universität innsbruck
Fakultät für Mathematik, Informatik und Physik

Martin Nocker

R-Paket für Kanalkodierung mit Faltungskodes

Bachelorarbeit

24. Mai 2016

Universität Innsbruck
Institut für Informatik
Technikerstr. 21a · 6020 Innsbruck · Österreich
http://informatik.uibk.ac.at/

R-Paket für Kanalkodierung mit Faltungskodes

Bachelorarbeit

vorgelegt von

Martin Nocker

geb. am 1. Mai 1993
in Innsbruck

angefertigt am

Institut für Informatik
Leopold-Franzens-Universität Innsbruck

Betreuer: **Univ.-Prof. Dr. Rainer Böhme**
Dr. Pascal Schöttle

Abgabe der Arbeit: **24. Mai 2016**

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties.
I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Martin Nocker)
Innsbruck, 24. Mai 2016

Zusammenfassung

Inhaltsverzeichnis

Kapitel 1

Einleitung

Kanalkodierung stellt einen wichtigen Teil der Nachrichtentechnik dar. Kanalkodierung stellt Methoden zur Verfügung, um Fehler, die während der Übertragung über einen verrauschten Kanal auftreten, zu korrigieren. Die Leistungsfähikeit und Zuverlässigkeit vieler digitaler Systeme basiert auf der Verwendung von Kanalkodierung. Eine Art der Kanalkodierung stellen Faltungskodes dar, auf welche sich diese Arbeit konzentriert. Verwendung finden Faltungskodes in der Mobil- und Satellitenkommunikation aber vor allem bilden sie die Basis für Turbokodes, welche die Faltungskodes mittlerweile aufgrund ihrer noch höheren Leistungsfähigkeit abgelöst haben.

Ziel dieser Arbeit ist die Implementierung von Faltungskodes mithilfe der Programmiersprache R. Das entwickelte R-Paket dient zukünftigen Studenten zu Lernzwecken und soll sie beim Verstehen von Faltungskodes unterstützen.

Kapitel 2

Grundlagen und ähnliche Arbeiten

Kapitel 3

Verwendete Technologien

Dieses Kapitel über die verwendeten Technologien bei der Implementierung setzt sich folgendermaßen zusammen: Kapitel 3.1 behandelt die Programmiersprache R und die verwendete Entwicklungsumgebung RStudio. In Kapitel 3.2 werden die Möglichkeiten der Einbindung von C/C++ Code, v.a. mithilfe des Pakets *Rcpp*, beschrieben. Schließlich wird in Kapitel 3.3 auf die Erstellung dynamischer Dokumente und Visualisierungen mittels *RMarkdown*, LaTeX und Ti*k*Z.

3.1 R, RStudio, Pakete

R ist eine, im Jahre 1992 entwickelte, schwach und dynamisch typisierte Programmiersprache, die vor allem in der Statistik für die Analyse von großen Datenmengen Anwendung findet. Ein weiteres Motiv für die Verwendung von R sind die vielseitigen Möglichkeiten, bei gleichzeitig einfacher Handhabung, graphischer Darstellungen großer Datenmengen. R-Code wird nicht kompiliert, sondern nur interpretiert und ist daher platformübergreifend verwendbar. Datentypen müssen zur Übersetzungszeit nicht bekannt sein. Die Typüberprüfung findet zur Laufzeit statt. Diese Eigenschaft erschwert das Finden von Fehlern im Code erheblich.

Der Funktionsumfang der Sprache kann durch sogenannte Pakete erweitert werden. Bei der Installation von R sind die wichtigsten Pakete inkludiert. Über Repositories wie CRAN[1] oder GitHub sind über 8000 zusätzliche Pakete

[1] The Comprehensive R Archive Network: `https://cran.r-project.org/`

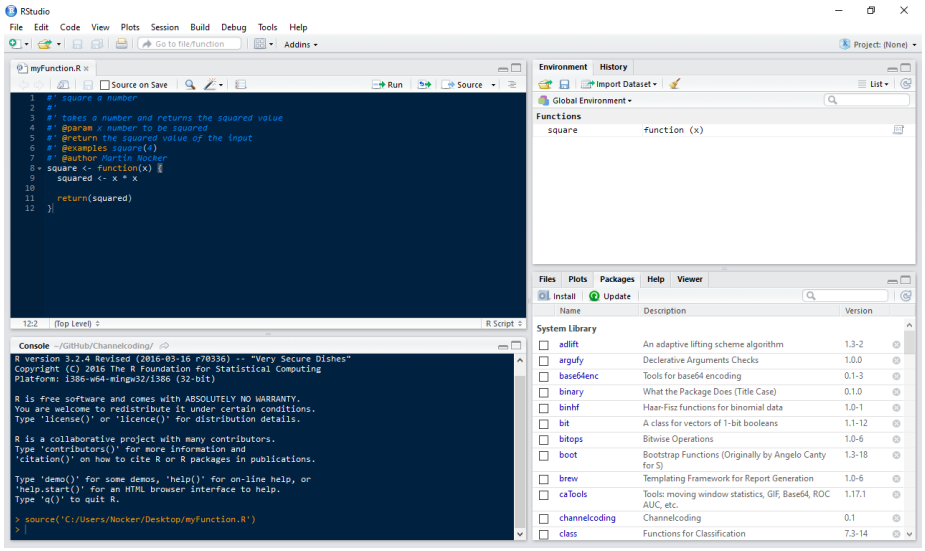

Abbildung 3.1 – RStudio Standardansicht

(Stand: Mai 2016) für die verschiedensten Anwendungsbereiche verfügbar. Diese Vielfalt an Paketen ist ein Grund für den Erfolg von R [R Packages]. Pakete werden laufend aktualisiert und verbessert. Selbst entwickelte Pakete können via CRAN für andere Entwickler veröffentlicht werden, müssen jedoch strenge Auflagen zur Aufrechterhaltung der Konsistenz bei Inhalt, Form und Dokumentation der Pakete einhalten [2].

Ein wichtiges Paket welches im Rahmen dieser Arbeit verwendet wird ist *roxygen*. Mithilfe dieses Pakets wird, ähnlich wie JavaDoc für Java, durch spezielle Kommentare und Annotations überhalb der Paketfunktionen automatisch die Paketdokumentation erstellt. Die roxygen-Kommentare der Paketfunktionen, die für wartbaren Code ohnehin unabdingbar sind, sind für den Entwickler erheblich angenehmer als die Paketdokumentation von Hand zu schreiben. Roxygen-Kommentare werden durch das Kommentarsymbol #' am Zeilenbeginn eingeleitet. Zu den wichtigsten Annotations gehören jene für die Beschreibung der Parameter (@param) und Rückgabewerte (@return) sowie Beispiele zur Ausführung der Funktion (@examples). Weiters wird über die @export Annotation geregelt welche Funktionen nach Auslieferung des Pakets von außen aufrufbar sind.

RStudio ist eine freie und open source Entwicklungsumgebung für R. RStudio

verfügt über alle notwendigen Funktionalitäten für die Softwareentwicklung mit R und bietet darüber hinaus Funktionen für eine vereinfachte Entwicklung von R-Paketen an. Abbildung 3.1 zeigt die Version 0.99.893.

3.2 C++, Rcpp

Manchmal ist R-Code einfach nicht schnell genug. Typische Flaschenhälse sind Schleifen und rekursive Funktionen. Die Performance kann in solchen Fällen durch Auslagern von Funktionen und Algorithmen in C oder C++ erheblich verbessert werden, da der Code kompiliert und somit optimiert werden kann anstatt nur interpretiert zu werden.
R bietet drei Möglichkeiten C/C++ Code aufzurufen:

- *.C* native Schnittstelle
- *.Call* Schnittstelle
- *Rcpp* Paket

Die *.C* Schnittstelle ist die einfachste Variante C Code auszuführen, jedoch auch jene mit den größten Einschränkungen. Im C Code sind keinerlei R Datentypen oder Funktionen bekannt. Alle Argumente sowie der Rückgabewert müssen als Zeiger in der Parameterliste übergeben werden deren Speicher vor dem Aufruf reserviert werden muss.

Bei der *.Call* Schnittstelle handelt es sich um eine Erweiterung der *.C* Schnittstelle. Die Implementierung ist komplexer, dafür sind R Datentypen verfügbar und es gibt die Möglichkeit eines Rückgabewerts mittels dem return Statements. [4]

Bei den ersten beiden Möglichkeiten muss der C Code vor dem Aufruf per Hand kompiliert und in der R Session geladen werden. Das *Rcpp* Paket ermöglicht die Verwendung von C++ Code ohne diesen Aufwand. Im C++ Code stehen R Datentypen wie Vektoren, Matrizen oder Listen ohne kompliziertem Syntax zur Verfügung. Die Funktionsaufrufe sehen, im Gegensatz zu den ersten beiden Ausführungen, aus wie normale R Funktionsaufrufe und macht dadurch den Code erheblich lesbarer. Weiters stehen Vektorfunktionen zur Verfügung, d.h. eine auf einen Vektor angewandte Funktion wird auf jedes

Vektorelement ausgeführt und erspart somit beispielsweise eine Schleife. Bei der Entwicklung eines eigenen Pakets ist es bei der Verwendung des *Rcpp* Pakets zusammen mit RStudio sehr einfach C++ Code zu integrieren. Durch all diese Vorteile ist das *Rcpp* Paket die zu wählende Schnittstelle. Während der Paketerzeugung kompiliert RStudio automatisch alle C++ Dateien und erstellt automatisch Wrapper-Funktionen, die den Zugriff auf die Funktionen erleichtern.

Die C++ Datei muss mit folgenden Zeilen starten:

```
1 #include <Rcpp.h>
2 using namespace Rcpp;
```

Sowie jede Funktion, die in R verfügbar sein soll muss folgenden Präfix erhalten:

```
1 // [[Rcpp::export]]
```

Die genaue Verwendung des *Rcpp* Pakets ist in [3] beschrieben/ nachzulesen.

3.3 RMarkdown, LaTeX, Ti*k*Z

Zur Erstellung von dynamischen Dokumenten wird das Paket *RMarkdown* verwendet werden. Durch die Kombination der Syntax von Markdown, R, LaTeX und HTML ergibt sich ein flexibles und einfaches Werkzeug. Die unterstützen Ausgabeformate beinhalten u.a. HTML, PDF, MS Word und Beamer (Präsentationen).

Abbildung 3.2 zeigt den Workflow für die Generierung eines dynamischen Dokuments mittels *RMarkdown*. Der Markdown, R und LaTeX Code wird zusammen mit dem gewünschten Ausgabeformat, wobei mehrere Angaben möglich sind, in die *RMarkdown*-Datei (Dateiendung .rmd) geschrieben. Die RMD-Datei wird dem *knitr* Paket übergeben, welches den R Code ausführt und eine neue Markdown-Datei (Dateiendung .md) erstellt, die den R Code und dessen Ergebnisse beinhaltet. Die erzeugte Markdown-Datei wird von *pandoc* weiterverarbeitet, was für die Erstellung des endgültigen Dokuments im gewünschten Format zuständig ist. Bei der Verwendung von RStudio ist *pandoc* automatisch verfügbar. Den eben beschriebenen Ablauf kapselt das *RMarkdown*-Paket in einen einzigen *render*-Funktionsaufruf.

Abbildung 3.2 – RMarkdown Überblick, Quelle: [1]

Für die Erzeugung dynamischer Grafiken wird die Sprache Ti*k*Z verwendet, die durch LaTeX interpretiert wird. Mithilfe des Dokumenttyps Beamer in LaTeX lassen sich Präsentationen erstellen. Die Grafiken und Inhalte können dadurch dynamisch ein- oder ausgeblendet werden oder farblich hervorgehoben werden. Dies ist insofern wertvoll, da Informationen, die Schritt für Schritt vervollständigt werden, es dem Benutzer leichter machen den Ablauf nachzuvollziehen. Damit zukünftige Studenten die Prinzipien von Faltungskodes besser verstehen werden die Visualisierungen der Kodierung und Dekodierung wie beschrieben sukzessive eingeblendet.

Kapitel 4

Implementierung

Dieses Kapitel gibt einen Einblick in die Konzepte der Implementierung.

Kapitel 4.1 beinhaltet den Entwurf der Faltungskodierer-Datenstruktur. Die Implementierung der Kodierung wird in Kapitel 4.2 beschrieben, die der Dekodierung in Kapitel 4.3.

Aus Performancegründen Kodierung, Dekodierung in C++
Weiters: Kodierer erzeugen, Depunktierung, Katastrophale Kodierer Prüfung (Polynom GGT mod 2)
Parameterprüfung, Punktierung, ApplyNoise, Aufruf Visualisierung in R
Referenzimplementierung

4.1 Faltungskodierer

Ein Faltungskodierer ist gegeben durch

- N: Anzahl an Ausgangsbits je Eingangsbit,
- M: Länge des Schieberegisters,
- G: Vektor von Generatorpolynomen.

Die Angabe von M ist hier redundant, jedoch Teil der Benutzereingabe zur Generierung eines Faltungskodierers, welche durch [AoECC] inspiriert wurde.

Zur leichteren Implementierung der Kodierung und Dekodierung wird die Kodierer-Datenstruktur um folgende Elemente erweitert:

- eine *Zustandsübergangsfunktion*, die angibt, in welchen Zustand der Kodierer bei einem Eingangsbit wechselt,
- eine *inverse Zustandsübergangsfunktion*, die angibt, aus welchem Zustand der Kodierer bei einem Eingangsbit kommt,
- eine *Outputfunktion*, die angibt, welche Kodebits der Kodierer bei einem Eingangsbit in einem bestimmten Zustand ausgibt,
- ein Flag zur Markierung rekursiver systematischer Kodierer (RSC),
- ein *Terminierungsvektor* die für rekursiver systematische Kodierer angibt, ob ein Eingangsbit 0 oder 1 in einem bestimmten Zustand für die Terminierung zu verwenden ist.

Die Zustandsübergangsfunktion, inverse Zustandsübergangsfunktion und Outputfunktion sind als Matrizen implementiert. Diese Art der Implementierung wurde (aus [REF] übernommen) von der Referenzimplementierung übernommen, musste jedoch erweitert werden, um für allgemeine Faltungskodes verwendbar zu sein. Für alle gilt, die Anzahl an Zeilen entspricht der Anzahl an Zuständen 2^M. Der Zeilenindex entspricht dem Zustand. Die *Zustandsübergangsmatrix* sowie die *Outputmatrix* besitzen jeweils zwei Spalten. Je eine Spalte steht für ein Eingangsbit (0 oder 1), wobei der Spaltenindex dem Eingangsbit entspricht. Die *inverse Zustandsübergangsmatrix* benötigt eine dritte Spalte. Für viele Kodierer (v.a. nicht-rekursive) tritt der Fall ein, dass nur durch *ein bestimmtes* Eingangsbit in einen bestimmten Zustand gewechselt werden kann. Sei ein Zustand bspw. nur durch das Eingangsbit 0 erreichbar, so bedeutet das, dass es für diesen Zustand mit dem Bit 0 *zwei* Vorgängerzustände gibt, für ein Eingangsbit 1 jedoch keinen Vorgänger. Diese zweite Möglichkeit wird in der dritten Spalte gespeichert.

Der Terminierungsvektor ist für nicht-rekursive Kodierer nicht notwendig, da ein Kode eines solchen Kodierers immer mit M 0-Bits terminiert wird. Bei einem rekursiven Kodierer ist es nicht trivial zu sagen mit welchem Eingangsbit in einem bestimmten Zustand terminiert wird, um den Kodierer in den Nullzustand zu bringen. Dies hängt von der Definition des Rekursionpolynoms ab. Der Terminierungsvektor wird bei der Erzeugung rekursiver Kodierer berechnet.

4.1.1 Katastrophaler Kodierer

Bei der Erzeugung von Faltungskodierern ist zu prüfen ob es sich um einen katastrophalen Kodierer handelt. RSC-Kodierer sind, wie in Kapitel **??** beschrieben, nicht zu prüfen.

4.2 Kodierung

Bei Faltungskodes stellt die Kodierung den bei Weitem einfacheren Teil dar. Es muss lediglich jedes Bit der zu kodierenden Nachricht zusammen mit dem aktuellen Zustand, der nach jedem Bit mithilfe der Zustandsübergangsmatrix aktualisiert wird, auf die Outputmatrix angewendet werden. Die Terminierung funktioniert analog, einzig das zu kodierende Bit muss ermittelt werden. Für RSC-Kodierer muss im Terminierungsvektor nachgeschaut werden, andernfalls ist das Bit immer 0. Abgeschlossen wird die Kodierung mit dem Abbilden der Kodebits 0 bzw. 1 auf die Signalwerte +1 bzw. -1.

```
1: state = 0, code = result = " "
2: for each bit in message do
3:    output = output.matrix[state][bit]
4:    code = concat(code, output)
5:    state = state.transition.matrix[state][bit]
6: end for
7: if terminate code then
8:    for i = 0 to M − 1 do
9:       termination.bit = rsc-coder ? termination.vector[state] : 0
10:      output = output.matrix[state][termination.bit]
11:      code = concat(code, output)
12:      state = state.transition.matrix[state][termination.bit]
13:   end for
14: end if
15: for each bit in code do
16:   signal = 1 − 2bit
17:   result = concat(result,signal)
18: end for
19: return result
```

Algorithmus 1 – Faltungskodierung

4.3 Dekodierung, Viterbi-Algorithmus

4.4 Rauschen

4.5 Visualisierung

Kapitel 5

R-Paket Schnittstelle

In diesem Kapitel wird die Schnittstelle für den Benutzer erläutert. Kapitel 5.1 listet Funktionen zur Erzeugung von Faltungskodierern, der Kodierung, Dekodierung und Simulation von Faltungskodes. Kapitel 5.3 beinhaltet Hilfsfunktionen für Faltungskodes. Schließlich beschreibt Kapitel 5.4 weitere nützliche Funktionen der Kanalkodierung.

5.1 Faltungskodierung

5.1.1 ConvGenerateEncoder

ConvGenerateEncoder
`ConvGenerateEncoder(N, M, generators)` Erzeugt einen Faltungskodierer für nichtrekursive Faltungskodes. **Argumente:** `N` - Anzahl an Ausgangssymbole je Eingangssymbol. `M` - Länge des Schieberegisters des Kodierers. `generators` - Vektor der N oktale Generatorpolynome enthält (ein Polynom je Ausgangssymbol). **Rückgabewert:**

Faltungskodierer, abgebildet als Liste mit folgenden Feldern:
- N
- M
- Generatorpolynome
- nextState Matrix
- previousState Matrix
- output Matrix
- RSC Flag (`FALSE`)
- Terminierungsvektor

Tabelle 5.1 – ConvGenerateEncoder Funktion

5.1.2 ConvGenerateRscEncoder

ConvGenerateRscEncoder
`ConvGenerateRscEncoder(N, M, generators)` Erzeugt einen Faltungskodierer für rekursive systematische Faltungskodes (rsc). **Argumente:** `N` - Anzahl an Ausgangssymbole je Eingangssymbol. `M` - Länge des Schieberegisters des Kodierers. `generators` - Vektor der oktale Generatorpolynome enthält (ein Polynom je nicht-systematischen Ausgang und ein Polynom für die Rekursion). **Rückgabewert:**

Faltungskodierer, abgebildet als Liste mit folgenden Feldern:
- N
- M
- Generatorpolynome
- nextState Matrix
- previousState Matrix
- output Matrix
- RSC Flag (`TRUE`)
- Terminierungsvektor

Tabelle 5.2 – ConvGenerateRscEncoder Funktion

5.1.3 ConvEncode

ConvEncode

```
ConvEncode(message, conv.encoder, terminate,
punctuation.matrix, visualize)
```

Erzeugt einen Faltungskode aus einer unkodierten Nachricht.

Argumente:
`message` - Nachricht die kodiert wird.
`conv.encoder` - Faltungskodierer der für die Kodierung verwendet wird.
`terminate` - Markiert ob der Kode terminiert werden soll. Standard: `TRUE`
`punctuation.matrix` - Wenn ungleich `NULL` wird die kodierte Nachricht mit der Punktierungsmatrix punktiert. Standard: `NULL`
`visualize` - Wenn `TRUE` wird ein PDF-Bericht der Kodierung erstellt. Standard: `FALSE`

Rückgabewert:
Die kodierte Nachricht mit den Signalwerten +1 und -1 welche die Bits 0 und 1 darstellen. Falls punktiert wurde Liste mit dem Originalkode (nicht punktiert) und dem punktiertem Kode.

Tabelle 5.3 – ConvEncode Funktion

5.2 Dekodieren

5.2.1 ConvDecodeSoft

ConvDecodeSoft

`ConvDecodeSoft(code, conv.encoder, terminate, punctuation.matrix, visualize)`
 	
Dekodiert einen Faltungskode mittels soft decision Dekodierung.
 	
Argumente:
`code` - Faltungskode der dekodiert wird. Genauer Signalpegel (soft input).
`conv.encoder` - Faltungskodierer der für die Kodierung verwendet wurde.
`terminate` - Markiert ob der Kode terminiert ist. Standard: `TRUE`
`punctuation.matrix` - Wenn ungleich `NULL` wird der Kode vor der Dekodierung depunktiert. Standard: `NULL`
`visualize` - Wenn `TRUE` wird ein PDF-Bericht der Dekodierung erstellt. Standard: `FALSE`
 	
Rückgabewert:
Liste die die dekodierte Nachricht und die soft output Werte enthält.

Tabelle 5.4 – ConvDecodeSoft Funktion

5.2.2 ConvDecodeHard

ConvDecodeHard
`ConvDecodeHard(code, conv.encoder, terminate, punctuation.matrix, visualize)` Dekodiert einen Faltungskode mittels hard decision Dekodierung. **Argumente:** `code` - Faltungskode der dekodiert wird.

`conv.encoder` - Faltungskodierer der für die Kodierung verwendet wurde.
`terminate` - Markiert ob der Kode terminiert ist. Standard: `TRUE`
`punctuation.matrix` - Wenn ungleich `NULL` wird der Kode vor der Dekodierung depunktiert. Standard: `NULL`
`visualize` - Wenn `TRUE` wird ein PDF-Bericht der Dekodierung erstellt. Standard: `FALSE`

Rückgabewert:
Vektor der Dekodierten Nachricht.

Tabelle 5.5 – ConvDecodeHard Funktion

5.2.3 ConvSimulation

ConvSimulation
`ConvSimulation(conv.coder, msg.length, min.db, max.db, db.interval, iterations.per.db, punctuation.matrix, visualize)` Simulation einer Faltungskodierung und -dekodierung nach einer Übertragung über einen verrauschten Kanal mit verschiedenen Signal-Rausch-Verhältnissen (SNR). **Argumente:** `conv.coder` - Faltungskodierer der für die Simulation verwendet wird. Kann mittels `ConvGenerateEncoder` oder `ConvGenerateRscEncoder` erzeugt werden. `msg.length` - Nachrichtenlänge der zufällig generierten Nachrichten. Standard: 100 `min.db` - Untergrenze der getesteten SNR. Standard: 0.1 `max.db` - Obergrenze der getesteten SNR. Standard: 2.0 `db.interval` - Schrittweite zwischen zwei getesteten SNR. Standard: 0.1 `iterations.per.db` - Anzahl der Iterationen (Kodieren und Dekodieren) je SNR. Standard: 100

`punctuation.matrix` - Wenn ungleich `NULL` wird die kodierte Nachricht punktiert. Kann mittels `ConvGetPunctuationMatrix` erzeugt werden. Standard: `NULL`
`visualize` - Markiert ob ein Simulationsbericht erzeugt wird. Standard: `FALSE`

Rückgabewert:
Dataframe das die Bitfehlerrate für die getesteten Signal-Rausch-Verhältnisse beinhaltet.

Tabelle 5.6 – ConvSimulation Funktion

5.3 Hilfsfunktionen

5.3.1 ConvGetPunctuationMatrix

ConvGetPunctuationMatrix
`ConvGetPunctuationMatrix(punctuation.vector, conv.coder)` Erzeugt aus dem gegebenem Punktierungsvektor und Faltungskodierer eine Punktierungsmatrix. **Argumente:** `punctuation.vector` - Vektor der die Punktierungsinformation enthält welche in eine Punktierungsmatrix transformiert wird. `conv.coder` - Faltungskodierer der für die Matrixdimension verwendet wird. **Rückgabewert:** Punktierungsmatrix die für `ConvEncode`, `ConvDecodeSoft`, `ConvDecodeHard` und `ConvSimulation` verwendet werden kann.

Tabelle 5.7 – ConvGetPunctuationMatrix Funktion

5.3.2 ConvOpenPDF

ConvOpenPDF

`ConvOpenPDF(encode, punctured, simulation)`
 	
Öffnet die mit `ConvEncode`, `ConvDecodeSoft`, `ConvDecodeHard` und `ConvSimulation` erzeugten PDF-Berichte.
 	
Argumente:
`encode` - Markiert ob Kodierungsbericht (`TRUE`) oder Dekodierungsbericht (`FALSE`) geöffnet wird. Standard: `TRUE`
`punctured` - Markiert ob Berichte mit Punktierung geöffnet werden. Standard: `FALSE`
`simulation` - Markiert ob Simulationsbericht geöffnet wird. Standard: `FALSE`

Tabelle 5.8 – ConvOpenPDF Funktion

5.4 Kanalkodierung

5.4.1 ApplyNoise

ApplyNoise

`ApplyNoise(msg, SNR.db, binary)`
 	
Verrauscht ein Signal basierend auf dem AWGN Modell (Additive White Gaussian Noise), dem Standardmodell für die Simulation eines Übertragungskanals.
 	
Argumente:
`msg` - Die zu verrauschende Nachricht
`SNR.db` - Signal-Rausch-Verhältnis (signal noise ratio) des Übertragungskanals in dB. Standard: 3.0
`binary` - Blockkode Parameter. Nicht zu verwenden. Standard: `FALSE`
 	
Rückgabewert:

Verrauschtes Signal.

Tabelle 5.9 – ApplyNoise Funktion

5.4.2 ChannelcodingSimulation

ChannelcodingSimulation

`ChannelcodingSimulation(msg.length, min.db, max.db, db.interval, iterations.per.db, turbo.decode.iterations, visualize)`
 	
Simulation von Block-, Faltungs- und Turbo-Kodes und Vergleich ihrer Bitfehlerraten bei unterschiedlichen SNR.
 	
Argumente:
`msg.length` - Nachrichtenlänge der zufällig generierten Nachrichten. Standard: 100
`min.db` - Untergrenze der getesteten SNR. Standard: 0.1
`max.db` - Obergrenze der getesteten SNR. Standard: 2.0
`db.interval` - Schrittweite zwischen zwei getesteten SNR. Standard: 0.1
`iterations.per.db` - Anzahl der Iterationen (Kodieren und Dekodieren) je SNR. Standard: 100
`turbo.decode.iterations` - Anzahl der Iterationen bei der Turbo-Dekodierung. Standard: 5
`visualize` - Wenn `TRUE` wird ein PDF-Bericht erstellt. Standard: `FALSE`
 	
Rückgabewert:
Dataframe das alle Simulationsergebnisse der 3 Kodierungsverfahren beinhaltet.

Tabelle 5.10 – ChannelcodingSimulation Funktion

5.4.3 PlotSimulationData

PlotSimulationData
`PlotSimulationData(...)` Stellt die mitgegebenen Dataframes bzw. die Bitfehlerraten für verschiedene Signal-Rausch-Verhältnisse in einem Diagramm dar. Dataframes können mittels `ConvSimulation`, `TurboSimulation` und `BlockSimulation` erzeugt werden. **Argumente:** `...` - Dataframes die mit den Simulationsfunktionen erzeugt wurden.

Tabelle 5.11 – PlotSimulationData Funktion

Kapitel 6

Visualisierung

LIMITS!

Um das Verständnis für Faltungskodes beim Benutzer dieses R-Pakets zu stärken, stehen Visualisierungen der Kodierung, Dekodierung und Simulation mithilfe des *RMarkdown* Pakets, wie in Kapitel 3.3 beschrieben, zur Verfügung.
Wird der `visualize` Parameter bei der Ausführung einer Simulation bzw. Funktion zur Kodierung oder Dekodierung auf `TRUE` gesetzt, wird ein *RMarkdown* Skript ausgeführt. Dieses generiert eine Beamer Präsentation mit Informationen und Visualisierungen.

6.1 Kodierung

Bei der Kodierung befinden sich auf den ersten Folien allgemeine Informationen zum verwendeten Faltungskodierer wie die Kode-Rate, Generatorpolynome, Zustandsübergangstabelle etc. Daraufhin folgt die Kodierungsvisualisierung. Diese zeigt zunächst die zu kodierende Nachricht (Input), das Zustandsübergangsdiagramm sowie eine noch nicht befüllte Kodierungstabelle. Um für einen noch besseren Lerneffekt zu sorgen wird Schritt für Schritt mittels Overlays ein Bit des Inputs, der aktuelle Zustand, Folgezustand sowie der resultierende Output in eine neue Zeile der Kodierungstabelle geschrieben. Der aktuelle Zustand sowie der entsprechende Übergang werden im Diagramm farblich hervorgehoben. Die kodierte Nachricht wächst mit jedem Schritt bis schlussendlich die gesamte Nachricht kodiert wurde. Da die Kodierungsfunktion nicht die Bitwerte des Kodeworts zurückliefert

- Nicht-Rekursiver Kodierer
- Anzahl von Ausgängen :

$$N = 2$$

- Anzahl von Registern :

$$M = 2$$

- Generatoren :

$$(7,5)_8 = \begin{pmatrix} 111 \\ 101 \end{pmatrix}$$

- Kode-Rate :

$$\frac{1}{2}$$

Abbildung 6.1 – Faltungskodierer Informationen

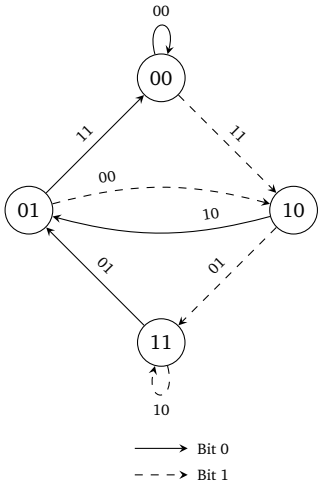

Abbildung 6.2 – Kodierung

sondern die Signalwerte (für eine Übertragung über einen Kanal) wird auf einer weiteren Folie dargestellt, wie die Kodebits in Signalwerte überführt werden.
Wird eine Punktierungsmatrix bei der Kodierung mitgegeben, wird eine zusätzliche Folie am Ende hinzugefügt. Auf dieser wird die Punktierung des Signals, d.h. das Entfernen von Signalwerten (definiert durch die Punktierungsmatrix) dargestellt. Dabei wird neben dem originalen Signal und der Punktierungsmatrix das punktierte Signal dargestellt, wobei zunächst die

punktierten Signalwerte, d.h. die entfernten Werte, durch Asterisk-Symbole ($*$) ersetzt werden. Diese Darstellung dient als visueller Zwischenschritt für das danach folgende tatsächlich punktierte Signal, bei dem die punktierten Werte fehlen, was auch dem Rückgabewert der Funktion entspricht.

6.2 Dekodierung

Bei der Dekodierung befinden sich ebenfalls, wie bei der Kodierung, allgemeine Informationen des Faltungskodierers auf den ersten Folien. Als Input erhält die Dekodierung das Kodewort als Signalwerte, die möglicherweise durch Anwendung der `ApplyNoise` Funktion verfälscht worden sind. Die soft decision Dekodierung verwendet zur Dekodierung zwar kontinuierliche Signalwerte, da aber sowohl die hard decision Dekodierung Bitwerde zur Dekodierung verwendet und Trellis-Diagramme mit Bitwerten beschriftet werden, wird auf einer Folie die Überführung der Signalwerte zu Bits dargestellt. Dieser transformierte Input wird auch als Input für die Visualisierung des Viterbi-Algorithmus verwendet. Anschließend folgt die Visualisierung des Viterbi-Algorithmus mithilfe des Trellis-Diagramms. Zunächst werden, zur besseren Übersicht bei großen Diagrammen, jene Pfade entfernt, für die es eine bessere Alternative gibt, d.h. die eine größere Metrik bei hard decision Dekodierung bzw. eine kleinere Metrik bei soft decision Dekodierung als ihre Alternative haben. Danach erfolgt Schritt für Schritt mittels Backtracking die Rekonstruktion der Nachricht. Der gewählte Pfad beim Backtracking wird farblich hervorgehoben. Die übrigen Pfade werden ausgegraut. Am Ende befindet sich unter dem Trellis-Diagramm die farblich hervorgehobene dekodierte Nachricht. Wird eine Punktierungsmatrix bei der Dekodierung mitgegeben, wird eine zusätzliche Folie nach den Kodiererinformationen hinzugefügt. Auf dieser wird die Depunktierung des Signals, d.h. das Einfügen des Signalwerts 0 (definiert durch die Punktierungsmatrix), dargestellt. Die eingefügten 0-Werte sind zur leichteren visuellen Erkennung farblich hervorgehoben.

6.3 Simulation

Weiters können Berichte der Simulation generiert werden, die die resultierenden Daten u.a. in einem Diagramm darstellen.

Kapitel 7

Beispiele

Kapitel 8

Fazit, Ausblick, Erweiterungen

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Literatur

[1] JJ Allaire u. a. *rmarkdown: Dynamic Documents for R*. R package version 0.9.5. 2016. URL: http://rmarkdown.rstudio.com/.

[2] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2016. URL: https://www.R-project.org.

[3] H. Wickham. *Advanced R*. CRC Press, 2015. URL: https://books.google.at/books?id=FfsYCwAAQBAJ.

[4] H. Wickham. *R Packages*. O'Reilly Media, 2015. URL: https://books.google.at/books?id=eqOxBwAAQBAJ.