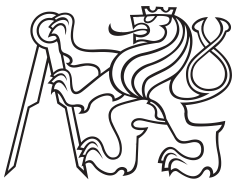
Bachelor Project



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Radio Engineering

Coding and Signal Processing Algorithms in Single-User Radio Communications

David Machát

Supervisor: prof. Ing. Jan Sýkora, CSc.
Field of study: Open Electronic Systems
January 2018

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Radioelectronics

BACHELOR PROJECT ASSIGNMENT

Student: **David Machát**

Study programme: Open Electronic Systems

Title of Bachelor Project: **Coding and Signal Processing Algorithms in Single-User Radio Communications**

Guidelines:

Student will design and analyze a set of coding and signal processing algorithms for a simple single-user radio communication link and validates them and evaluates their theoretically predicted performance by a computer simulation and/or, where appropriate, by an over-the-air experiment using an experimental TxR platform. It should include selected forms of single/multiple-input/output modulation, coding and receiver processing (demodulation, decoding and synchronisation) algorithms as well as selected signal processing algorithms (e.g. power spectrum density estimation). The simulation and validation should have a modular structure with an instructive visual form.

Bibliography/Sources:

[1] J. G. Proakis: Digital Communications. McGraw-Hill.

Bachelor Project Supervisor: prof. Jan Sýkora Ing., CSc.

Valid until the end of the summer semester of academic year 2017/2018

L.S.

doc. Mgr. Petr Páta, Ph.D.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 27, 2017

Acknowledgements

I would like to thank my supervisor, prof. Ing. Jan Sýkora, CSc., for his patience and for the thoroughness with which he helped with the creation of this work.

Declaration

I declare that I have completed the presented thesis independently and that I wrote down all used sources in accordance with the metholodogical instruction on ethical principles in academic theses.

In Prague, 9. January 2018

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací

V Praze dne 9. ledna 2018

Abstract

This thesis focuses on elementary signal processing algorithms. Its main focus is on various forms of modulations, coding and synchronisation. Its goal is creation of working digital communication models in MATLAB environment and verifying their functionality.

Keywords: Modulation, Coding, Viterbi algorithm

Supervisor: prof. Ing. Jan Sýkora, CSc.

Abstrakt

Tato bakalářská práce se zabývá elementárními algoritmy pro zpracování digitálních signálů. Zaměřuje se především na různé formy kódování, modulace a synchronizace. Jejím cílem je vytvoření funkčních modelů digitální komunikace v prostředí MATLAB a ověření jejich funkčnosti.

Klíčová slova: Modulace, Kódování, Viterbiho algoritmus

Contents

1 Introduction **1**

2 Theory **3**

2.1 Source encoding 3

2.2 Channel encoding 4

2.2.1 Finite Field 4

2.2.2 Linear codes 4

2.2.3 Code properties 6

2.3 Complex envelope 6

2.4 Digital modulation 7

2.4.1 Signal space 8

2.4.2 Modulation pulse 8

2.4.3 Linear modulators 10

2.5 Channel 12

2.6 Demodulator 12

2.6.1 Synchronisation 13

2.6.2 Matched filter 13

2.7 Decoder 14

2.7.1 Hard and Soft decision decoding 14

2.7.2 Syndrome-based decoding . . . 14

2.7.3 Viterbi decoding 15

3 Practical **17**

3.1 Signal Processing algorithms . . . 17

3.1.1 Convolution 18

3.1.2 Periodogram 18

3.1.3 Coding 19

3.1.4 Hamming code 19

3.1.5 Convolutional code 20

3.2 Modulator 21

3.3 Channel 23

3.4 Demodulator 23

3.5 Decoder 25

3.5.1 Block codes decoding 25

3.5.2 Convolutional codes decoding 26

3.5.3 Communication models 28

4 Performance **31**

5 Conclusion **37**

Bibliography **39**

A CD contents **41**

Figures

2.1 Impulse response of filter $g'[k]$ . . . 9

2.2 Frequency response of shortened RRC pulse 9

2.3 OOK and 4ASK constellations . 10

2.4 16QAM signal constellation 11

2.5 BPSK and 4PSK constellations . 11

2.6 Matched filter response to RCS pulse unaffected by noise 14

2.7 Viterbi algorithm forward step . 16

2.8 Viterbi algorithm backwards phase 16

4.1 BPSK error probability 32

4.2 Block error probability for Hard decision decoding 33

4.3 Hamming code bit error rates . . 33

4.4 Convolutional code bit error rates 34

4.5 M-PSK bit error rates comparison 35

4.6 4PSK convolutional code bit error rates 35

Chapter 1

Introduction

The goal of this work is to design group of selected digital signal processing algorithms. Visualise them and test their performance in computer simulation. Main focus is on algorithms describing channel encoding, modulation, channel, demodulation and decoding parts of digital communications model. Although these building block are not completely isolated and frequently overlap, we will try to describe them separately where possible. First part of this thesis describes mechanisms and theoretical background of these algorithms. Second part shows their implementation in MATLAB environment, visualizes how they operate and builds functioning digital communication models. Third part verifies algorithm performance and shows the effectivity of various modulation and coding combinations.

Chapter 2

Theory

2.1 Source encoding

First step in digital data communication is making sure the data we want to transmit are actually digital. If not, It is necessary to employ source encoding, also called data compression, Its goal is to reduce the overall number of bits while preserving the information that is being transmitted by reducing redundancy present in the source or interpreting arbitrary data in the form of bits. As an example we can look at analog source of information. To get the data in binary form, it is necessary to quantize the source's range. When encoding each state into n bits, we can differentiate between up to 2^n states of the source. To measure the amount of data being sent, we use bit rate and symbol rate. Bit rate shows the amount of bits sent per second.

$$R_b = \frac{1}{T_b} \tag{2.1}$$

The symbol rate shows the amount of symbols sent through the channel per second. For modulations with two states, the Bit rate is equal to symbol rate. For modulations which have more than two states the symbol rate is smaller than bit rate as every data symbol of modulation with 2^n possible states contains within itself n data bits.

2.2 Channel encoding

Channel encoding, also called error-correcting coding or forward error correction (FEC) coding differs from the source encoding both in purpose and methods. While source encoding reduces the amount of bits being sent. The channel encoding codes parts of bit stream into codewords by adding additional bits, called parity bits. These bits improve our ability to find and correct errors introduced by noisy or unreliable communication channel.

The two main types of FEC codes are block codes and convolutional codes. Block codes work with fixed amounts of bits and convolutional work with bit streams of arbitrary length.

2.2.1 Finite Field

Finite field with q elements is a set with defined arithmetic for a limited number of elements. All finite fields are in the form q^n, where q is a prime number and n is a positive integer and are denoted as GF(q^n). Arithmetic operations within such field are defined modulo(q^n). Our focus will be mainly on the smallest galois (finite) field, GF(2) with elements called one and zero which we will directly asociate with 1 and 0 values of data bits. In the table 2.1 are the addition and multiplication operations for GF(2).

+	0	1
0	0	1
1	1	0

.	0	1
0	0	0
1	0	1

Table 2.1: Addition and multiplication operations for GF(2)

The additive identity of GF(2) is zero and the multiplicative identity is one.

2.2.2 Linear codes

Linear codes are codes for which any linear combination of codewords is also a codeword. In this thesis we will work with two biggest families of linear

codes, block codes and convolutional codes. In linear block codes, we use the Generator matrix $\mathbf{G}$ which generates codewords of set length n from segments of the data input vector of length k by adding m parity bits. The relationship between n and k is called code rate and is defined as

$$R = \frac{n}{k} \tag{2.2}$$

Every generator matrix can be reduced to its systematic form, which is defined as:

$$\mathbf{G} = \begin{bmatrix} \mathbf{P} \\ \mathbf{I_k} \end{bmatrix} \tag{2.3}$$

Where $\mathbf{I_k}$ is an Identity matrix of size $k \times k$ and $\mathbf{P}$ is a matrix which governs the added parity bits. The generator matrix can be built in a way that the original sequence of bits is a part of the created codeword. Codes that possess this property are called systematic. Parity bits help us with decoding from uncertainties introduced by the channel. The rows of the generator matrix form the basis of a linear code and the codewords are its linear combinations. This combined with the fact that equivalent codes are codes that differ only by permutation of codeword elements [Sý05], makes every linear code equivalent to linear systematic code. From generator matrix we can construct the parity check matrix $\mathbf{H}$. For generator matrices in systematic forms, the parity check matrix is:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I_{n-k}} & \mathbf{P}^T \end{bmatrix}. \tag{2.4}$$

With $-\mathbf{P}^T$ being the transpose of the $\mathbf{P}$ matrix. The parity check matrix of a linear code is a generator matrix of its dual code.

The main difference between convolutional and block codes is that convolutional coders require memory. Bits encoded are dependent on several previous incoming bits. The general form of a linear convolutional code is

$$c_n = \sum_{i=0}^{K} \mathbf{G}_i b_{n-i} \tag{2.5}$$

Where K is called constraint length and shows us by how many previous bits is the coding of the current bit affected. Convolutional codes can be described by their generator matrices, state tables or by graphical diagrams: code tree, trellis or state diagram.

2.2.3 Code properties

Quality of linear codes is dictated by their codeword weight and free distance ρ_{free}. Codeword weight is the codeword's Hamming distance from zero codeword, free distance is smallest Hamming distance between two codewords. Hamming distance is defined as the number of bits in which the two codeword differ.[HV94]

2.3 Complex envelope

Complex envelope, also called a complex baseband is a frequency shifted analytic representation of real signal. For an arbitrary non-stationary real signal $s(t)$, its Fourier transform has Hermitian symmetry around $f = 0$.

$$\mathcal{F}_s(f) = \mathcal{F}_s^*(-f) \tag{2.6}$$

This symmetry allows us to discard the negative frequencies of the spectrum without any loss of information. For this purpose an analytical representation of the original signal is formed. It consists of the original signal $s(t)$ and its Hilbert transform $\hat{s}(t) = s(t) * \frac{1}{\pi t}$. Hilbert transform shifts components on positive frequencies by $-\frac{\pi}{2}$ and components on negative frequencies by $\frac{\pi}{2}$.

$$s_a(t) = s(t) + j\hat{s}(t) \tag{2.7}$$

This abandons the negative frequencies and the signal becomes complex.

To represent modulated signals we will be using the constellation diagrams. For that it is useful to show the analytical signals of sine and cosine waves.

$$s_a(t) = \cos(2\pi f_0 t) + j\mathcal{H}[\cos(2\pi f_0 t)] = \cos(2\pi f_0 t) + j\sin(2\pi f_0 t) = e^{2\pi f_0 t} \tag{2.8}$$

$$s_a(t) = \sin(2\pi f_0 t) + j\mathcal{H}[\sin(2\pi f_0 t)] = \sin(2\pi f_0 t) - j\cos(2\pi f_0 t) = -je^{2\pi f_0 t} \tag{2.9}$$

Complex envelope is then created from the analytical signal by a shift in frequency towards 0 Hz.

$$x(t) = s_a(t)e^{-j2\pi f_0 t} \tag{2.10}$$

Where f_0 is called carrier frequency which is an arbitrary real positive number. The carrier frequency and complex envelope together fully represent both analytical and the original signals. The complex envelopes of sine and cosine waves can be respectively interpreted as real and imaginary components of

linearly modulated signal displayed in the signal space at the symbol sampling instants. The real part is called In-phase component and the imaginary is called quadrature component The restoration of original signal from complex envelope can be done with the knowledge of the carrier frequency.

$$s(t) = \mathcal{R}[x(t)]\cos(2\pi f_0 t) - \mathcal{I}[x(t)]\sin(2\pi f_0 t) \tag{2.11}$$

Given that complex envelope and carrier frequency fully represent real signal, we will perform our simulations on complex envelopes.

2.4 Digital modulation

Digital modulators allow us to map a series of binary data symbols into continuous signal waveforms. This process is separated in modulator into two parts which often overlap. First is the discrete part which maps incoming data message **d** to channel symbols **q**. For modulators with memory, state σ of modulator is also used. Second half of modulator is expansion part where channel symbols are assigned modulation pulses with finite energy. This system can be described as:

$$s(t) = \sum_n g(d_n, \sigma_n, t - nTs) = \sum_n g(q_n, t - nTs) \tag{2.12}$$

Where d_n is the n-th data symbol, q_n is the n-th channel symbol and σ_n is the corresponding modulator state.[Sý16]

Channel symbol q_n is one out of channel symbol alphabet $\mathcal{A}_q = \{q^{(i)}\}_{i=0}^{M_q-1}$ with $q^i \in \mathbb{C}$. Mean symbol energy is calculated as follows

$$\bar{\mathcal{E}}_S = \frac{1}{2}\lim_{L\to\infty}\frac{1}{L}E\left[\int_{-\infty}^{\infty}|s_L(t)|^2 dt\right] \tag{2.13}$$

Free distance ρ^2_{free} is a property determines resistance to noise of chosen modulation. It is defined as a half of the smallest distance between two constellation points.

$$\rho^2_{free} = \frac{1}{2}min_{i\neq i'}|q^{(i)} - q^{(i')}|^2 \tag{2.14}$$

Where $q^{(i)}$ and $q^{(i')}$ are two different channel symbols. Free distance shows us the minimal distance between two channel symbols. This quality is useful while demodulating signals.

2.4.1 Signal space

Signal space is a Hilbert space with orthonormal basis

$$\{\zeta_{n,i}(t)\} = \{\zeta_i(t - nT_S)\}_{i=1}^{N_s} \tag{2.15}$$

where n is the position in sequence and i is the modulation dimension index. Modulated signal can be decomposed into orthogonal basis functions. [Sý16]

$$s_{n,i} = \int_{-\infty}^{\infty} s(t)\zeta_i^*(t - nT_S)dt \tag{2.16}$$

2.4.2 Modulation pulse

In this thesis we will be using some of the most commonly used modulation pulses. The main reason for their usage are their favourable spectral properties. We want these pulses to fulfill the Nyquist intersymbol interference criterion. In time domain Nyquist criterion is defined as orthogonality w.r.t. sequence position [Sý05]

$$\mathcal{R}_{g1,g2}^{\mathcal{E}}[m-n] = \int_{-\infty}^{\infty} = g_1(t - nT_s)g_2^*(t - mT_s)dt = 0 \qquad \forall m \neq n \tag{2.17}$$

Another option is to choose a pulse which best conforms with the Nyquist criterion in frequency domain

$$\frac{1}{T_s}\sum_{k=-\infty}^{\infty} H\Big(f - \frac{k}{T_s}\Big) = 1 \qquad \forall f \tag{2.18}$$

In which T_s is the symbol period and $H(f)$ is the Fourier transform of the impulse response. In order to minimise intersymbol interference we will use rectangular (REC), raised cosine spectrum (RCS) and root raised cosine spectrum (RRC) pulses. The Rectangular pulse is a pulse of constant amplitude and finite energy. The raised cosine spectrum pulse is defined in frequency domain[Sý16].

$$\mathcal{F}_{v_{RCS}}(f) = \begin{cases} T_s, & \text{for} \quad |f| < \frac{1-\alpha}{2T_s} \\ T_s \cos^2\left(\frac{\pi T_s}{2\alpha}\left(|f| - \frac{1-\alpha}{2T_s}\right)\right), & \text{for} \quad \frac{1-\alpha}{2T_s} \leq |f| \leq \frac{1+\alpha}{2T_s} \\ 0, & \text{for} \quad \frac{1+\alpha}{2T_s} \leq |f| \end{cases} \tag{2.19}$$

where α is the roll-off factor ($0 < \alpha < 1$). Another, similar pulse is the root raised cosine spectrum pulse. It is defined in frequency domain as

$$\mathcal{F}_{v_{RRC}}(f) = \begin{cases} \sqrt{T_s}, & \text{for} \quad |f| < \frac{1-\alpha}{2T_s} \\ \sqrt{T_s}\cos\left(\frac{\pi T_s}{2\alpha}\left(|f| - \frac{1-\alpha}{2T_s}\right)\right), & \text{for} \quad \frac{1-\alpha}{2T_s} \leq |f| \leq \frac{1+\alpha}{2T_s} \\ 0, & \text{for} \quad \frac{1+\alpha}{2T_s} \leq |f| \end{cases} \tag{2.20}$$

These pulses are defined in frequency domain due to their unique spectral properties. When using their perfect time domain representations, they achieve the Nyquist condition in frequency domain. However, achieving this would require the usage of infinite duration pulses which would violate causality. As such we have to work finite and causal discrete time approximations $g'[k]$.[Sý16]

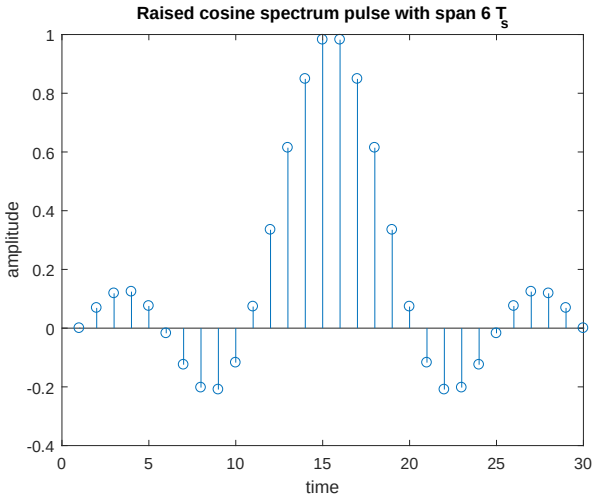


Figure 2.1: Impulse response of filter $g'[k]$

Finiteness is accomplished by applying rectangular masking function and causality is accomplished by shifting the pulse by half of its length. Applying rectangular masking function and thus shortening the pulse causes power spectrum density changes.

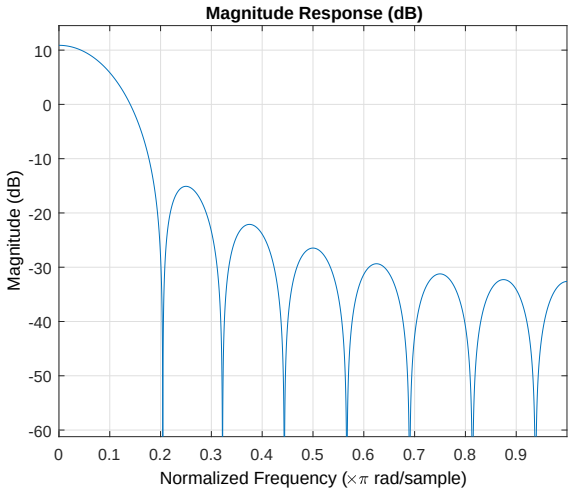


Figure 2.2: Frequency response of shortened RRC pulse

2.4.3 Linear modulators

Modulators with modulation dimension one are called linear and their complete signal space basis is

$$\{\zeta_{n,1}(t)\}_n = \{g(t - nT_S)\}_n \tag{2.21}$$

Linear modulators are divided according to the characteristic of carrier signal that they alter. One of the carrier signal characteristics that modulators can alter is its amplitude. These modulators assign differing values of amplitude to different data symbols.

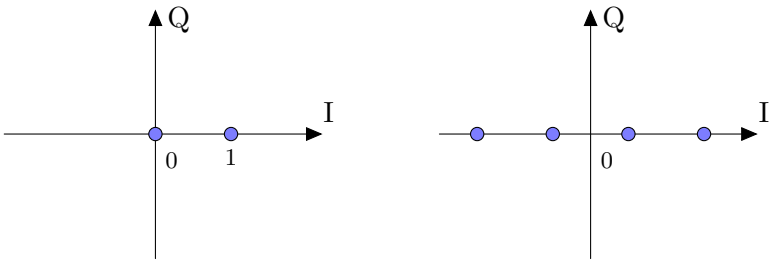


Figure 2.3: OOK and 4ASK constellations

The simplest amplitude modulation is the On-Off keying (OOK). OOK modulators have only two channel symbols, one and zero. These symbols represent the states in which the sent signal is turned on and off. They may be mapped directly to incoming data symbols. Another group of modulators changing amplitude are the Amplitude Shift Keying (ASK) modulators. These modulators map groups of N data symbols to 2^N possible channel symbols. Possible channel symbols for M-ary amplitude shift keying are $q_n = \{\pm 1, \pm 3, ... \pm (M_q - 1)\}$

One of the most popular modulations is the quadrature amplitude shift keying (QASK).Sometimes also called modulation Quadrature Amplitude modulation (QAM) which consists of superposition of two amplitude shift keying modulations.[Sý05] In the signal constellation space the quadrature amplitude modulation is represented by a rectangular raster.

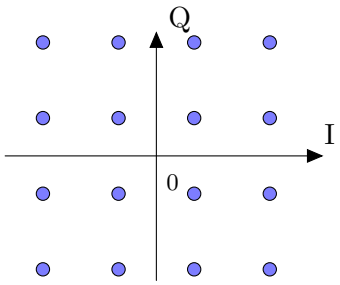


Figure 2.4: 16QAM signal constellation

Another characteristic of the carrier signal that can be modulated is its phase. Modulation method which change the phase of carrier signal are classified as Phase Shift Keying (PSK) modulations. M-ary phase shift keying modulation usually has channel symbols $q_n\{e^{j\frac{2\pi}{M_q}i}\}_{i=0}^{M_q-1}$ and mean symbol energy $\bar{\mathcal{E}}_S = \frac{1}{2}$

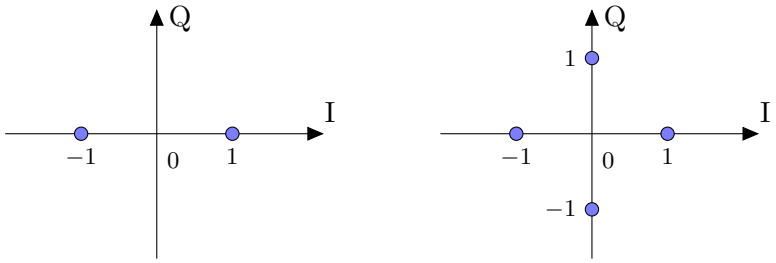


Figure 2.5: BPSK and 4PSK constellations

It is also possible to combine these two modulation methods. Such approach is called Amplitude-Phase Shift Keying (APSK) [Sý16].

Another way to classify linear modulations is into memoryless modulations and modulations with memory. All of the modulations described above are memoryless. An example of modulation with memory is the Differential Phase Shift Keying (DPSK). This phase modulation encodes the information by the change of constellation phase.

2.5 Channel

Channel is the medium between one or more transmitters and one or more receivers. Channel effect on the sent data is prepresented by channels Input-Output relation between input signal $y(t)$ and output signal $x(t)$ with channel operator $\mathcal{H}[.]$.

$$y(t) = \mathcal{H}[x(t), \theta] \tag{2.22}$$

Where θ is introduced channel parameter. Channels can be classified according to their time domain behaviour to time-invariant or time-variant channels, by their frequency domain behaviour to frequency selective and non-selective (flat), by their stochastic behaviour to deterministic and random, by their linearity or by their input and output dimensionality. Composite channels are composed from several channel models. The most common channel is the Additive White Gaussian Noise (AWGN) model. Used to simulate the random processes affecting real signals, this model adds to the transmitted data noise with uniform power across the whole spectrum. In time domain the noise has normal distribution and zero mean. This noise is added to any other channel model. The relation of channel operator with AWGN $w(t)$ parameter can be represented as

$$y(t) = x(t) + w(t) \tag{2.23}$$

Another channel model used is frequency non-selective (flat) Linear Time-Invariant (LTI) channel. This model multiplies the signal by α which represents the loss of power over the distance, shifts it in time by τ and in phase by ϕ. These channel parameters change during the observation very little or not at all. Thanks to that we can consider them constant. This channel model also does not distort the signal shape. [Sý05].

$$y(t) = \alpha e^{j\phi} x(t - \tau) \tag{2.24}$$

2.6 Demodulator

In demodulator, we try to compensate the effect of channel to successfully recover sent data by finding the data message estimate $\hat{d}$ and mapping from the observation space into data message space. For this we use minimum error probability detector (MEP) which minimises mean message error and symbol error probabilities by maximising the probability of correct detection over all possible mappings $\check{d}$.[Sý16]

$$\hat{d} = arg\max_{\check{d}} Pr\{d = \check{d}\} \tag{2.25}$$

2.6.1 Synchronisation

In order to compensate for the phase shift introduced by channel, it is necessary to know by how much the signal was shifted. One of the possible ways and the one we will be using to determine the size of the shift is to use pilot signal. Before transmitting the data message we will transmit unmodulated pulse and determine its phase shift.

$$x(t) = g(t - T_s)e^{j\phi} + w(t) \tag{2.26}$$

The phase estimation will be corrupted by the AWGN. One way to decrease the influence of additive noise is to increase the length of pilot signal. Phase shift changes in time very slowly or not at all allowing us to write:

$$E[x] = E[e^{j\phi}] + E[W(t)] = e^{j\phi} \tag{2.27}$$

2.6.2 Matched filter

In this thesis we will work with a matched filter demodulator. The name comes from the fact that the impulse response of these filters has the same shape as the modulating pulse. One of the demodulation methods is the use of matched filters which maximise the output signal to noise ratio[Pro95]. For an arbitrary complex signal $s(t)$, the definition of its matched filter is given by its time inverted complex conjugate impulse response which has the following form

$$h(t) = s^*(T - t) \tag{2.28}$$

The response to a signal $t(s)$ is then

$$y(t) = \int_0^t s(\tau)s^*(T - t + \tau)d\tau \tag{2.29}$$

Which is essentially an autocorrelation function of signal $s(t)$, achieving peak at time $t = T$. Given that the length of convolution of two signals is the sum of their lengths minus one and that we convolve two signals of same length, we can see that the perfect time for symbol sampling is at $t = T$ and that its final part at time $t = 2T - 1$ does not interfere with sampling of next symbol at time $2T$.

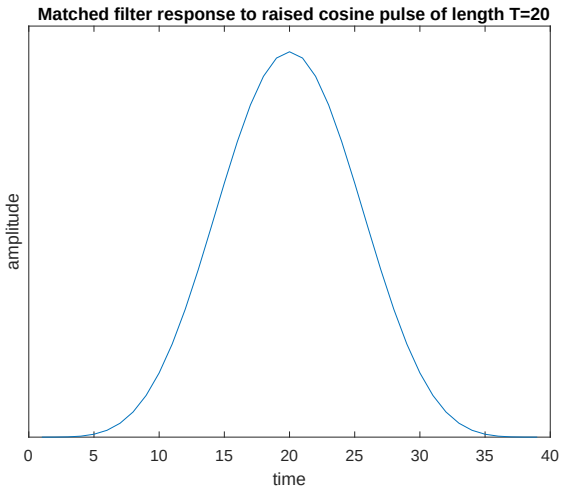


Figure 2.6: Matched filter response to RCS pulse unaffected by noise

2.7 Decoder

2.7.1 Hard and Soft decision decoding

When decoding data after demodulation, we use the parity bits introduced into the data stream earlier by encoders. They tell us whether what we have received is a valid codeword or whether we were unable to correctly determine the originally sent data. To this end we use two decoding methods. Hard and soft decision decoding. Hard decision decoding discards any information from the demodulator other than its constellation threshold-based decision and maximises the probability of decoding correctly by picking a valid codeword with smallest Hamming distance. Soft decision decoding works directly with demodulator output and chooses the replacement codeword with smallest Euclidean distance. This allows for more flexibility with cases near thresholds which Hard decision decoding may interpret incorrectly.

2.7.2 Syndrome-based decoding

Linear block codes can be decoded by using their syndrome. Syndrome decoding uses the fact that parity matrix $\mathbf{H}$ is a generator matrix to the dual code of the one generated by $\mathbf{G}$ and therefore

$$\mathbf{H}^T\mathbf{G} = \mathbf{0} \tag{2.30}$$

This property can be exploited when decoding data with possible errors because matrix multiplication is distributive. Then for received codeword $z = x + e$, where x is the original codeword and e is the introduced error, applies [Sý05]

$$\mathbf{H}^T z = \mathbf{H}^T x + \mathbf{H}^T e = \mathbf{H}^T e. \tag{2.31}$$

With a precomputed table mapping $\mathbf{H}e$ to e we know exactly which bit to repair.

2.7.3 Viterbi decoding

Given that convolutional encoder uses preceding data symbols to determine state of the coder, It can be described as finite-state machine. Optimum decoder is then maximum-likelyhood sequence estimator [Gol07]. Method which we will use to find this sequence is called Viterbi algorithm. Viterbi algorithm searches through the trellis for the most probable series of events that led to the current state of the code. The metric used to represent the most probable path ending in state m is called path metric p_m and the metric for determining likelyhood of transitions between states n and m is called branch metric $b_{n>m}$. We are looking for paths which maximise the probability of decoded sequence being correct. Such paths are those that minimise the probability of erroneous detection. Therefore, we are looking for allowed paths with lowest distances incoming data. Soft decision decoding uses euclidean distances and Hard decision decoding uses hamming distances. Viterbi algorithm works in two phases. In first, forward phase, the algorithm checks upcoming state m and looks at all branches that lead into it. Then compares the sums of branch metrics $b_{n>m}$ and corresponding path metrics p_n. Out of these, the path with optimum metric is chosen and set as the new path metric p_m and the chosen branch is called survivor. Rest of the paths leding into this state are discarded as they are no longer able to beat the survivor path.

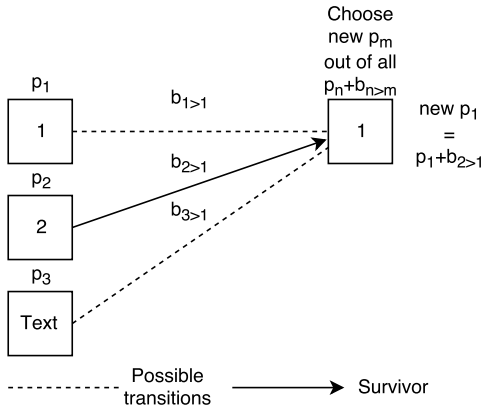


Figure 2.7: Viterbi algorithm forward step

This is done for all possible states m. After each stage there is only one surviving path per decoder state. When the algorithm reaches the end of data sequence, first phase ends and out of the terminal paths the one with optimum metric is chosen as the definitive coder state sequence. With the sequence to be decoded selected, second, backward phase begins. Starting with the last state, algorithm looks where did the survivor path leading into it came from and tracks the survivor paths back to the start.

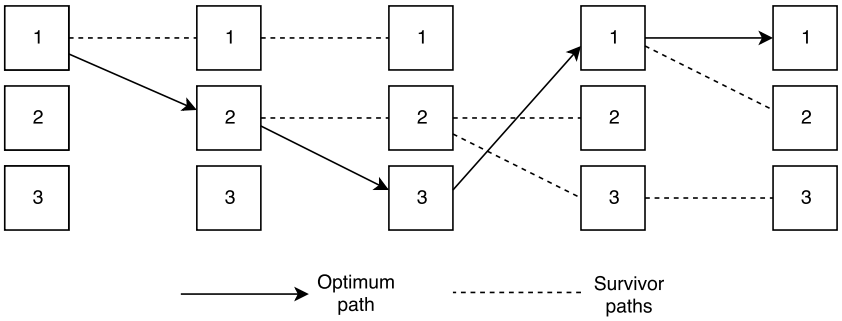


Figure 2.8: Viterbi algorithm backwards phase

Decoding is then just a matter of comparing the state sequence with precomputed state table.

Chapter 3

Practical

This section describes MATLAB implementations of used signal processing algorithms. Algorithms are banded together according to their role in digital signal communication which allows us to build observation models depending on one's goals. In this thesis we will first describe standalone signal processing algorithms, then describe building blocks such as modulator or coder. In the end we will build several observation models from these blocks. One for computing performance of given algorithms and one for possible data visualisation.

3.1 Signal Processing algorithms

In addition to helping visualise the function of these algorithms they are also built in a way which allows for their possible reproduction in other programming languages, namingly in C. Given this role most of the algorithms are built without the use of any advanced functions with the sole exceptions being fast Fourier transform (FFT) and functions generating random sequences. First exception is due to FFT libraries being commonly found in many programming languages and second is due to being necessary for simulating values which would given in real applications (source data and additive white gaussian noise).

3.1.1 Convolution

First function is discrete time convolution.

$$(f[k] * g[k])[i] = \sum_{i=-\infty}^{\infty} f[i]g[k-i] \tag{3.1}$$

It is performed by function *convo* with input arguments being two vectors that enter discrete time convolution and output being vector of their convolution. Code first extends one input vector by zeros to the length of length(f)+length(g)-1. Then creates empty vector of identical length for convolution output. The algorithm then keeps shifting the non-prolonged input vector and one by one computes additions to the output vector.

```
function [z]=convo(f,g)
gn=[g zeros(1,length(f)-1)];
z=[zeros(1,length(f)+length(g)-1)];
for i=1:length(f)+length(g)-1
    for k=1:length(f)
        if((i-k+1)>0)
            z(i)=z(i)+f(k)*gn(i-k+1);
        end
    end
end
```

3.1.2 Periodogram

To compute the spectral density of signal we will use periodogram. Periodogram is a power spetrum density estimate obtained as squared-magnitude Fourier Transform of a signal divided by its length (in our case, the number of samples).

$$\mathcal{S}_{x_T}(f) = \frac{|X_T(f)|^2}{2T} \tag{3.2}$$

This method of estimating power spectrum density works only for deterministic signals as it operates only with one realization of random parameter. For random signals there is necessary additional averaging of the periodogram. This can be done by averaging over independent realizations or by sliding window [Sý16].

$$\hat{S}_x(f) = \frac{1}{KN} \sum_{k=0}^{K-1} |X[k]|^2 \tag{3.3}$$

Where K is the number of segments over which we are averaging and N is their length. Following code applies rectangular window of length N and

performs Fourier transform on resulting signals. This method of estimating power spectrum density by averaging estimates of non-overlapping segments is called Bartlett's method.

```
for K=1:N:(length(y)/N)
    wnd= y(1,K:K+N-1);
    yfft=fftshift(fft(wnd));
    sum=sum+yfft
end
PSDE=sum/(length(ynew)/N);
```

3.1.3 Coding

In this thesis we will work exclusively with binary data and leave the analog-to-digital conversion or the transformation from an arbitrary format to binary of the input data to the user. First building block described here is the channel coding part in which we introduce redundancy into the input data stream by applying channel coding methods. It is called by function *coding* with two input arguments, input data vector and chosen coding method. Then it returns data vector of coded input data. Depending on the code method chosen, this method may increase the amount of data to be sent and thus reduces the rate at which the data are being transmitted.

3.1.4 Hamming code

For demonstration, we will be using a Hamming(4,7) code given by generator matrix [Pro95]

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

We can see that this code is systematic (it contains identity matrix) and adds three parity bits which are placed at the first three positions of the codeword. This code has free distance of length three. Thanks to this it can repair any defect that changes one bit. It can detect defect that causes two bits to change but it will repair them incorrectly. Errors with Hamming distance

equal to multiple of free distance cannot be detected nor corrected. In this implementation the code checks the length of input data and if necessary adds zeros to the end so the incoming data sequence has usable length. Then the input vector is divided into blocks of length 4 and multiplies them with Generator matrix. As we are computing within the GF(2), the result of matrix multiplication is taken Modulo-2 and is put at the end of output vector.

```
%Signal length adjustment
if mod(length(y),4)~=0;
    yc=[y zeros(1,4-mod(length(y),4))];
else
    yc=y;
end
for i=1:4:length(yc)
    %Block selection
    Ax=yc(i:i+3,1)
    Ax=[yc(1,i);yc(1,1+i);yc(1,i+2);yc(1,i+3)];
    A=mod(Ax'*G,2);
    yenc=[yenc A];
end
```

3.1.5 Convolutional code

As an example of convolutional codes, we will use a simple code from: [HV94].

$$G_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \qquad G_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad G_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{3.5}$$

which has a code rate of 0.5 and constraint length $K = 3$. Generalized code for an arbitrary generator matrix G(m,n) is initialized by adding $K - 1$ zeros before the bit sequence to calculate the codeword for first bits. Each incoming bit then generates m bits at the output which are dependent on current and $n - 1$ previous bits of the coded sequence.

```
G=[1 1 1;1 0 1];
m=size(G,1);
n=size(G,2);
y=[zeros(1,n-1) y];
yenc=NaN(1,(length(y)-(n-1))*2);
for k=n:length(y)
    for l=1:m
        yenc(1,m*(k-1)-n+l-1)=mod(G(l,:)*y(k:-1:k-2)
            ',2);
```

```
    end
end
```

3.2 Modulator

Modulator implementation consists of two parts. First one assigns each bit combination one of n possible channel symbols. Second part upsamples data stream to avoid intersymbol interference and convolves chosen channel symbol with finite impulse response (FIR) filter with response in the shape of selected modulation pulse. Function modulator.m assigns constellation space points. Following code shows the implementation of Binary Phase Shift keying

```
yout=2*y-1
```

Assigning constellation space point 1 to bits with value 1 and constellation space point -1 to bits with value zero. BPSK is identical with 2ASK. On-Off keying is implemented in the same way. Only difference being bits with value 0 are assigned to constellaton point space 0. M-ary Phase shift keying assigns values $e^{j\frac{2\pi}{M}i}$ to groups of bits. Code padds the input vector to aproppriate length and assigns bits to constellation points. Following code shows the implementation of 8PSK.

```
C=[0 0 0;0 0 1;0 1 1;0 1 0;1 1 0;1 1 1;1 0 1;1 0 0];
if mod(length(y),3)~=0
    y=[y zeros(1,3-mod(length(y),3))];
end
for i=1:3:length(y)-2
    for a=1:8
        if y(1,i:i+2)==C(a,:)
            yn((i+2)/3)=exp(1i*pi/4*(a-1));
        end
    end
end
```

As an example of mulations with memory here follows the implementation of DPSK modulation in which data are modulated base modulator state.

```
state=1;
for i=1:2:length(y)-1
    for a=1:4
        if y(1,i:i+1)==C(a,:)
```

```
            yn((i+1)/2)=exp(1i*pi/2*(a-1));
        end
    end
yn((i-1)/2+1)=state*yn;
state=yn((i-1)/2+1);
end
```

Second function is *Pulse*. This function upsamples data stream by the entered number of samples per pulse and adds a pilot signal of required signal before the data. Instead of ideal pulses $g[k]$ we have to work with their finite and causal approximations $g'[k]$. Causality is accomplished by shifting the pulse by half of its length. Finiteness is accomplished by applying rectangular masking function $u[k] = 0, k \notin [0, ..., N_S]$.

$$g'[k] = g[k - \frac{N_S}{2}]u[k] \tag{3.6}$$

```
yUpsampled=[];
%Adding pilot signal
yn=[ones(1,pilotlength) yn];
%Upsampling by pulse length Ts
for i=1:length(yn)
yUpsampled=[yUpsampled yn(i) zeros(1,Ts-1)];
end
switch PulseType;
    ...
    %RCS pulse
    span=2;
    T=1;
    x=linspace(-span*T,span*T,Ts);
    alpha=0;
    for i=1:Ts
        z(i)=(T^3*sin(pi*x(i)/T)*cos(alpha*pi*x(i)...
        /T))/(pi*x(i)*(T^2-4*alpha^2*x(i)^2));
    end
    %normalisation to unit energy
    pow=sum(z.^2);
    z=z/sqrt(pow);
    ...
end
%Pulse train creation
y=conv(yUpsampled,Pulse);
```

3.3 Channel

Here we add the white gaussian noise and introduce a phase shift to the transmitted signal. Considering we are working with complex baseband representations of real signals it is also necessary to work with complex noise. To both I and Q signal components we will add a white gaussian noise with variance N_0. This will generate a noise of total variance $2N_0$. We also have to take into account the number of samples T_s. Total noise added per sample is

$$\frac{2N_0}{T_s} \tag{3.7}$$

Following code calculates average symbol power and modifies the variance to match given signal to noise ratio (SNR). Code rate R is also included, as during observation, we are usually interested in the energy per useful bit.

```
pow=(yin*yin')/(length(yin));
sigma=sqrt(Ts*pow./(2*R*10^(snr/10)));
addnoise=(randn(1,length(yin))+1i*randn(1,length(yin
   )))*sigma;
yout=yin+addnoise;
```

Where MATLAB function randn creates vector of random numbers with variance equal to one and zero mean.

3.4 Demodulator

In the demodulator we will focus on successfully recognizing transmitted symbols. Values received from sampling matched filter demodulator output at appropriate times will be projected into the signal space. Demodulator is divided into several functions. First one is *MatchedFilter* with inputs of noisy signal, number of samples per symbol, length of used pilot signal which correlates incoming signal with complex conjugate time reversed version of original pulse used, estimates phase shift introduced by channel and compensates for it

```
%%Phase estimation
ph=y(1,1:Pilot*Ts);
pha=convo(ph,Pulse);
for k=1:Pilot
    phas(k)=pha(k*Ts);
```

```
end
phas=sum(phas)/Pilot;
y=y/exp(1i*atan2(imag(phas),real(phas)));
```

Then works with the waveforms containing data and samples them at appropriate time.

```
matched=[ones(1,Ts)];
c=convo(y,matched);
for p=1:length(y)/Ts
    yz(p)=c(p*Ts)
end
yz=sqrt(2)*yz/sqrt(Ts);
```

Third part is function *metric* with two outputs: first is closest constellation point and second is a matrix of euclidean distances from all possible constellation points. Necessary inputs for function *metric* are input data and modulation method chosen. Following sample code shows approach to 4PSK.

```
%preallocation
yout=NaN(1,length(yin));
ydist=NaN(4,length(yin));
for k=1:length(yin)
    B=[-1 1 -1i 1i];
    %finding euclidean distances
    A=abs(yin(k)-B);
    ydist(1:end,k)=A;
    %choosing the smallest
    [~,b]=min(A);
    yout(k)=B(1,b);
end
```

In case of choosing hard decision decoding there is another function *mapper* which maps incoming channel symbols to data according to their closest constellation space point. Its inputs are sequence of channel symbols and appropriate type of modulation. Following snippet shows 16QAM segment of *mapper* function. Matrices B and C contain constellation points and their respective bit combinations.

```
yout=NaN(1,4*length(yz));
B=[-1-1i 1+1i ...
C=[0 1 1 1;1 1 0 1;...
for k=1:length(yz)
    for m=1:16;
        if(yz(k)==B(1,m))
```

```
            yout(1,(4*k-3):4*k)=C(m,:);
        end
    end
end
```

3.5 Decoder

3.5.1 Block codes decoding

Linear block codes coded by generator matrix **G** are decoded by their parity-check matrix **H**. We will work with a parity-check matrix for the code defined in 3.2.3. This code has the following parity-check matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

Multiplying a valid codeword by parity-check matrix would result in a zero vector. Multiplying an invalid codeword will give us a vector of values. This vector is the same as the n-th column of parity-check matrix. Then we assume that the eror is appeared on the n-th symbol of the demodulated sequence and it will be repaired. Another option is to repair bits according to precomputed table.

```
for i=1:7:length(yc)
    Ax=yc(1,i:i+6);
    A=(Ax*H');
    A=(mod(A,2));
    d=0;
    %Syndrome check
    for b=1:3
        if mod(A(1,b),2)~=0
            d=d+1;
        end
    end
    if d>0
        %Searching which bit to repair
        for a=1:7
            if A'==H(:,a)
                m=a;
            end
```

```
        end
    %Codeword correction
    Ax(1,m)=mod(Ax(1,m)+1,2);
    end
ydecoded=[ydecoded Ax(1,1:4)];%Parity bits disposal
end
```

This algorithm checks the output of matrix multiplication and if necessary, compares it with columns fo parity check matrix and corrects resulting errors. After the correction it discards the parity bits and passes the rest of the codeword further. Soft decision decoding of the block code is implemented by comparing the distances from all possible codewords and choosing the closest. Following code shows the soft decision decoder implementation.

```
%list of all possible codewords
codewords=[0 0 0 0 0 0 0;0 0 0 1 0 1 1;...
codewords2=codewords*2-1;
for k=1:7:length(mfout)
    A=NaN(1,16);
    %calculating distance from all codewords
    for m=1:16
        A(1,m)=sum(abs(mfout(1,k:k+6)-codewords2(m
            ,:)).^2);
    end
    [~,b]=min(A);
    ydecoded=[ydecoded codewords(b,1:4)];
end
```

3.5.2 Convolutional codes decoding

First step in decoding convolutional codes with the Viterbi algorithm is comparison of incoming data with all possible paths and computing their metrics. To this end we will use 2 functions to compute the Hamming and Euclidean distances. First is *HDist* for computation of Hamming distance between two input vectors.

```
for a=1:length(b)
    if b(1,a)~=c(1,a)
        dist=dist+1;
    end
end
```

Second is function *Softdist* for computation of euclidean distance which depends also on used type of modualion. Its input are two data vectors and chosen modulation type. Following code snippet shows the computation of euclidean distances for 4PSK modulations.

```
statetab=[0 0 1j;0 1 -1j;1 1 1;1 0 -1];
for a=1:4
    if c==statetab(a,1:2)
    s=abs(b-statetab(a,end)).^2;
end
```

Following code performs the standalone Viterbi algorithm. Matrix C shows the state sequences and their metrics. Each state has its row which will always end in it. First position in a row is always the metric of the path. Followed by a sequence of states which end by the nominal state.

```
C=[0 1 NaN(1,(length(yc)/2))];
C=repmat(C,4,1);
for i=1:2:length(yc)
    H=[];
    for a=1:8  %computing branch metrics for one
       step
        c=[1 1 0 0;1 2 1 1;2 3 1 0;...
        %initial state, ending state and branch
           between them
        f=[yc(i) yc(i+1)];
        g=[c(a,3) c(a,4)];
        [b]=HDist(f,g);
        H=[H;c(a,1) c(a,2) b];
        %initial state, ending state and branch
           metric
    end
```

First part computes the distance of incoming data from all possible trellis transitions. Second part looks at all path leading into a node and chooses the one with lowest path metric. This path is then copied into new matrix of paths Cn. This process is then repeated for all nodes until Cn is complete. The C matrix is then replaced by Cn.

```
    for e=1:4
        comparison=[];
        for m=1:size(H,1)
            if H(m,2)==e
                comparison=[comparison;H(m,1) H(m,3)
                   +C(H(m,1),1)];
            end
```

```
        end
    [amin,bmin]=min(comparison(:,2));%Searching
       optimum metric
    Cn(e,:)=C(comparison(bmin,1),:);%preparing new
       state table
    Cn(e,1)=amin;%new metric
    Cn(e,((i+1)/2)+2)=e; %comparison(bmin,1);
    end
%new state table
C=Cn;
end
```

Last part chooses the path with lowest overall metric and decodes it according to precomputed table.

```
[~,minz]=min(C(:,1));
%choosing best path
Cfin=C(minz,2:end);
%precomputed decoding table
DecTable=[0 1 NaN NaN;NaN NaN 0 1;0 1 NaN NaN;NaN
   NaN 0 1];
for j=1:(length(yc)/2)
    ydecoded(j)=DecTable(Cfin(1,j),Cfin(1,j+1));
end
```

3.5.3 Communication models

To demonstrate the manner in which data travel in digital communication system we will build from aforementioned blocks two functioning models. First one

```
%% Initialisation
CodeMethod='convols';
ModType='4PSK';
PulseType='RCS';
Ts=16;%symbol samples
yinitial=round(rand(1,200));
snr=6;
Pilotlength=1;
R=1
%% Modulation
[ycoded]=coding(yinitial,CodeMethod);
[ymodulated]=Modulator(ycoded,ModType);
```

```
[yenvelope]=Pulse(ymodulated,Ts,PulseType,
   Pilotlength);
yenvelope(length(yenvelope)-Ts+2:length(yenvelope))
   =[];
%% Channel
yphaseshifted=LTIch(yenvelope);
yshiftednoisy=Noise(yenvelope,snr,Ts,R);
%% Demodulator
ymfoutput=MatchedFilter(yshiftednoisy,Ts,PulseType,
   Pilotlength);
[ysymbols,~]=metric(ymfoutput,ModType);
yout=Mapper(ModType,ysymbols);
%% Decoder
[ydecoded]=decoding(yout,CodeMethod,ymfoutput);
perf=PerfCheck(yinitial,ydecoded);
```

Where function *Perfcheck* is function counting the number of differences between two input sequences of same length.

```
errorcount=0;
for i=1:length(y1)
    if y1(i)~=y2(i)
        errorcount=errorcount+1;
    end
end
```

This composition allows for examination and visualisation of data at any stage. As an examples Function of second model is to measure bit or symbol error rates. It uses modified version of previous model. It runs the simulation with given SNR until it reaches given amount of errors and returns the rate at which they appear.

```
datalength=50;
n=30;
snrdb=linspace(-3,10,n);
plotprob=NaN(1,n);
for k=1:n
    bitsent=0;
    errors=0;
    snr=snrdb(k);
    while errors<100
        yinitial = round(rand(1,datalength));
        bitsent=bitsent+datalength;
        ...
        ...
        ...
```

```
        ...
        newerrors=PerfCheck(ynew,ydecoded);
        errors=errors+newerrors;
    end
    perf(1,k)=errors/bitsent;
end
semilogy(snrdb,perf);
```

Chapter 4

Performance

Let's start by evaluating the error performance of BPSK modulation in AWGN channel without coding. In binary modulation each channel symbol carries information of about one data bit and there is only one possible error which happens when added noise exceeds half of the distance between constellation points. Since our noise has normal distrubution, the probability of random variable achieving given value can be described by complementary error function or by the Q-function with the latter defined as follows

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \exp\left(-\frac{u^2}{2}\right) du \tag{4.1}$$

Amounf of noise that is introduced to the channel is described by γ_b.

$$\gamma_b = \frac{E_b}{N_0} = \frac{E_s}{N_0 \log_2 M} \tag{4.2}$$

where γ_b and γ_s show the ratio between noise and energy dedicated to bit or symbol. For BPSK modulation the energy per bit equal to energy per symbol and the probability of error is:

$$P_{berr} = Q(\sqrt{(2\gamma_s)}) \tag{4.3}$$

Following figure shows both predicted and simulated error probaility of BPSK modulation.

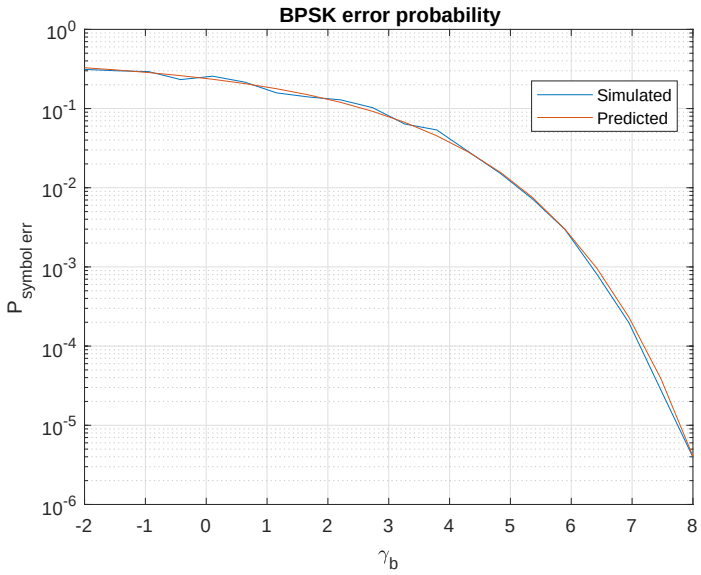


Figure 4.1: BPSK error probability

Now that we have confirmed that the basis works we can examine the error performance of various coding methods. 4,7 Hamming code described in previous chapters is a linear block code capable of correcting up to $t = 1$ errors. Probability of a block error for such a code is given as.

$$P_e = \sum_{j=t+1}^{n} \binom{n}{j} p^j (1-p)^{n-j} \tag{4.4}$$

Where p is the probability of bit error [Gol07]. Given the correcting capabilities of our code it is preferable to calculate the probability of error through the probability of correct detection which in our case is the sum of the probability of all bits being detected properly and the probability of one bit error

$$P_{blockerr} = 1 - P_c = 1 - ((1 - P_{biterr})^7 + 7(1 - P_{biterr})^6 P_{biterr}) \tag{4.5}$$

Following figure shows comparison of predicted and simulated block error rate of Hamming(7,4) code.

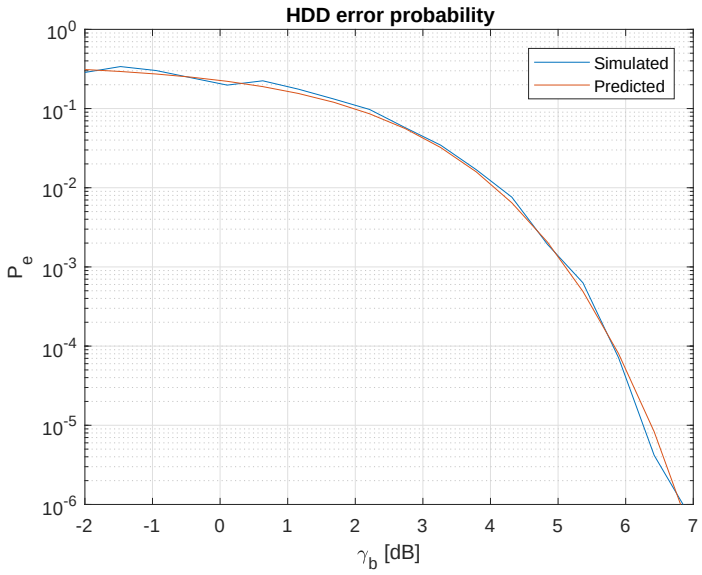


Figure 4.2: Block error probability for Hard decision decoding

The error rate can be improved by applying Soft decision decoding which should perform better than Hard decision decoding especially at higher signal to noise ratios. Following figure shows the effectivity of Soft and Hard decision decoding of linear block code compared with uncoded transmission.

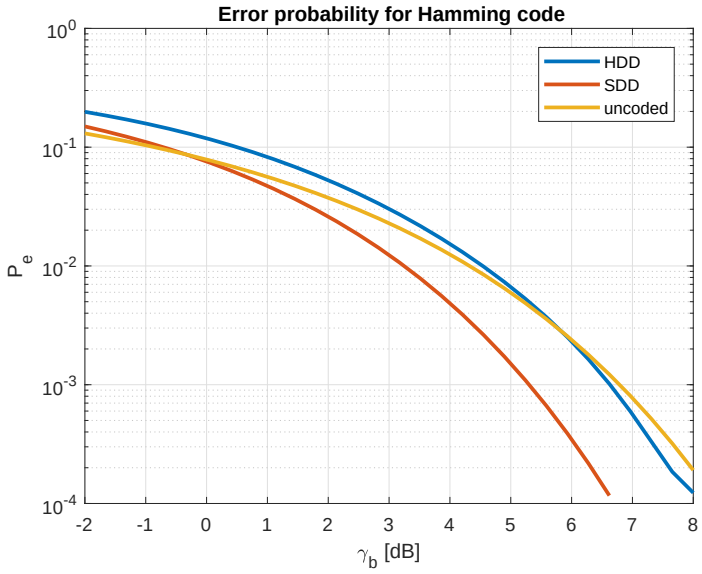


Figure 4.3: Hamming code bit error rates

We can see that at low signal-to-noise ratios, coding can even be detrimental as incorrect repairs introduce more errors. As the SNR increases, both coding

methods pull ahead of uncoded transmissin with Soft decision decoding being significantly more effective. The price for this improvement tends to lie in more complicated implementation.

Second implemented code was a convolutional code decoded by the Viterbi algorithm. Again, the decoding process was done in two versions. Once for hard decision decoding and once for Soft decision decoding. Following figure shows comparison of both methods together with uncoded transmission.

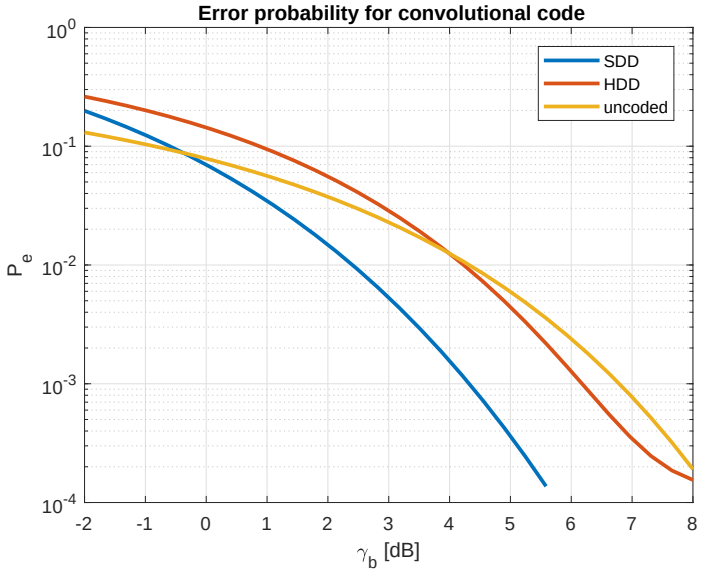


Figure 4.4: Convolutional code bit error rates

Besides BPSK, also 4 and 8 PSK modulations were implemented. These modulations, which were encoded into constellation space by Gray mapping (close neighbours are Hamming distance one apart), have bit error rates according to:

$$P_e = \frac{2}{\log_2 M} Q\left(\sqrt{2\gamma_b \log_2 M} \sin\left(\frac{\pi}{M}\right)\right) \tag{4.6}$$

Following figure shows bit error rates comparison for various M values depending on the γ_b.

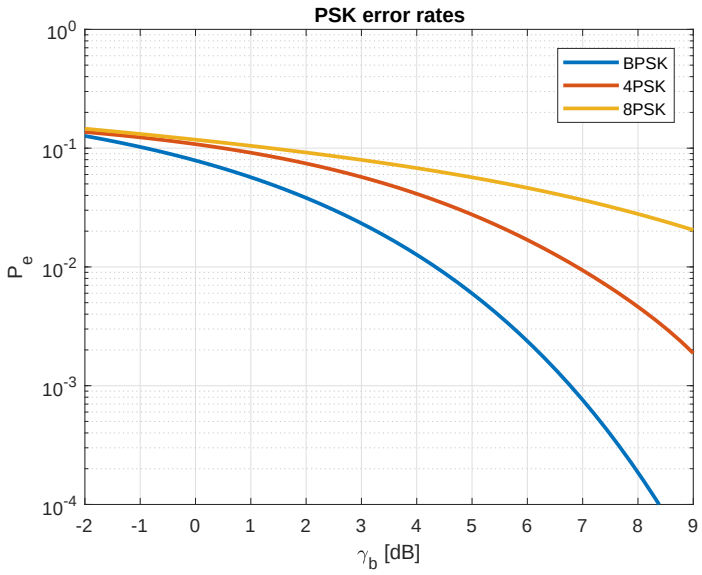


Figure 4.5: M-PSK bit error rates comparison

Soft decision decoding methods was also implemented for modulations which encode bits in groups of two.

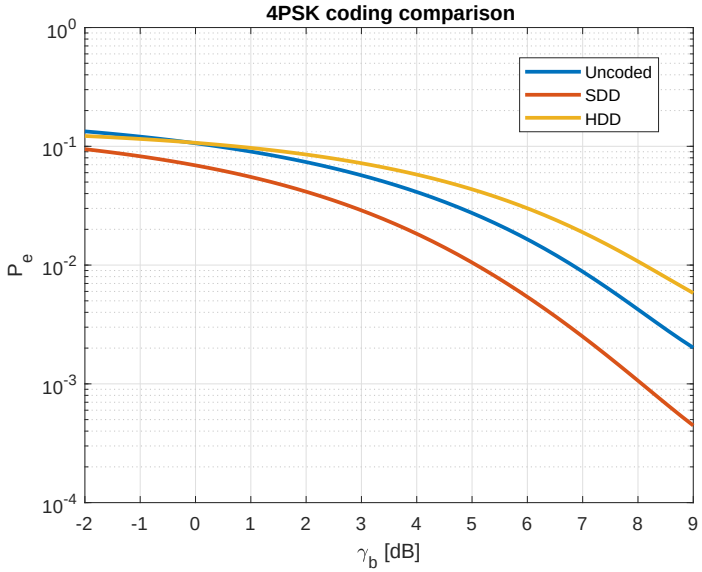


Figure 4.6: 4PSK convolutional code bit error rates

Chapter 5

Conclusion

This thesis was dedicated to creating coding and signal processing algorithms and distributing them where possible according to their role in the communication model into standalone blocks. From these blocks we have built several functioning models which allow us to examine the path that data takes and validate effectivity of built algorithms by comparing them with theoretically predicted performance. Chosen approach created versatile and highly customizable models which can be easily modified for given goal or expanded upon. Possible expansions include adding more coding or modulation methods or expansion for transmitting real signals instead of complex envelopes.

Bibliography

[Gol07] Andrea Goldsmith, *Wireless communications*, Posts & Telecom Press, 2007.

[HV94] Zdeněk Hrdina and František Vejražka, *Digitální radiová komunikace*, Vydavatelství ČVUT, 1994.

[Pro95] John G. Proakis, *Digital communications*, 3rd ed. ed., McGraw-Hill, c1995.

[Sý05] Jan Sýkora, *Teorie digitální komunikace*, dotisk 1. vyd. ed., Vydavatelství ČVUT, 2005.

[Sý16] ———, *Digital modulations and coding - lecture slides*, 2016.

Appendix A

CD contents

Scripts:
-envelope.m
-erroratеplotter.m
-Main.m
-Rx.m
-Tx.m

Functions:
-convo.m
-decoding.m
-HDist.m
-LTIch.m
-Mapper.m
-MatchedFiler.m
-metric.m
-Modulator.m
-Noise.m
-PerfCheck.m
-Periodo.m
-Pulse.m
-softdist.m
-Viterb.m
-Viterbs.m