MULTIKANALSYSTEME

MATLAB Rechenübung

DIPL. ING. LUTZ MOLLE
PROF. DR.-ING. MARKUS NÖLLE

3. April 2020

Sommersemester 2020

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Studiengang Informations- und Kommunikationstechnik

Fachbereich 1, Ingenieurwissenschaften - Energie und Information
Hochschule für Technik und Wirtschaft (HTW) Berlin

1 EINFÜHRUNG

In dieser Rechenübung sollen die Grundlagen eines einfachen Funksystems mit Diversitäts- und Raummuliplexoptionen mit Hilfe eines numerischen Computeralgebrasystems nachvollzogen werden. Dazu werden die einzelnen Komponenten des Systems, nämlich der Sender, ein Funkkanal und ein Empfänger simulativ am Computer nachgebildet. Als Simulationsumgebung soll dazu entweder die proprietäre Software MATLAB [1] oder die Open-Source Alternative GNU Octave [2], [3] eingesetzt werden. MATLAB bietet eine komplette Entwicklungsumgebung (Editor, Kommandozeile, Debugger,...) und kann auch kostengünstig als Studentenversion bezogen werden [1].

GNU Octave ist in weiten Teilen kompatibel zu MATLAB und ist in den meisten Linuxdistributionen enthalten. Mittlerweile existiert auch eine recht stabile Windowsversion, die, ähnlich wie MATLAB, eine komplette Entwicklungsumgebung darstellt [2].

Die Rechner an der HTW haben eine vorinstallierte MATLAB Version, die für die Rechenübungen benutzt werden kann.

Für die Vor- und Nachbereitung der Rechenübung sollten Sie allerdings auch privat Zugriff auf eine der oben genannten Programme (entweder MATLAB oder GNU Octave) sicherstellen [1]–[3].

Die HTW bietet eine für Studenten kostenlose MATLAB Campus Lizenz an. Nutzen Sie am besten diese Gelegenheit um MATLAB auch auf Ihrem privaten Computer zu installieren und machen Sie sich in groben Zügen mit ihr vertraut. Eine Anleitung zur Installation dieser Lizenz finden Sie in dem Moodle Kurs zu dieser Veranstaltung.

Versuchen Sie im Vorfeld folgende Fragen zu beantworten:

- Wie finden Sie Hilfe für bestimmte Funktionen?
- Wie können Kommandos / Skripte ausgeführt werden?
- Wie werden Breakpoints gesetzt und der Debugger benutzt?
- Wie können Variablen inspiziert werden?

Für die grundlegende Einarbeitung in MATLAB / GNU Octave gibt es zahlreiche, online verfügbare Tutorials[1]. Diese reichen von einfachen, kurzen Einführungen bis hin zu sehr komplexen und spezialisierten Anleitungen. Auch einige Links zu Trainingsvideos können dort gefunden werden. Wenn nötig, nutzen Sie dieses Angebot und machen Sie sich grob mit weiteren Eigenheiten von MATLAB / GNU Octave vertraut. Während des ersten Übungstermins (und evtl. auch Zuhause als Nachbereitung des ersten Termins) sollten Sie folgende grundlegenden Fragestellungen bearbeiten:

- Was bedeuten die folgenden MATLAB / GNU Octave Funktionen: `transpose (') , colon (:), length, size, min, max, abs`. Sind diese auch für Vektoren oder Matrizen benutzbar?
- Was bedeuten die Operatoren .* oder ./ ?
- Wie definiert man einen linear ansteigenden / absteigenden Vektor?
- Welche Möglichkeiten gibt es in MATLAB / GNU Octave Teile eines Vektors zu indizieren?
- Wie definiert man Kommentare in einem Skript?

1 http://www.mathworks.de/academia/student_center/tutorials/launchpad.html

- Was bedeutet ein Semikolon nach einer Anweisung?
- Welche Strukturelemente gibt es? (Bedingte Verzweigungen, Schleifen, ...). Schreiben Sie ein Skript, welches einen linear ansteigenden Vektor generiert und berechnen Sie die Quadrate der einzelnen Elemente mittels einer Schleife. Wie kann man diese Aufgabe in MATLAB / GNU Octave eleganter (und effizienter) lösen?
- Wie können Sie eigene Funktionen definieren? Schreiben Sie eine einfache Funktion `quadMittel`, die den Mittelwert aus den quadrierten Elementen eines Vektors berechnet (am besten ohne eine Schleife zu benutzen).

Damit sollte eine erste sehr grobe Einarbeitung in MATLAB / GNU Octave abgeschlossen sein. Diese einfachen Grundfunktionen werden in den weiteren Terminen vorausgesetzt um die Simulation des Funksystems zu implementieren.

Im weiteren Verlauf der Rechenübung sollen Sie Schritt für Schritt einzelne Funktionen implementieren die bestimmte Aufgaben in der Simulation übernehmen. Es wird neben der funktionalen Beschreibung der Funktion nur der Funktionsaufruf (Name der Funktion und die Ein-/Ausgabeparameter) vorgegeben, so dass Sie bei der eigentlichen Implementierung völlig frei sind. Allerdings werden in jedem Abschnitt nützliche MATLAB / GNU Octave Grundfunktionen angegeben die zur Implementierung der einzelnen Aufgaben hilfreich sein können. Die Funktionsweise und Parameter der einzelnen Funktionen können Sie der MATLAB / GNU Octave Hilfe Funktion entnehmen. Zusätzlich werden zu jedem Abschnitt einige Fragen gestellt, die einerseits Hilfestellung geben, Sie aber auch andererseits dazu anhalten sollen sich detaillierter mit den einzelnen Funktionen und ihrer systemtechnischen Bedeutung auseinander zu setzen.

Ein weiterer guter Startpunkt für spezifische MATLAB / GNU Octave Problemstellungen bietet auch MATLAB Central [4] und dort besonders File Exchange, zu erreichen unter[2].

Zum Ende des Semesters (in einem der beiden letzten Termine) findet eine mündliche Aussprache pro Studentengruppe statt, in der Sie Ihre Implementation vorstellen und die implementierten Funktionen genauer erklären sollen. Bitte dokumentieren Sie Ihre Arbeit und vor allem die (Zwischen)ergebnisse während des Semesters in einem Bericht. Diese schriftliche Ausarbeitung muss spätestens zu der mündlichen Aussprache abgegeben werden. Die Endnote zu dieser Rechenübung setzt sich dann zu gleichen Teilen aus der Bewertung der schriftlichen Ausarbeitung und der mündlichen Aussprache zusammen.

2 http://www.mathworks.de/matlabcentral/fileexchange/

2 SIMULATION EINES SISO SYSTEMS

Genau wie in der Vorlesung soll zuerst ein Funksystem mit jeweils nur einer Sende- und Empfangsantenne betrachtet werden. Dazu wird die Simulation eines solchen Systems Schritt für Schritt (vom Sender zum Empfänger) aufgebaut.

2.1 *Grundsätzliches zur Simulation*

Als eigentliche Simulation fungiert ein MATLAB Skript in dem die grundsätzlichen Parameter der Simulation eingestellt werden können. Aus diesem Skript heraus werden dann einzelne Funktionen (die Sie im weiteren Verlauf implementieren werden) aufgerufen. Diese Funktionen führen einzelne Funktionalitäten des Senders, des Kanals und des Empfängers aus.

Versuchen Sie in die einzelnen Funktionen möglichst viele Plausibilitätsprüfungen einzubauen und Fehler so früh wie möglich mit Aussagekräftigen Fehlermeldungen abzufangen. So können Sie z.B. die Eingabeparameter auf eine korrekte (plausible) Größe oder auf die Art ihrer Elemente hin überprüfen. Falls Sie z.B. einen Zeilenvektor `x` der Länge `N` erwarten, können Sie folgende Fehlerprüfung zu Beginn der Funktion durchführen.

Listing 1: Beispiel zur Fehlerprüfung zu Beginn einer Funktion

```
...
[r, c] = size(x);
if (r ~= 1) || (c ~= N) || (ndims(x) ~= 2))
    error(sprintf('functionname: input vector x needs to be a row...
    vector of length N (N = %d)...',N));
end
...
```

Diese einfachen Fehlerabfragen werden die Simulation deutlich stabiler machen und eine spätere Fehlersuche erheblich vereinfachen.

Für verschiedene Kanalzustände (z.B. SNR Werte) werden Signale am Sender generiert, übertragen und empfangen, bis eine gewisse Anzahl von Fehlern erreicht sind oder eine maximale Anzahl von Bits simuliert wurden.

Ein solches Simulationsskript (beschrieben in Pseudo-Code) sieht aus wie in Listing 2.

Erzeugen Sie nun ein Simulationsskript mit dem Namen `simulation.m` welches wie in Listing 3 beginnt.

Im Folgenden werden nun die einzelnen Funktionen besprochen, die implementiert und innerhalb der Schleife aufgerufen werden sollen.

2.2 *Erzeugung der Sendesignale am Sender*

Zuerst müssen in dem Sender die Signale, die übertragen werden sollen erzeugt werden. Dazu werden zu allererst zufällige Bits benötigt.

2.2.1 *Erzeugung der Bitsequenzen*

Daher implementieren Sie bitte eine Funktion `y = generateBits(nBits)`, die eine als Eingabeparameter die Anzahl der zu erzeugenden Bits (`nBits`) erhält und als Ausgabeparameter (`y`) die erzeugten Bits ausgibt. Dabei soll `y`

Listing 2: Beispiel eines Simulationsskripts

```
%% Definition von Simulationsparametern
SNR=1...10
Modulationformat=QPSK
...
% Simulationsschleifen
for SNR
    while ((Anzahl Fehler kleiner 100) or
        (Max Anzahl simulierter Bits erreicht))

        Senderfunktion1(...)
        (evtl. Visualisierung von Zwischenergebnissen)
        Senderfunktion2(...)
        ...

        Kanalfunktion1(...)
        Kanalfunktion2(...)
        (evtl. Visualisierung von Zwischenergebnissen)
        ...

        Empfaengerfunktion1(...)
        Empfaengerfunktion2(...)
        (evtl. Visualisierung von Zwischenergebnissen)
        ...

        Bestimmung der Bitfehler / BER
    end
    Visualisierung von Endergebnissen (z.B. BER vs. SNR)
end
```

ein Zeilenvektor sein, der als Elemente die einzelnen erzeugten Bits enthält. Diese Funktion wird als erste Funktion innerhalb der Simulationsschleife aufgerufen.

Erzeugen Sie zum Test nur eine kleine Anzahl von Bits (z. B. 50) und visualisieren sie diese.

- Welche Möglichkeiten gibt es in MATLAB / GNU Octave zufällige Daten zu erzeugen? Wie sind diese Daten verteilt und welche Verteilung ist bei der Erzeugung von Bits sinnvoll?
- Wie kann man Daten (Vektoren, Matrizen,...) graphisch darstellen?
- MATLAB / GNU Octave Funktionen: `randn, rand, randi, logical, plot, stem`

2.2.2 *Zuordnung der Bitsequenzen zu Modulationssymbolen*

Nachdem nun die Bits erzeugt wurden, müssen diese Bits verschiedenen Konstellationspunkten (abhängig vom verwendeten Modulationsformat) zugeordnet ("gemapped"von englisch to map) werden. Dafür implementieren Sie bitte eine Funktion `y = mapper(bits, const)`, die als Eingabeparameter sowohl den gerade erzeugten Bitvektor `bits`, als auch die einzelnen Konstellationspunkte eines Modulationsformats als Vektor `const` erhält. So würde einer BPSK (binary phase shift keying) Modulation z.B. ein Vektor [-1, 1] entsprechen. Für eine QPSK hingegen enthält der Konstellationspunkte-Vektor

Listing 3: Beginn des eigentlichen Simulationsskrpts

```
% radio communication system simulation script
%

clc; clear variables; % clear all variables
addpath('subfunctions'); % add directory "subfunctions" to path

% global simulation parameters
ebN0dB = 0:30; % SNR (per bit) in dB
nBitsPerLoop = 10e3; % simulate nBitsPerLoop bits per simlation loop
nMinErr = 100; % simulate at least nMinErr bit errors...
nMaxBits = 100 * nBitsPerLoop; % ... or stop after nMaxBits

const = [-1-1j, 1-1j, -1+1j, 1+1j]; % constellation of the
% modulation format here: QPSK wiht Gray mapping)

% here goes the simulation loop...
```

Bitwert	Dezimalzahl	Index	Konstellationspunkt
00	0	1	-1-j
01	1	2	-1+j
10	2	3	1-j
11	3	4	1+j

Tabelle 1: Beispiel für ein Gray Mapping zwischen Bitwerten und Konstellationspunkten für eine QPSK. Konstellationspunkte-Vektor [-1-j, -1+j, 1-j, 1+j]

die vier Elemente [-1-j, -1+j, 1-j, 1+j]. Der Ausgabeparameter y soll ein Zeilenvektor sein, in dem jedes Element einem (komplexen) Modulationssymbol entspricht.

Die Zuordnung der einzelnen komplexen Zahlen (Konstellationspunkte) zu den jeweiligen Bitwerten kann damit über die Position der Konstellationspunkte innerhalb des Vektors bestimmt / variiert werden. So wird der erste Konstellationspunkt innerhalb des Vektors dem Bitwert 0 zugeordnet. Der zweite Konstellationspunkt dem Bitwert 1, usw.

Als Beispiel soll nun eine QPSK betrachtet werden. Für eine QPSK werden jeweils zwei Bits zusammengefasst und in eine Dezimalzahl konvertiert. Zu dieser wird dann der Wert 1 addiert und das Ergebnis kann als Index für den Konstellationspunkte-Vektor benutzt werden um den jeweiligen Konstellationspunkt zu bestimmen. Für eine QPSK und Gray Mapping von Bits zu Konstellationspunkten ist dieser Zusammenhang in Tabelle 1 dargestellt. Dies entspricht einem Konstellationspunkte-Vektor von [-1-j, -1+j, 1-j, 1+j].

Variieren Sie das Modulationsformat (z.B. 16QAM und 8-PSK) und visualisieren Sie die so erzeugten Symbole (Konstellationspunkte) in einem Konstellationsdiagramm um die korrekte Funktionsweise der implementierten Funktion zu prüfen. Für die weitere Simulation soll allerdings im Folgenden eine QPSK benutzt werden.

- Wie wird eine binäre Zahl in eine Dezimalzahl umgewandelt?

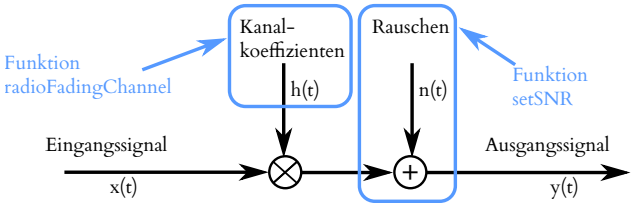

Abbildung 1: Schematische Darstellung der Simulation eines Rayleighkanals für ein SISO System

- Wie kann diese Aufgabe effizient in MATLAB / GNU Octave implementiert werden?
- MATLAB / GNU Octave Funktionen: `reshape, mod, length, log2, sum, '`

2.3 *Simulation eines gedächtnislosen Rayleigh Funkkanals*

Nachdem nun das Sendesignal (die Modulationssymbole) erzeugt wurden, soll dieses Signal nun über einen Funkkanal übertragen werden. In einem ersten Schritt soll hier ein Rayleigh-Funkkanal implementiert werden.

Der Rayleigh-Funkkanal besteht zum einen aus einem zeitliche veränderlichen Schwundkanal und einem anschließenden additiven, weißen, gaußverteilten Rauschen (AWGN). Daher sollen zwei verschiedene Funktionen implementiert werden, die jeweils eine dieser Aufgaben erledigen.

Eine schematisches Diagramm eines solchen Kanals ist in Abbildung 1 dargestellt. Der Schwundkanal kann als Multiplikation des Signals mit den komplexen Kanalkoeffizienten simuliert werden, wohingegen das Rauschen dem Signal additiv hinzugefügt wird. Die Kanalkoeffizienten des Schwundkanals sollen mit der Funktion `radioFadingChannel` erzeugt werden, wohingegen die Funktion `setSNR` das Rauschen zu dem Signal addieren soll.

2.3.1 *Kanalkoeffizienten*

Zuerst implementieren Sie bitte die Funktion `y = radioFadingChannel(nSamp)`. Diese erhält als Eingabeparameter `nSamp` die Anzahl der Kanalkoeffizienten, die erzeugt werden sollen. Als Ausgabeparameter `y` werden die komplexen Kanalkoeffizienten ausgegeben.

Bei dieser Simulation soll die Geschwindigkeit der Änderung des Kanals durch die Dopplerverschiebung (siehe Vorlesungsfolien) nicht simuliert werden. Es wird einfach angenommen, dass der Kanal während eines Übertragung eines Modulationssymbols konstant ist, sich beim nächsten Modulationssymbol aber wieder anders verhält. Das heißt, da das Sendesignal jeweils nur einen Abtastwert pro Modulationssymbol enthält, wird, pro Modulationssymbol was über den Kanal übertragen werden soll, ein einzelner Kanalkoeffizient erzeugt.

Wie in der Vorlesung besprochen besteht sowohl der Real- als auch der Imaginärteil eines Kanalkoeffizienten durch die Mehrwegeausbreitung aus einer Überlagerung vieler einzelner Empfangssignale. Daher ist Real- als auch der Imaginärteil guaßverteilt mit einem Mittelwert von Null.

Das heißt die Funktion muss `nSamp` komplexe Zahlen erzeugen, deren Real- und Imaginärteile jeweils einer mittelwertfreien Gaußverteilung entsprechen. Bitte beachten Sie jedoch, dass der Kanal im Mittel weder dämpfen noch

verstärken soll. Daher sollten die Kanalkoeffizienten `y` so normiert werden, dass ihre mittlere Leistung 1 entspricht.

Bitte visualisieren Sie die Verteilungsdichtefunktion (oder besser: das Histogramm) des Betrags und der Phase der erzeugten Kanalkoeffizienten. Vergleichen Sie dieses Histogramm mit den aus der Vorlesung bekannten theoretischen Verteilungsdichtefunktionen von Amplitude und Phase der Kanalkoeffizienten eines Rayleighkanals.

- Was bedeutet eine Gaußverteilung in Real- und Imaginärteil für die Verteilungsdichtefunktion von Amplitude und Phase der Kanalkoeffizienten?
- MATLAB / GNU Octave Funktionen: `randn, mean, abs, sqrt, hist, bsxfun`

2.3.2 *Additives weißes gaußverteiltes Rauschen*

Nachdem das Sendesignal über den gedächtnislosen Schwundkanal übertragen wurde, d.h. jedes Modulationssymbol mit dem komplexen Kanalkoeffizienten multipliziert wurde, soll dem Signal Rauschen mit einem bestimmten Signal-Rauschabstand (signal to noise ratio, SNR) zugefügt werden. Dazu implementieren Sie bitte die Funktion `y = setSNR(x, snrdB)`. Diese Funktion erhält als Eingabeparameter das Signal `x` als Zeilenvektor von komplexen Zahlen und das gewünschte SNR pro Symbol in dB (`snrdB`). Der Ausgabeparameter `y` beschreibt das verrauschte Eingangssignal, wieder als Zeilenvektor aus komplexen Zahlen.

Bitte beachten Sie, dass in der Vorlesung alle Bitfehlerratenkurven und -formeln als Funktion des SNR pro Bit angegeben wurden. Des Weiteren soll auch die komplette Simulation als Parameter das SNR pro Bit beschreiben.Daher müssen Sie das SNR pro Symbol, welches der Funktion übergeben werden soll, erst einmal aus dem SNR pro Bit und dem zu simulierenden Modulationsformat errechnen. Wenn das Modulationsformat M Konstellationspunkte besitzt, dann lässt sich das SNR pro Symbol wie folgt berechnen

$$\mathrm{SNR_s} = \mathrm{SNR_b} \cdot \log_2(M) \tag{1}$$

Bitte beachten Sie, dass sowohl das SNR pro Bit ($\mathrm{SNR_b}$), als auch das SNR pro Symbol ($\mathrm{SNR_s}$) in Gleichung 1 das lineare SNR bezeichnet. Gegebenenfalls müssen Sie daher SNR Werte die vorher in dB vorliegen zuerst ins lineare umrechnen und das Ergebnis anschließend wieder in dB zurück rechnen.

In der Funktion berechnen Sie dann zuerst die mittlere Leistung des Signals `x`. Mit dem gegebenen $\mathrm{SNR_s}$ und der mittleren Signalleistung können Sie dann die Rauschleistung P_N berechnen. Erzeugen Sie nun einen Vektor aus komplexen Zufallszahlen mit gaußverteiltem Real- und Imaginärteil, welcher die gleiche Länge wie das Signal und eine mittlere Leistung von P_N hat. Diesen addieren sie zu dem unverrauschten Signalvektor `x`.

- Wie berechnet man die mittlere Leistung eines Signals?
- Wie ändert man die Varianz (oder die Standardabweichung) einer gaußverteilten Zufallsvariable?
- MATLAB / GNU Octave Funktionen: `randn, mean, abs, sqrt, size`

2.4 *Empfänger*

In dieser Simulation wird angenommen, dass der Empfänger den Kanal ideal geschätzt hat und somit alle Kanalkoeffizienten kennt. Aus diesem Grund können sie Amplituden- und Phasenschwankungen des Schwundkanals am Empfänger ideal kompensiert werden. Teilen Sie also das empfangene Signal durch die bekannten Kanalkoeffizienten und führen Sie damit eine ideale Kompensation durch. Ohne den Einfluss des Rauschens wäre also eine perfekte Rekonstruktion des Sendesignals möglich.

Anschließend müssen die einzelnen Modulationssymbole des Empfangssignals entschieden, aus diesen entschiedenen Symbolen die Bits bestimmt und zuletzt die Anzahl der Bitfehler gezählt werden. Dafür implementieren Sie bitte die folgenden drei Empfängerfunktionen, die diese Aufgaben übernehmen.

2.4.1 *Entscheidung der empfangenen Symbole*

Die Funktion `y = decision(x, const)` erhält einen Zeilenvektor `x`, der das empfangene Signal enthält und einen Vektor mit den Konstellationspunkten des zu detektierenden Modulationsformats `const`. Die Funktion soll nun für jeden Abtastwert des Eingangssignals den Abstand zwischen diesem Eingangssignal und jedem möglichen Konstellationspunkt berechnen. Anschließend wird angenommen, dass der Konstellationspunkt mit dem kleinsten Abstand das gesendete Modulationssymbol war. Es wird also am Empfänger eine harte Entscheidung zugunsten eines Modulationssymbols getroffen.

Das Ausgangssignal `y` ist nun ein Zeilenvektor, der nun die entschiedenen Modulationssymbole als Elemente enthält.

Bitte beachten Sie, dass das Signal vor der Entscheidung eventuell zuerst auf die mittlere Leistung der originalen Konstellationspunkte normiert werden muss, damit eine sinnvolle Entscheidung (Abstandsberechnung) getroffen werden kann.

Visualisieren Sie das empfangene Signal einmal vor und nach der Entscheidung und beschreiben Sie die Unterschiede. Ist diese Visualisierung plausibel? Warum?

- Wie berechnet man die mittlere Leistung eines Signals (oder Vektors)?
- Wie ändert man die Varianz (oder die Standardabweichung) einer gaußverteilten Zufallsvariable?
- MATLAB / GNU Octave Funktionen: `randn, mean, abs, sqrt, size, bsxfun`

2.4.2 *Zuordnung der entschiedenen Symbole in Bits*

Nun soll mit Hilfe der Funktion `y = demapper(x, const)` eine inverse Funktion zu der Funktion `mapper` des Senders implementiert werden. Diese Funktion erhält einerseits einen Zeilenvektor mit den entschiedenen Symbolen `x` und andererseits die originalen Konstellationspunkte des Modulationsformats `const`. Der Ausgabeparameter `y` enthält die laut Zuordnungsvorschrift (siehe Unterunterabschnitt 2.2.2) zugehörigen Bitwerte. Das heißt, wenn das Modulationsformat M Konstellationspunkte besitzt, enthält der der Vektor `y` $\log_2(M)$ mal so viele Elemente wie der Vektor `x`.

Dafür ist folgendes Vorgehen vorteilhaft: Zuerst muss für jedes entschiedene Symbol im Vektor `x` ermittelt werden an welcher Position dieses Symbol innerhalb des Vektors `const` auftaucht. Mit anderen Worten, es wird für jedes Element im Vektor `x` der Index des korrespondierenden Konstellationspunkts im Vektor `const` bestimmt. Danach können, da dieser Dezimalzahl Index, subtrahiert mit dem Wert 1, und konvertiert in eine binäre Zahl genau dem Bitwert entspricht (siehe Unterunterabschnitt 2.2.2), die zugehörigen Bitwerte so leicht bestimmt werden.

Simulieren Sie zuerst nur wenige Bits (10-20) und vergleichen Sie sowohl die Symbole vor und nach der Entscheidung, als auch die gesendeten Bits mit den entschiedenen. Sind die Entscheidungen und die Zuordnungen plausibel? Falls nicht, vereinfachen Sie die Simulation (vernachlässigen Sie den Kanal und das Rauschen) und vergleichen Sie die Entscheidungen und die Zuordnungen erneut.

- MATLAB / GNU Octave Funktionen: `bitget, logical, log2, nan, isnan, any`

2.4.3 *Fehlerzählung*

Nun sollen mit Hilfe der Funktion `[nErr, idx, ber] = countErrors(x, bits)` die entstandenen Bitfehler gezählt werden. Dafür erhält diese Funktion einerseits einen Zeilenvektor `x` mit den empfangenen und entschiedenen Bits und andererseits einen Zeilenvektor `bits` mit den ursprünglich gesendeten Bits.

Als Ausgabeparameter soll die Funktion ide Anzahl der Bitfehler (die Anzahl der Elemente die in den beiden Eingangsvektoren unterschiedlich sind) `nErr`, deren Positionen (Indices) innerhalb des Eingangsvektors `idx` und die Bitfehlerrate `ber` ausgeben. Die Bitfehlerrate ist dabei gegeben als das Verhältnis zwischen der Anzahl der Fehler und der Anzahl der gesendeten Bits.

Wie schon in Unterabschnitt 2.1 erwähnt soll die Simulationsschleife so lange durchlaufen werden, bis entweder die maximale Anzahl von Bits simuliert wurden, oder bis eine Mindestanzahl von Bitfehlern aufgetreten ist. Der Rückgabewert `nErr` kann also zu einer zweiten Variable `nTotalErr`, die der Gesamtanzahl der Bitfehler über alle Schleifendurchläufe entspricht hinzu addiert werden. Die Variable `nTotalErr` kann dann als zweites Abbruchkriterium für die Schleife benutzt werden. Im Regelfall sollte die Simulation mindestens so lange durchgeführt werden, dass 100 Bitfehler auftreten.

Damit ist die erste, einfach Simulation eines Rayleighkanals abgeschlossen.

Bitte simulieren Sie den Kanal für verschiedene SNR Werte und bestimmen Sie die resultierenden Bitfehlerraten. Visualisieren Sie diese Ergebnisse (z.B. in einem Plot, der die Bitfehlerrate als Funktion des SNR (in dB) darstellt, ähnlich zu dem Graphen in Abbildung 2). Vergleichen Sie dazu die aus der Vorlesung bekannte theoretisch erreichbare Bitfehlerrate für eine QPSK Übertragung über einen Rayleighkanal [5], gegeben in Gleichung 2.

$$P_{b\,|\,QPSK}^{Rayleigh} = \frac{1}{2} \cdot \left(1 - \sqrt{\frac{SNR_b}{1 + SNR_b}}\right) \tag{2}$$

- Warum sollten in einer Simulation mindestens 100 Bitfehler auftreten?
- MATLAB / GNU Octave Funktionen: `=, find, size, sum, semilogy, xlim, ylim, legend, xlabel, ylabel, axis`

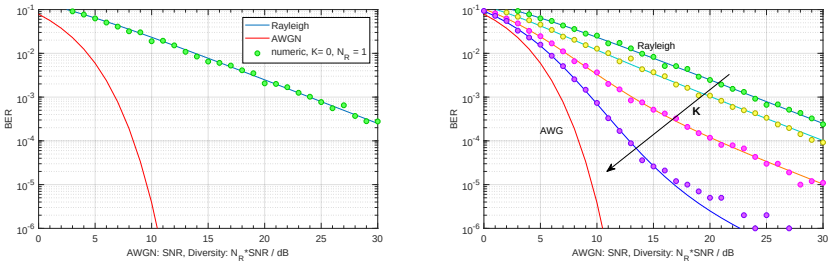

Abbildung 2: Bitfehlerrate als Funktion des SNR pro Bit für einen Rayleigh Kanal (links) und für einen Rice Kanal mit unterschiedlichen Werten für den Parameter K ($K = 1, 2, 5, 10$) (rechts). Die Linien zeigen analytische Kurven, während die Punkte numerisch ermittelte Simulationswerte darstellen.

2.5 *Simulation eines Rice Funkkanals*

Wie aus der Vorlesung bekannt, enthält ein Rice Funkkanal sowohl einen direkten Übertragungspfad (eine line of sight, LOS Komponente) als auch Streukomponenten (non line of sight, NLOS Pfade). Um nun einen Ricekanal zu simulieren muss die Funktion `radioFadingChannel` aus dem Unterunterabschnitt 2.3.1 um diese Funktionalität erweitert werden.

Der Parameter K gibt dabei das Verhältnis der Leistungen zwischen LOS und NLOS Pfaden an und ist daher definiert als

$$K = \frac{P_{LOS}}{P_{NLOS}} = \frac{a_{LOS}^2}{2\sigma^2} \tag{3}$$

Erweitern Sie nun die Kanalfunktion um einen weiteren Eingabeparameter K, so dass der Funktionsaufruf nun `y = radioFadingChannel(nSamp, K)` lautet. Generieren Sie nun in der Funktion pro Kanalkoeffizient, also pro Abtastwert, eine zusätzliche LOS Komponente mit einer zufälligen Phase und einer Leistung von $P_{LOS} = K \cdot P_{NLOS}$. Addieren Sie diese LOS Komponenten zu den zuvor erzeugten Koeffizienten der NLOS Komponenten und geben Sie diese so erzeugten Kanalkoeffizienten (LOS+NLOS Komponente) als Zeilenvektor y zurück.

Bitte beachten Sie, dass der Kanal nach hinzufügen der LOS Komponente nochmals normiert werden muss, damit er weder verstärkt noch dämpft (also eine mittlere Leistung von 1 aufweist).

Um die Implementation der neuen Funktion auf Plausibilität zu überprüfen können Sie die Simulation einmal mit der ursprünglichen Funktion `radioFadingChannel` durchführen und ein zweites Mal mit der neuen Funktion und einem Eingabeparameter von $K = 0$. Beide Simulationen müssen die gleichen Ergebnisse liefern, da der Fall $K = 0$ genau einem Rayleighkanal entspricht.

Führen Sie die Simulation für verschiedene SNR Werte und unterschiedliche Ricekanäle (verschiedene Werte von K) durch und vergleichen Sie die Ergebnisse mit den bekannten, theoretisch erreichbaren Bitfehlerraten für eine QPSK Übertragung über einen Ricekanal [5], gegeben in Gleichung 4. Visualisieren Sie Ihre Ergebnisse ähnlich wie in Abbildung 2 dargestellt.

$$P_{b|QPSK}^{Rice} = \frac{1}{\pi} \int_0^{\pi/2} \frac{(1+K)\sin^2(\theta)}{(1+K)\sin^2(\theta) + SNR_b} \exp\left(-\frac{K \cdot SNR_b}{(1+K)\sin^2(\theta) + SNR_b}\right) d\theta$$

(4)

Erhöhen Sie den Wert für `K` und vergleichen Sie die Ergebnisse mit den theoretisch erreichbaren Bitfehlerraten für eine QPSK Übertragung über einen AWGN Kanal. Diese können Sie z.B. mir der MATLAB Funktion `berawgn` erzeugen.

- Warum nähern sich die Ergebnisse für einen steigenden Parameter K denen eines AWGN Kanals an?
- MATLAB / GNU Octave Funktionen: `randn, rand, mean, trapz, quad, integral, berawgn`

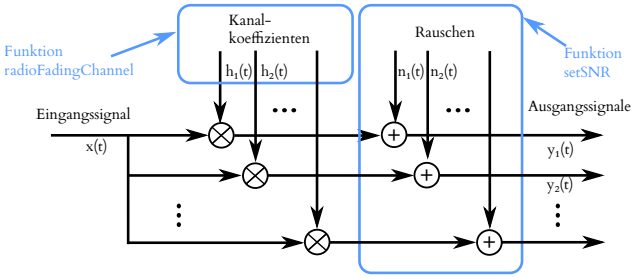

Abbildung 3: Schematische Darstellung der Simulation eines Rayleighkanals für ein SIMO System

3 SIMULATION EINES SIMO SYSTEMS

In diesem Teil der Rechenübung soll ein System mit mehreren Empfangsantennen (also ein single input multiple output, SIMO System) simuliert werden. Dafür müssen manche Funktionen erweitert, bzw. neue Funktionen implementiert werden.

3.1 *Änderung des Funkkanals*

Da weiterhin nur eine Sendeantenne angenommen wird, kann die Signalerzeugung am Sender unverändert bleiben.

Allerdings müssen die Funktionen des Funkkanals angepasst werden um die Funktionalität der Empfangsdiversität abzubilden. Dies ist schematisch in Abbildung 3 dargestellt.

Kanalkoeffizienten

Erweitern Sie daher die Kanalfunktion um einen weiteren Eingabeparameter `Nr`, so dass der Funktionsaufruf nun `y = radioFadingChannel(nSamp, K, Nr)` lautet. Der neue Parameter `Nr` gibt die Anzahl der Empfangsantennen und damit die Anzahl der verschiedenen Kanalkoeffizienten pro Abtastwert an. Das bedeutet, dass innerhalb der Funktion nun nicht mehr nur ein, sondern `Nr` Kanalkoeffizienten pro Abtastwert erzeugt werden müssen. Jeder einzelne dieser `Nr` Kanalkoeffizienten beschreibt einen eigenen Ricekanal zu einem Zeitpunkt.

Aus diesem Grund ändert sich auch die Gestalt des Ausgabeparameters: `y` ist nun nicht mehr zwangsläufig ein Zeilenvektor, sondern eine Matrix mit der Dimension `Nr` x `nSamp`. Das heißt, die einzelnen Spalten geben weiterhin den Kanalzustand für einen Zeitpunkt (Abtastwert) an. Die einzelnen Zeilen hingegen beschreiben den Kanal für jeweils eine einzelne Empfangsantenne zu unterschiedlichen Zeitpunkten. In anderen Worten wird der Gesamtkanal zu einem Zeitpunkt nun nicht mehr mit einem Skalar, sondern mit einem Spaltenvektor (der Länge `Nr`) beschrieben.

Bitte beachten Sie, dass der *Gesamtkanal* weder dämpfen noch verstärken soll. Dass heißt, die Kanalkoeffizienten der einzelnen Empfangsantennen müssen jeweils so skaliert werden, dass ihre mittlere Leistung `1/Nr` entspricht.

Rauschen

Auch die Funktion `y = setSNR(x, snrdB)` die das Rauschen erzeugt, muss nun mit einem Eingabeparameter `x` in Matrixform umgehen können und ihrerseits eine (verrauschte) Ausgangsmatrix der Dimension `Nr x nSamp` erzeugen. Bitte prüfen Sie, ob Ihre Implementierung der Funktion diese Funktionalität leistet. Falls nicht, ändern Sie bitte die Implementierung.

Bitte beachten Sie außerdem, dass nun der Parameter `snrdB` das SNR_s für jede einzelne Empfangsantenne (d.h. für jede einzelne Zeile der Matrix `y`) angibt. Daher könnte, unter Annahme von unterschiedlichen Signalleistungen in den einzelnen Antennen auch die Rauschleistung pro Empfangsantenne unterschiedlich sein. Das bedeutet, dass zuerst die mittlere Signalleistung der einzelnen Zeilen (Antennen) ermittelt werden muss, bevor das Rauschen mit einer dem jeweiligen SNR_s entsprechenden Rauschleistung für die einzelnen Zeilen der Ausgangsmatrix `y` generiert werden kann.

- Natürlich ist es möglich die Funktionalität der Empfangsdiversität innerhalb der Funktionen als eine Schleife über die Anzahl der Empfangsantennen zu realisierten. Gibt es in MATLAB / GNU Octave eine effizientere Alternative zu einer Schleife?
- MATLAB / GNU Octave Funktionen: `bsxfun`

3.2 *Vergleich verschiedener Kombinationsvektoren*

Um die Empfangssignale y_i der einzelnen Antennen zu kombinieren wird eine Linearkombination aus diesen Signalen gebildet. Das heißt, die einzelnen Signale y_i werden mit einem Skalierungsfaktor z_i multipliziert und dann zu einem einzigen Signal r aufaddiert. Dies geschieht für jeden einzelnen Zeitpunkt oder Abtastwert. Wie in der Vorlesung gesehen ergibt sich damit das Ausgangssignal r (vor dem Entscheider) in Matrixschreibweise zu

$$r = \mathbf{z}^T \mathbf{h} \cdot x + \mathbf{z}^T \mathbf{n}. \tag{5}$$

Es wird im Folgenden angenommen, dass die Kanalkoeffizienten am Empfänger bekannt sind und damit eine ideale Kanalkorrektur vorgenommen werden kann.

Der Skalierungs- oder Kombinationsvektor $\mathbf{z}$ kann nun unterschiedlich gewählt werden.

Eine (und für die Fehlerrate des Systems unvorteilhafte) Möglichkeit ist eine einfach Summation aller Antennensignale, was einem Kombinationsvektor von $\mathbf{z}^T = \begin{bmatrix} 1 & \dots & 1 \end{bmatrix}$ entspricht. Da sich die einzelnen Antennensignale nun weiterhin destruktiv aufaddieren können und damit zur Entzerrung das Rauschen stark angehoben werden muss kann mit einem solchen Kombinationsvektor kein Diversitätsgewinn erzielt werden.

Weitere Möglichkeiten für die Auswahl des Kombinationsvektors und ihre Leistungsfähigkeit für das Gesamtsystem wurden in der Vorlesung besprochen. Diese Methoden sind:

- maximum ratio combining (**MRC**), $\mathbf{z}^T = \begin{bmatrix} h_1^* & \dots & h_{N_R}^* \end{bmatrix}$
- equal gain combining (**EGC**), $\mathbf{z}^T = \begin{bmatrix} e^{-j \cdot \arg(h_1)} & \dots & e^{-j \cdot \arg(h_{N_R})} \end{bmatrix}$

- selection diversity combining (**SDC**), $\mathbf{z}^T = \begin{bmatrix} 0 & \dots & z_i & \dots & 0 \end{bmatrix}$, mit $z_i = 1, i \in 0 \dots N_R$.

Implementieren Sie nun bitte eine Funktion `y = antennaCombining(x, h, combMethod)`, die die einzelnen Antennensignale nach diesen unterschiedlichen Methoden kombinieren kann. Diese Funktion erhält als Eingabeparameter die Antennensignale `x` und die Kanalkoeffizienten `h`, jeweils als Matrix der Dimension `Nr` x `nSamp`.

Zusätzlich gibt der Parameter `combMethod` an welche Kombinationsmethode (oder welcher Kombinationsvektor) benutzt werden soll. Der Eingabeparameter `combMethod` kann folgende Zeichenketten enthalten: 'sum', 'MRC', 'EGC' und 'SDC'.

Der Ausgabeparameter `y` ist ein Vektor, der das kombinierte Signal `r` enthält. Mit dieser Funktion kann also die Kombination der Antennensignale vor dem Entscheider durchgeführt werden.

Führen Sie Simulationen für verschiedene Kombinationsvektoren unter der Annahme von zwei Empfangsantennen durch und vergleichen Sie die Ergebnisse. Visualisieren Sie Ihre Resultate und vergleichen Sie sie mit den theoretischen Bitfehlerratenkurven für eine QPSK Übertragung und SDC / MRC für einen Rayleighkanal (ähnlich wie in Abbildung 4 dargestellt). Diese sind gegeben durch

$$P_b^{SDC} = N_R \int_0^\infty Q\left(\sqrt{\frac{2E_b}{N_0}\gamma}\right)(1-e^{-\gamma})^{N_R-1}e^{-\gamma}d\gamma \tag{6}$$

einerseits und

$$P_b^{MRC} = \frac{1}{\Gamma(N_R)} \int_0^\infty Q\left(\sqrt{\frac{2E_b}{N_0}\gamma}\right)\gamma^{N_R-1}e^{-\gamma}d\gamma \tag{7}$$

andererseits. Wobei $Q(x) = 0.5 \cdot \operatorname{erfc}\left(\frac{x}{\sqrt{2}}\right)$, $\operatorname{erfc}(x) = 1 - \operatorname{erf}(x)$ und $\Gamma(x)$ die Gamma-Funktion ist.

- Erklären Sie anschaulich warum sich für die einfache Summation aller Antennensignale (Option 'sum') kein Diversitätsgewinn ergibt.
- Welches ist die beste Methode? Und Warum?
- MATLAB / GNU Octave Funktionen: `switch, case, bsxfun, integral, trapz, quad`

3.3 *Vergleich unterschiedlicher Diversitätsordnungen*

Führen Sie Simulationen für maximum ratio combining (Option 'MRC') und verschiedene Anzahlen von Empfangsantennen (Diversitätsordnungen) durch. Nehmen Sie einen klassischen Rayleighkanal an. Visualisieren Sie Ihre Resultate (ähnlich wie in Abbildung 5) und vergleichen Sie sie mit der theoretischen Bitfehlerratenkurve für MRC aus Gleichung 7.

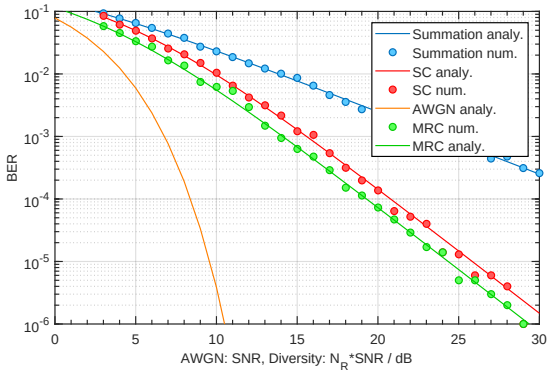

Abbildung 4: Bitfehlerrate als Funktion des SNR pro Bit (skaliert mit der Diversitätsordnung) für verschiedene Kombinationsvektoren bei Übertragung über einen Rayleighkanal. Die Linien zeigen analytische Kurven, während die Punkte numerisch ermittelte Simulationswerte darstellen. SDC: selection diverstity combining, MRC: maximum ratio combining, AWGN: additive white Gaussian noise.

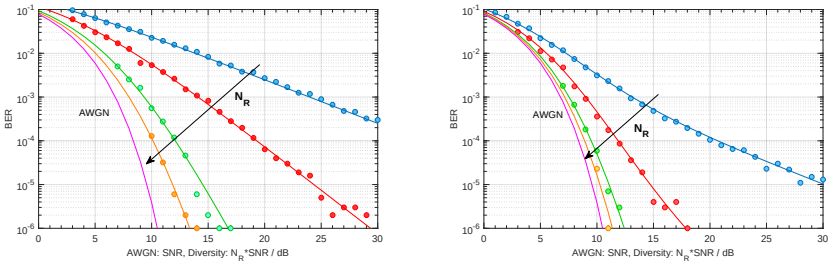

Abbildung 5: Bitfehlerrate als Funktion des SNR pro Bit (skaliert mit der Diversitätsordnung) für verschiedene Diverstitätsordnungen ($N_R = 1, 2, 5, 10$) bei Übertragung über einen Rayleighkanal (links) und über einen Ricekanal mit dem Parameter $K = 5$ (rechts). Die Linien zeigen analytische Kurven, während die Punkte numerisch ermittelte Simulationswerte darstellen. Als Kombinationsmethode wurde maximum ratio combining (MRC) gewählt. AWGN: additive white Gaussian noise.

3.4 *Vergleich unterschiedlicher Kanäle*

Führen Sie die gleichen Simulationen wie under Unterabschnitt 3.3 durch, aber nehmen Sie einen Ricekanal mit unterschiedlichen Werten für den Parameter K an. Visualisieren Sie Ihre Resultate und (siehe Abbildung 5) vergleichen Sie sie mit der theoretischen Bitfehlerratenkurve für MRC mit der MATLAB eigenen Funktion `berfading`.

- MATLAB Funktion: `berfading`

LITERATUR

[1] Mathworks, *Mathworks website*, available online at http://www.mathworks.de/.

[2] Octave, *Octave website*, available online at http://www.gnu.org/software/octave/.

[3] OctaveForge, *Octave Forge Website*, available online at http://octave.sourceforge.net/.

[4] Mathwoks, *MATLAB Central*, available online at http://www.mathworks.de/matlabcentral/, Apr. 2014.

[5] K. D. Kammeyer, *Nachrichtenübertragung*, 5rd. Vieweg + Teubner, 2011.