Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Signal and Information Processing Laboratory
Institut für Signal- und Informationsverarbeitung

Fachpraktikum Signalverarbeitung

# SV4: Equalization and Adaptive Filters

## 1 Introduction

In wireless and wired data transmission, a signal is sent through a channel. Channels imperfections (e.g., external disturbances, thermal noise, quantization error, etc.) introduce some noise in the original signal. A transmission channel can thus be modeled by a linear filter and the addition of a noise signal, as represented by the dashed box in figure 1. The task of de-filtering is known as *equalization* or *deconvolution*. As of today, many possible solutions exist. Several equalization methods are tested and compared experimentally (in software) in the experiment. In addition, channel estimation employing two adaptive methods will also be discussed.

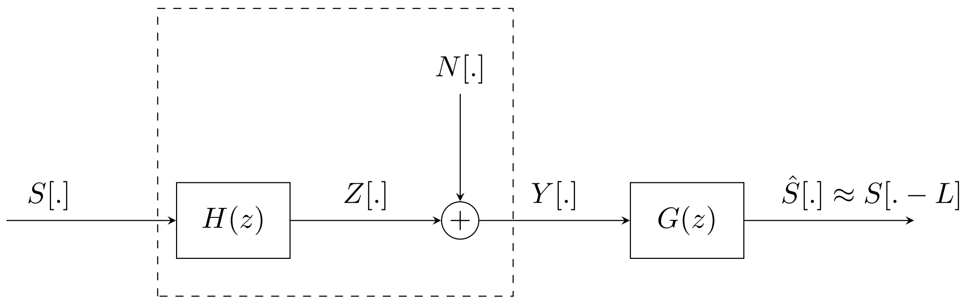

Figure 1: Equalization (with delay $L$) of a noisy LTI transmission channel.

## 2 Preparation

For the experiments, we will use filters of different orders. To be able to compare the results, we will mostly start from the following transmission channel:

$$\begin{aligned} H(z) &= h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3} + h_4 z^{-4} && (1)\\ &= 0.5 + -1.575 z^{-2} + 0.025 z^{-3} + 1.2012 z^{-4} && (2)\\ &= 0.5 \cdot \frac{(z+1.1)(z-1.2)(z-1.3)(z+1.4)}{z^4}. && (3) \end{aligned}$$

The channel distorts the transmitted signal like a linear time-invariant filter. The amplitude response $|H(e^{i\Omega})|$ is shown in figure 2.

Furthermore, we assume that we only have access to measurements of the channel output, which were noisy by white Gaussian noise with power $\sigma^2$. The goal now is to use some equalization algorithm $G(z)$ to estimate the input signal as well as possible. We will see that delaying the estimation often improves the estimate's quality. The process is depicted in figure 1.

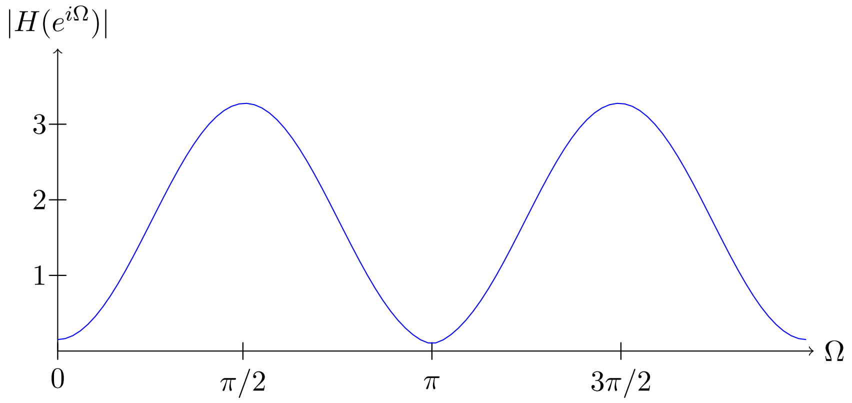

Figure 2: Amplitude response of the band-pass filter $H(z)$.

## 2.1 Zero-forcing equalization [1]

An inverse filter $G(z)$ to a filter $H(z)$ is a filter with a transfer function that satisfies $G(z)H(z) = 1$. The idea of using such a filter is obvious since it eliminates the effect of linear filtering through the channel. The first drawback of this approach is that noise is not considered. This quite naive approach is known as *zero-forcing* equalization.[2]

It is often impossible to construct exact inverse filters, so one must settle for an approximation. On the other hand, we require the filter $g[.]$ to be stable and causal.

**Example 2.1.** *With the channel defined in equation (1), we obtain*

$$1/H(z) = \frac{2z^4}{(z+1.1)(z-1.2)(z-1.3)(z+1.4)}. \tag{4}$$

*Because the poles lie outside the unit circle, the right-sided expansion is not stable, and the left-sided expansion is not causal. A causal and stable inverse filter can be constructed, allowing some delays, i.e., requiring only $G(z)H(z) \approx z^{-L}$, for some nonnegative integer $L$. We rewrite $H(z)$ as*

$$H(z) = z^{-4}\left(\tilde{h}_0 + \tilde{h}_1 z + \tilde{h}_3 z^2 + \tilde{h}_3 z^3 + \tilde{h}_4 z^4\right). \tag{5}$$

*The (stable) left-sided expansion of $1/H(z)$ is thus given by:*

$$G_\ell(z) = z^4(g_0 + g_1 z + g_2 z^2 + \dots). \tag{6}$$

*Shifting this signal to the right, we add some delay $L$*

$$z^{-L}G_\ell(z) = z^{4-L}(g_0 + g_1 z + g_2 z^2 + \dots). \tag{7}$$

*Finally, the non-causal part of the shifted signal is truncated and we obtain*

$$z^{-L}G_\ell(z) \ mod \ z = g_{L-4} + g_{L-4+1}z^{-1} + \dots + g_0 z^{-(L-4)}. \tag{8}$$

[1]Discussed in the section 1.7 of the DSSP Lecture Note

[2]The name zero-forcing is derived from the behavior of the zero-forcing filter $\tilde{G}(z)$ when combined with the distortion (in this case the channel $H(z)$). Because $\tilde{G}(z)H(z) \approx 1$, all coefficients of $z^n$ for $n \neq 0$ are transformed to zero

**Question 1.** *Let $N_c$ be the order of a causal filter $h[.]$, i.e. $H(z) = h_0 + h_1 z^{-1} + \cdots + h_{N_c} z^{-N_c}$. Specify the delay $L$ of an inverse filter with order $N$ according to the above scheme.*

To calculate the coefficients $g_k$ of the inverse filter $G(z)$, the following recurrence relation can be used:

$$g_k = -\frac{1}{\tilde{h}_0} \sum_{i=0}^{k-1} \tilde{h}_{k-i} g_i, \tag{9}$$

for $k > 0$ and $g_0 = 1/\tilde{h}_0$.

**Question 2.** *Transform the recursion in (9) so that the recursion can be implemented in a Matlab program. (Hint: Matlab matrix/vector indices start from $1$ and not from $0$.)*

## 2.2 Wiener Filter[3]

The Wiener filter is a particular case of filter, which uses the information about the channel, the input, and noise signals for robust equalization. For this purpose, specific properties (the so-called autocorrelation functions) of the signals must be known.

In the modeled transmission channel (see figure 1), the equalizer takes as input the noisy transmitted signal $Y(z) = H(z)S(z) + N(z)$. The signal $S[.]$ is assumed to be i.i.d., uniformly distributed in $\{-1, 1\}$. Thus, we have $\mathbb{E}\left[S[.]^2\right] = 1$. Moreover, $W[.]$ is white noise with power $\mathbb{E}\left[N[.]^2\right] = \sigma^2$, and is independent of $S[.]$.

**Example 2.2.** *With the channel defined in equation (1), the cross-correlation function from $S$ to $Y$ is given by $R_{SY}[.] = h[-.]$.*

**Question 3.** *For the channel defined in equation (1), compute the autocorrelation function $R_Y[n]$, using*

$$R_Y[.] = (h \star R_{SY})[.] + R_N[.], \tag{10}$$

Let $L$ and $M$ be nonnegative integers such that $g_w[n] = 0$ for $n < -L$ and $n > M$. The coefficients of the Wiener filter of order $N = L + M$ are determined by solving a system of linear equations called the *Wiener-Hopf equations*:

$$\sum_{n=-L}^{M} g_w[n] R_Y[j-n] = R_{XY}[j] \qquad j = -L, \ldots, M. \tag{11}$$

## 2.3 Decision-feedback equalizer (DFE)[4]

Next, we consider a popular nonlinear equalization method, the *decision feedback equalizer (DFE).* This method is applicable when the unknown signal $S[.]$ in figure 1 takes only value in a discrete set $S$, as is common in digital communication, e.g., $S = \{-3, -1, 1, 3\}$.

The structure of the original DFE[5] is shown in figure 3. It consists of a linear forward filter $G_f(z)$, a linear backward filter $G_b(z)$, and a (static) decision function $q : \mathbb{R} \to S$ (or $q : \mathbb{C} \to S$), that rounds to the nearest value in $S$. This decision function makes the DFE nonlinear. There are several versions of DFE, differing mainly in the choice of the forward filter $G_f(z)$.

[3]Discussed in the section 5.6 of the DSSP Lecture Notes

[4]Discussed in the section 1.7 of the DSSP Lecture Notes.

[5]The DSSP Lecture Notes introduce a slightly different one with $G'_f(z) = G_f(z)$ and $G'_b(z) = G_b(z)G_f(z)$. Of course, both are equivalent.

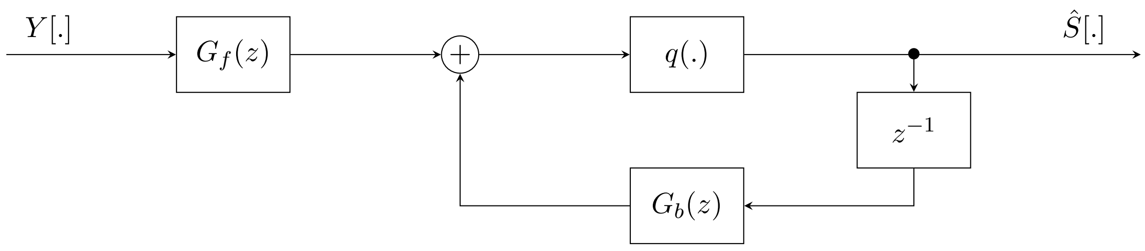

Figure 3: Decision-Feedback Equalizer (DFE).

**Question 4.** *For the channel defined in equation 1, compute the DFE with latency $L = 2$. Proceed as follows:*

1. *Decompose the transfer function of the channel into two parts : $H(z) = H_1(z) + z^{-L-1}H_2(z)$ with $H_1(z) = h_0 + h_1z^{-1} + h_2z^{-2}$ and $H_2(z) = h_3 + h_4z^{-1}$.*
2. *The forward filter should inverse $H_1(z)$ as well as possible. Compute $G_f(z) = z^{-L}F(z) \bmod z$ where $F(z)$ is the (stable) left-sided inverse of $H_1(z)$. Before you calculate, consider what order $G_f(z)$ will be.*
3. *The backward filter is simply $G_b(z) = -G_f(z)H_2(z)$.*

## 2.4 LMS algorithm[6]

Often the exact channel transfer function $H(z)$ is not available or, even worse, is not time-invariant (e.g., wireless communication). In this case, adaptive methods are used for equalization. The most popular methods are based on the *least mean squares (LMS)* algorithm. Figure 4 shows the basic structure of an adaptive filter for an equalization task with a quantizer $q(.)$. The measured signal $Y[.]$ is filtered by a *time-varying* linear FIR filter with coefficients $g_k[.]$.

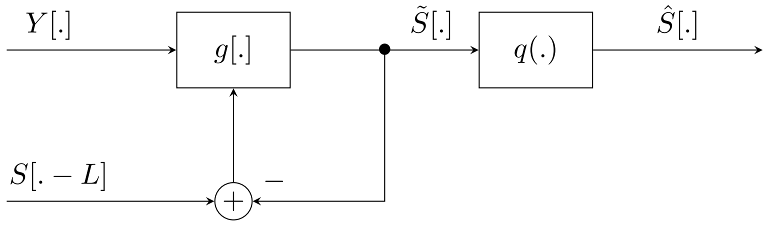

Figure 4: Adaptive filter for equalization.

The algorithm has two modes of operation: in the *frozen phase* or *operating phase*, the coefficients are fixed, i.e. $g_{k+1}[.] = g_k[.]$; in the *adaptive phase* or *training phase*, the signal $S[.]$ must be available[7] and the coefficients $g_k[.]$ are modified according to a gradient descent learning rule on the expected squared error, i.e.,

$$g_{k+1}[.] = g_k[.] + \beta\mathbb{E}\left[(S[k-L] - \hat{S}[k])Y[k-.]\right] \tag{12}$$
$$= g_k[.] + \beta(R_{XY}[L-.] - (g_k \star R_Y)[.]), \tag{13}$$

[6]Discussed in the section 5.10 in the DSSP Lecture Notes

[7]Under certain circumstances, adaptation can also be performed in the operating phase. However, a suitable reference signal is necessary.

where $\beta > 0$ is the learning rate.

However, this iteration rule is useless because the channel is unknown, and therefore the expected value cannot be computed. At this point, assuming that the reference signal is known, the LMS starts with an approximation. The current signal values replace the expected value.

**Question 5.** *Find the LMS learning rule for estimating $S[.-L]$ from $Y[.]$ by approximating the expected value by the current signal values as described above.*

### 2.4.1 Bonus: Adaptive decision feedback equalizer

In this section, we will see that a good estimation is even possible without a training phase, i.e., when the input signal is unavailable. Here, the LMS algorithm is used to estimate the forward and backward filters of a DFE. $S[.-L]$, $Y[.]$, and $\hat{S}[.]$ must be suitably substituted in the LMS learning rule found in question 5. However, $S[.-L]$ is not available anymore for training. As a substitute, the output $\hat{S}_d[.]$ of the quantizer $q(.)$ can be used as a reference. Thus, the learning rule tries to minimize the difference between the input and output of the quantizer. This algorithm is shown in figure 5. The LMS learning rule for the time-varying backward filter is given by:

$$g_{b,k+1}[.] = g_{b,k}[.] + \beta(\hat{S}[k] - \tilde{S}[k])\hat{S}[k-1-.]. \tag{14}$$

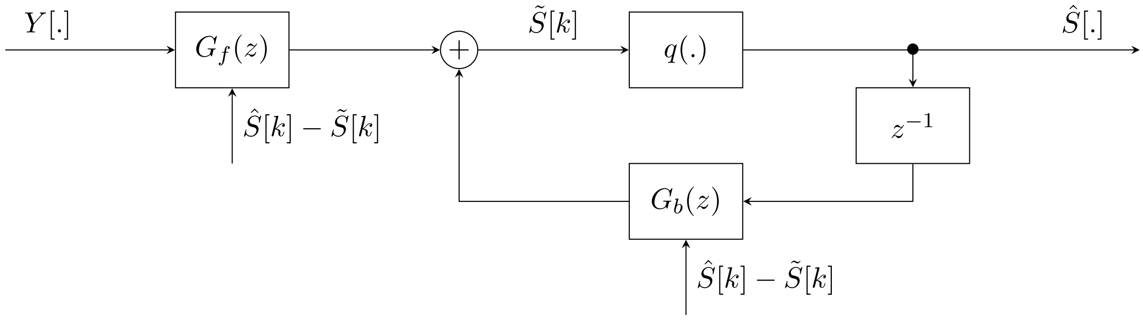

Figure 5: Adaptive decision-feedback equalizer.

**Question 6.** *Find the LMS learning rule for the time-varying forward filter $g_{f,k}[.]$.*

## 3 Experiment

1. Copy `/home/isistaff/glf/SV4` in the home directory (with `cp -irL /home/isistaff/glf/SV4 ./`)
2. Move in the `matlab` folder and launch Matlab (with `matlab &`)

For the experiments, various scripts have to be completed. It is worth it to look at the different files to understand what each file is computing. All files are located in the folder `matlab` and can be modified. It is not necessary to make copies. The main object of the simulation part is the `struct`[8] variable `s_Settings`. Each variable `s_Settings` specifies

[8]A `struct` variable is a list of fields with names and values. With `myStruct.fieldName`, you have access to the value of the `myStruct` variable in the `fieldName` field. In a `struct` array, each element is a `struct` variable and all elements have the same fields (with different values). With `myArrayStruct(2).fieldName` you can access the `fieldName` field of the second `struct` in the `myArrayStruct` array.

| Algorithm | `str_EstimationAlgorithm` | Parameters |
|---|---|---|
| Zero Forcing Filter | `ZF` | `InverseFilterLength` |
| Wiener Filter | `WF` | `InverseFilterLength` |
| DFE | `DFE` | - |
| LMS Filter | `LMS` | `InverseFilterLength`, `TrainingLength`, `LearningBeta`, `RunningBeta` |
| adaptive DFE | `Adaptive DFE` | `TrainingLength`, `LearningBeta` |

Table 1: Labels in `str_EstimationAlgorithm` and parameters of the equalization methods.

a simulation. It includes information about the transmitter, transmission channel, and equalization algorithm. This variable is always the first argument passed to a function, and it is also returned. Thus, it is possible to return selected data (e.g., filter coefficients or the bit error rate[9]). In the simulation environment, only the scripts `runSimPlot.m`, `runSim.m`, and `ZFSim.m` should be modified.

An overview of the simulation process and environment can be found in section 4. The main parameters controlling the simulations are `NumberOfSimulationRuns`, which specifies the number of Monte Carlo simulation runs, and the array `SNRValuesToSimulate`, which specifies the SNR points to simulate. Note that the SNR points are specified in dB. Both variables can be defined in `defaultSettings.m` (or directly in `s_Settings`).

The following Matlab files have to be adapted at marked positions in the source code:

`defaultSettings.m` Default values are defined for all simulations and automatically set. For most simulation parameters it is sufficient to adjust and use these values.

`runSimPlot.m` Each simulation is launched with this script (or the similar script `runSim.m`). Here, at the marked location in the source code, any number of variables `s_Settings` can be added to the array `as_SimSettings` and initialized. For each entry in the array `as_SimSettings`, a simulation is executed and the error rate is output graphically. It is recommended to follow the pattern of `ZFSim.m` (see below), as it should make it easier to combine different simulations and reuse older simulations. Note that no Matlab function is executed here. As a result, all variables in the script appear in the workspace. This way, variables can still be viewed in the Command window after the simulation. *Tip: Some important functions and scripts also have short documentation. Call them e.g. with* `doc` *followed by the name.*

`runSim.m` Performs the same operations as `runSimPlot.m` except for the graphical output of the error rates.

`ZFSim.m` This sample script is used to simulate the zero-forcing method. In detail, the script adds a new entry to the array `as_SimSettings`. The last occupied index in the array is always indicated by the variable `INDEX`. If a new entry is added, this variable must be incremented by one. Most entries are already initialized with the default values (from `defaultSettings.m`) and are therefore commented out and listed in `ZFSim.m` for illustrative purposes only.

Calculating the coefficients for the equalization filters is done in the functions listed below. A few of these functions need to be completed in the experiments.

[9] The bit error rate (*BER*) is defined as the number of bit errors per bit transmitted, thus BER = number of bit errors/number of bits transmitted).

`ComputeZeroForcingFilter.m` Compute a (causal and stable) inverse filter $\tilde{G}(z)$ to a transfer function $H(z)$. Here, we need to insert the recursion found in question 2 and the calculation of the delay from question 1.

`ComputeWienerFilter.m` Compute a Wiener filter $G_w(z)$ to the transfer function $H(z)$. This file does not need to be edited.

`ComputeDFEFilter.m` Computes the filters $G_f(z)$ and $G_b(z)$ for the DFE equalization procedure. In this function, the coefficients from question 4 must be inserted.

`ComputeLMSFilter.m` Computes the adaptive filter $g_k[.]$ using the LMS method. The filter order `InverseFilterLength`, the number of training symbols `TrainingLength`, and the learning rate $\beta$ `LearningBeta` must be specified. The learning rule from question 5 for filter $g_k[.]$ must be inserted here.

`ComputeAdaptiveDFEFilter.m` Computes the adaptive DFE filter from 2.4.1.

## 3.1 Equalization of a known channel

In the first part of the experiments, it is assumed that the receiver knows the channel. The zero-forcing algorithm, the Wiener filter, and the DFE are compared. Creating a new script file for each simulation as described above is recommended.

3. Complete the script `ComputeZeroForcingFilter.m` with the recursion found in question 2 and the calculation of delay $L$ from question 1.

4. Calculate an inverse filter to $H(z)$ with length 20 by running the script `runSim.m`. In `runSim.m` the script `ZFSim.m` should be called. Create a plot[10] of the filter coefficients of $\tilde{G}(z)$. *Tip: The filter $\tilde{G}(z)$ was stored in the array `as_SimSettings` in the field `Gf` after running `runSim.m`.*

5. Use the Matlab command `conv` to find out if the calculated filter is approximately inverse to $H(z)$. Plot the approximately equalized impulse response. What would the equalized impulse response look like with an ideal inverse filter? Can you read off the delay?

6. Now, compute zero-forcing filters with other lengths and look at the filter coefficients of $\tilde{G}(z)$ and the convolution with $H(z)$. To do this, change the value of `InverseFilterLength` in the file `ZFSim.m`. *Tip: You can use a loop (e.g. `for`) in `ZFSim.m` to calculate multiple filter lengths at once.*

In the error rate simulations, bit sequences of length 5000 (constant `SEQUENCE_LENGTH`) should be selected. For the number of simulation runs (field `NumberOfSimulationRuns` or default constant `NUM_SIMULATION_RUNS`), one has to trade-off between the duration of a simulation and the accuracy of the results. Therefore, experiment with this parameter. Choose the SNR range so that the meaningful part of the bit error rate (between $10^{-1}$ and $10^{-3}$) is shown.

7. First create a plot of the bit error rate for a zero-forcing filter of length 20 by setting the necessary values in `runSimPlot.m` (for the default values) or in `ZFSim.m`. Execute the script `runSimPlot.m`.

8. Simulate and plot now the bit error rates for the zero-forcing filter with lengths 10, 20, 30, 40 and 50. In what order can hardly any improvement of the error rates be seen?

[10]The Matlab function `stem` is a nice way to plot discrete-time signals

Now let us compare the previous equalization method with the Wiener filter method. The Wiener filter is calculated in the script `ComputeMyWienerFilter.m`. However, nothing more has to be changed there.

9. The Wiener filter depends on the noise's strength - that is, the noise variance or SNR. It is instructive to compare the Wiener filter with the zero-forcing filter at different SNR points. To do this, compare the convolution of the filter with the channel (see point 5) for SNR points 0 dB, 15 dB, and 30 dB. For this, you have to run a separate simulation for each SNR point. What do you see?

10. Now create a plot analogous to point 8 with the error rates for different Wiener filter lengths. Plot all curves and compare the performance. Which delays are sufficient for the optimal performance of the Wiener filter?

11. As you may have noticed from the plot in 10, the error rates with the zero-forcing method approach those of the Wiener filter at high SNR. Plot the filter coefficients of the Wiener filter $g_w[.]$ and the zero-forcing filter $\tilde{g}[.]$ for length 50 and SNR=30 dB on top of each other. Can you confirm the observation on the error rates by this?

Next, we want to simulate the error rates of the DFE.

12. Complete the m-file `ComputeDFEFilter.m` with the values found in task 4.

13. Again, create a plot of the bit error rate and compare it with the Wiener filter. Can you explain the difference in performance? *Tip: consider which assumptions about the sequence $S[.]$ are inherent in the DFE (and not in the Wiener filter)? Is the DFE a linear filter?*

## 3.2 Bonus: Equalization with adaptive methods

We will consider the equalization problem when the channel is unknown in the following experiments. We will therefore use adaptive filters. Furthermore, we assume that the transmitter sends a known training sequence of length `TrainingLength` and then the data sequence of length `SequenceLength` = 5000 symbols.

14. Complete the script `ComputeLMSFilter.m` with the LMS learning rule for the filter $g_k[.]$ found in question 5. Note that the step size parameter $\beta$ is called `betaT` in the Matlab code.

15. Choose a training sequence of length (`TrainingLength`) 2000, set the step size parameter $\beta = 0.005$ (`LearningBeta`). The SNR should be 15 dB and the equalization filter should have a length of 20. To find out how well the LMS algorithm has converged, create a plot showing the evolution over time of the last 3 filter coefficients.[11] *Tip: Again, the filter coefficients are calculated before the simulation runs and stored in `s_Settings`.*

16. Using the same setting as in the previous point, compare the learned filter coefficients $g_{2000}[.]$ with those of the Wiener filter and zero-forcing filter. Can you tell if the learned filter converges to one of the known filters? To which filter does it converge?

[11] The function from `ComputeLMSFilter.m` returns not only the final value of the filter coefficients but also the intermediate values. Therefore, the filter is stored in Matlab as a two-dimensional array. For example, the filter coefficient $g_k[n]$ can be found in row $k$ and column $n$ of the two-dimensional return value.

17. Choose 2 as the length for the adaptive filter and compute the filters with different values for $\beta$. Call the function `plotErrorSurface`[12] with one of the returned `s_Settings` variable. This function plots the error surface as a function of the two filter coefficients. On the surface, the iteration steps of the LMS are also shown as a path. Depending on the value of $\beta$ and SNR, the LMS converges very fast near the minimum of the surface. For small SNR and/or large $\beta$, the path may not even go towards the minimum anymore.

18. Compare the error rates of the Wiener filter, zero-forcing and the LMS algorithm. Play with the filter length and the length of the training sequence for the LMS. How long must the training sequence be for the LMS to have lower error rates than the zero-forcing filter for the same SNR and filter length?

### 3.2.1 LMS Tuning and Adaptive DFE

To improve the error rate of the LMS algorithm without changing the length of the training sequence, we can extend the learning rule. We want to use the estimated symbols $\hat{S}[.]$ as a reference signal after the training phase and thus train the filter also during normal operation. Since errors can also occur during this process, the step size parameters for this learning rule (`RunningBeta` field in `s_Settings`) should be $5 - 10$ times smaller than that of the training phase.

19. Put the new learning rule for the operational phase in `ComputeLMSFilter.m`. Note that there are now two step-size parameters: `betaT` for the step size during the training phase and `betaR` for the step size during normal operation. Compare the error rates with the previous LMS. Does the new algorithm need fewer training symbols?

Finally, examine the error rate of the adaptive DFE from section 2.4.1. In this case, there is also a training phase. However, it differs from the training phase of the LMS because no reference signals are available.

20. Simulate the error rate for the adaptive DFE for different length training periods and compare it with that of the non-adaptive DFE.

## 4 Summary

Three widely used equalization methods (zero-forcing, Wiener filter, and DFE) were introduced and tested on a practical communication model. The second part applied adaptive methods (namely the LMS algorithm) to the channel inversion problem and the DFE. The experiments show that good inversion requires some delay, depending on the channel. Furthermore, inverting a distortion is usually not the most optimal method, as the comparisons with the Wiener filter and the DFE show. Adaptive methods can be used if the channel is unknown to the receiver but often need a certain number of training symbols. It was briefly shown how, by clever choice of alternative reference signals, one can sometimes even dispense with reference symbols altogether.

*Congratulations! You have worked through all experiments!*

[12] Use `doc plotErrorSurface` to know how to see a short documentation.

## Appendix

The simulation environment consists of a loop that performs the steps listed below for each SNR value in the `SNRValuesToSimulate` times `NumberOfSimulationRuns`. The following steps are performed in this order:

`simulate.m` Runs the simulation specified in the argument `s_Settings`. A simulation always consists of the three same steps:

1. `GenerateRandSignal.m` Generates a random symbol sequence of a given length (`s_Settings.SequenceLength`) from the symbols in the alphabet vector `s_Settings.InputAlphabet`. As in figure 1, this sequence is distorted by a channel defined in `s_Settings.Channel` (in this experiment the channel from equation (1)). Finally, noise is added to the channel output[13].
2. EstimateInputSignal.m This function corresponds to the equalization block in figure 1. As return values, besides the variable `s_Settings`, the estimated input symbols $\hat{S}[.]$ are calculated. In the beginning, the coefficients of the equalization filter are calculated. The used equalization method is read from the `str_EstimationAlgorithm` field according to the table 3.
3. `ComputeErrors.m` Here the error rate is computed by comparing the input sequence $S[.]$ with the estimates $\hat{S}[.]$.

[13]An important parameter of this function is the field `Seed` in `s_Settings`. Thanks to this number it can be determined which random sequences will be generated. In turn, it can be guaranteed that the simulation results are reproducible