ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
School of Computer and Communication Sciences

Handout 19 Modern Digital Communications
Assignment 12 December 11, 2024 (due Dec 17, 2024)

MIMO Communications

In this assignment we consider a communication system with multiple transmitter and multiple receiver antennas (a MIMO system) and in this context, we compare the LS approximation to the LMMSE estimate of the transmitted signal.

If there is a single path between the transmitter and the receiver, the inputs and the outputs of the symbol-level channels in such a system will be related as

$$\boldsymbol{y}[n] = H[n]\boldsymbol{x}[n] + \boldsymbol{z}[n]. \tag{1}$$

In the above,

- $\boldsymbol{x}[n] = \begin{bmatrix} x_1[n] & x_2[n] & \cdots & x_t[n] \end{bmatrix}^T$ is the vector of t transmitter antenna inputs;
- $H[n] = \begin{bmatrix} H_{11}[n] & H_{12}[n] & \cdots & H_{1t}[n] \\ H_{21}[n] & H_{22}[n] & \cdots & H_{2t}[n] \\ \vdots & \vdots & \ddots & \vdots \\ H_{r1}[n] & H_{r2}[n] & \cdots & H_{rt}[n] \end{bmatrix}$ is the $r \times t$ channel matrix, which in general can be time dependent and consists of i.i.d. circularly-symmetric complex Gaussian entries.
- $\boldsymbol{y}[n] = \begin{bmatrix} y_1[n] & y_2[n] & \cdots & y_r[n] \end{bmatrix}^T$ is the vector of r receiver antenna outputs; and
- $\boldsymbol{z}[n] = \begin{bmatrix} z_1[n] & z_2[n] & \cdots & z_r[n] \end{bmatrix}^T$ is the vector of r i.i.d. white Gaussian noise samples.

Usually, the transmitter has no knowledge about the channel matrix $H[n]$, but we assume that the receiver knows it. Indeed, as you have seen in the previous asignments, the receiver can estimate the channel matrix with the help of a training sequence. If the channel changes with time, the channel estimation can be performed periodically, assuming the channel remains constant during the interval in between two channel estimations. For simplicity, we assume throughout this assignment that the receiver knows the exact channel matrix $H[n]$.

The transmitter sends out data symbols from a constellation (e.g. 4-QAM constellation) over its t antennas. Our goal is to develop a receiver that decodes the transmitted symbols $\boldsymbol{x}[n]$ with low error probability, upon observing the received symbols $\boldsymbol{y}[n]$.

Download the code framework for this assignment[1] from the course webpage. Read through the function `mimo_channel()`. This function simulates the channel described in Equation (1). It generates a new channel matrix each time it is called, and returns it as a second output argument. This is useful because we are not going to implement channel estimation; instead we use the exact channel matrix returned by the channel simulator.

As in the previous assignments, we provide the solutions for every function that you should write. You can use these solutions to test a whole working system, maybe just substituting one function at a time with your implementation. For MATLAB, the script `function_mapper` is used to select for each function whether to use the provided solution or your own implementation. For Python, the functions that you need to write are grouped under `my_utilMIMO.py`.

Exercise 1. In principle, given the channel matrix $H[n]$, we can formulate a decision problem using the signal model (1), and derive the optimal decoder for deciding on the information symbols $\boldsymbol{x}[n]$. However this would be impractical: First, the computational complexity of the decoder can be high. Assume data symbols $x_1[n], x_2[n], \ldots, x_t[n]$ are chosen from an M-ary constellation. Then the observable $\boldsymbol{y}[n]$ will be the noisy version of a "super-symbol" $H[n]\boldsymbol{x}[n]$ that can take M^t different possible values. This means instead of an M-ary decision rule, the decoder has to implement a much more complex M^t-ary decision. Second, this decision rule depends on the channel, which varies over time.

In practice, to decode we often follow a two-step approach: we first *equalize* the 'mixing' effect of the channel and create a new set of observables $\tilde{\boldsymbol{y}}[n] \in \mathbb{C}^t$ obtained from $\boldsymbol{y}[n]$ in such

[1] `mimo_assignment.zip`

a way that $\tilde{y}_i[n]$ is a noisy version of $x_i[n]$, $i = 1, 2, \ldots, t$. Then we feed the equalized symbols to a standard M-ary decoder to decode each of the t components independently. This way, we bring down the complexity from one M^t-ary decision to t M-ary decisions.

An easy-to-implement equalizer is simply a linear equalizer that multiplies the received symbols with a matrix $B[n] \in \mathbb{C}^{t \times r}$ (called the equalization matrix), to obtain

$$\tilde{\boldsymbol{y}}[n] = B[n]\boldsymbol{y}[n]. \tag{2}$$

A first attempt would be to choose $B[n]$ so that $\tilde{\boldsymbol{y}}[n]$ is the LS approximation of $\boldsymbol{x}[n]$. We know that in that case, $B[n] = (H^\dagger[n]H[n])^{-1}H^\dagger[n]$ which is the Moore-Penrose inverse of $H[n]$ (also called pseudoinverse).

If we assume for simplicity that $r = t$ and the channel matrix is invertible, the previous expression simplifies to $B[n] = H^{-1}[n]$, where we have used the fact that $(AB)^{-1} = B^{-1}A^{-1}$ where A and B are non-singular square matrices. In this case, we obtain

$$\tilde{\boldsymbol{y}}[n] = \boldsymbol{x}[n] + \boldsymbol{v}[n]$$

where $\boldsymbol{v}[n] = H^{-1}[n]\boldsymbol{z}[n]$. This method is called *zero-forcing*, as it 'forges' t effective channels through the MIMO channel such that on each channel the effect of other transmitted symbols is zeroed out. Observe also that $\boldsymbol{v}[n]$ does not have independent components anymore, so performing a component-wise minimum-distance decoding on $\tilde{\boldsymbol{y}}[n]$ is not optimal. Nevertheless, as the optimal decision rule will (again) depend on $H[n]$, we accept to use the sub-optimal component-wise minimum-distance decoding.

1. Implement `zeroForcing_equalizer()`. As it is clear from the comments, the function implements a zero-forcing equalizer as we discussed above. Recall that to solve $\boldsymbol{y} = A\boldsymbol{x}$, you can use the backslash operator (see `mldivide` for MATLAB, and `numpy.linalg.lstsq` for Python). This is much faster than computing the matrix inverse. The advantage of using this operator is that it automatically computes the pseudoinverse if A is not invertible.

   Once you have implemented your function, open the script `mimo_performance` and set the flag `test_zf = true`. Now you can run this script to simulate the performance of your communication system at different signal-to-noise ratios. Note that you can control how many different channel realizations are generated per signal-to-noise ratio, and the duration (in terms of number of symbols) over which the channel is constant through parameters `trials_per_snr` and `symbols_per_trial`, respectively. The script then plots the symbol error rate and the effective signal-to-noise ratio after equalization (noise after equalization is $\boldsymbol{v}[n] = B[n]\boldsymbol{z}[n]$).

   Look at the plot of effective signal-to-noise ratio versus channel signal-to-noise ratio. What can you conclude?

2. As you have probably discovered, the problem with zero-forcing equalization is *noise amplification*. Let us go through a more concrete example here. For simplicity, let us drop the time indices in this part.

   Derive the expression for the covariance matrix of $\boldsymbol{v} = H^{-1}\boldsymbol{z}$,

   $$\Sigma_{\boldsymbol{v}} = E\left[\boldsymbol{v}\boldsymbol{v}^\dagger\right]$$

   in terms of H and $\Sigma_{\boldsymbol{z}}$, assuming $\boldsymbol{z} \sim \mathcal{N}_{\mathcal{C}}(0, \sigma^2 I_2)$.

   Suppose

   $$H = \begin{bmatrix} 1 & e^{0.001j} \\ e^{0.001j} & 1 \end{bmatrix}.$$

   Use MATLAB/Python to evaluate $\Sigma_{\boldsymbol{v}}$. How do $\mathrm{var}(v_1)$ and $\mathrm{var}(v_2)$ compare to σ^2?

Exercise 2. In order to mitigate the noise-amplification problem of the zero-forcing equalizer, we can choose $B[n]$ so that $\tilde{\boldsymbol{y}}[n]$ is a Linear Minimum Mean Square Estimate (LMMSE) of $\boldsymbol{x}[n]$.

1. Find the matrix B such that $B\boldsymbol{y}$ is the LMMSE of $\boldsymbol{x}$, where $\boldsymbol{y} = H\boldsymbol{x} + \boldsymbol{z}$. Assume that $\boldsymbol{z} \sim \mathcal{N}_{\mathcal{C}}(0, \sigma^2 I_r)$, $E\left[\boldsymbol{x}\boldsymbol{x}^\dagger\right] = I_t$, and $\boldsymbol{x}$ and $\boldsymbol{z}$ are statistically independent.

2. Using the results of part 1, implement `lmmse_equalizer()`. Then set the flag `test_lmmse ... = true` in `mimo_performance`. Run the script again to compare the performance of different equalizers.

You should see that the LMMSE equalizer results in considerably higher signal-to-noise ratio at the decoder input, hence a lower symbol error rate. Also observe that the symbol error rate for the LMMSE receiver decays almost as the inverse of the signal-to-noise ratio. This is what we actually expect from a fading channel.