Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Signal and Information Processing Laboratory
Institut für Signal- und Informationsverarbeitung

Fachpraktikum Signalverarbeitung

SV4: Egalisation und Adaptive Filter

1 Einführung

Bei der drahtlosen oder drahtgebundenen Datenübertragung wird das gesendete Signal linear gefiltert und verrauscht. Ein Übertragungskanal kann also durch ein lineares Filter und die Addition eines Rauschsignals modelliert werden. Die gestrichelte Box in Abbildung 1 stellt das Modell für einen Übertragungskanal dar.

Die 'Entfilterung' (Entzerrung, Entfaltung, Egalisation) der empfangenen Signale ist eine Aufgabe, für die es viele Lösungsmöglichkeiten gibt. Im Versuch werden mehrere Egalisationsverfahren experimentell (in Software) erprobt und verglichen. Zudem wird auch auf die Kanalschätzung mittels zweier adaptiver Verfahren eingegangen.

2 Vorbereitung

Für die Versuche werden wir Filter verschiedener Ordnung verwenden. Damit wir die Resultate vergleichen können, werden wir meistens vom folgenden Übertragungskanal ausgehen:

$$H(z) = 0.5 - 1.575z^{-2} + 0.025z^{-3} + 1.2012z^{-4} \quad (1)$$
$$= h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3} + h_4 z^{-4}. \quad (2)$$

Dieser Kanal verzerrt das gesendete Signal also wie ein linear zeitinvariantes Filter. Ausserdem nehmen wir an, dass wir nur zu Messungen des Kanalausgang Zugang haben, die durch weisses Gauss'sches Rauschen mit Leistung $E[|N[k]|^2] = \sigma^2$ verrauscht wurden. Ziel ist es nun einen Egalisationsalgorithmus zu erstellen, der das Eingangssignal möglichst gut schätzt. Wir werden sehen, dass oft eine Verzögerung der Schätzung die Qualität der Schätzung verbessert. Abbildung 1 stellt das gesamte System von Kanal, Rauschen und Egalisation mit Verzögerun L schematisch dar. Der Frequenzgang von $H(z)$ ist in Abbildung 2 dargestellt.

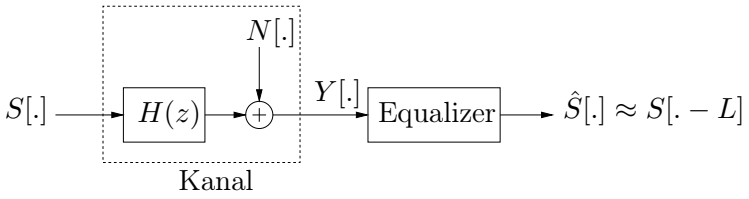

Abbildung 1: Übertragungskanal und Egalisation mit Verzögerung L

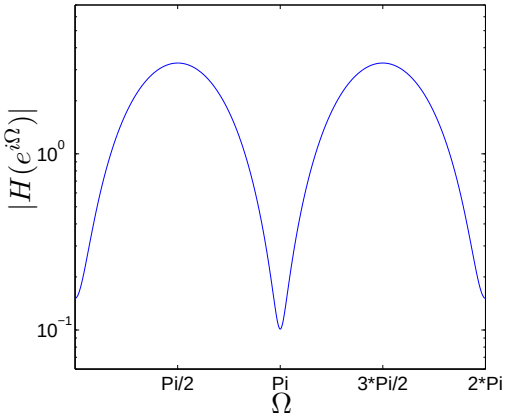

Abbildung 2: Frequenzgang von $H(z)$. $H(z)$ hat Bandpasscharakter.

2.1 Zero Forcing Egalisation

Ein inverses Filter $G(z)$ zu einem Filter $H(z)$ ist ein Filter mit einer Übertragungsfunktion, die $G(z)H(z) = 1$ erfüllt. Die Idee, ein zum Kanal inverses Filter zur Egalisation zu verwenden, liegt auf der Hand, da dadurch der Effekt der linearen Filterung durch den Kanal beseitigt wird. Dem Rauschen wird allerdings bei dieser Methode nicht Rechnung getragen. Diese Methode zur Egalisation ist bekannt als *Zero Forcing* Egalisation[1].

Häufig ist es allerdings nicht möglich, exakte inverse Filter zu konstruieren, man muss sich deshalb mit Aproximationen begnügen. Wir werden deshalb approximierte inverse Filter mit verschiedenen Verzögerung L berechnen.

Beispiel 2.1. *Für den Kanal (1) ist ersichtlich, dass alle Pole von*

$$G(z) = H^{-1}(z) = \frac{z^4}{0.5z^4 - 1.575z^2 + 0.025z^1 + 1.2012} \tag{3}$$

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

ausserhalb des Einheitskreises liegen. Also ist nur das dazugehörige linksseitige Signal stabil [2]*. Da wir aber nur an rechtsseitigen inversen Signalen (kausale Egalisationsmethoden) interessiert sind, muss das inverse Signal also in drei Schritten gebildet werden* [3]*. Zuerst wird die linksseitige Inverse berechnet. Der Einfachheit halber wird die Impulsantwort des Kanals zeitverkehrt als Vektor geschrieben:*

$$H(z) = h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3} + h_4 z^{-4} \tag{5}$$

$$= z^{-4}(\tilde{h}_0 + \tilde{h}_1 z + \tilde{h}_2 z^2 + \tilde{h}_3 z^3 + \tilde{h}_4 z^4). \tag{6}$$

Die linksseitige Inverse ist dann $G(z) = z^4(g_0 + g_1 z + g_2 z^2 + \ldots)$. Danach wird das linksseitige Signal L Zeitschritte nach rechts verschoben (verzögert), also $z^{-L}G(z) = z^{4-L}(g_0 + g_1 z + g_2 z^2 + \ldots)$. Schlussendlich wird der nicht kausale Teil des verschobenen Signals ab-

[1] Die Bezeichnung Zero Forcing leitet sich aus dem Verhalten des Zero Forcing Filters $\tilde{G}(z)$ ab wenn es mit der Verzerrung (hier dem Kanal $H(z)$) kombiniert wird. Weil das Ziel $\tilde{G}(z)H(z) \approx 1$ ist, spricht man auch davon dass alle Koeffizienten von z^n ausser demjenigen von z^0 auf Null transformiert werden.

[2] Siehe auch das Skript zur Vorlesung ZSS ('Zeitdiskrete und statistische Signalverarbeitung'), Kapitel 1

[3] Siehe Details im Skript zur Vorlesung ZSS, Kapitel 1, Abschnitt Inverse Filter

geschnitten und man erhält [4].

$$\tilde{G}(z) = z^{-L}G(z) \ mod \ z = g_{L-4} + g_{L-4+1}z^{-1} + \ldots + g_0 z^{-(L-4)}. \tag{7}$$

Aufgabe 1. *Sei N_c die Filter Ordnung von $H(z)$. Gebe an welche Verzögerung L eine Inverse mit Ordnung N nach obigem Schema hat.*

Für die Berechnung der Koeffizienten g_k der Inversen $G(z)$, kann folgende Rekursionsbeziehung mit den Koeffizienten h_k von $H(z)$ benutzt werden:

$$g_0 = \frac{1}{\tilde{h}_0} \tag{8}$$

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

Durch Multiplikation von $\tilde{G}(z)$ und $H(z)$ kann verifiziert werden, ob $\tilde{G}(z)$ ungefähr invers zu $H(z-L)$ ist.

Aufgabe 2. *Forme die Rekursion in (9) so um, dass die Rekursion in einem Matlab Programm implementiert werden kann. (Hinweis: Matlab Vektoren haben Indizes von* 1 *bis* n *und nicht von* 0 *bis* $n-1$*).*

2.2 Wiener Filter

Das Wiener Filter ist eine spezielle Klasse von Filtern, welche zur Egalisation nicht nur die Information über den Kanal, sondern auch Informationen über das Informationssignal und das Rauschen verwendet. Dazu müssen gewisse Eigenschaften (die sogenannten Autokorrelationsfunktionen) der Signale bekannt sein[5].

Im modellierten Übertragungskanals (siehe Abb. 1) ist $Y(z) = H(z)S(z) + N(z)$ das Ausgangssignal und somit das Eingangssignal unseres Equalizers. Das Nutzsignal $S[.]$ sei i.i.d gleichverteilt auf {+1,-1} und hat deshalb Varianz $E[|S[k]|^2] = 1$. Das reellwertige Störsignal $N[.]$ sei weisses Rauschen mit $E[|N[k]|^2] = \sigma^2$ und von $X[.]$ unabhängig.

Beispiel 2.2. *Für den Beispiel-Kanal aus (1) lautet die zur Berechnung des Wiener Filters nötige Kreuzkorrelationsfunktion folgendermassen:*

$$R_{SY}[k] = \begin{cases} h_4, & k = -4 \\ h_3, & k = -3 \\ h_2, & k = -2 \\ h_1, & k = -1 \\ h_0, & k = 0 \\ 0, & sonst \end{cases} \tag{10}$$

Aufgabe 3. *Berechne für den Kanal (1) die Werte der Autokorrelationsfunktion $R_Y[k]$. Mithilfe von Beispiel 2.2 kann $R_Y[k]$ aus*

$$R_Y[.] = (h \star R_{SY})[.] + R_N[.], \tag{11}$$

also der Faltung von $h[.]$ mit $R_{SY}[.]$ und $R_N[.] = \sigma^2\delta[.]$, erhalten werden.

[4] Im folgenden wird die Notation $G(z)$ mod z verwendet, wenn bei einer rationalen z-Transformation $G(z)$ alle nichtkausalen Terme, also diejenigen mit einer positiven Potenz von z zu Null gesetzt werden.

[5] Das Wiener Filter wird im ZSS-Skript Kapitel 5 behandelt.

Um ein Wiener Filter $G_w(z)$ der Ordnung M mit Verzögerung L zu berechnen, werden die Koeffizienten des Filters als Lösung eines $M+1 \times M+1$ dimensionales linearen Gleichungssystem ausgedrückt. Voraussetzung dazu ist die Kenntnis[6] der Autokorrelationsfunktion $R_Y[.]$ und der Kreuzkorrelation $R_{SY}[.]$. Durch auflösen des Gleichungssystems

$$\sum_{n=-L}^{M} g_w[n] R_Y[j-n] = R_{SY}[j] \qquad j = -L, \ldots, -L+M \tag{12}$$

erhält man das Wiener Filter $g_w[n] \qquad j = -L, \ldots, -L+M.$

2.3 Decision-Feedback-Equalizer

Als nächstes betrachten wir ein beliebtes nichtlineares Egalisationsverfahren, den 'Entzerrer mit Entscheidungsrückführung', *decision-feedback equalizer* (*DFE*). Dieses Verfahren ist anwendbar, wenn das unbekannte Signal $S[.]$ in Abbildung 1 wie in der digitalen Nachrichtentechnik üblich nur diskrete Werte annimmt, z.B. $S[.] = \{-3, -1, +1, +3\}$.

Die Struktur eines DFE ist in Abbildung 3 gezeigt. Der Algorithmus besteht aus einem linearen Vorwärts-Filter $G_f(z)$, einem linearen Rückwärtsfilter $G_b(z)$ und einer Entscheidungsfunktion (auch Quantisierer genannt), die zum nächstliegenden möglichen Wert des diskreten Eingangssignals rundet. Eine solche Entscheidungsfunktion ist nichtlinear; sie ist das einzige nichtlineare Element im DFE. Es gibt verschiedene DFE-Versionen, die sich hauptsächlich durch die Wahl des Vorwärtsfilters $G_f(z)$ unterscheiden.

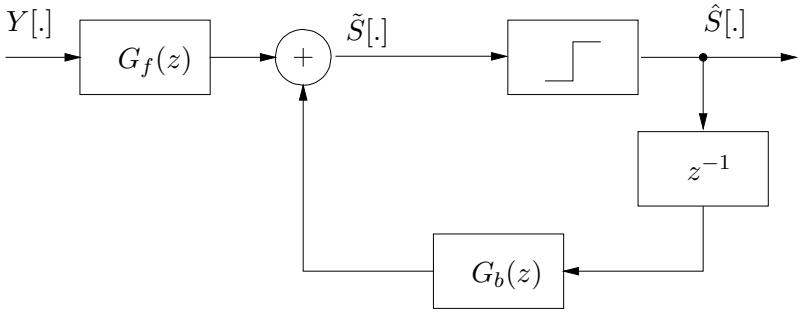

Abbildung 3: Decision-Feedback Equalizer

Aufgabe 4. *Berechne den DFE für den Kanal (1) mit einer Struktur gemäss Abbildung 3 und mit Latenz $L = 2$. Gehe dazu folgendermassen vor:*

- *$H(z)$ wird aufgeteilt in $H(z) = H_1(z) + z^{-L-1} H_2(z)$ mit $H_1(z) = h_0 + h_1 z^{-1} + h_2 z^{-2}$ und $H_2(z) = h_3 + h_4 z^{-1}$.*
- *$G_f(z) \triangleq z^{-2} F(z)$ mod z, wobei $F(z)$ die linksseitige Inverse zu $H_1(z)$ (vgl. Abschnitt 2.1 zum inversen Filter und Formeln (8) und (9)). Bevor du rechnest, überlege dir, von welcher Ordnung $G_f(z)$ sein wird.*
- *$G_b(z) \triangleq -G_f(z) H_2(z)$*

[6]Genauer gesagt, damit das Wiener Filter eine optimale Performance hat und sinnvoll definiert ist, müssen die Signale stationäre Prozesse sein und die Statistiken 1. und 2. Ordnung (Autokorrelation) müssen bekannt sein.

2.4 Least-Mean-Square Algorithmus

Häufig steht die genaue Kanalübertragungsfunktion $H(z)$ nicht zur Verfügung oder ändert sich sogar mit der Zeit (z.B. drahtlose Kommunikation). In diesem Fall werden adaptive Verfahren zur Egalisation eingesetzt. Abbildung 4 zeigt die Grundstruktur eines adaptiven Filters als Methode zur Egalisation. Die empfangenen Symbole $Y[.]$ werden mit dem Filter $g_k[.]$ entzerrt. Dabei sind die Filterkoeffizienten $g_k[.]$ nicht zeitinvariant, sondern werden nach jedem Zeitschritt so angepasst, dass das Referenzsignal (in der Abbildung ist das Referenzsignal der Fehler zwischen Datensignal $S[.]$ und geschätztem Signal $\tilde{S}[.]$) kleiner oder sogar null wird.

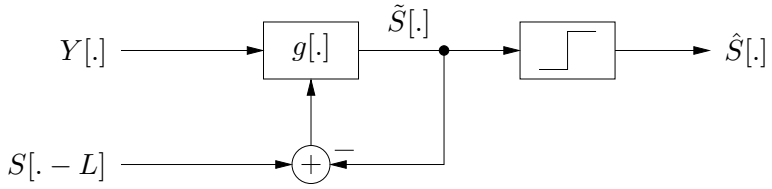

Abbildung 4: Adaptives Filter mit Quantisierer

Für diese Art der Adaption muss also das zu schätzende Signal $S[.]$ für eine gewisse Zeit zur Verfügung stehen. Das Filter hat dann zwei Betriebsphasen: die adaptive Phase, in welcher die Koeffizienten mit Hilfe des Referenzsignales adaptiert werden (Trainingsphase); und die eingefrorene Phase, in der das Eingangssignal nicht mehr zur Verfügung steht (Betriebsphase). Unter bestimmten Umständen kann aber auch in der Betriebsphase adaptiert werden. Allerdings muss ein geeignetes Referenzsignal gefunden werden. In 2.4.1 werden wir sehen, dass eine gute Schätzung sogar gänzlich ohne Trainingsphase möglich ist, also wenn das Eingangssignal $S[.]$ gar nicht zur Verfügung steht.

Eine einfache und weit verbreitete Methode zur Adaption ist der Least-Mean-Square Algorithmus (LMS). Im Prinzip entspricht das Verfahren dem klassischen Gradientenverfahren angewendet auf die Suche desjenigen Filters $g[.]$, welches den quadratischen Fehler minimiert. Es lässt sich zeigen, dass die Iterationsregel in diesem Fall

$$g_{k+1}[.] = g_k[.] + \beta \, \mathrm{E}\left[(S[k-L] - \tilde{S}[k])Y[k-.]\right] \quad (13)$$

$$= g_k[.] + \beta(R_{SY}[L-.] - (g_k \star R_Y)[.]) \quad (14)$$

ist, wobei β die Schrittweite für das Gradientenverfahren ist. Diese Iterationsregel ist jedoch unbrauchbar, da der Kanal unbekannt ist und deshalb der Erwartungswert in (13) nicht berechnet werden kann. An diesem Punkt setzt der LMS unter der Annahme, dass das Referenzsignal bekannt ist, mit einer Approximation an; in der Lernregel des LMS wird der Erwartungswert in (13) durch die aktuellen Signalwerte ersetzt.

Aufgabe 5. *Finde die LMS-Lernregel zur Schätzung von $S[.-L]$ aus $Y[.]$ indem du in 13 den Erwartungswert wie oben beschrieben durch die aktuellen Signalwerte annäherst.* [7].

2.4.1 Bonus: Adaptiver Decision Feedback Equalizer

Der LMS Algorithmus kann auf die gleiche Weise wie vorhin auch benutzt werden um die Filter $g_f[.]$ und $g_b[.]$ zu schätzen. Dazu müssen Referenzsignal $S[.-L]$, Filter Eingangssignal $Y[.]$ und geschätztes Signal $\tilde{S}[.]$ in der LMS-Lernregel in Aufgabe 5 geeignet ersetzt werden. Allerdings stehe nun für das Training das Referenzsignal $S[.]$ nicht zur Verfügung. Als Ersatz kann in diesem Fall der Ausgang des Quantisierers $\hat{S}[.]$ als Referenz benutzt

[7] Interessierte finden mehr Informationen im Skript zur Vorlesung 'ZSS', Kapitel 5, Abschnitt 'LMS-Algorithmus'.

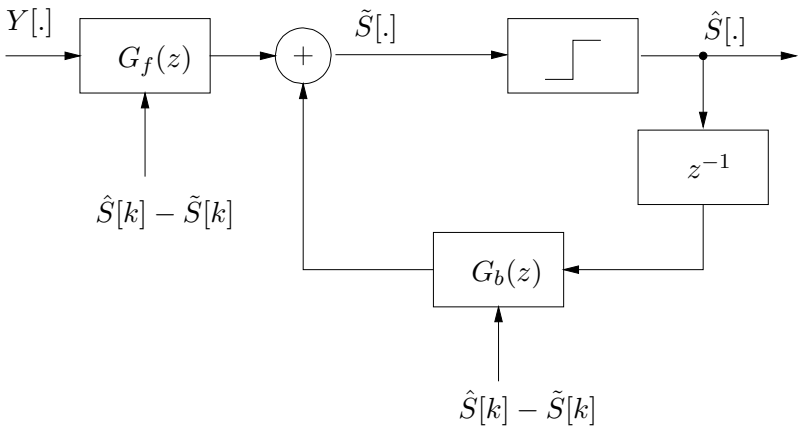

Abbildung 5: Adaptiver Decision Feedback Equalizer

werden. Die Lernregel versucht also die Differenz zwischen Eingang und Ausgang des Quantisierers zu minimieren. Dieser Algorithmus ist in Abbildung 5 gezeigt. Die Lernregel für das (zeitvariante) Filter $g_{b,k}[.]$ lautet nun

$$g_{b,k+1}[.] = g_{b,k}[.] + \beta(\hat{S}[k] - \tilde{S}[k])\hat{S}[k-1-.]. \tag{15}$$

Aufgabe 6. *Finde analog zu (15) die LMS Lernregel für das Filter $g_{f,k}[.]$.*

3 Versuche

1. Kopiere `/home/isistaff/glf/fachprak_isi/SV4` in dein Homeverzeichnis: (`cp -irL /home/isistaff/glf/fachprak_isi/SV4 ./`)
2. Starte Matlab im Ordner `matlab` (in einer Shell in den Ordner wechseln und dann mit dem Befehl `matlab &`).

Für die Versuche müssen auch diverse Matlab-Skripte komplettiert werden. Bei den vollständigen Dateien ist es wertvoll einen Blick hinein zu werfen um zu verstehen was berechnet wird. Alle Matlab-Dateien befinden sich im Ordner `matlab` und können verändert werden, es ist nicht nötig, Kopien zu machen. Kernstück der Simulationsumgebung ist die `struct`[8] Variable `s_Settings` bzw. der Array dieser Variablen `as_SimSettings`. Jede Variable `s_Settings` spezifiziert eine Simulation. Das beinhaltet Angaben zum Sender, Übertragungskanal und Egalisationsalgorithmus. Diese Variable ist immer das erste Argument, das einer Funktion übergeben wird, und sie wird auch zurückgegeben. Dadurch ist es möglich ausgewählte Daten (z.B. Filterkoeffizienten oder die Bit-Error-Rate[9]) zurück zu geben. In der Simulationsumgebung sollten für die Versuche nur die Skripte `runSimPlot.m`, `runSim.m` und `ZFSim.m` verändert werden.

Eine Übersicht über den Ablauf einer Simulation und der Simulationsumgebung kann in Abschnitt 4 gefunden werden. Die wichtigsten Parameter zur Steuerung der Simulationen sind `NumberOfSimulationRuns`, welche die Anzahl an Monte-Carlo Simulationsdurchgängen vorgibt und der Array `SNRValuesToSimulate`, der die zu simulierenden SNR-Punkte

[8] Eine `struct` Variable ist eine Liste von Feldern mit Namen und Werten. Mit `myStruct.fieldName` hat man Zugriff auf den Wert der Variable `myStruct` im Feld `fieldName`. In einem `struct` Array ist jedes Element eine `struct` Variable und alle Elemente haben die gleichen Felder (mit unterschiedlichen Werten). Mit `myArrayStruct(2).fieldName` hat man Zugriff auf das Feld `fieldName` des zweiten `struct` im Array `myArrayStruct`.

[9] Die Bit-Error-Rate (*BER*) ist definiert als die Anzahl der Bitfehler pro übertragenes Bit, also BER = $\frac{\text{Anzahl Bitfehler}}{\text{Anzahl übertragene Bits}}$)

Algorithmus	`str_EstimationAlgorithm`	Parameter
Zero Forcing Filter	`ZF`	`InverseFilterLength`
Wiener Filter	`WF`	`InverseFilterLength`
DFE	`DFE`	-
LMS Filter	`LMS`	`InverseFilterLength`, `TrainingLength`, `LearningBeta`, `RunningBeta`
adaptiver DFE	`Adaptive DFE`	`TrainingLength`, `LearningBeta`

Tabelle 1: Bezeichnungen in `str_EstimationAlgorithm` und Parameter der Egalisationsmethoden.

vorgibt. Beachte, dass die SNR-Punkte in dB angegeben werden. Beide Variablen können in `defaultSettings.m` (oder direkt in `s_Settings`) definiert werden.

Für diesen Versuch müssen folgende Matlab-Dateien an markierten Stellen im Quelltext angepasst werden:

`defaultSettings.m` An dieser Stelle werden default-Werte für alle Simulationen definiert. Diese Werte werden dann automatisch gesetzt. Für die meisten Simulations-Parameter genügt es diese Werte anzupassen und zu verwenden.

`runSimPlot.m` Jede Simulation wird durch dieses Skript (oder das ähnliche Skript `runSim.m`) gestartet. Hier kann an der markierten Stelle im Quelltext eine beliebige Anzahl an `s_Settings` Variablen dem Array `as_SimSettings` hinzugefügt und initialisiert werden. Für jeden Eintrag im Array `as_SimSettings` wird dann eine Simulation ausgeführt und die Fehlerrate grafisch ausgegeben.

Es empfiehlt sich das Muster von `ZFSim.m` (siehe unten) einzuhalten. Dadurch sollte die Kombination von verschiedenen Simulationen und das wiederverwenden von älteren Simulationen erleichtert werden. Beachte, dass hier keine Matlab Funktion ausgeführt wird. Das hat zur Folge, dass alle Variablen im Skript im Workspace erscheinen. So können Variablen auch nach der Simulation noch im Command Fenster betrachtet werden. *Tipp: Ein paar wichtige Funktionen und Skripts haben auch kurze Dokumentationen. Rufe sie z.B. mit* `doc` *gefolgt vom Namen auf.*

`runSim.m` Führt bis auf die grafische Ausgabe der Fehlerraten, die gleichen Operationen aus wie `runSimPlot.m`.

`ZFSim.m` Mit diesem Beispiel-Skript kann eine Simulation der Zero-Forcing Methode ausgeführt werden (siehe Punkt 4 in 3.1). Im Detail fügt das Skript einen neuen Eintrag dem Array `as_SimSettings` hinzu. Der letzte besetzte Index im Array ist immer durch die Variable `INDEX` angezeigt. Wird ein neuer Eintrag hinzugefügt muss diese Variable folglich um eins erhöht werden. Die meisten Einträge sind bereits mit den default-Werten (aus `defaultSettings.m`) initialisiert und werden deshalb auskommentiert und nur zu Anschauungszwecken in `ZFSim.m` aufgeührt.

Die Berechnung der Koeffizienten für die Egalisationsfilter geschieht in den unten aufgeführten Funktionen. Ein paar dieser Funktionen müssen in den Versuchen komplettiert werden.

`ComputeZeroForcingFilter.m` Berchnet ein stabiles inverses Filter $\tilde{G}(z)$ zu einer Übertragungsfunktion (linksseitig stabilen) $H(z)$. Hier muss die in Aufgabe 2 gefundene Rekursion und die Berechnung der Verzögerung aus Aufgabe 1 eingefügt werden.

`ComputeWienerFilter.m` Berechnet ein Wiener Filter $G_w(z)$ zur Übertragungsfunktion $H(z)$. Diese Datei muss nicht bearbeitet werden.

`ComputeDFEFilter.m` Berechnet die Filter $G_f(z)$ und $G_b(z)$ für das DFE Egalisationsverfahren aus 2.3. In dieser Funktion müssen die Koeffizienten aus Aufgabe 4 eingefügt werden.

`ComputeLMSFilter.m` Berechnet das adaptive Filter $g_k[.]$ mit dem LMS-Verfahren. Nebst der Filterordnung `InverseFilterLength`, müssen auch die Anzahl Trainingssymbole `TrainingLength` und der Schrittweiten-Parameter β `LearningBeta` vorgegeben werden. Die Lernregel aus Aufgabe 5 für das Filter $g_k[.]$ muss hier eingefügt werden.

`ComputeAdaptiveDFEFilter.m` Diese Funktion berechnet das adaptive DFE Filter aus 2.4.1.

3.1 Egalisation eines bekannten Kanals

Im ersten Teil der Versuche wird angenommen, dass der Empfänger den Kanal kennt. Es werden der Zero Forcing Algorithmus, das Wiener Filter und der DFE komplettiert und miteinander verglichen. Es wird empfohlen, für jede Simulation wie oben beschrieben eine neue Skript-Datei zu erstellen.

3. Komplettiere das Skript `ComputeZeroForcingFilter.m` mit der in Aufgabe 2 gefundenen Rekursion und der Berechnung der Verzögerung L aus Aufgabe 1.

4. Berechne ein inverses Filter zu $H(z)$ mit Länge 20 indem du das Skript `runSim.m` ausführst. In `runSim.m` sollte das Skript `ZFSim.m` aufgerufen werden. Erstelle einen Plot[10] der Filterkoeffizienten von $\tilde{G}(z)$. *Tipp: Das Filter $\tilde{G}(z)$ wurde nach der Ausführung von `runSim.m` im Array `as_SimSettings` im Feld `Gf` gespeichert.*

5. Benutze den Matlab-Befehl `conv` um herauszufinden, ob das berechnete Filter ungefähr invers zu $H(z)$ ist. Plotte die annähernd egalisierte Impulsantwort. Wie würde die egalisierte Impulsantwort mit einem idealen inversen Filter aussehen? Kann man die Verzögerung ablesen?

6. Berechne nun Zero Forcing Filter mit anderen Längen und schaue dir die Filterkoeffizienten von $\tilde{G}(z)$ und die Faltung mit $H(z)$ an. Ändere dazu den Wert von `InverseFilterLength` in der Datei `ZFSim.m`. *Tipp: Mit einer Schlaufe (z.B. `for`) in `ZFSim.m` kannst du mehrere Filterlängen auf einmal berechnen.*

Für die folgenden Fehlerrraten Simulationen, sollten Bit-Sequenzen der Länge 5000 (Konstante `SEQUENCE_LENGTH`) gewählt werden. Bei der Anzahl der Simulationsdurchgänge (Feld `NumberOfSimulationRuns` oder default Konstante `NUM_SIMULATION_RUNS`) muss man abwägen zwischen der Dauer einer Simulation und der Genauigkeit der Resultate. Deshalb sollte mit diesem Parameter experimentiert werden. Wähle den SNR-Bereich so, dass der aussagekräftige Teil der Bit-Error-Rate (zwischen 10^{-1} und 10^{-3}) gezeigt wird.

7. Erstelle als erstes einen Plot der Bit-Error-Rate für ein Zero Forcing Filter der Länge 20. Setze dazu die nötigen Werte in `runSimPlot.m` (bei den default Werten) oder in `ZFSim.m`. Führe das Skript `runSimPlot.m` aus.

8. Simuliere und erstelle nun einen Plot der die Bit-Error-Raten für das Zero Forcing Filter mit den Längen 10, 20, 30, 40 und 50. Ab welcher Ordnung lässt sich kaum mehr eine Verbesserung der Fehlerraten erkennen?

[10] Mit der Matlab Funktion `stem` lassen sich zeitdiskrete Signale schön darstellen.

Nun wollen wir die vorherige Egalisationsmethode mit der Wiener Filter Methode vergleichen. Das Wiener Filter wird im Skript `ComputeMyWienerFilter.m` berechnet. Dort muss aber nichts mehr verändert werden.

9. Das Wiener Filter ist abhänig von der Stärke des Rauschens - also der Rauschvarianz bzw. dem SNR. Es ist aufschlussreich, das Wiener Filter an verschiedenen SNR-Punkten mit dem Zero Forcing Filter zu vergleichen. Vergleiche dazu die Faltung des Filters mit dem Kanal (siehe Punkt 5) für die SNR-Punkte 0 dB, 15 dB und 30 dB. Dazu musst du für jeden SNR-Punkt eine separate Simulation laufen lassen. Was erkennt man?

10. Erstelle nun analog zu Punkt 8 ein Plot mit den Fehlerraten für verschiedene Wiener Filter Längen. Stelle alle Kurven in einem Plot dar und vergleiche die Performance. Welche Verzögerungen reichen für die optimale Performance des Wiener Filters?

11. Wie du vielleicht auf dem Plot im Punkt 10 erkennen konntest nähern sich die Fehlerraten mit der Zero Forcing Methode denen des Wiener Filters bei hohem SNR. Plotte die Filterkoeffizienten des Wiener Filters $g_w[.]$ und des Zero Forcing Filter $\tilde{g}[.]$ für die Länge 50 und SNR=30 dB übereinander. Kannst du die Beobachtung bei den Fehlerraten dadurch bestätigen?

Als nächstes wollen wir die Fehlerraten des DFE simulieren.

12. Komplettiere dazu das m-File `ComputeDFEFilter.m` mit den in Aufgabe 4 gefundenen Werten.

13. Erstelle auch hier einen Plot der Bit-Error-Rate und vergleiche mit dem Wiener Filter. Kannst du die unterschiedliche Performance erklären? *Tipp: Überlege dir welche Annahmen zur Sequenz $S[.]$ im DFE inherent enthalten sind (und im Wiener Filter nicht)? Ist das DFE ein lineares Filter?*

3.2 Bonus: Egalisation mit adaptiven Verfahren

In den nächsten Versuchen betrachten wir das Egalisationsproblem, falls der Kanal nicht bekannt ist. Wir werden deshalb adaptive Filter einsetzen. Ausserdem treffen wir die Annahme, dass der Sender eine bekannte Trainingssequenz der Länge `TrainingLength` und dann die Datensequenz der Länge `SequenceLength` = 5000 Symbolen sendet.

14. Komplettiere das Skript `ComputeLMSFilter.m` mit der in Aufgabe 5 gefundenen LMS-Lernregel für das Filter $g_k[.]$. Beachte, dass der Schrittweitenparameter β im Matlab-Code `betaT` genannt wird.

15. Wähle eine Trainingssequenz der Länge (`TrainingLength`) 2000, setzte den Schrittweiten-Parameter $\beta = 0.005$ (`LearningBeta`). Das SNR sollte 15 dB sein und das Egalisationsfilters sollte Länge 20 haben. Um herauszufinden wie gut der LMS Algorithmus konvergiert ist, erstelle einen Plot der die zeitliche Entwicklung der letzten 3 Filterkoeffizienten[11] zeigt. *Tipp: Auch in diesem Fall werden die Filterkoeffizienten vor den Simulationsdurchgängen berechnet und in `s_Settings` gespeichert.*

16. Mit den gleichen Einstellung wie im vorherigen Punkt, vergleiche die gelernten Filterkoeffizienten $g_{2000}[.]$ mit denjenigen von Wiener Filter und Zero Forcing Filter. Kannst du erkennen ob das gelernte Filter zu einem der bekannten Filter konvergiert? Zu welchem Filter konvergiert es?

[11] Die Funktion aus `ComputeLMSFilter.m` liefert nicht nur den endgültigen Wert der Filterkoeffizienten, sondern auch die Zwischenwerte. Deshalb wird das Filter in Matlab als zweidimensionaler Array gespeichert. Den Filterkoeffizient $g_k[n]$ findet man z.B. in Zeile k und Spalte n des zweidimensionalen Rückgabewert.

17. Wähle als Länge für das adaptive Filter 2 und berechne die Filter mit verschiedenen Werten für β. Rufe die Funktion `plotErrorSurface`[12] mit einer der zurückgegebenen `s_Settings` Variable auf. Diese Funktion zeichnet die Fehlerfläche (quadratische Fehler $\mathrm{E}\left[(S[.] - \tilde{S}[.])^2\right]$) in Abhängigkeit der beiden Filterkoeffizienten. Auf der Fläche werden ausserdem die Iterationsschritte des LMS als Pfad dargestellt. Je nach Wert von β und SNR, konvergiert der LMS sehr schnell nahe dem Minimum der Fläche. Für kleine SNR und/oder grosse β kann der Pfad nicht einmal mehr Richtung Minimum gehen.

18. Vergleiche die Fehlerraten von Wiener Filter, Zero Forcing und dem LMS Algorithmus. Spiele mit der Filterlänge und der Länge der Trainingssequenz für den LMS. Wie lang muss die Trainingssequenz sein damit das LMS bei gleichem SNR und Filterlänge tiefere Fehlerraten als das Zero Forcing Filter hat?

3.2.1 LMS Tuning und adaptives DFE

Um die Fehlerrate des LMS Algorithmus zu verbessern ohne die Länge der Trainingssequenz zu verändern, kann man die Lernregel erweitern. Wir wollen nach der Trainingsphase die geschätzten Symbole $\hat{S}[.]$ als Referenzsignal nuzten und so das Filter auch während dem normalen Betrieb trainieren. Da hierbei auch Fehler auftreten können, sollten die Schrittweiten Parameter für diese Lernregel (Feld `RunningBeta` in `s_Settings`) $5-10$ Mal kleiner als derjenige der Trainingsphase sein.

19. Setze die neue Lernregel für die Betriebsphase in `ComputeLMSFilter.m` ein. Beachte, dass es nun zwei Schrittweitenparameter gibt: `betaT` für die Schrittweite während der Trainingsphase und `betaR` für die Schrittweite während dem normalen Betrieb. Vergleiche die Fehlerraten mit dem vorherigen LMS. Braucht der neue Algorithmus weniger Trainingssymbole?

Zum Schluss soll noch die Fehlerrate des adaptiven DFE aus Abschnitt 2.4.1 untersucht werden. In diesem Fall gibt es auch eine Trainingsphase. Sie unterscheidet sich aber von der Trainingsphase des LMS, weil keine Referenzsignale zur Verfügung stehen.

20. Simuliere die Fehlerrate für den adaptiven DFE für verschieden lange Trainingsphasen und vergleiche sie mit derjenigen des nicht adaptiven DFEs.

4 Zusammenfassung

Es wurden drei weit verbreitete Egalisationsmethoden (Zero Forcing, Wiener Filter und DFE) eingeführt und an einem praktischen Kommunikationsmodel getestet. Im zweiten Teil wurden adaptive Verfahren (namentlich der LMS Algorithmus) auf das Kanal-Inversion Problem und den DFE angewendet. Aus den Versuchen ist ersichtlich, dass gute Inversion bei bestimmten Kanälen oft nur mit grosser Verzögerung möglich ist. Ausserdem ist die Inversion einer Verzerrung in der Regel nicht das optimalste Verfahren, wie die Vergleiche mit dem Wiener Filter und dem DFE zeigen. Adaptive Verfahren lassen sich einsetzen, falls der Kanal beim Empfänger unbekannt ist brauchen aber oft eine bestimmte Anzahl an Trainingssymbolen. Es wurde kurz aufgezeigt wie man durch geschickte Wahl von alternativen Referenzsignalen, manchmal sogar ganz auf Referenzsymbole verzichten kann.

Gratuliere! Du hast alle Versuche durchgearbeitet.

[12] Eine kurze Dokumentation zur Funktion lässt sich wie üblich in Matlab mit `doc plotErrorSurface` aufrufen.

Appendix

Die Simulationsumgebung besteht aus einer Schleife welche die unten aufgeführten Schritte für jeden SNR-Wert aus dem Vektor `SNRValuesToSimulate` mal der in `NumberOfSimulationRuns` gegebenen Anzahl durchgeführt. Die folgenden Schritte werden in dieser Reihenfolge ausgeführt:

`simulate.m` Führt die im Argument `s_Settings` spezifizierte Simulation aus. Eine Simulation besteht immer aus den gleichen Schritten. Die Schritte entsprechen jeweils einer Funktion:

1. `GenerateRandSignal.m` Generiert eine zufällige Symbol Sequenz der Länge `s_Settings.SequenceLength` aus den Symbolen im Vektor `s_Settings.InputAlphabet`. Wie in 1 wird diese Sequenz durch einen Kanal, der in `s_Settings.Channel` (in diesem Versuch der Kanal aus (1)) definiert ist, verzerrt. Schlussendlich wird dem Kanalausgang Rauschen hinzugefügt[13].
2. `EstimateInputSignal.m` Diese Funktion entspricht dem Egalisationsblock in Abb. 1. Als Rückgabewerte werden nebst der Variable `s_Settings` auch die geschätzten Input Symbole $\hat{S}[.]$ berechnet. Am Anfang werden die Koeffizienten des Egalisationsfilters berechnet. Die benutzte Egalisationsmethode wird aus dem Feld `str_EstimationAlgorithm` gemäss Tabelle 3 ausgelesen.
3. `ComputeErrors.m` Hier wird die Fehlerrate berechnet indem die Input Sequenz $S[.]$ mit den Schätzungen $\hat{S}[.]$ verglichen wird.

[13] Ein wichtiger Parameter dieser Funktion ist das Feld `Seed` in `s_Settings`. Dank dieser Zahl lässt sich bestimmen welche Zufallssequenzen generiert werden. Im Gegenzug kann gewährleistet werden, dass die Simulationsergebnisse reproduzierbar sind.