

BER-Basisband

October 12, 2017

1 Bitübertragung - Basis- und Breitbandverfahren

Wir betrachten hier die Übertragung von Daten mit unterschiedlichen Modulationstechniken und die sich dabei einstellenden Bitfehlerraten.

```
In [ ]: # setup:
import numpy as np
from pprint import pprint as pp
%matplotlib notebook
from matplotlib import pyplot as plt

In [ ]: # Parameters
numBits = 10000

# Welche SNRs betrachten? Von SNR [dB] zur Standardabweichung des AWGN-Kanals
logsnrs = np.linspace(-10,15)
snrs = 10**(0.1*logsnrs)
sigmas = 1/snrs**0.5
```

1.1 Basisband

Einige Hilfsfunktionen, zunächst Basisband. Dann die eigentliche Simulation der Basisbandübertragung.

```
In [ ]: def generate_bits(numBits):
        """Zufällige Folge von Bits erzeugen"""
        return np.random.randint(0, 1+1, numBits)

def modulate(bits):
    """Einfache Amplituden-Modulation"""
    modulated = 2*bits-1
    return modulated

def channel(bits, sigma):
    """Simple Kanalmmodell: AWGN
    Eingabe: bits ein Array von modulierten Bits (als Phasenwert)
            sigma die Standardabweichung des AWGN-Kanals
    Ausgabe: Array gleicher Länge wie bits, mit verrauschten Werten
```

```

    """
    ### YOUR SOLUTION HERE

    return noisy

def demodulate(noisy_bits):
    """Amplitudenmodulation demodulieren: einfache Entscheidung um 0
    """
    ### YOUR SOLUTION HERE

    return bits

def transmit(numBits, sigma,
             modulate=modulate,
             demodulate=demodulate,
             channel=channel):
    """Einfaches Übertragungsmodell: bits erzeugen, modellieren, durch Kanal
    demodulieren und die BER ausrechnen.
    """
    r = {}
    r['bits'] = generate_bits(numBits)
    r['modulated'] = modulate(r['bits'])
    r['noisy'] = channel(r['modulated'], sigma)
    r['received'] = demodulate(r['noisy'])
    r['error'] = [ t != r for t, r in zip(r['bits'], r['received'])]
    r['ber'] = sum(r['error']) / numBits

    return r

results = [transmit(numBits, sigma) for sigma in sigmas]
# pp([(sigma, r['ber']) for sigma, r in zip(sigmas, results)])

```

1.1.1 Eine Folge von Bits als Beispiel

Wir nehmen hier als Beispiel die Übertragung beim besten SNR.

```

In [ ]: def plot_specific_sigma(traces):
        f, ax = plt.subplots(2, sharex=True)
        # plt.plot(trace['noisy'], 's')
        ax[0].plot(traces[0]['noisy'], 's')
        ax[0].set_title('Worst SNR')
        ax[1].plot(traces[-1]['noisy'], 's')
        ax[1].set_title('Best SNR')

        plt.xlabel('Bit #')
        plt.ylabel('Signal')

```

```

        # plt.show()

plot_specific_sigma(results)

```

1.1.2 BER over SNR

```

In [ ]: plt.figure()
        # plt.plot(sigmas, [r['ber'] for r in results])
        plt.semilogy(logsnrs, [r['ber'] for r in results])
        plt.xlabel('SNR [dB]')
        plt.ylabel('BER')
        plt.show()

```

1.2 Breitband-Modulation

1.2.1 QPSK

Gleiche Struktur wie oben: wir brauchen modulation und Demodulationsfunktion. Und einen Kanal. Diesmal sind das nicht Amplitudenwerte, sondern komplexe Konstellationspunkte. Rauschen ist hier AWGN, unabhängig im Real- und Imaginärteil. Ergänzen Sie den Code für den Kanal!

```

In [ ]: # QPSK modulation and corresponding channel:

import itertools

def qpsk_modulate(bits):
    # behold the power of python: elegant AND efficient!
    bitpairs = zip(bits[::2], bits[1::2])
    return [complex(2*a-1, 2*b-1) for (a,b) in bitpairs]

def qpsk_demodulate(symbols):
    tmp = [(1 if s.real > 0 else 0,
            1 if s.imag > 0 else 0,
            ) for s in symbols]
    # flatten the list of tuples into a simple list
    return itertools.chain(*tmp)

def qpsk_channel(symbols, sigma):
    """Assumption: quadrature and phase noise the same"""

    ### YOUR SOLUTION HERE

    return result

qpsk_results = [transmit(numBits, sigma,
                        modulate=qpsk_modulate,
                        demodulate=qpsk_demodulate,

```

```

channel=qpsk_channel) for sigma in sigmas]
# pp([(sigma, r['ber']) for sigma, r in zip(sigmas, qpsk_results)])

```

Ein Beispielplot für einige SNR-Werte

```

In [ ]: f, axes = plt.subplots(1, 2)

for ax, qr in zip(axes, (qpsk_results[0], qpsk_results[-1])):
    # pp(r['noisy'])
    noisy = qr['noisy']
    x, y = list(zip(*[(c.real, c.imag) for c in noisy]))
    ax.set_adjustable('box')
    ax.set(xlim=[-10, 10], ylim=[-10, 10], aspect=1)
    ax.plot(x, y, '*')

```

1.3 16 QAM

Let's move from simple QPSK to a 16 QAM and compare resulting BER.

```

In [ ]: # 16 QAM using a modulation table

```

```

qam16_table = [complex(-1, -1), complex(-1/3, -1), complex(1/3, -1), complex(1, -1),
               complex(-1, -1/3), complex(-1/3, -1/3), complex(1/3, -1/3), complex(1, -1/3),
               complex(-1, +1/3), complex(-1/3, +1/3), complex(1/3, +1/3), complex(1, +1/3),
               complex(-1, +1), complex(-1/3, +1), complex(1/3, +1), complex(1, +1),
               ]

```

```

def qam16_modulate(bits):
    bit4tuples = list(zip(bits[::4], bits[1::4], bits[2::4], bits[3::4],))
    # pp(bit4tuples)
    symbolnumbers = [8*a + 4*b + 2*c + d
                    for (a, b, c, d) in bit4tuples]
    # pp(symbolnumbers)
    return [qam16_table[s] for s in symbolnumbers]

```

```

def qam16_demodulate(symbols):
    """Find the closest smybol.
    This is horribly inefficient, but good enough here"""

```

```

def closest_symbol(s):
    distances = [(i, abs(s-q)) for i, q in enumerate(qam16_table)]
    mindist = min(distances, key=lambda x: x[1])[0]
    return mindist

```

```

def bitstring(s):
    """return a four-digit bit string for the integer s"""
    tmp = bin(s)[2:]
    tmp = '0'*(4-len(tmp)) + tmp

```

```

    return tmp

    # map to the closest symbol:
    symbols = [closest_symbol(s) for s in symbols]
    # pp(symbols)

    # turn symbols back into bit sequences:
    tmp = [[int(x) for x in bitstring(s)] for s in symbols]
    # pp(tmp)
    return list(itertools.chain(*tmp))

```

```

In [ ]: fewbits = generate_bits(40)
        # pp(fewbits)
        qamsymbols = qam16_modulate(fewbits)
        # pp(qamsymbols)
        noisy = qpsk_channel(qamsymbols, 0.01)
        received = qam16_demodulate(noisy)
        # pp(received)

```

```

In [ ]: qam16_results = [transmit(numBits, sigma,
                                modulate=qam16_modulate,
                                demodulate=qam16_demodulate,
                                channel=qpsk_channel) for sigma in sigmas]
        # pp([(sigma, r['ber']) for sigma, r in zip(sigmas, qam16_results)])

```

```

In [ ]: # Do a loglog plot
        plt.figure()
        plt.semilogy(logsnrs, [r['ber'] for r in qpsk_results], 'g')
        plt.semilogy(logsnrs, [r['ber'] for r in qam16_results], 'r')
        plt.xlabel('SNR [dB]')
        plt.ylabel('BER')

```