ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Kommunikationstechnik

Prof. Dr. H. Bölcskei
Sternwartstrasse 7
CH-8092 Zürich

# PPS: Bits on Air
# 2. Teil

Markus Gärtner & Felix Kneubühler

Revidierte Version vom 16. August 2016

## 1 Einleitung

Im ersten Teil des Praktikums sollen Sie drei einfache Module programmieren: Den Sender (source) eines digitalen Kommunikationssystems, den Empfänger (drain), und den dazwischen liegenden (Funk-) Kanal (channel). Das System wird in Bezug auf wichtige Grundbegriffe der Stochastik, die ebenfalls eingeführt werden, untersucht.

Während des gesamten PPS sollten Sie unbedingt darauf achten, dass Sie sinnvolle Variablennamen verwenden. Damit ist gemeint, dass die Funktion der Variable im Namen ersichtlich wird. Ausserdem sollte Ihr Code benutzerfreundlich sein, was Sie unter anderem durch ausreichendes Kommentieren Ihres Codes erreichen können. Dies spart letzten Endes sehr viel Zeit bei der Fehlerbehebung und beim Schreiben des Berichts.

## 2 Programmierung eines einfachen Kommunikationssystems

Zunächst wollen wir das zu implementierende digitale Kommunikationssystem so einfach wie möglich gestalten; stark abstrahiert besteht ein solches System aus drei Teilen: dem Sender, dem Empfänger, und dem Kanal (Übertragungstrecke). Wir werden diese Teile einzeln programmieren.

### 2.1 Der Sender

Aufgabe des Senders wird es sein, Bitsequenzen, also Zahlenfolgen $\{b_i\}$ mit den Werten $b_i \in \{0, 1\}$ herzustellen. Der Einfachheit wegen nehmen wir an, dass von Bit zu Bit *zufällig* entschieden wird, welcher Wert (0 oder 1) gesendet wird. Wir werden dabei kennenlernen, was überhaupt mit zufällig gemeint ist.

#### 2.1.1 Zufallsvariablen und ihre Realisierungen

`Matlab` stellt eine Menge Befehle zur Verfügung, mit deren Hilfe Zufallszahlen, also „zufällig gewürfelte“ Zahlen erzeugt werden können. Eine Folge $\{b_1, ..., b_N\}$ von $N$ Bits ist eine

Folge solcher Zufallszahlen mit den Werten 0 oder 1. Zufallszahlen wiederum sind *Realisierungen von Zufallsvariablen*. Die Einführung dieser beiden ähnlichen Begriffe ist notwendig, um zwischen *möglichen* Werten, also der Zufallsvariablen, oder *konkreten* Werten, der tatsächlich gezogenen Zufallszahl, zu unterscheiden. Das Ziehen dieser Zahlen bezeichnet man als *Zufallsexperiment*.

Zufallsvariablen werden üblicherweise durch Grossbuchstaben beschrieben, während die dazugehöhrenden Kleinbuchstaben die entsprechenden Zufallszahlen sind. Eine Folge $\{b_1, ..., b_N\}$ von Bits ist also das Ergebnis eines Zufallsexperimentes, das durch die Folge von Zufallsvariablen $\{B_1, ..., B_N\}$ beschrieben wird. Ein weiteres Beispiel für ein Zufallsexperiment ist das Werfen eines Würfels. Die Zufallsvariable $W$ dieses „Würfelexperimentes" kann Werte $W \in \{1, ..., 6\}$ annehmen; das Ergebnis eines konkreten Wurfes wird dann mit $w$ bezeichnet.

Im Falle von Bits muss bestimmt werden, mit welcher *Wahrscheinlichkeit* (*probability*) $p$ eine 0 oder eine 1 auftritt. Die Zahl $p$ ist selbst immer ein Wert zwischen 0 und 1, wobei 0 bedeutet, dass der Wert nie (mit Wahrscheinlichkeit 0) auftritt, und 1, dass er sicher auftritt. Ein Wert von $p = 1/2$ bedeutet demnach, dass der Wert in 50% aller Fälle vorkommt.

Ist $p$ die Wahrscheinlichkeit, dass eine 0 auftritt, $p = Pr(B_i = 0)$, so gilt $Pr(B_i = 1) = 1 - p$, da die Wahrscheinlichkeit, dass *entweder* $B_i = 0$ *oder* $B_i = 1$ ist, gleich 1 ist. Das Bit kann andere Werte als 0 oder 1 gar nicht annehmen.

`Matlab` selbst stellt einen Zufallzahlengenerator zur Verfügung, mit dessen Hilfe Zufallszahlen $a$ im kontinuierlichen Intervall von $a \in (0, 1)$ erzeugt werden. Die Erzeugung geschieht so, dass jeder Wert von $a$ mit gleicher Wahrscheinlichkeit auftritt, also $Pr(a = a_i) =$ const., $\forall a_i \in [0, 1]$. Die Zufallsvariable $A$, deren Realisierung die Zahlen $a$ sind, wird daher als *gleichverteilt* bezeichnet. Es gibt noch viele weitere als nur gleichverteilte Zufallsvariablen; eine weitere, die *gaussverteilte* Zufallsvariable, wird später behandelt.

### 2.1.2 Erzeugung und Untersuchung von Zufallszahlen mit `Matlab`

Mit `rand(1,N)` wird in `Matlab` ein $N$-dimensionaler Vektor aus Realisierungen einer gleichverteilten Zufallsvariablen $A \in (0, 1)$ erzeugt. Jedes erneute Aufrufen dieses Befehls führt zu einer neuen Folge von $N$ Realisierungen.

Das Verhalten zufälliger Grössen wird häufig anhand von Histogrammen untersucht. Ein Histogramm ist ein Balkendiagramm, bei dem Ereignisse, die innerhalb eines bestimmten Intervalls liegen, gezählt werden. In `Matlab` wird ein Histogramm eines Vektors **v** durch Aufrufen des Befehls `hist(v,M)` gebildet, wobei $M$ die Anzahl der Balken des Histogramms ist. Weglassen von $M$ liefert die Standardeinstellung $M = 10$. Weitere Details zu den Funktionen finden Sie durch `help hist`, bzw. `help rand`.

Tipp zu Matlab: Unter Home->Preferences->Keyboard->Shortcuts können die Shortcuts auf „Windows Default Set“ gesetzt werden, falls man diese angenehmer findet. Die kleinen Pfeile oben rechts im Command Window sind dazu da das Fenster in der Umgebung anzudocken.

**Aufgaben:**

- Veranschaulichen Sie sich die Arbeitsweise des Zufallszahlengenerators `rand` mit Hilfe des `hist` Befehls. Wählen Sie dazu Zufallsvektoren verschiedener Längen (z.B. $N = 5$, $N = 1 \cdot 10^5$). Woran erkennt man, dass es sich um eine gleichverteilte Zufallsvariable handelt? Was passiert, wenn man die Zahl der Balken des Histogramms ändert? Beachten Sie, dass ein Balkendiagramm nur dann zuverlässig ist, wenn genügend Ereignisse in den einzelnen Balken zusammengefasst werden.
- Schreiben Sie ein Programm `source.m`, das eine Quelle eines digitales Kommunikationssystems ist. Das Programm soll zwei Eingabeparameter haben: `seq_length`, die Länge der auszugebenden Bitsequenz, und `p`, die Wahrscheinlichkeit dafür, dass ein ausgegebendes Bit den Wert 0 annimmt. Ausgegeben werden soll nur die realisierte Bitsequenz.

  Gehen Sie dabei folgendermassen vor:

  1. Der Befehl `edit` öffnet den internen Editor von Matlab. Das Program muss die Kopfzeile

     ```
     function bitsequence = source(sequence\_length,p)
     ```

     haben, da Dateiname und Funktionsname identisch sein müssen. Kommentare können hinter % eingegeben werden. *Wenn Sie das Programm speichern, achten Sie darauf, dass es in dem von Ihnen angelegten Verzeichnis abgelegt wird!*
  2. Benutzen Sie den Befehl `rand`, um gleichverteilte Zufallszahlen zu erzeugen. Überlegen Sie sich, wie Sie mit diesen Zahlen eine Bitfolge erzeugen können.

## 2.2 Der Empfänger

Nach dem Sender soll der Empfänger programmiert werden. Da das zunächst zu programmierende Kommunikationssystem keinen Fehler verursachenden Kanal enthalten wird, sind die empfangenen Bits gleich den gesendeten Bits; da ausserdem noch kein Modulator im Sender implementiert ist, ist die Struktur des Empfängers denkbar einfach. Im Grunde soll der Empfänger zuerst nur benutzt werden, um wichtige statistische Aussagen über die gesendeten Bitsequenzen zu erhalten.

### 2.2.1 Einfache statistische Kenngrössen eines Übertragungssystems

Die Quelle des Übertragungssystems wurde so programmiert, dass sie bei jedem Sendevorgang, also bei jedem Aufrufen des Programms, eine bestimmte Anzahl Bits sendet. Da die Bits durch einen Zufallsgenerator erzeugt werden, sieht jede Bitsequenz anders aus. Trotzdem gibt es Kenngrössen, die die Bitsequenzen allgemein beschreiben.

Im folgenden beschreibe $\{X_1, ..., X_N\}$ eine Folge von Zufallsvariablen, die durch das wiederholte Ausführen eines durch die Zufallsvariable $X$ beschriebenen Zufallsexperimentes erzeugt werde. Die hier eingeführten Begriffe gelten also nicht nur für Bitsequenzen $\{B_1, ..., B_N\}$.

**Mittelwert**
Eine intuitiv sehr einfache Grösse zur Charakterisieung von $\{X_1, ..., X_N\}$ ist der *Mittelwert*. Ist eine Realisierung $\{x_1, ..., x_N\}$ gegeben, so errechnet sich der Mittelwert dieser Sequenz, $\mu_N$, als

$$\mu_N = \frac{1}{N} \sum_{k=1}^{N} x_k.$$

Wird der Mittelwert wie oben bestimmt, so ist dieser Wert nur für eine gegebene Sequenz eine feste Grösse. Werden mit demselben Zufallszahlengenerator verschiedene Sequenzen erzeugt und ihre Mittelwerte berechnet, so ergeben sich zwischen diesen Werten Abweichungen. Diese Abweichungen werden jedoch mit steigender Sequenzlänge N immer kleiner bis man für genügend grosse $N \longrightarrow \infty$ einen konstanten Wert $\mu$ erhält. Diese Konvergenz von $\mu_N$ gegen $\mu$ ist als *stochastische Konvergenz* oder auch als *schwaches Gesetz grosser Zahlen* bekannt.

Der Grenzwert $\mu$ wird *Mittelwert der Zufallsvariable* $X$ genannt und ist eine wichtige Charakterisierung für $X$. Die Grösse $\mu_N$, die den Mittelwert approximiert, bezeichnet man in der Stochastik als *Schätzer* für den Mittelwert.

**Bit error rate**
Eine für ein Kommunikationssystem sehr wichtige Kenngrösse ist die *Bit Error Rate*, kurz *BER*. Die BER gibt an, welcher Teil der empfangenen Bits fehlerhaft ist. Um die BER zu bestimmen, muss die Differenz der gesendeten und der detektierten Sequenz gebildet werden. Um bei einem allfälligen Fehler im erlaubten Intervall [0,1] zu bleiben, wird nach der Differenz die Operation *modulo 2* angewendet. Überall dort, wo in dieser Sequenz eine 1 steht, wurde ein Bit falsch übertragen bzw. falsch detektiert. Die BER ist die normierte Summe dieser Sequenz,

$$BER_N = \frac{1}{N} \sum_{k=1}^{N} mod_2(x_k^{sent} - x_k^{det}),$$

wobei $x_k^{sent}$ die von der Quelle gesendeten Daten, und $x_k^{det}$ die evtl. fehlerhaften detektierten Daten sind. Wie im Falle des Mittelwerts konvergiert auch dieser Schätzer für grosse $N$ stochastisch gegen einen Grenzwert BER.

**Varianz**
Eine weitere wichtige Grösse ist die *Varianz*. Die Varianz gibt Auskunft über die quadratische Abweichung vom Mittelwert, charakterisiert also Fluktuationen einer Zufallsvariable um ihren Mittelwert. Die Varianz wird häufig mit $\sigma^2$ bezeichnet; die Grösse $\sigma$ bezeichnet die *Standardabweichung* und ist die Wurzel der Varianz. Für eine gegebene Sequenz $\{x_1, ..., x_N\}$ erhält man einen Schätzer für die Varianz als

$$\sigma_N^2 = \frac{1}{N-1} \sum_{k=1}^{N} (x_k - \mu)^2,$$

wenn der Mittelwert bekannt ist. Anderfalls muss statt $\mu$ der geschätzte Mittelwert $\mu_N$ verwendet werden. Auch $\sigma_N$ ist eine Zufallsvariable, die dem schwachen Gesetz grossen Zahlen unterworfen ist.

### 2.2.2 Mehrfache Übertragung zufälliger Sequenzen

Die Schätzung von Mittelwerten und Standardabweichungen wird umso genauer, je länger die gesendeten Sequenzen sind. Anstatt aber die Sequenzlänge immer weiter zu erhöhen, kann man auch kürzere Sequenzen mehrfach hintereinander senden, vorausgesetzt, dass jede Sequenz für sich neu erzeugt wird. Das hat auch den Vorteil, dass das statistische Verhalten der Grössen $\mu_N$ oder $\sigma_N$ anhand von Histogrammen untersucht werden kann, da ja gleich mehrere Sequenzen und deren Mittelwerte und Varianzen vorliegen. Anstelle also zusätzlich zum Sender nur noch einen Empfänger zu programmieren, sollen Sender und Empfänger in eine Schleife eingebettet werden, so dass das Kommunikationssystem viele Sequenzen hintereinander senden kann.

**Aufgaben:**

- Schreiben Sie ein Programm `drain.m`, das als Eingabewert die Bitsequenz `bitsequenz`, und als Ausgabewerte die Grössen $\mu_N$ und $\sigma_N$ liefert. Benutzen Sie hierfür die Befehle `mean` und `std`.
- Schreiben Sie ein Programm `loop.m`, welches es ermöglicht, die Programme `source.m` und `drain.m` in einer Schleife `loopsize` mal laufen zu lassen. Schreiben Sie das Programm so, dass es am Ende des Ablaufes in einem Histogramm die ermittelten Grössen $\mu_N$ ausgibt. Probieren Sie dabei verschiedene Werte für $N$ und `loopsize`. Gegen welchen Wert konvergiert $\mu_N$?

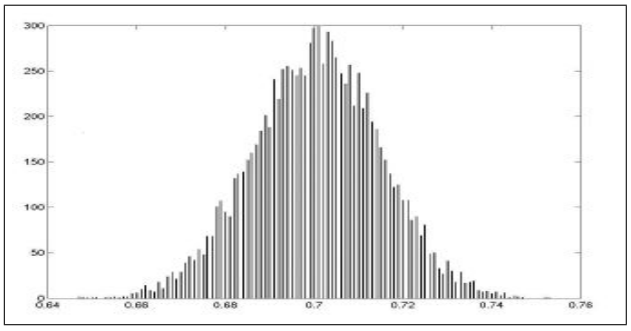


Abbildung 1: Histogramm von *loop(n)*

- Ein sehr wichtiges Theorem in der Stochastik ist der *zentrale Grenzwertsatz*. Der zentrale Grenzwertsatz sagt aus, dass für eine Zufallsvariable $X$ das Histogramm der normierten Summe $\frac{1}{N}\sum X_k$ von $N, N \gg 1$, dieser Zufallsvariablen durch eine *Gausskurve* approximiert werden kann. Die Gausskurve hat folgende Form:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp -\frac{(x-\mu)^2}{2\sigma^2}.$$

Die Gausskurve enthält die zwei Grössen $\mu$ (ihren Mittelwert) und $\sigma$ (ihre Standardabweichung) als Parameter. Da auch $\mu_N$ aus einer Summe von Zufallsvariablen gebildet wird, lässt sich das Histogramm des Mittelwertes $\mu_N$ durch eine Gausskurve beschreiben.

Schreiben Sie ein Programm `gauss.m`, das die Gausskurve für einen beliebigen Mittelwert und eine beliebige Varianz zeichnet. Verwenden sie die Funktion `lookfor` um in der Dokumentation nach einer Funktion, die die Gaussverteilung implementiert zu suchen (Tipp: Verteilungsfunktionen sind oftmals mittels fuzzy logic mittels sogenannten membership functions implementiert). Schätzen Sie für einen Durchlauf von `loop.m` den Mittelwert $\mu$ und Standardabweichung $\sigma$ und plotten Sie damit eine Gausskurve. Vergleichen Sie das Histogramm mit ihrem Plot der Gausskurve. Hängen $\sigma$ und $p$ voneinander ab?
(Tip: Der Befehl `figure(n)` öffnet ein Fenster mit der Nummer $n$ für den nächsten Graphen. Mit Hilfe dieses Befehls können mehrere Graphen nebeneinander ausgegeben werden. Ein ähnliches Ergebnis wird auch mit `subplot` erreicht.)

### 2.3 Der Kanal

Zunächst wollen wir den Kanal in sehr abstrahierter Form betrachten, also ohne jeglichen Bezug zu einer physikalischen Realität. Der Kanal sei einfach ein System, das mit Wahrscheinlichkeit $q$ Übertragungsfehler verursacht. Wird vom Sender eine 1 gesendet, so wird nur mit Wahrscheinlichkeit $1 - q$ eine 1 empfangen, und mit Wahrscheinlichkeit $q$ eine 0. Wird eine 0 gesendet, so wird diese nur mit Wahrscheinlichkeit $1 - q$ empfangen und statt dessen mit Wahrscheinlichkeit $q$ eine 1. Eine graphische Darstellung dieses Systems ist in Abbildung 2 gegeben. Wegen seiner symmetrischen Struktur wird dieser Kanal auch Binary Symmtetric Channel (BSC) genannt.

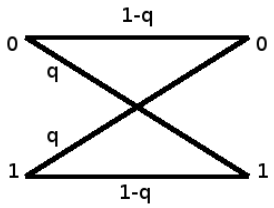


Abbildung 2: Graphische Darstellung des *Binary Symmetric Channel*. Die Variable $q$ bezeichnet die Wahrscheinlichkeit eines Bitfehlers.

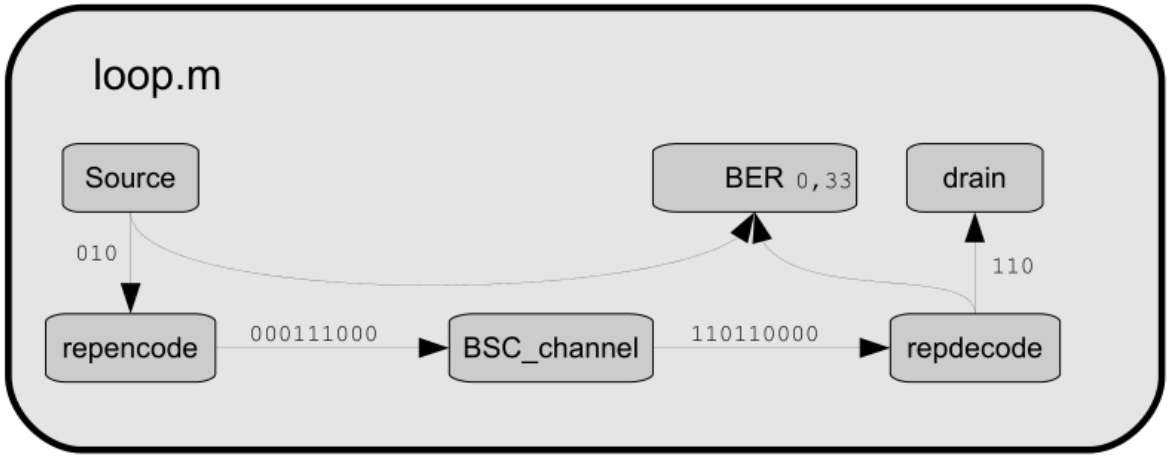


Abbildung 3: Graphische Darstellung des gesammten Programmflusses

**Aufgaben:**

- Schreiben Sie ein Programm `BSC_channel.m`, das `bit_sequence` und die Wahrscheinlichkeit $q$ als Eingabeparameter, und `channel_bit_sequence` als Ausgabewert hat. Schreiben Sie das Programm `calcBER.m`, welches aus der empfangenen und der gesendeten Bitsequenz die $BER$ berechnet. Verifizieren Sie, dass $BER \simeq q$, zumindest für lange Sequenzen.

  (Tipp: `help mod, help reshape, help repmat` zeigt Ihnen jeweils Hilfen zu den 3 Funktionen an. Die Funktionen `repmat` und `reshape` erlauben es die Methoden ohne `for`-schleifen zu schreiben, was wesentlich effizienter ist!)

- Die BER eines Systems kann ganz wesentlich durch Codierung verbessert werden. Ein sehr einfacher Code ist der Repetitionscode, bei dem jedes Bit nicht einmal sondern $n$ mal gesendet wird. Der Empfänger entscheidet sich dann für dasjenige Bit, das in dieser Sequenz am häufigsten vorkommt. Soll also eine 1 übertragen werden, und es ist $n = 3$, so wird 111 gesendet. Der Empfänger entscheidet sich dann für eine 1, wenn mindestens zwei aller empfangenen Bits Einser sind, bzw. wenn `mean(`$x_1x_2x_3$`)>0.5`. Nachteil dieses Verfahrens ist natürlich, dass die Übertragungsrate um den Faktor $n$ geringer ist als bei einem uncodierten System.

  Ändern Sie Ihre Programme so, dass Sie einen Repetitionscode variabler Länge verwenden können. Überprüfen Sie, dass die BER sinkt, wenn Sie $n$ erhöhen. Wieso ist ein Repetitionscode viel besser, wenn $q$ klein ist?

- Ihr Programm sollte nun einen Programmfluss haben wie in Abbildung 3. Falls nicht, passen Sie Ihr Programm diesem an.