DRAHTLOSE KOMMUNIKATION

MATLAB Rechenübung

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

20. Oktober 2020

Wintersemester 2020-2021

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Studiengang Computer Engineering

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 Hilfe eines numerischen Computeralgebraprogramms 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 kommerzielle Software MATLAB [1] oder die Open-Source Alternative GNU Octave [2], [3] eingesetzt werden. MATLAB bietet eine komplette Entwicklungsumgebung (Editor, Kommandozeile, Debugger,...) [1].

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

Da in diesem Semester die Rechenübung nur Online per Zoom-Videokonferenz stattfindet, ist es notwendig, dass Sie sich eines der genannten Programme (MATLAB oder GNU Octave) auf ihrem privaten Rechner installieren.

Studierende der HTW können eine kostenlose MATLAB Campus Lizenz herunterladen. Eine Anleitung zum Bezug dieser Lizenz finden Sie im Moodle Kurs zu dieser Veranstaltung.

Installieren Sie die bevorzugte Software auf Ihrem Computer und machen Sie sich in groben Zügen mit ihr vertraut [1]–[3], z. B.:

- 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 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 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 Funktionen: `transpose (') , colon (:), length, size, min, max, abs`? Sind diese auch für Vektoren oder Matrizen benutzbar?
- Was bewirkt der vorangestellte Punkt z.B. bei den Operatoren `.*` oder `./` ?
- Wie definiert man einen linear ansteigenden / absteigenden Vektor?
- Welche Möglichkeiten gibt es in MATLAB Teile eines Vektors zu indizieren?
- Wie definiert man Kommentare in einem Skript?

1 https://de.mathworks.com/academia.html#learn-basics
https://de.mathworks.com/products/matlab/getting-started.html
https://de.mathworks.com/help/matlab/examples.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 eleganter (und effizienter) lösen?
- Wie können Sie eigene Funktionen definieren? Schreiben Sie eine einfache Funktion `y = quadMittel(x)`, 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 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. Es werden in jedem Abschnitt nützliche MATLAB Funktionen angegeben, die zur Implementierung der Aufgaben hilfreich sein können.

Die Funktionsweise und Parameter der einzelnen Funktionen können Sie der MATLAB -Hilfe entnehmen. Diese ist sehr detailliert und umfangreich und enthält in den allermeisten Fälle auch nützliche Beispiele. Sie ist zwar nur auf Englisch verfügbar, aber nach einer Eingewöhnung in den Aufbau und die Syntax ist sie gut verständlich und eine unverzichtbare Informationsquelle. Ich rate Ihnen, die MATLAB Hilfe als erste Anlaufstelle für die Erklärung der Funktionsweise bestimmter Funktionen zu benutzen, bevor Sie weitere Quellen (wie das Internet) zu Rate ziehen. Ein weiterer guter Startpunkt für spezifische MATLAB Problemstellungen bietet auch MATLAB Central [4], insbes. *File Exchange* [2].

Zusätzlich zu den angegebenen Grundfunktionen 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. Diese und ähnliche Fragen können auch Bestandteil der mündlichen Rücksprache (s.u.) sein!

Die Ergebnisse der PCÜ werden gemeinsam mit dem Vorlesungsinhalt in einer (voraussichtlich mündlichen) Prüfung nach dem Semester abgeprüft. Die Prüfung findet gruppenweise entsprechend der PCÜ-Gruppeneinteilung statt. Jede Gruppe muss einige Tage *vor* der Prüfung ihren gemeinsam erstellten Simulationscode auf den Moodle-Server hochladen. In der Prüfung sollen Sie dann Ihren Simulationscode vorstellen und die implementierten Funktionen sowie deren Funktionsweise detailliert erklären. In die Benotung geht auch die Qualität des abgegebenen Codes und insbesondere dessen Kommentierung ein. Ausführliche Kommentare im Code können Ihnen also während der Rücksprache durchaus von Nutzen sein.

Die **Gesamtmodulnote** setzt sich zu **35% aus der Bewertung des PCÜ-Prüfungsteils** und zu **65% aus Bewertung des Vorlesungsteils** der Prüfung zusammen. Die Benotung erfolgt individuell für jedes Gruppenmitglied.

Hinweis: Falls Sie Code-Teile von anderen Gruppen übernehmen, ist es zwingend erforderlich, die entsprechenden Abschnitte in Ihrem Quellcode eindeutig zu kennzeichnen, um einen Plagiatsverdacht zu vermeiden! Übernommene (und entsprechend gekennzeichnete) Code-Abschnitte fließen nicht mit in Ihre PCÜ-Bewertung ein.

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

2 GRUNDSÄTZLICHES ZUR SIMULATION

Analog zu der Vorlesung wird ein Funksystem mit jeweils einer Sende- und Empfangsantenne betrachtet (`SISO: single input - single output`). Dazu wird die Simulation eines solchen Systems Schritt für Schritt (vom Sender zum Empfänger) aufgebaut. In Abbildung 1) ist eine schematische Übersicht über ein allgemeines digitales Übertragungssystem dargestellt, wobei in dieser PCÜ lediglich die nicht-gestrichelten Blöcke implementiert werden sollen. Die beiden grau hinterlegten Blöcke zur Kanal-Codierung und -Decodierung sollen optional nur implementiert werden, wenn Sie gegen Ende der PCÜ noch Zeit haben und die anderen Funktionalitäten bereits implementiert haben. Genauere Informationen zu diesen beiden Funktionen folgen im Laufe des Semesters in diesem Skript.

Als eigentliche Simulation fungiert ein MATLAB-Skript, in dem die grundsätzlichen Parameter der Simulation eingestellt werden sowie der Ablauf der einzelnen Simulationsschritte festgelegt ist. Aus diesem Skript heraus werden einzelne Funktionen (die Sie im weiteren Verlauf implementieren werden) nacheinander aufgerufen. Diese Funktionen führen einzelne Funktionalitäten des Senders, des Kanals und des Empfängers aus. Die vorgegebenen Funktionsnamen finden sich in Abbildung 1 bei den zugehörigen Funktionsblöcken.

Für verschiedene Kanalzustände (im Weiteren vor allem verschiedene SNR-Werte) werden eine bestimmte, feste Anzahl von Bits am Sender (*transmitter*) generiert, über einen Kanal (*channel*) übertragen und im Empfänger (*receiver*) ausgewertet. Ein solches Simulationsskript (beschrieben in Pseudo-Code) ist in Listing 1 dargestellt.

Listing 1: Beispiel eines Simulationsskripts

```
% definition of simulation parameters
SNR=1...10
modulationFormat=QPSK
nBits = 100e3
...

% simulation loop(s)
for SNR
        transmitterfunction1(...)
        (potential visualization of results)
        transmitterfunction2(...)
        ...

        channelfunction1(...)
        channelfunction2(...)
        (potential visualization of results)
        ...

        receiverfunction1(...)
        receiverfunction2(...)
        (potential visualization of results)
        ...

        determination of bit errors / BER
end

visualization of final simulation results (e.g. BER vs. SNR)
```

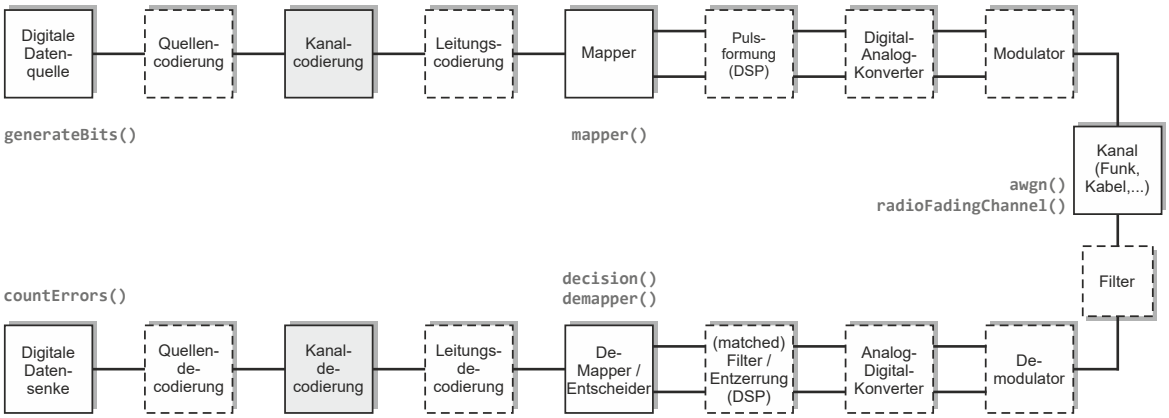

Abbildung 1: Blockdiagramm eines digitalen Übertragungssystems. Die gestrichelt dargestellten Blöcke werden nicht mit in die Simulation einbezogen.

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 entsprechend Listing 2 zu Beginn einer Funktion durchführen:

Listing 2: 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. Eine weitere, einfache und sehr elegante Methode die Eigenschaften einer Matrix zu testen bietet die MATLAB eigene Funktion `validateattributes()`. Bitte machen Sie sich mit der Funktionsweise vertraut und benutzen Sie diese Funktion so häufig wie nötig um potentielle Fehlerquellen zu finden, bzw. zu vermeiden.

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

Im den nun folgenden Abschnitten werden die einzelnen Funktionen besprochen, die implementiert und innerhalb der Schleife(n) aufgerufen werden sollen.

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
nBits = 10e3; % simulate nBits bits per simulation loop

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

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

3 ERZEUGUNG DER SENDESIGNALE AM SENDER

Zuerst müssen in dem Sender die Signale, die übertragen werden sollen erzeugt werden. Dazu werden zuallererst zufällige Datenbits benötigt.

3.1 *Erzeugung der Bitsequenzen*

Daher implementieren Sie bitte eine Funktion **`y = generateBits(nBits)`**, die als Eingabeparameter die Anzahl der zu erzeugenden Bits (`nBits`) erhält und als Ausgabeparameter (`y`) die erzeugten Bits (als {`1,0`} oder {`true,false`}) ausgibt. Dabei soll `y` 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 zunächst nur eine kleine Anzahl von Bits (z.B. 40) und visualisieren sie diese.

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

3.2 *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, constellation)`**, die als Eingabeparameter sowohl den gerade erzeugten Bitvektor `bits`, als auch die einzelnen Konstellationspunkte eines Modulationsformats als Vektor `constellation` 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 z.B. 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 (das Mapping) der einzelnen komplexen Symbole (Konstellationspunkte) zu den jeweiligen Bitwerten wird über die *Position* der Konstellationspunkte innerhalb des Vektors `constellation` bestimmt (im Sinne einer Lookup-Table). So wird im BPSK-Beispiel der erste Konstellationspunkt innerhalb des Vektors (-1) dem Bitwert 0 zugeordnet, der zweite Konstellationspunkt (+1) dem Bitwert 1.

Als Beispiel soll nun eine QPSK betrachtet werden. Für eine QPSK werden jeweils zwei aufeinanderfolgende Bits zusammengefasst (2-Tupel) und in eine Dezimalzahl konvertiert. Da dies Dezimalzahlen zwischen 0 und 3 ergibt, die Indizierung eines Vektors in MATLAB jedoch mit der Zahl 1 beginnt, muss zu den berechneten Dezimalzahlen noch der Wert 1 addiert werden, bevor das Ergebnis als Index für den Konstellationspunkte-Vektor `constellation` benutzt werden kann, um den jeweiligen Konstellationspunkt zu bestimmen. Für eine QPSK mit `Gray`-Mapping von Bits zu Konstellationspunkten ist dieser Zusammenhang in Tabelle 1 dargestellt. Dies entspricht einem Konstellationspunkte-Vektor `constellation = [-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

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]

Simulation soll dann jedoch zunächst die QPSK benutzt werden.

4 EMPFÄNGER

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

Natürlich ist das resultierende Bitfehlerverhältnis am Empfänger ohne den Einfluss eines Kanals (der ja noch nicht implementiert ist) immer ideal Null, da ja keine Kanalstörungen und damit keine Bitfehler entstehen.

4.1 *Entscheidung der empfangenen Symbole*

Die Funktion **`y = decision(x, constellation)`** erhält einen Zeilenvektor `x`, der die empfangenen Symbole enthält (ohne Kanal also direkt das Sendesignal) und den Vektor `constellation` mit den Konstellationspunkten des zu detektierenden Modulationsformats. Die Funktion soll nun für jeden Abtastwert des Empfangssignals `x` den (euklidischen) Abstand zwischen diesem Eingangssignal und jedem möglichen Konstellationspunkt berechnen[3]. Anschließend wird angenommen, dass der empfangene Konstellationspunkt mit dem kleinsten Abstand das ursprünglich gesendete Modulationssymbol war. Es wird im Empfänger also eine harte Entscheidung (hard decision) bezüglich eines Modulationssymbols getroffen.

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

Bitte beachten Sie, dass das Signal vor der Entscheidung eventuell zuerst auf die mittlere Leistung der gesendeten Konstellationspunkte normiert werden muss, da der potentielle Kanal die mittlere Leistung des Signals unter Umständen ändert. Nur so kann gewährleistet werden, dass eine sinnvolle Entscheidung (Abstandsberechnung) getroffen wird.

- Wie berechnet man die mittlere Leistung eines Signals (oder Vektors)?
- MATLAB Funktionen: `randn, mean, abs, sqrt, size, bsxfun, var`

4.2 *Zuordnung der entschiedenen Symbole in Bits*

Nun soll mit Hilfe der Funktion **`y = demapper(x, constellation)`** eine inverse Funktion zur `mapper` Funktion des Senders implementiert werden. Diese Funktion erhält einerseits einen Zeilenvektor mit den entschiedenen Symbolen `x` und andererseits die originalen Konstellationspunkte des Modulationsformats `constellation`. Der Rückgabeparameter `y` enthält die laut Zuordnungsvorschrift (siehe Kapitel 3.2) zugehörigen Bitwerte. Das heißt, wenn das Modulationsformat M Konstellationspunkte besitzt, enthält der Vektor `y` $\log_2(M)$ mal so viele Elemente (Bits) 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 `constellation` auftaucht. Mit anderen Worten, es wird für jedes Element im Vektor `x` der Index des korrespondierenden Konstellationspunkts im Vektor `constellation` bestimmt. Danach kann von jedem einzelnen Index der Wert `1` subtrahiert werden. Dies muss (analog zur `mapper` Funktion) gemacht werden, da die Indizierung in MATLAB bei `1`

3 Eine eigentlich notwendige *zeitliche Synchronisation* zwischen Sender und Empfänger wird hier als bereits gegeben vorausgesetzt

beginnt, die ermittelten Dezimalzahlen allerdings zwischen 0 und $M-1$ liegen müssen. Die resultierenden Werte entsprechen nun den empfangenen Symbolen (in Dezimalzahlen $[0 \dots M-1]$) und können wieder jeweils in einzelne binäre Zahlen konvertiert werden, um die empfangene Bitfolge zu bestimmen (vgl. Kapitel 3.2).

Simulieren Sie zuerst nur wenige Symbole (z.B. 16) und vergleichen Sie sowohl die Symbole vor und nach der Entscheidung, als auch die gesendeten Bits mit den entschiedenen. Diese müssen, ohne Kanaleinflüsse, identisch sein. Sind die Entscheidungen und die Zuordnungen plausibel?

- MATLAB Funktionen: `bitget, logical, log2, nan, isnan, any, de2bi, dec2bin`

4.3 *Fehlerzählung*

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

Als Ausgabeparameter soll die Funktion die Anzahl der Bitfehler (die Anzahl der Elemente die in den beiden Eingangsvektoren unterschiedlich sind) `nErr` und das Bitfehlerverhältnis `ber` ausgeben. Das Bitfehlerverhältnis ist dabei gegeben als das Verhältnis zwischen der Anzahl der Bitfehler und der Gesamtanzahl der empfangenen Bits.

Damit ist die erste, einfache Simulation, bis jetzt noch ohne jegliche Kanaleinflüsse, abgeschlossen.

Bis jetzt, da noch keine Störungen implementiert wurden, sollte die Anzahl der Fehler und damit das Bitfehlerverhältnis immer Null sein. Testen Sie Ihre Empfängerimplementierung, indem Sie vor dem Empfänger absichtlich einzelne Fehler in die empfangenen Symbole einbauen. Dies müsste sich entsprechend in der Anzahl der Fehler und dem ermittelten Bitfehlerverhältnis niederschlagen.

- MATLAB Funktionen: `~=, find, size, sum, semilogy, xlim, ylim, legend, xlabel, ylabel, axis`

5 SIMULATION EINES GEDÄCHTNISLOSEN FUNKKANALS

In einem ersten Schritt soll nun zuerst ein Kanal mit additivem, weißen und gaussverteiltem Rauschen simuliert werden. Erst danach soll ein numerische Simulation eines Rayleigh- bzw. ein Rice-Kanalmodels durchgeführt werden.

5.1 *Additives weißes gaußverteiltes Rauschen (AWGN)*

Nachdem das Sendesignal nun erzeugt wurden, soll dem Signal Rauschen mit einem bestimmten Signal-Rauschabstand (signal to noise ratio, SNR) zugefügt werden. Dazu benutzen Sie bitte die existierende MATLAB Funktion `awgn()`. Die Funktion kann Rauschen hinzufügen und damit ein Signal mit einem bestimmten SNR erzeugen. Bitte konfigurieren Sie die Funktion so, dass zuerst die mittlere Leistung des Eingangssignals `x` bestimmt und dann Rauschen mit der Leistung P_N hinzugefügt wird, so dass das geforderte SNR am Ausgang entsteht.

Bitte beachten Sie, dass in der Vorlesung alle BER-kurven und -formeln als Funktion des *SNR pro Bit* (SNR_b) angegeben wurden. Des Weiteren soll auch die komplette Simulation als Parameter das SNR pro Bit beschreiben. Daher müssen Sie das *SNR pro Symbol* (SNR_s), 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

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

Beachten Sie weiterhin, dass sowohl das SNR pro Bit (SNR_b), als auch das SNR pro Symbol (SNR_s) in Formel (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 nach dB konvertieren.

Bitte simulieren Sie den Kanal für verschiedene SNR Werte und bestimmen Sie die resultierenden Bitfehlerverhältnis-Werte (engl. Bit Error Ratio, *BER*). Visualisieren Sie diese Ergebnisse (z.B. in einem Plot, der das Bitfehlerverhältnis als Funktion des SNR (in dB) darstellt, ähnlich zu dem Graphen in Abbildung 2). Stellen Sie die y-Achse dabei logarithmisch dar, damit auch sehr kleine Bitfehlerverhältnisse dargestellt werden können. Vergleichen Sie diese Werte mit theoretisch erreichbaren Bitfehlerverhältnissen für eine Übertragung einer QPSK über einen AWGN Kanal. Diese sind gegeben als [5, Kap. 11.4.4]

$$P_{b|QPSK}^{AWGN} = \frac{1}{2}\operatorname{erfc}\left(\sqrt{SNR_b}\right), \tag{2}$$

wobei die komplementäre Fehlerfunktion `erfc()` mit

$$\operatorname{erfc}(x) = \frac{2}{\sqrt{\pi}}\int_x^\infty e^{-\tau^2}\,d\tau, \tag{3}$$

gegeben ist. Diese Funktion steht auch in MATLAB zur Verfügung und muss daher nicht selbst implementiert werden.

Ändern Sie nun das Modulationsformat (benutzen Sie z.B. eine 8-QAM, 16-QAM oder 64-QAM) indem Sie den Vektor der Konstellationspunkte `constellation` ändern und vergleichen Sie das resultierende Bitfehlerverhältnis. Diskutieren Sie die Veränderung und

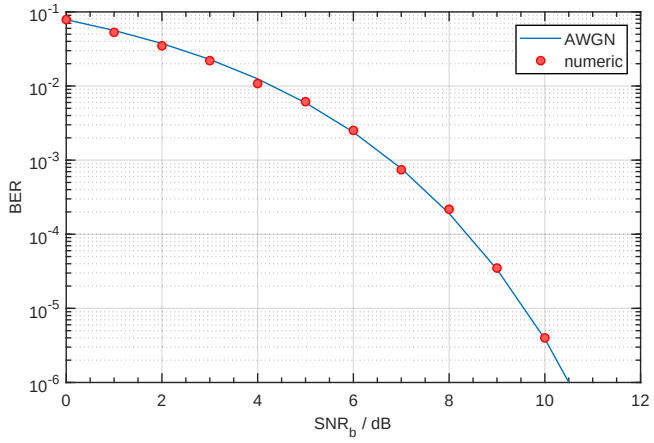

Abbildung 2: Bitfehlerverhältnis als Funktion des SNR pro Bit für eine QPSK über einen AWGN Kanal. Die Linie zeigt die analytische Kurve, während die Punkte numerisch ermittelte Simulationswerte darstellen.

stellen Sie die numerisch ermittelten BER-Werte auch in einem Diagramm (ähnlich wie in Abbildung 2) dar.

- Wie berechnet man die mittlere Leistung eines Signals?
- Diskutieren Sie evtl. auftretende Abweichungen zwischen den numerisch ermittelten Fehlerverhältnissen zu den theoretisch erwarteten Werten
- Wie ändert man die Varianz (oder die Standardabweichung) einer gaußverteilten Zufallsvariable? Wie kann der Mittelwert verändert werden?
- MATLAB Funktionen: `randn, mean, abs, sqrt, size, erfc, semilogy`

5.2 *Änderung der Abbruchbedingung für die Simulation*

Bit jetzt wurde jede Simulation mit der gleichen Anzahl von Bits durchgeführt. Um ein aussagekräftiges Bitfehlerverhältnis zu bekommen, ist es allerdings ausreichend etwa 100 Fehler abzuwarten. Das heißt, in dem Fall von einer festen Anzahl von Bits erhalten Sie für große BER-Werte (geringes SNR) viel zu viele Fehler, während Sie für kleine Fehlerverhältnisse (großes SNR) zu wenige Fehler für eine aussagekräftige BER-Schätzung erhalten. Dies sollen Sie nun in Folgenden ändern.

Die Simulationsschleife soll so lange durchlaufen werden, bis eine einstellbare Anzahl von Fehlern gefunden wurde. Dies soll mit dem Parameter `nMinErr` eingestellt werden können.

Andererseits würde dies bei sehr kleinen Fehlerverhältnissen (BER$< 10^{-5}$) zu einer unrealistisch großen Anzahl von simulierten Bits führen, so dass eine zusätzliche Abbruchbedingung angegeben werden muss. Hier soll die absolut maximale Anzahl von Simulationsbits mit dem Parameter `nMaxBits` angegeben werden können. Dass heißt, für große Bitfehlerverhältnisse bricht die Simulation nach `nMinErr` Fehlern und bei kleinen BER-Werten nach `nMaxBits` simulierten Bits ab.

Ändern Sie dazu Ihr Simulationsskript wie in Listing 4 gezeigt ab.

Listing 4: Änderung des Abbruchkriteriums für die Simulation

```
% definition of simulation parameters
...
nMinErr=100
nBitsPerLoop = 10e3
nMaxBits= 100*nBitsPerLoop
...

% simulation loops
for SNR
    nBits = nBitsPerLoop
    while ((number of errors smaller nMinErr) or
        (max. number of simulated bits reached))

        transmitterfunktion1(...)

        ...

        determination of bit errors / BER

    end

    visualization of end results (e.g. BER vs. SNR)
    loopCnt = loopCnt + 1
end
```

Prinzipiell sollte der Rückgabewert `nErr` der Funktion `countErrors` dazu benutzt werden, um die Anzahl der Fehler aus mehreren Schleifendurchläufen in einer zweiten Variable `nTotalErr` aufzusummieren, welche dann auch als Abbruchkriterium für die Schleife benutzt werden.

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

5.3 *Implementierung eines Rayleighkanals*

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 zeitlich veränderlichen Schwundkanal und einem anschließenden additiven, weißen, gaußverteilten Rauschen (AWGN).

Eine schematisches Diagramm eines solchen Kanals ist in Abbildung 3 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 wieder die Funktion `awgn()` das Rauschen zum Signal addiert.

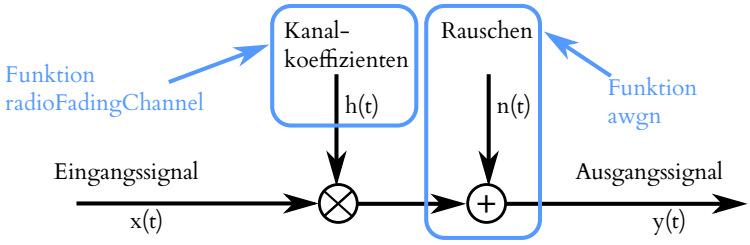

Abbildung 3: <Darstellung der Simulation eines Rayleighkanals

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 (vgl. Vorlesung) 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 gaußverteilt 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 Funktionen: `randn, mean, abs, angle, real, imag, sqrt, hist, bsxfun`

Bitte simulieren Sie anschließend den Rayleigh-Kanal für verschiedene SNR Werte und bestimmen Sie die resultierenden Bitfehlerverhältnisse. Visualisieren Sie diese Ergebnisse (z.B. in einem Plot, der das Bitfehlerverhältnis als Funktion des SNR (in dB) darstellt, ähnlich zu dem Graphen in Abbildung 4). Vergleichen Sie dazu das aus der Vorlesung bekannte theoretisch erreichbare Bitfehlerverhältnis für eine QPSK Übertragung über einen Rayleighkanal [5, Kap. 15.2.1], welche als

$$P_{b|QPSK}^{Rayleigh} = \frac{1}{2} \cdot \left(1 - \sqrt{\frac{SNR_b}{1+SNR_b}}\right) \tag{4}$$

gegeben ist.

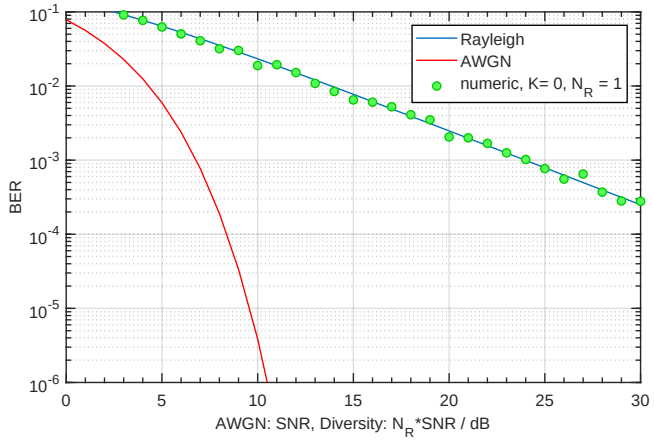

Abbildung 4: Bitfehlerverhältnis als Funktion des SNR pro Bit für einen Rayleigh Kanal. Die Linien zeigen analytische Kurven, während die Punkte numerisch ermittelte Simulationswerte darstellen.

ACHTUNG: In dieser Simulation wird angenommen, dass der Empfänger den Kanal ideal geschätzt hat und somit alle Kanalkoeffizienten kennt. Aus diesem Grund können die Amplituden- und Phasenschwankungen des Schwundkanals am Empfänger *ideal* kompensiert werden. Teilen Sie also das empfangene (und durch den Schwundkanal gestörte) Signal durch die bekannten, mit der Funktion `radioFadingChannel()` erzeugten Kanalkoeffizienten. Damit führen Sie eine ideale Kompensation der Rayleigh-Kanaleinflüsse durch. Ohne den Einfluss des Rauschens wäre so eine perfekte Rekonstruktion des Sendesignals möglich.

5.4 *Erweiterung des Kanals auf ein Rice-Kanalmodel*

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 Unterabschnitt 5.3 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

$$K = \frac{P_{LOS}}{P_{NLOS}} = \frac{a_{LOS}^2}{2\sigma^2}, \tag{5}$$

wobei σ^2 die Varianz (und damit die Leistung) des Real- bzw. des Imaginärteils der Rayleigh-Kanalkoeffizienten bezeichnet.

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 (zeitliche konstanten) 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.

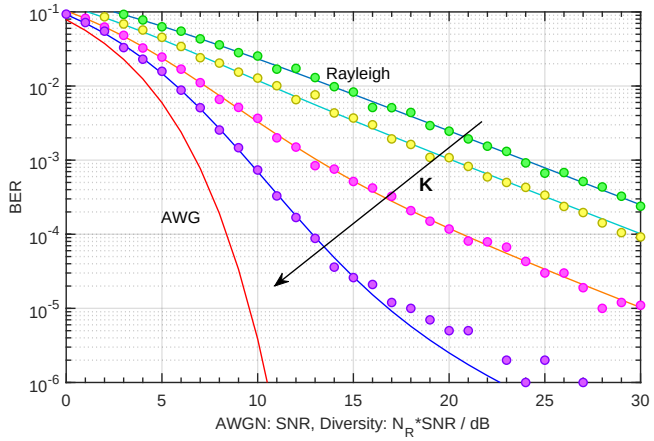

Abbildung 5: Bitfehlerverhältnis als Funktion des SNR pro Bit für einen Rice Kanal mit unterschiedlichen Werten für den Parameter K ($K = 0, 1, 2, 5, 10$). Die Linien zeigen analytische Kurven, während die Punkte numerisch ermittelte Simulationswerte darstellen.

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 Bitfehlerverhältnissen für eine QPSK Übertragung über einen Ricekanal [5, Kap. 15.2.1], gegeben in Gleichung 6. Visualisieren Sie Ihre Ergebnisse ähnlich wie in Abbildung 5 dargestellt.

$$p_{b|QPSK}^{Rice} = \frac{1}{\pi}\int_0^{\pi/2} \frac{(1+K)\sin^2(\theta)}{(1+K)\sin^2(\theta)+SNR_b} \exp\left(-\frac{K \cdot SNR_b}{(1+K)\sin^2(\theta)+SNR_b}\right) d\theta \qquad (6)$$

Erhöhen Sie den Wert für K und vergleichen Sie die Ergebnisse mit den theoretisch erreichbaren BER-Werte 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 Funktionen: `randn, rand, mean, trapz, quad, integral, berawgn`

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/.

[5] K. D. Kammeyer, *Nachrichtenübertragung*, 3rd. Vieweg + Teubner, 2004.