Design Example: OFDM

1 Introduction

In this lecture we will have a look at some of the principles used in modem design for Digital Subscriber Line (DSL) communication. Asymetric Digital Subscriber Line (ADSL), which is a specific instance of DSL technology, is currently one of the main techniques used to provide broadband internet access.

DSL operates over twisted pair copper telephone lines. Telephone lines have originally been used for transmission of voice only, using the spectrum up to 3.4kHz. DSL shares the telephone line with regular voice transmission, but operates at frequencies roughly from 10kHz up until 1.1MHz.

Transmitting over a twisted pair copper line requires digital-to-analog conversion at the transmitter and analog-to-digital conversion an the receiver. We will consider only the discrete-time domain and treat A/D and D/A conversion as a part of our communication channel.

If one transmits over a twisted pair copper telephone line at high frequencies, the signal will get corrupted. Since we transmit over a fixed connection, we can assume that the system is time-invariant. It turns out that we can also use the assumption the the system is linear. Our time-discrete channel is therefore a linear time-invariant (LTI) system. We will interchangeably use the terms system and channel.

One of the techniques that are used to transmit over LTI channels is Orthogonal Frequency Division Multiplexing (OFDM). This technique is also known as Discrete Multi-Tone (DMT). In this lecture we will look at the principles behind OFDM. We will also give a complete MATLAB implementation of these ideas.

2 Problem

We will design a DSL communication system. As seen in the introduction we can model the communication channel as a Linear Time-Invariant system, see Figure 1.

Let $h[n]$ be the impulse response of the LTI channel. We assume that it is of finite-length, i.e. there is a ν such that

$$h[n] = 0, \quad \text{for } n < 0, \text{and } n > \nu - 1.$$

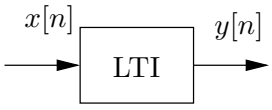

Figure 1: DSL communication channel modelled as a linear time-invariant system.

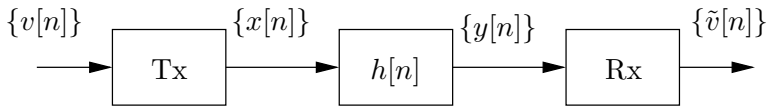

Figure 2: Tranmission of a block of symbols $\{v[n]\}$. The transmitter (Tx) produces N symbols $\{x[n]\}$ that are sent through the channel. At the output of the channel, the receiver (Rx) takes $\{y[n]\}$ and produces $\{\tilde{v}[n]\}$.

We also assume that the receiver knows $h[n]$. In the example and MATLAB implementation we consider in this handout, we have a channel for which $\nu = 128$.

The specific problem that we will consider is that of transmission of a sound signal using a DSL system. We denote the sequence that is to be transmitted by $\ldots$, $v[-2]$, $v[-1]$, $v[0]$, $v[1]$, $v[2]$,$\ldots$

We will split up this sequence in blocks of N symbols. For each block of symbols, say the symbols $v[0], \ldots, v[N-1]$, a transmitter produces N symbols $x[0], \ldots, x[N-1]$ that are sent through the channel. At the output of the channel, a receiver takes $y[0], \ldots, y[N-1]$ and produces $\tilde{v}[0], \ldots, \tilde{v}[N-1]$, see Figure 2. We use the notation $\{v[n]\}$ to denote the block $v[0], \ldots, v[N-1]$.

We restrict ourselves to transmitter and receiver structures that produce one input symbol to the channel for each symbol of the sound sequence. One can think of this as streaming of audio over the LTI channel. Implicitely we consider the audio samples arriving at the transmitter at a certain rate (the number of symbols per second). If we want to have any hope of successfully streaming the sequence over the channel, we can not send more than one channel input for each audio symbol.

A trivial transmitter and receiver would directly sent each incoming symbol over the channel and directly use it, i.e. $\{x[n]\} = \{v[n]\}$ and $\{\tilde{v}[n]\} = \{y[n]\}$. The MATLAB implementations for these trivial transmitter and receiver are given in Figure 5. The code to test this system is given in Figures 3 and 4.

If we run this code and listen to the received sound sequence $\tilde{v}[n]$, we notice that it is corrupted. This should not be surprising, since we know that if $y[n] = h[n] * x[n]$, then

$$Y\left(e^{j\omega}\right) = H\left(e^{j\omega}\right) X\left(e^{j\omega}\right).$$

This means that the different frequency components in the signal $x[n]$, get attenuated differently.

In the next sections we will develop a different transmitter/receiver structure.

3 Basic Idea

The basic idea of our new transmitter and receiver is to use the fact that complex exponentials are eigenfunctions of LTI systems. This follows immediately from the following DTFT relation

$$x[n] * h[n] \Leftrightarrow X\left(e^{j\omega}\right) H\left(e^{j\omega}\right). \tag{1}$$

Suppose we want to transmit a single value v. If we let $x[n] = \text{DTFT}^{-1}\{v\delta(\omega - \omega_0)\}$, with ω_0 arbitrarily chosen, we have

$$y[n] = x[n] * h[n]$$

and

$$Y\left(e^{j\omega}\right) = H\left(e^{j\omega_0}\right) v\delta(\omega - \omega_0). \tag{2}$$

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Loading audio
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Load the sound sequence that we will
% use for testing.
[v,fs] = wavread('handel.wav');

% Let's listen to it...
soundsc(v,fs);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Divide the sequence in blocks
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% We choose to use a blocklength
% of N = 1024.
N = 1024;

% Add zeros to the sound sequence such that the
% total length is a multiple of N.
v = [v; zeros(1024-mod(length(v),N),1)];

% Divide the sequence in blocks.
% Each column of v_block is a block of N symbols
v_block = reshape(v,N,length(v)/N);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% The channel
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
impulse_response = [ <<removed for handout>> ];
```

Figure 3: MATLAB implementation: Initialization. Loading audio and dividing it into blocks. The channel impulse response is known at the receiver, so we give it here.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Transmit the signal over the channel. We use a
% trivial transmitter and receiver.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initialization
x = zeros(size(v_block,1),1);
y = zeros(size(v_block,1),1);
v_tilde_block = zeros(size(v_block));
% Now do all operations per block
for i=1:size(v_block,2)
x = trivial_transmitter(v_block(:,i));
y = lti_channel(x);
v_tilde_block(:,i) = trivial_receiver(y);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Check the result.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Put all blocks back together in one sequence.
v_tilde = reshape(v_tilde_block,size(v));

% ... and listen to what we received.
soundsc(v_tilde,fs);
```

Figure 4: MATLAB implementation: Transmission using the trivial transmitter and receiver from Figure 5.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Trivial transmitter.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function x = trivial_transmitter(v)
x = v;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Trivial receiver.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function v_tilde = trivial_receiver(y)
v_tilde = y;
```

Figure 5: MATLAB implementation: A trivial transmitter and receiver.

From (2) we see that we can recover v as

$$\tilde{v} = H\left(e^{j\omega_0}\right)^{-1} \text{DTFT}\left\{y[n]\right\}.$$

Building on this idea, if we want to transmit $v[0]$, ..., $v[N-1]$ we can select some frequencies ω_0, ..., ω_{N-1} and let

$$x[n] = \text{DTFT}^{-1}\left\{\sum_{l=0}^{N-1} v[l]\delta(\omega - \omega_l)\right\}. \tag{3}$$

After receiving $y[n]$ we evaluate the $Y(e^{j\omega})$ at the points $\omega_0, \ldots, \omega_{N-1}$ to get

$$Y\left(e^{j\omega_l}\right) = H\left(e^{j\omega_l}\right) v[l], \qquad l = 0, \ldots, N-1.$$

We see that we can find $v[0], \ldots, v[N-1]$ as

$$v[l] = H\left(e^{j\omega_l}\right)^{-1} \text{DTFT}\left\{y[n]\right\}, \qquad l = 0, \ldots, N-1.$$

There is a fundamental problem with the idea presented above. The sequence $x[n]$ in (3) is of infinite length and can never be transmitted over the LTI channel. We see that the above is a nice mathematical construction, but it can not be implemented. In the next section, however, we will see that we can obtain similar results if we start from the fact that $x[n]$ is a finite-length sequence.

4 Finite Length Inputs

The DTFT equivalent for finite-length sequences is the DFT. The DFT equivalent to relation (1) for N-length sequences is

$$x'[n] \circledast_N h[n] \Leftrightarrow X'[k]H[k], \tag{4}$$

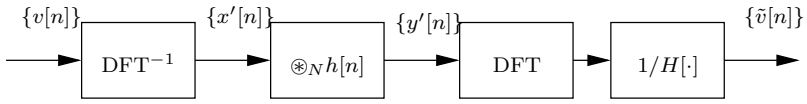

Figure 6: Basic idea of the communication system, based on DFT relation (4).

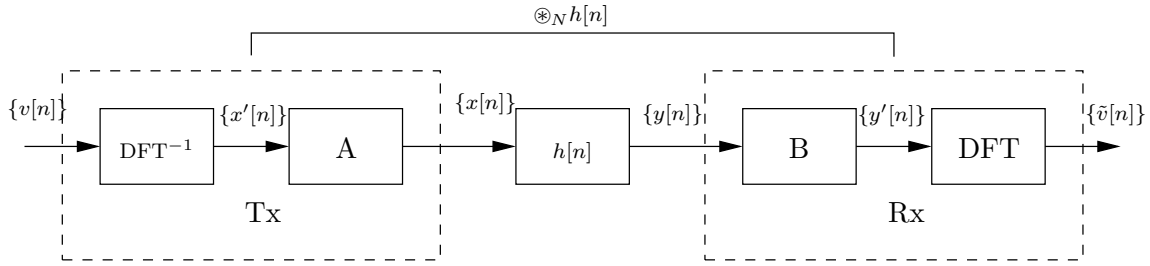

Figure 7: Expansion of Figure 6. The transmitter and receiver perform additional processing A and B in order to get an overall behavior of circular convolution.

where $\circledast_N$ is the circular convolution. The idea is to exploit relation (4) in a way as depicted in Figure 6. We see that if the LTI channel would perform a circular convolution, we would be done. The channel, however, performs a linear convolution, i.e. $y[n] = x[h] * h[n]$. Therefore, we need to perform a trick to get an overall behavior of a circular convolution. See Figure 7, where the circular convolution comes from the LTI channel together with some additional processing in the transmitter (A) and in the receiver (B). We will now analyze how to implement the systems A and B in order to get an overall circular convolution.

This is where we need the assumption that $h[n]$ has finite length. Remember from Section 2 that

$$h[n] = 0, \quad \text{for } n < 0, \text{and } n > \nu - 1.$$

We have

$$y'[l] = x'[l] \circledast_N h'[l] = \sum_{k=0}^{\nu-1} h[k] x'[(l-k)_N], \tag{5}$$

$l = 0, \ldots, N-1$, where

$$(l-k)_N = \begin{cases} l-k, & \text{if } l-k \geq 0 \\ N+l-k, & \text{if } l-k < 0. \end{cases}$$

Now, the trick is to add a *cyclic prefix*. If $x'[0] \ldots x'[N-1]$ is the output of the inverse DFT in Figure 7, we send $\nu - 1$ additional symbols over the channel. To simplify notation we start indexing $\{x[n]\}$ at $-(\nu-1)$, we have $\{x[n]\} = x[-\nu] \ldots x[N-1]$. If we now let

$$x[l] = \begin{cases} x'[l], & \text{if } 0 \leq l \leq N-1 \\ x'[N+l], & \text{if } -(\nu-1) \leq l < 0, \end{cases}$$

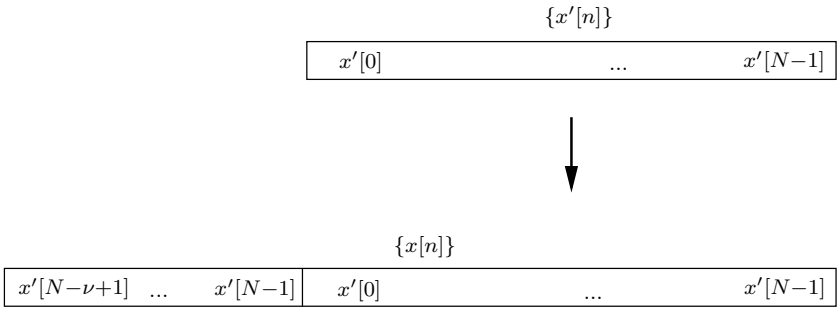

Figure 8: The cyclic prefix.

we see that at the output of the channel we have, for $l = 0, \ldots, N-1$.

$$\begin{aligned} y[l] &= h[l] * x[l] \\ &= \sum_{k=0}^{\nu-1} h[k]x[(l-k)] \\ &= \sum_{k=0}^{\nu-1} h[k]x'[[(l-k)_N] \\ &= x'[l] \circledast_N h'[l] \end{aligned}$$

The overall behavior is that of a circular convolution. Note that the receiver will only have to keep $y[0], \ldots, y[N-1]$ and process these symbols to retrieve $\{v[n]\}$. Figure 8 shows how the cyclic prefix is added.

There is one problem left to solve. In Section 2 we specified that the transmitter needs to create blocks of length N. In our current solution it creates an output of length $N + \nu$.

5 Downsampling

To make sure that our transmitter outputs a block of N samples we will downsample the sequence $\{v[n]\}$ before applying the inverse DFT.

We need to determine the downsampling factor. Let N' be the length of the downsampled block. After applying the cyclic prefix of length $\nu - 1$ we need a length N, i.e. we need $N' + \nu = N$. Therefore we need to resample with a factor

$$\left(1 - \frac{\nu - 1}{N}\right).$$

The receiver, after removing the cyclic prefix, performing the DFT and scaling according to $1/H[\cdot]$, resamples with a factor $1/(1 - (\nu - 1)/N)$.

Figures 9 and 10 give the complete MATLAB implementation of the transmitter and the receiver. The code from Figure 3 can be easily adjusted to use the new transmitter and receiver. If we know listen to the received sound sequence we notice that there is no audible difference with the original.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% OFDM transmitter.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function x = ofdm_transmitter(v,nu)

% The receiver needs nu as an additional argument
% to know how long the cyclic prefix needs to be

% Resampling
v_down = resample(v,7,8);
% (We could compute the factors 7/8 from the function arguments
%  but we cheat a bit here...)

% Inverse DFT
x_prime = ifft(v_down);

% Apply the cyclic prefix
x = [x_prime(end-nu+1:end); x_prime];
```

Figure 9: MATLAB implementation: OFDM transmitter.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% OFDM receiver.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function v_tilde = ofdm_receiver(y,nu,impulse_response)

% The OFDM receiver has 2 additional arguments, the length
% of the cyclic prefix and the impulse response of the
% channel.

% Remove the cyclic prefix
y_cut = y(nu+1:end);

% DFT
Y = fft(y_cut);

% Scale
H = fft(impulse_response,length(Y));
v_tilde_down = Y./H.';
% ...we are still working in the "downsampled regime"

% We know that v_tilde_down is real, but MATLAB keeps it as a
% complex variable.
v_tilde_down = real(v_tilde_down);

v_tilde = resample(v_tilde_down,8,7);
v_tilde = v_tilde(1:1024);
% (Again, we could have computed the upsampling factor and the
% block length from the function arguments, but we cheat a bit
% to keep the code easy.)
```

Figure 10: MATLAB implementation: OFDM receiver.