Acoustic modem project

Session 5: OFDM over the acoustic channel: channel estimation and equalization

Paschalis Tsiaflakis, Hanne Deprez [1]

Goal: Using the OFDM modem over an acoustic channel. Estimating and equalizing the channel response based on training symbols.

Requirements: Matlab/Simulink in Windows, a sound card, a loudspeaker and a microphone.

Required files from DSP-CIS website: *recplay.mdl* (see session 1)

Required files from previous sessions: *initparams.m*, *qam_mod.m*, *qam_demod.m*, *ofdm_mod.m*, *ofdm_demod.m*, *ber.m*

Outcome: 4 m-files (+1 elite) *ofdm_channelest.m*, *ofdm_demod.m* (updated), *initparams.m* (updated), *alignIO.m*, *ofdm_channelest_pilots.m* (optional)

Deliverables: See session 5

1 Exercise 5-1 (Core): Using training frames for channel estimation and equalization

Following exercise 4-2 (point 6), it should be clear that an OFDM modem has to include a compensation for the channel filtering operation. This compensation is often referred to as channel equalization. In session 4, we have constructed the equalizer based on an a-priori known channel response. However, in real-life applications, this channel response is not known and needs to be estimated. In session 2, we have estimated the channel impulse response (time-domain), based on a least-squares (LS) procedure (remember the Toeplitz matrix?). Fortunately, this can be done more efficiently inside an OFDM framework, based on a frame-based frequency domain procedure, avoiding the computationally-intensive time-domain LS estimation from session 2. Channel estimation in OFDM uses so-called training frames, which consist of training (QAM) symbols that are known to both the transmitter and receiver.

Exercise: create an m-file *ofdm_channelest.m* in which the following steps are implemented:

1. Let N denote the DFT-size that will be used by your modem (this variable should be set in the beginning of the m-file, make sure you can easily change it later). Generate a vector `trainblock` that contains $\frac{N}{2} - 1$ random QAM symbols (e.g., use *qam_mod.m* on a random bit stream).

[1] Post your questions and remarks on the Discussion Board of toledo (English course) or look at question that are already there.

2. Using *ofdm_mod.m*, generate an OFDM time-domain signal `Tx` of length $100(N+L)$, in which all OFDM frames are identical and contain the QAM symbols from `trainblock`.

3. Filter (=convolve) the signal `Tx` with the acoustic channel impulse response (saved in vector `h`) that was measured in session 2 based on *IR2.m*. Save this (simulated) received signal in the variable `Rx`.

4. Modify the function *ofdm_demod.m* such that it estimates the channel frequency response in the frequency domain, based on `trainblock` and the received signal `Rx` (use a separate least-squares procedure for each frequency point). Use this estimate to construct a channel equalizer and apply it to the received data in `Rx` (see also exercise 4-2, point 6).

   The function *ofdm_demod.m* should return the following outputs: the demodulated QAM-symbol stream and the estimate of the channel frequency response.

5. Generate a figure with 2 subplots containing 1) the acoustic impulse response `h` (see step 3), and 2) its frequency response.

6. Generate a second figure with 2 subplots containing 1) the estimated channel impulse response (after transformation from the frequency domain to the time domain), and 2) the estimated channel frequency response.

7. Use *qam_demod.m* to generate the demodulated bitstream and compare with the initial random bitstream using *ber.m*.

Only if the BER is equal to zero, and if the generated figures are the same, you can proceed to the next exercise! (the frequency response on both figures should be *exactly* the same, except for the two frequency bins which are set to zero)

2 Exercise 5-2 (Core): OFDM over the acoustic channel

1. First, we need an automatic synchronization procedure that aligns the audio input and output:

   - Modify the function *[simin,nbsecs,fs]=initparams(toplay,fs)* such that it adds a synchronization pulse (= the signal *pulse*) in front of the signal *toplay* (after the 2 seconds of silence). Also add as many zeros as the length of the IR after the pulse to avoid that its response mixes up with the signal. The choice for the shape of the pulse is up to you.
   - Create a Matlab function *[out_aligned]=alignIO(out,pulse)* that aligns the output signal out with the signal *pulse*. This function calculates

the maximum of the cross-correlation function of the two signals in order to determine the delay introduced in the output signal. This information is used to synchronize the two signals.

- The output signal *out_aligned* contains the vector *out* where the silence period and the synchronization pulse are cut off. However, make sure that you do not cut off anything from the actual data samples, i.e., leave a safety margin of about 20 samples! Basically, this means that the first data sample of *toplay* should (more or less) correspond to sample 20 of the signal *out*.
- Explain why this alignment is important for the modem (see also exercise 2-2).
- An alternative for estimating the delay is to detect the synchronization pulse, based on a user-defined threshold. In this case the function becomes *[out_aligned]=alignIO(out,threshold)*. What could be the disadvantage of this approach?

2. Modify your m-file *ofdm_channelest.m* from exercise 5-1 as follows.
   - Feed the signal `Tx` to the function *initparams* as the input *toplay*. Run the simulink model *recplay.m*. Align the microphone output signal by applying the function *alignIO.m*. The resulting signal vector is stored in `Rx`.
   - Besides the generation of `Rx`, everything remains the same as in the previous exercise. Only the figure showing the estimated channel responses is plotted (since the true channel responses are not known).
   - Make a pragmatic choice for the DFT-size, the CP length, and the QAM constellation.
3. Run the *ofdm_channelest.m*, and check the channel impulse response and frequency response. Does it look ok? Also, check the BER. Try to optimize your settings for the DFT-size, the CP length, and the QAM constellation.
4. Which method do you prefer for channel estimation: the one used in this exercise (frequency domain), or the time-domain estimation with the Toeplitz matrix (in terms of memory usage, computational complexity, and performance).
5. What happens if you put your hand in front of the microphone? How does this affect the channel response? How does this affect the BER? Try to answer first, then confirm experimentally.
6. How does the sampling frequency affect the performance of your modem?

3 Exercise 5-3 (Elite): Channel estimation based on pilot tones

The Elite part(s) are good for 20% of the grades of the upcoming milestone. However, it does not need to be implemented for the core parts of the exercise sessions that will follow. In case you run out of time, you should focus on understanding and implementing the core part(s) (80% of the grade for the milestone) and spend less time on the Elite part(s).

In general, the DFT-size used by OFDM is much larger than the length of the channel impulse response (why?). However, this means that the frequency response of the channel is oversampled, yielding a rather smooth frequency response (why?). Therefore, it is not necessary to estimate the frequency response in all the frequencies or tones. One can estimate the channel by only transmitting training symbols over a selection of tones, and interpolate between them to estimate the channel in the intermediate tones. The tones that are used for training are called 'pilot tones'. In the other tones, not used for training, one can then transmit actual data symbols.

Exercise: create a new m-file *ofdm_channelest_pilots.m* that estimates the channel frequency response in only half of the tones (in a uniform spacing, e.g., use the odd tones as pilot tones). The channel response in the intermediate frequencies is then estimated based on interpolation. You do not have to put actual data in the intermediate tones (this will be covered in the next session).

Hint: You may consider using linear or polynomial interpolation, but here an optimal interpolation can be devised by exploiting the fact that the time-domain channel impulse response is short (i.e., has a non-zero part with length equal to (at most) the CP length L, and hence a zero tail of $N - L$ samples). Using the IDFT matrix, you can set up a linear system of equations (with the channel response on the non-pilot tones as unknowns) that forces this tail to be zero. Furthermore, if a uniform spacing of pilot tones is used, one can rely on resampling theory to implement the same optimal interpolation with a simple and efficient IFFT/FFT operation. Try and find out how this can be done.[2].

[2]Think of a time-domain signal that needs to be upsampled. First, zeros are inserted between subsequent samples, generating additional mirror images in the frequency domain. Then, a low-pass filtering is used, i.e., a multiplication with a rectangular window in the frequency domain, to remove these mirror images. This effectively results in an interpolation in the time domain. Use duality between time domain and frequency domain to perform a similar interpolation for the pilot-tone case, i.e, in the frequency domain.