we can devise much more powerful compression algorithms. Today, at 64 kb/s, we can get very high quality speech, instead of just telephone quality speech. Current speech coders allow for audio bandwidths of above 10 kHz at that bit rate, which means speech sounds like if we are right there with the speaker. Also, **at 64 kb/s**, we can transmit high quality **audio/music** signals, using for instance, the **MPEG-AAC** standard, which is also used in iTunes. This, the transmission of music over ISDN lines, was actually the original motivation for the development of MPEG audio compression. ISDN lines are outdated nowadays, but compression remains useful, for instance for wireless connections (downloading or streaming to your wireless phone).
Observe: ISDN has a fixed bit rate (64kb/s), and has a fixed connection. You dial to the person you would like to talk to, and have a fixed connection, and each bit you generate is send off to the receiver right away without delay.
In contrast: The **internet has no fixed bit rate**, it depends on the individual connection, and there is also no fixed connection. There, first we assemble a certain number of bits into **packets**, these packets get headers with the receiver address, and then the switches in the internet rout the packet to the receiver. Observe that this also leads to delay, first for the assembling of the packet, and then for the switching in the internet. This also shows that originally, the internet was not made for real time communications, but just for data traffic. But the internet is becoming faster, with less delay, because it is constantly expanding, for higher bit rates.

3. A/D and D/A Conversion of a Signal, Quantization

Example:

4-channel stereo multiplexed analog-to-digital converter WM8775SEDS made by Wolfson Microelectronics placed on an X-Fi Fatal1ty Pro sound card. From: Wikipedia, Analog-to-digital converter.
This is on the sound card of your computer, where you connect your microphone and speakers to.



This A/D converter measures the voltage at its input and assigns it to an index or codeword. A Python example is: Assume our A/D converter has an input range of -1V to 1V, 4 bit accuracy (meaning we have a total of 2^4 codewords or indices), and the A/D converter has **0.2 V** at its **input**. One

possibility to obtain the **quantization stepsize** or **quantization interval** (in **Python**) is:
Open Terminal, type:

```
python
stepsize=(1.0-(-1.0))/(2**4) # or pow(2,4)
stepsize
```

```
0.12500
```

Next we get **quantization index** which is then encoded as a **codeword**:

```
index=  round(0.2/stepsize);
```

which results in:

```
index
```

```
2
```

Observe: If the quantization **stepsize is constant**, independent of the signal, we call it a "**uniform quantizer**"
The index then is **coded** using the 4 bits and sent to a decoder, for instance using the 4 bit binary **codeword** "0010". The first bit usually is the sign bit. The **decoder reconstructs** the voltage by first decoding the codeword to an index, and for instance by **multiplying the index** with the **stepsize**:

```
reconstr=stepsize*index
```

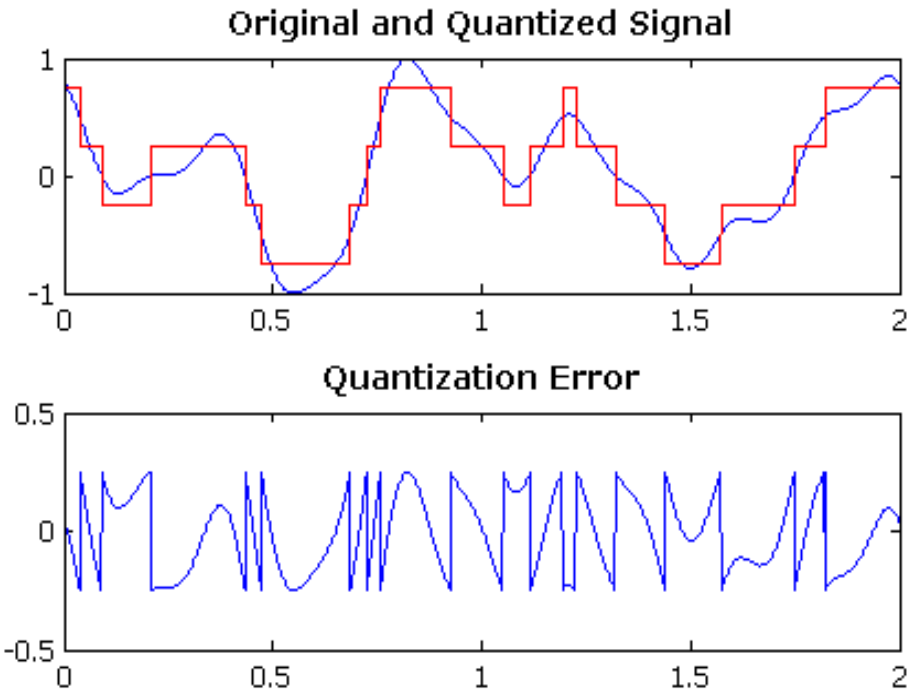
resulting in:

```
reconstr
```

```
  0.25000 V
```

This is also called the **(de-)quantized signal**, and its difference to the original value or signal is called the **quantization error**.
In our example the quantization error is
Quantized Value – Original Value = 0.25V-0.2V=**0.05 V**

Observe: There is always a range of voltages which is mapped to the same codeword. We call this range Δ , or **stepsize**. These steps represent the quantization in the A/D conversion process, and they lead to quantization errors.
The output after quantization is a linear "**Pulse Code Modulation**" (**PCM**) signal. It is linear in the sense that the code values are proportional to the input signal values.



Quantization error for a full range signal. Quantization and De-Quantization included.
(From: Wikipedia, quantization error, the red line shows the quantized signal, the blue line in the above image is the original analog signal)

Let's now call our quantization error "e". Then the **quantization error power** is the **expectation** value of the squared quantization error e:

$$E(e^2) = \int_{-\Delta/2}^{\Delta/2} e^2 \cdot p(e)\, de$$

where $p(e)$ is the probability of error value e. Here we compute the power of each possible error value e by squaring it, and multiply it with its probability to obtain the **average power**.

This number will give us some impression of the signal quality after quantization, if we set it in **relation to the signal energy**. Then we get a **Signal to Noise Ratio** (SNR) for our quantizer and A/D converter.
Assume the quantization error e is uniformly distributed (all possible values of the quantization error e appear with equal probability), which is usually the case if the signal is much larger than the quantization step size Δ (large signal condition). Since the integral over the probabilities of all possible values of e must be 1, and the possible values of e are between $-\Delta/2$ and $+\Delta/2$,

$$1 = \int_{-\Delta/2}^{\Delta/2} p(e)\, de = p \cdot \int_{-\Delta/2}^{\Delta/2} de = p \cdot \Delta$$

we have

$$p(e) = 1/\Delta$$

which yields

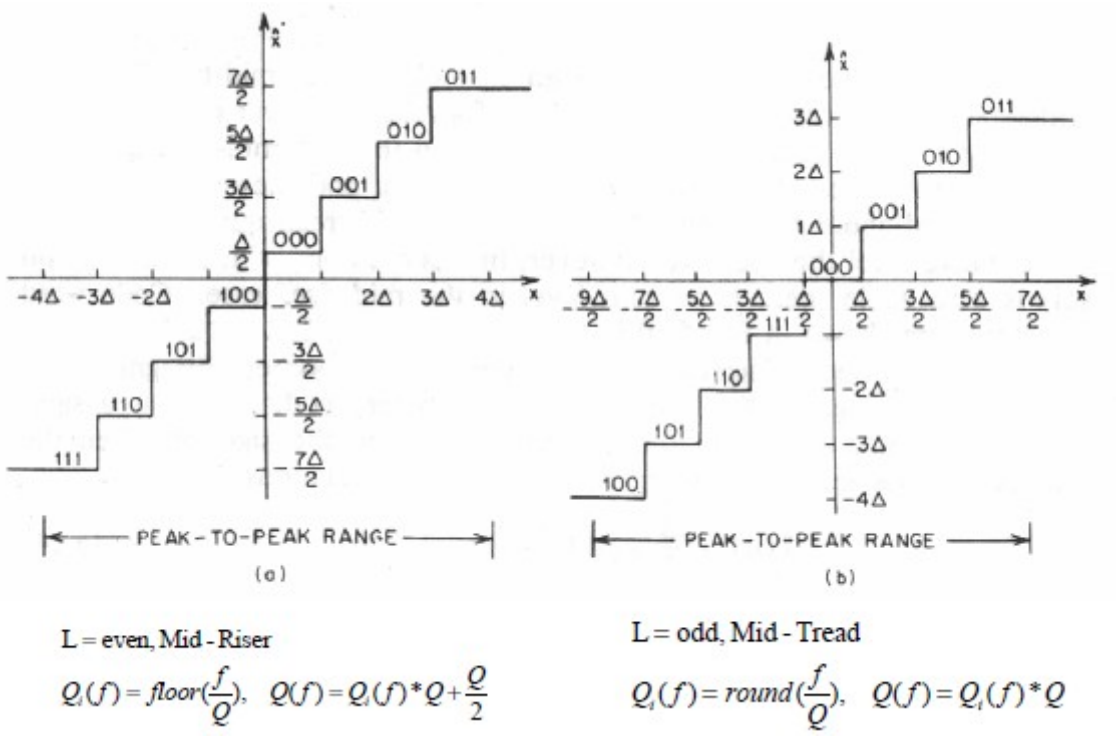
$$E(e^2) = \frac{1}{\Delta} \cdot \int_{-\Delta/2}^{\Delta/2} e^2\, de = \frac{1}{\Delta}\left(\frac{(\Delta/2)^3}{3} - \frac{(-\Delta/2)^3}{3} \right) = \frac{\Delta^2}{12}$$

Hence the **quantization error power for a uniform quantizer** with stepsize Δ and with a large signal is:

$$E\left(e^2\right)=\frac{\Delta^2}{12}$$

Mid-rise and Mid-tread quantization

Depending on if the quantizer has the input voltage 0 at the center of a quantization interval or on the boundary of that interval, we call the quantizer a mid-tread or a mid rise quantiser, as illustrated in the following picture:



L = even, Mid-Riser

$Q_i(f) = floor(\frac{f}{Q}), \quad Q(f) = Q_i(f) * Q + \frac{Q}{2}$

L = odd, Mid-Tread

$Q_i(f) = round(\frac{f}{Q}), \quad Q(f) = Q_i(f) * Q$

(From: http://eeweb.poly.edu/~yao/EE3414/quantization.pdf)
See also: https://en.wikipedia.org/wiki/Quantization_%28signal_processing%29

Here, $Q_i(f)$ is the index after quantization (which is then encoded and send to the receiver), and Q(f) is the de-quantization, which produces the quantized reconstructed value at the receiver.

This makes mainly a difference at **very small input values**. For the mid-rise quantiser, very small values are always quantized to +/- half the quantization interval ($+/-\Delta/2$), whereas for the mid-tread quantizer, very small input values are always rounded to zero. You can also think about the **mid-rise** quantizer as **not having a zero** as a reconstruction value, but only very small positive and negative values.
So the mid-rise can be seen as more accurate, because it also reacts to very small input values, and the mid tread can be seen as saving bit-rate because it always quantizes very small values to zero.
Observe that the expectation of the quantization error power for large signals stays the same for both types.

Python Example:
Open a terminal, type:

```
python
import numpy as np
q=0.1; #Stepsize
x = np.array([0.012,-1.234, 2.456, -
3.789])
x
   0.012000  -1.234000   2.456000  -
3.789000
```

Hint: In case the library numpy is not installed, you can install it under **Linux** with:

```
sudo apt install python.numpy
```

or using the pip command:

```
pip install numpy
```

In the same way other missing libraries can be easily installed.

Mid-Tread quantizer (Encoder):

```
import numpy as np
index=np.round(x/q)
index =
    0  -12   25  -38
```

De-quantization (Decoder):

```
reconstr=index*q
reconstr
   0.00000  -1.20000   2.50000  -3.80000
```

Mid-Rise quantizer (Encoder):

```
index=np.floor(x/q)
index
    0  -13   24  -38
```

De-quantization (Decoder):

```
reconstr=index*q +q/2
reconstr
   0.050000  -1.250000   2.450000  -
3.750000
```

Real-time audio python example: The audio signal is between -32000 and +32000, and the quantization step size is q=4096. Start it in a terminal with:

```
python pyrecplay_quantizationblock.py
```
Observe:
the Mid-Tread quantizer “swallows” small signal levels, since they are all rounded to zero.
The Mid-Rise quantizer still captures small levels, but distorted.