

# Sparse Regression Codes for Networked Control Systems

By:

Edwin G. W. Peters  
egwp08@student.aau.dk

Group 1075

Signal Processing and Computing 9th and 10th semester

June 6, 2013





**AALBORG UNIVERSITY**  
STUDENT REPORT

**Department of  
Electronic Systems**  
Fredrik Bajers Vej 7  
DK-9220 Aalborg Øst  
Phone: +45 96 35 86 00  
Internet: es.aau.dk

Title:  
Sparse Regression Codes for  
Networked Control Systems

Theme:  
Master's thesis

Project period:  
September 2012 - June 2013

Project group:  
13gr1075

Members of the group:  
Edwin G. W. Peters

Supervisors:  
Jan Østergaard  
Daniel E. Quevedo

Number of copies: 4

Number of pages: 69

Attachments: CD

Appendices: 2

Project completed: June 6, 2013

**Abstract:**

We study a Networked Control System (NCS) architecture for Linear Time Invariant (LTI) plants, where an unreliable data rate limited network, with random Independent and Identically Distributed (IID) packet dropouts occurring, connects the plant and the controller. To achieve robustness of the control system with respect to IID dropouts, a receding horizon controller is used which minimizes a finite horizon cost function. To deal with rate limited networks, we, in this thesis, wish to design sparse packets using  $l_0$  penalized optimization. This is done on Sparse Regression Code (SPARC) dictionaries containing lattices or IID Gaussian samples.

We transmit the current and expected future control signals, such that on reception of the packet, the current signal as well as the next  $N - 1$  future control signals can be reconstructed in the plant. The distinguishing factors of this thesis regarding to other available studies is that we use a fixed rate vector quantizer based on SPARC, featuring a finite support which can be overloaded in case of heavy oscillations in the plant.

We design different SPARC dictionaries and simulate these in an NCS with different packet dropout rates on the network. Results show good performance at bit rates down to 2.75 bit/symbol with an IID packet dropout probability up to 0.20 when Gaussian IID SPARC dictionaries are used. The performance of a lattice SPARC dictionary is not able to reach these bit rates, and generally requires bit rates of 3.75 or 4 bit/symbol to stabilize the NCS.

Finally we state an alternative network model featuring two network states with different dropout probabilities. A different SPARC dictionary is designed for each network state. We stated equations to analyze whether the system with given transition and dropout probabilities is stable using Markov Jump Linear System (MJLS) theory. This is followed by simulations of NCSs featuring these networks.

Contents of this report is freely available,

but publication (with specification of source) may only be done upon arrangement with the authors.





**AALBORG UNIVERSITET**  
STUDENTERRAPPORT

**Institut for  
Elektroniske Systemer**  
Fredrik Bajers Vej 7  
9220 Aalborg Øst  
Telefon: 96 35 86 00  
Internet: es.aau.dk

**Titel:**

Sparse Regression Koder for  
Netværksbaserede  
Kontrolsystemer

**Tema:**

Kandidatspeciale

**Projektperiode:**

September 2012 - juni 2013

**Projektgruppe:**

13gr1075

**Medlemmer af gruppen:**

Edwin G. W. Peters

**Vejleder:**

Jan Østergaard

Daniel E. Quevedo

**Antal kopier:** 5

**Antal sider:** 69

**Bilag:** CD

**Appendikser:** 2

**Projekt afsluttet:** 6. juni, 2013

**Synopsis:**

I denne afhandling undersøges Netværksbaserede Control Systemers (NCS) architectureer for Lineare Tidsinvariante (LTI) systemer, hvor et upålideligt båndbredde begrænset netværk med random IID pakkeab, forbinder systemet (plant) og controlleren. For at gøre kontrolsystemet robust m.h.t. IID pakkeab, bruges en aftagende horizon controller som minimerer en finite horizon cost funktion. For at kunne håndtere båndbredde begrænsede netværk, designes der i denne specialeafhandling sparse control pakker ved brug af  $l_0$  penaliserede optimerings algoritmer på SPARC baserede ordbøger indeholdende IID Gaussiske vektorer eller latticer. Der sendes expectede fremtidige samt et nutidigt control signal til planten, således at  $N - 1$  fremtidige controlsignaler kan genskabes såfremt pakken modtages af planten. De faktorer der adskiller denne specialeafhandling fra andre bidrag er, at der bruges fixed rate vektor quantizere baseret på SPARC, der har et finitivt support og derved kan overloade quantizeren i tilfælde af at der sker oscillationer i planten. Der designes forskellige SPARC ordbøger, der simuleres i et NCS med forskellige pakkeabrate på netværket. Resultaterne viser en god ydelse for bitrate ned til 2.75 bit/symbol når sandsynligheden for IID pakkeab ikke overstiger 0.20. Dette ved brug af Gaussiske IID SPARC ordbøger. Lattice SPARC ordbøger opnår derimod ikke den samme ydelse ved disse bitrate, og kræver generelt bit rate på 3.75 eller 4 bit/symbol for at stabilisere et NCS. Til sidst opstilles der en netværksmodel indeholdende to states med hver deres pakkeabrate. Der designes forskellige SPARC ordbøger for hver state netværket kan antage. Der opstilles ligninger baseret på MJLS teori, således stabilitet for et system med givne transitions- og pakkeab sandsynligheder kan analyseres. Til sidst er systemer baseret på disse netværk blevet simuleret.



# Preface

---

This thesis is composed during the 9<sup>th</sup> and 10<sup>th</sup> semester during the Autumn of 2012 and the Spring of 2013. It is written by a student studying “Signal Processing and Computing” at Aalborg University in Denmark in cooperation with the University of Newcastle in Australia.

The target audience for this thesis are students, researchers and others with a background in vector quantization and signal compressing, optimization, NCSs design and others interested in these topics.

The citations in this thesis are referred by a number in square brackets, e.g. [1], with details to be found in the bibliography located on page 69. The abbreviations used in this thesis are explained in the Abbreviation list on page ix. All matrices are denoted with a bold capital letter, i.e.  $\mathbf{M}$  where  $\mathbf{M}^T$  denotes its transpose. Vectors are denoted with a bold lower case letter i.e.  $\mathbf{v}$ , and scalars in normal lower case i.e.  $u$ .  $\mathbf{I}$  denotes the identity matrix, containing 1’s on the diagonal.  $\text{diag}\{\mathbf{v}\}$  diagonalizes the vector  $\mathbf{v}$ . We define  $\|\mathbf{v}\|_P = \sqrt{\mathbf{v}^T \mathbf{P} \mathbf{v}}$  for a positive definite matrix  $P$  and  $\|\mathbf{v}\|_2 = \sqrt{\mathbf{v}^T \mathbf{v}}$ . The spectral radius of the matrix  $\mathbf{M}$  is denoted by  $r_\sigma(\mathbf{M})$ . The operator  $\mathbb{E}\{x\}$  defines the statistical expectation of a random variable  $x$ , whereas  $\text{var}\{x\}$  denotes its variance.

A CD is attached containing a digital version of this thesis and the scripts used for the simulations performed herein.

Finally, the author would like to thank Associate Professor Daniel Quevedo at the University of Newcastle and Associate Professor Jan Østergaard at Aalborg University for their support and guidance during the writing of this thesis.

---

Edwin G. W. Peters



# Contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	State of the art . . . . .	1
1.2	Problem description . . . . .	2
<b>2</b>	<b>Rate-distortion theory</b>	<b>5</b>
2.1	Rate-distortion . . . . .	5
2.2	Calculating the rate . . . . .	5
<b>3</b>	<b>Dictionary search</b>	<b>7</b>
3.1	Matching pursuit . . . . .	7
3.2	Homotopy . . . . .	10
3.3	Summary . . . . .	18
<b>4</b>	<b>Control theory</b>	<b>21</b>
4.1	Control and feedback . . . . .	21
4.2	Controller design . . . . .	23
4.3	Model predictive control . . . . .	25
4.4	Summary . . . . .	28
<b>5</b>	<b>Quantized PPC</b>	<b>29</b>
5.1	Packet Predictive Controller (PPC) with a quantizer . . . . .	29
5.2	Closed loop PPC . . . . .	30
5.3	Summary . . . . .	31
<b>6</b>	<b>Dictionary design</b>	<b>33</b>
6.1	Gaussian dictionary . . . . .	33
6.2	Gaussian dictionary with scaled refinements . . . . .	37
6.3	Lattice dictionary . . . . .	38
6.4	Dictionaries with variable scaling . . . . .	40
6.5	Summary . . . . .	45
<b>7</b>	<b>Simulations</b>	<b>47</b>
7.1	Dictionary scaling - fixed gain . . . . .	47
7.2	Dictionary scaling - variable gain . . . . .	49
7.3	Dropout probabilities . . . . .	51
7.4	Summary . . . . .	54
<b>8</b>	<b>Alternative dropout scenarios</b>	<b>55</b>
8.1	Scenario with two states . . . . .	55
8.2	Dictionary considerations . . . . .	56
8.3	Stability of Markov Jump Linear Systems . . . . .	56
8.4	Stability for MJLS with different dropout scenarios . . . . .	58
8.5	Verification . . . . .	60
8.6	Summary . . . . .	62

<b>9</b>	<b>Conclusions</b>	<b>63</b>
<b>10</b>	<b>Future research</b>	<b>65</b>
	<b>Bibliography</b>	<b>67</b>
	<b>Appendix</b>	<b>73</b>
<b>A</b>	<b>Lattice Gram matrices</b>	<b>73</b>
<b>B</b>	<b>Parameters for simulation</b>	<b>75</b>

# Abbreviations

---

<b>AWGN</b>	Additive White Gaussian Noise	<b>MSE</b>	Mean Squared Error
<b>BIBO</b>	Bounded Input Bounded Output	<b>MP</b>	Matching Pursuit
<b>BP</b>	Basis Pursuit	<b>MJLS</b>	Markov Jump Linear System
<b>ECDQ</b>	Entropy-Coded Dithered (lattice) Quantizer	<b>MSS</b>	Mean Square Stable
<b>IID</b>	Independent and Identically Distributed	<b>NCS</b>	Networked Control System
<b>IP</b>	Integer Programming	<b>OMP</b>	Orthogonal Matching Pursuit
<b>LTI</b>	Linear Time Invariant	<b>PPC</b>	Packet Predictive Controller
<b>LQR</b>	Linear Quadratic Regulator	<b>SNR</b>	Signal to Noise Ratio
<b>MPC</b>	Model Predictive Control	<b>SPARC</b>	Sparse Regression Code



# 1 Introduction

---

LTI control systems are today used in many different places, occasions and situations. These systems all have in common, that there is a feedback from the system to be controlled (plant) to the controller, maintaining the plant in a desired state. In some cases it might be desired to have the controller at another physical location than the plant, in which case a wired or wireless network connects these. This topology is called a NCS and can have many advantages, such as lower cost, higher reliability and easier maintenance, but other challenges arise, including bit rate limitations, random delays and breakdowns, which leave the plant in open loop operation and can have severe consequences depending on the situation.

This chapter describes recent research in NCS and relevant topics. Afterwards the problem considered in this thesis is described.

## 1.1 State of the art

NCSs with random IID dropouts occurring and packet delays have been introduced and analyzed in various studies [1, 2, 3, 4, 5, 6, 7, 8, 9]. General literature [10, 11, 12, 13] explains methods for Linear Quadratic Regulator (LQR) controller design and receding horizon controllers for linear plants as well as stability for those. The paper [6] shows stability results for cases where the maximum number of consecutive packet dropout is bounded, whereas [7] investigates NCS with bounded time-delays. [3, 4] analyzes mean square- and stochastic stability of NCS based on a Markov dropout where an unbounded number of consecutive packet dropouts can occur.

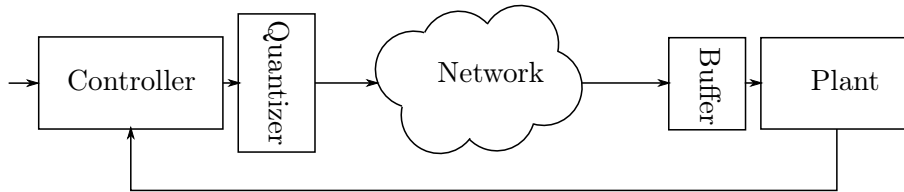
Quantization within the NCS has been done in e.g. [4, 5, 8], where the controller is forced to select the control vector from a finite constrained set of vectors using a nearest neighbor vector quantizer, reducing the bandwidth required on the network, and analyses the closed loop behavior of these. In [5] an Entropy-Coded Dithered (lattice) Quantizer (ECDQ) is used and closed loop stability is investigated using linear matrix inequalities based on MJLSs. Optimal rates for the entropy coder are calculated based on the statistics of the NCS. MJLS stability using ECDQ is also investigated in [4], where stochastic stability and bounds on the maximum packet dropout are investigated. The work [2] relates the PPC to problems solved in compressed sensing, and investigates sparse representation of the control vector using Orthogonal Matching Pursuit (OMP) as well as providing sufficient conditions for stability when the controller is used on a network with bounded packet dropouts.

Source coding using SPARC is introduced in [14] as codes to compress memoryless Gaussian sources over AWGN channels and reaches rate-distortion results close to the optimal rate-distortion function. In [15] a computationally efficient method is proposed for

SPARCs, which achieves performance close to the optimal rate-distortion function when used on IID Gaussian sources. The SPARC dictionary is constructed with the sections being interpreted as refinements of the previous sections using a scaling function. The dictionary search in [15] is performed using an algorithm closely resembled to Matching Pursuit (MP). In [16] the homotopy continuation algorithm is described to perform searches on Gaussian IID Dictionaries, which traces all solutions for Basis Pursuit (BP) as a function on the regularization parameter. The homotopy continuation algorithm in [16] can additionally be used to select a regularization parameter based on the desired sparsity of the representation of the signal.

## 1.2 Problem description

The setup of the problem to be considered in this thesis is shown in Figure 1.1. The controller and plant are connected through a network with random packet dropouts occurring, also called an NCS. The feedback path is throughout this thesis assumed to be stable with no packet dropouts occurring.



**Figure 1.1.** An illustration of the controller setup considered in this thesis.

The next state of the plant is described with the following function

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}_1 u_k + \mathbf{B}_2 \omega_k, \quad k \in \mathbb{N}_0, \quad (1.1)$$

with  $\mathbf{x}_k \in \mathbb{R}^m$  describing the plant state at time  $k$ ,  $\mathbf{A} \in \mathbb{R}^{m \times m}$  the system matrix of the plant,  $u \in \mathbb{R}$  the control signal or input,  $\mathbf{B}_1$  the input matrix and  $\mathbf{B}_2 \omega_n$  describing the noise in the plant. Although  $u_k$  can be an input vector, this thesis is limited to the scalar case.

Since there is a network connecting the controller and plant, there exists a probability for signals (or packets) to be delayed or lost. This has to be taken into account, such that the plant is kept stable in case it does not receive the control data, which has been described in e.g. [8, 10, 11, 12, 13], and other literature under the terms Receding Horizon Control and Model Predictive Control (MPC).

Since we in this thesis focus on bandwidth limited networks, it is of high interest that the size of the control packets is reduced, which can be done using vector quantization. The papers [2, 4, 5] investigated this using different quantizers and methods. This thesis contributes by applying SPARCs, presented in [14], on NCSs. The sparse regression codes are constructed as

$$\begin{array}{c}
 \leftarrow L \text{ columns} \rightarrow \leftarrow L \text{ columns} \rightarrow \quad \quad \quad \leftarrow L \text{ columns} \rightarrow \\
 \mathbf{\Gamma} = \left[ \begin{array}{ccc|ccc|ccc}
 \gamma_{1,1} & \cdots & \gamma_{1,N} & \gamma_{1,N+1} & \cdots & \gamma_{1,2N} & \cdots & \gamma_{1,(L-1)N+1} & \cdots & \gamma_{1,LN} \\
 \gamma_{2,1} & \cdots & \gamma_{2,N} & \gamma_{2,N+1} & \cdots & \gamma_{2,2N} & \cdots & \gamma_{2,(L-1)N+1} & \cdots & \gamma_{2,LN} \\
 \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & & \vdots & \ddots & \vdots \\
 \gamma_{M,1} & \cdots & \gamma_{M,N} & \gamma_{M,N+1} & \cdots & \gamma_{M,2N} & \cdots & \gamma_{M,(L-1)N+1} & \cdots & \gamma_{M,LN}
 \end{array} \right] \quad (1.2) \\
 \boldsymbol{\beta} = \left[ \begin{array}{ccc|ccc|ccc}
 0, \dots, 0, 1, 0 & 1, 0, 0, \dots, 0 & \cdots & 0, 0, \dots, 0, 1
 \end{array} \right], \quad (1.3)
 \end{array}$$

with  $\mathbf{\Gamma} \in \mathbb{R}^{N \times ML}$ , which is split up in  $M$  sections, each consisting of  $L$  code words, which can be used to generate a set  $\mathbb{W}$  of linear combinations. The vector  $\boldsymbol{\beta}$  being a  $ML \times 1$  vector, containing only one entry equaling 1 in every section  $m \in \{1, 2, \dots, M\}$ . These can be used to estimate a signal  $\mathbf{x}$  to compress

$$\hat{\mathbf{x}} = \mathbf{\Gamma}\boldsymbol{\beta}. \quad (1.4)$$

We need to find the closest code word  $\mathbf{\Gamma}\boldsymbol{\beta}$  over all  $\boldsymbol{\beta} \in \mathcal{B}$  to  $\mathbf{x} \in \mathbb{R}^N$ . Thus given a vector  $\mathbf{x}$  and some fixed code book  $\mathbf{\Gamma}$  and  $\mathcal{B}$ , we have to solve the minimization problem

$$\boldsymbol{\beta}^* = \arg \min_{\boldsymbol{\beta} \in \mathcal{B}} \|\mathbf{x} - \mathbf{\Gamma}\boldsymbol{\beta}\|^2. \quad (1.5)$$

The set  $\mathcal{B}$  is non-convex which means, that (1.5) is a non-convex optimization problem, requiring methods that are able to solve these.

In this thesis we will, based on the results in [4], present two methods to solve non-convex dictionary searches, and modify these to operate on (1.5) to compare the performance of these compared to traditional Gaussian dictionaries in Chapter 3. Chapter 4 describes the basics on quadratic control and receding horizon control, followed by the incorporation of the quantizer in Chapter 5. Chapter 6 describes different dictionaries, that can be used in the NCS followed by simulations in Chapter 7. The remaining chapters investigate different drop out scenarios in the network and stability criteria for these.





# 2 Rate-distortion theory

---

When using a lossy source coding, we are interested in keeping the bit rate, measured in bit/symbol, as low as possible, while also maintaining a distortion on the signal. This chapter briefly introduces the methods used to calculate the bit rate and distortion of the signals as well as the theoretic lower bounds on these. The measurement methods introduced in this chapter are used throughout the rest of the thesis.

## 2.1 Rate-distortion

When signals are quantized, we wish to reduce the bit rate of this signal. This introduces an error, since parts of the signals are lost. Some error in the signal can be allowed in many occasions, including NCS. The error, calculated by the difference between the original and quantized signal is called distortion, and can be calculated in different ways, such as Mean Squared Error (MSE) and Signal to Noise Ratio (SNR), which are used in this thesis. The MSE is calculated as [17]

$$D_{\text{MSE}} = \mathbb{E} \{ (\tilde{x}_i - x_i)^2 \}, \quad (2.1)$$

with  $\mathbb{E}$  being the expectation operator. The SNR is given by [17]

$$D_{\text{SNR}} = \frac{\mathbb{E} \{ (x_i)^2 \}}{D_{\text{MSE}}}. \quad (2.2)$$

The rate of a signal is often represented in the amount of bit needed to represent a signal, also denoted bit/symbol. In source coding it is often desired to minimize the bit rate for a given distortion, also called rate-distortion. There exist a theoretical lower bound for the rate that can be obtained for a given distortion, which for memoryless Gaussian sources is defined by [18]

$$R(D) = \frac{1}{2} \log_2 \left( \frac{\sigma^2}{D} \right), \quad (2.3)$$

with  $R$  denoting the rate,  $D$  denoting the distortion, and  $\sigma^2$  being the variance of the signal. Practical source coders result in distortion levels higher than the rate-distortion lower bound, since the algorithms often are sub-optimal [17].

Equation (2.3) can be rewritten to calculate the theoretical lower bound for the distortion at a given rate by

$$D(R) = \sigma^2 2^{-2R}. \quad (2.4)$$

## 2.2 Calculating the rate

In this thesis the dictionary and vectors are constructed as

$$\hat{\mathbf{x}} = \mathbf{\Gamma}\boldsymbol{\beta}, \tag{2.5}$$

where  $\mathbf{\Gamma} \in \mathbb{R}^{N \times ML}$  is the dictionary matrix containing  $M$  sections with  $L$  columns of Gaussian distributed samples with variance  $\sigma^2$ .  $\boldsymbol{\beta}$  is a  $ML \times 1$  binary vector containing only zeros and a single 1 in each section  $M$ . This results in the non-operational rate

$$R = M \frac{\log_2(L)}{N} \tag{2.6}$$

since the vector  $\boldsymbol{\beta}$  is coded as  $M$  binary numbers indicating which index equals 1. The operational rate is calculated by ceiling (2.6) to the nearest integer value.

# 3 Dictionary search

---

This chapter describes two different algorithms to perform a dictionary search to estimate a signal  $\mathbf{x}$  by a linear combination  $\hat{\mathbf{x}} = \mathbf{\Gamma}\boldsymbol{\beta}$ . These algorithms are described using Gaussian dictionaries, after which the algorithms are modified to perform a search on SPARCs containing Gaussian entries.

## 3.1 Matching pursuit

As described in Equation (1.5) in Chapter 1, we need to solve the optimization problem

$$\boldsymbol{\beta}^* = \arg \min_{\boldsymbol{\beta} \in \beta} \|\mathbf{x} - \mathbf{\Gamma}\boldsymbol{\beta}\|_2^2, \quad (3.1)$$

and find sparse solutions to this optimization problem. This can be enforced by penalizing  $\boldsymbol{\beta}$  with the  $l_0$ -norm enforcing a limited number of non-zero entries in  $\boldsymbol{\beta}$ , which results in the reformulated optimization problem

$$\boldsymbol{\beta}^* = \arg \min_{\boldsymbol{\beta} \in \beta} \|\mathbf{x} - \mathbf{\Gamma}\boldsymbol{\beta}\|_2^2 + \|\boldsymbol{\beta}\|_0. \quad (3.2)$$

This can be done by e.g. performing an exhaustive search, comparing all combinations of  $\mathbf{\Gamma}\boldsymbol{\beta}$  and picking the best solution. The complexity of an exhaustive search is  $O(L^N)$ , where  $L$  is the dictionary size, and  $N$  the number of elements to use. This makes it very time consuming to use an exhaustive search on larger dictionaries.

An alternative to performing an exhaustive search is MP, which performs the inner product between columns of the dictionary  $\mathbf{\Gamma}$  containing orthonormal rows, and  $\mathbf{x}$  after which it picks the solution containing the largest energy using the inner product. [19]

$$\boldsymbol{\beta} = \arg \max_{\gamma_i} |\langle \gamma_i, \mathbf{x} \rangle| \quad (3.3)$$

MP is a greedy algorithm finding the projection of the vector  $\mathbf{x}$  onto a dictionary  $\mathbf{\Gamma}$  with the maximal reduction in the residual. Afterwards the algorithm is repeated on the residual, until certain stopping criteria are reached. Afterwards, on the receiving side, the signal is reconstructed by a linear combination of the vectors selected. The complexity of the algorithm is reduced to  $O(LN)$ .

The general MP algorithm is shown in algorithm 3.1[20]. Here the dictionary index, maximizing the energy, is stored in  $g$  and the energy for each vector in  $\alpha$ . The result is subtracted from the residual. The algorithm can be stopped after a fixed amount of iterations, or after  $R$  is decreased sufficiently. The vector  $\boldsymbol{\beta}$  contains a 1 in every index  $g_k \in g$ , and  $\alpha_k$  contains the scaling of the vector. The scaling,  $\alpha_k$  has to be quantized and coded.

Figure 3.1 shows the performance of MP on a Gaussian dictionary. The signal  $\mathbf{x} \in \mathbb{R}^{20 \times 1}$  is quantized using the dictionary  $\mathbf{\Gamma} \in \mathbb{R}^{20 \times 3000}$ . The scaling factor  $\alpha$  has been

**Algorithm 3.1** The general MP algorithm

---

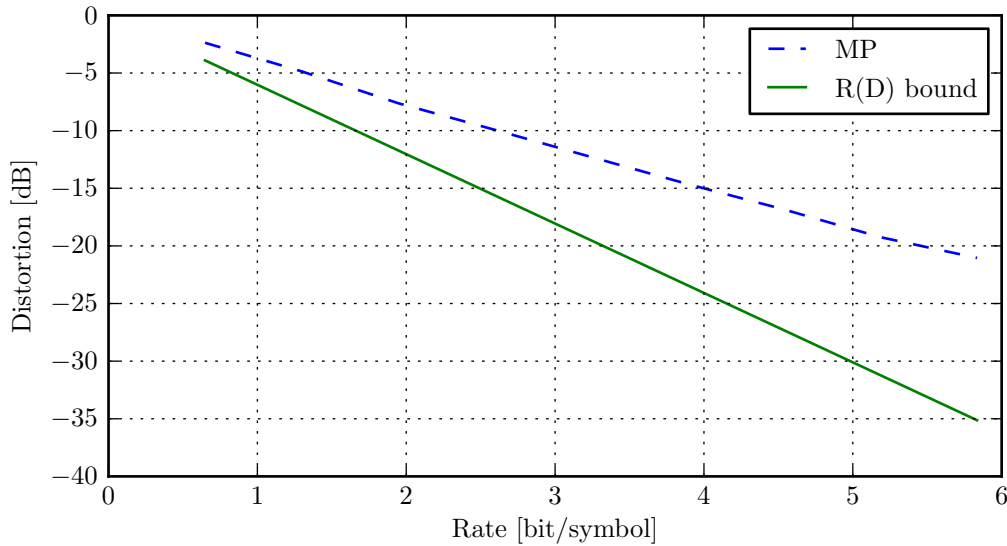
```

1: Dictionary  $\mathbf{\Gamma}$ 
2: Vector  $\boldsymbol{\beta} = \mathbf{0}$  ▷ Initialize with zeros
3:  $\mathbf{R}_0 \leftarrow \mathbf{x}$  ▷ Input signal
4: while stopping criteria not met do
5:    $g_k \leftarrow \arg \max_{\gamma_i} |\langle \mathbf{R}_k, \gamma_i \rangle|$ 
6:    $\alpha_k \leftarrow \langle \mathbf{R}_k, \gamma_{g_k} \rangle$ 
7:    $\mathbf{R}_{k+1} \leftarrow \mathbf{R}_k - \alpha_k \gamma_{g_k}$ 
8:    $\beta_{g_k} = 1$ 
9:   Increase  $k$ 
10: end while

```

---

limited to be in the set  $\{-2, -1.5, -1, -0.5, 0.5, 1, 1.5, 2\}$ , resulting in 3 bits of storage for every vector  $\gamma_i \in \mathbf{\Gamma}$  used. This results in 3 bits for the gain and  $\log_2 \left( \frac{M}{N} \right)$  indicating the index in  $\boldsymbol{\beta}$ , for every iteration  $k$  in algorithm 3.1. This plot is created running MP for  $k = \{1, \dots, 10\}$ , and calculate the rate afterwards.



**Figure 3.1.** The performance of MP on a Gaussian IID source compared to the rate-distortion bound.

Code: `mplots.py`.

### 3.1.1 Matching Pursuit on SPARC

When considering a SPARC dictionary, we are only interested in the index in the dictionary such that  $\boldsymbol{\beta}$  only contains a single 1 in each section  $m \in M$ . The gain,  $\alpha$ , is not used. For this reason, the MP algorithm has to be modified, such that the vector in the dictionary maximizes the inner product directly instead of the energy, such that the residual is minimized the most, without any scaling. The problem is rewritten as

$$\boldsymbol{\beta} = \arg \max_{\mathbf{a}_i} \langle \gamma_i, \mathbf{x} \rangle, \quad (3.4)$$

such that only linear combination with a positive result are chosen, resulting in the residual

$$\mathbf{R}_{k+1} = \mathbf{R}_k - \gamma_i, \quad (3.5)$$

with  $\mathbf{R}_0 = x$ .

The signal is constructed from the linear combination of the  $M$  sections in the dictionary  $\mathbf{\Gamma}$ . This means, that the linear combination must contain the same energy as the signal  $x$  for a perfect reconstruction. For this reason, the variance of the linear combination of  $\mathbf{\Gamma}$  must match the variance of the signal  $x$ , such that

$$\sigma_x^2 \approx M\sigma_a^2, \quad (3.6)$$

resulting in the scaling

$$c_k = \sqrt{\frac{1}{M}}. \quad (3.7)$$

The algorithm for performing MP on a regression dictionary, which is split into  $M$  sections with each the  $L$  vectors is shown in algorithm 3.2.

---

**Algorithm 3.2** The MP algorithm modified to work on sparse regression codes.

---

- 1: Dictionary  $\mathbf{\Gamma} \sim \mathcal{N}(0, \frac{1}{M}\sigma^2)$
  - 2: Vector  $\boldsymbol{\beta} = \mathbf{0}$  ▷ Initialize with zeros
  - 3:  $\mathbf{R}_0 \leftarrow \mathbf{x}$  ▷ Input signal
  - 4: **for**  $m = 1 \rightarrow M$  **do**
  - 5:      $g_m \leftarrow \arg \max \langle \mathbf{R}_k, \gamma_i \rangle \quad k \in \{(m-1)L + 1, \dots, mL\}$
  - 6:      $\mathbf{R}_{m+1} \leftarrow \mathbf{R}_m - \gamma_{g_m}$
  - 7:      $\beta_{g_m} = 1$
  - 8: **end for**
- 

Figure 3.2 shows MP performed on a regression dictionary, split up into  $M$  sections with  $M = \{1, 10, \dots, 300\}$  and the  $L = M$  columns per section. The vector length  $N$  is calculated to fit the desired rate such that

$$N = \text{round} \left( M \frac{\log 2(L)}{R} \right), \quad (3.8)$$

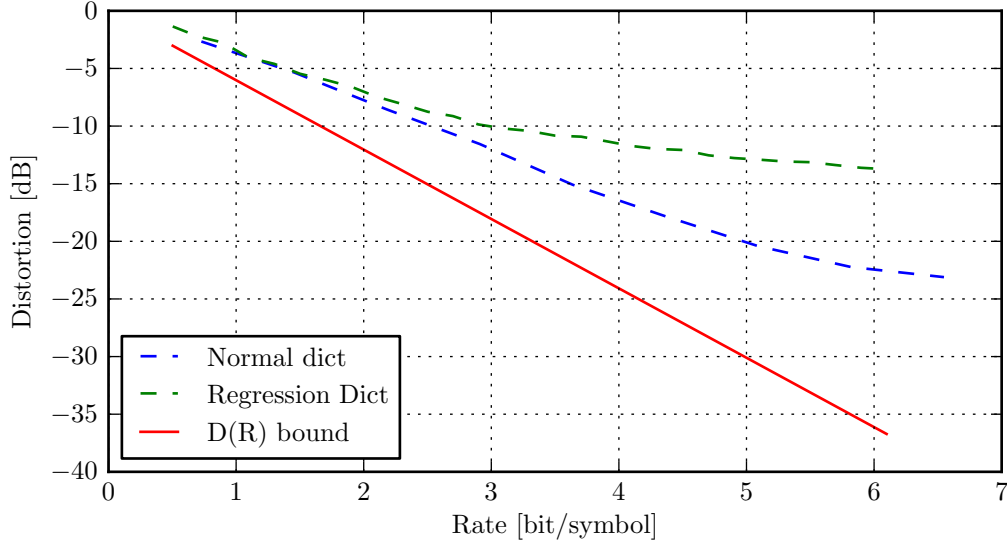
with the dictionary being Gaussian and scaled according to (3.6). The regular MP is performed on a orthonormal Gaussian dictionary as in algorithm 3.1.

When using lower bit rates, the performance is very similar until the bit rate reaches 2 bit/symbol, where the normal dictionary has a clear advantage. This is mostly due to the scaling  $\alpha$ , which is fixed to  $\sqrt{\frac{1}{M}}$  when using the regression dictionary.

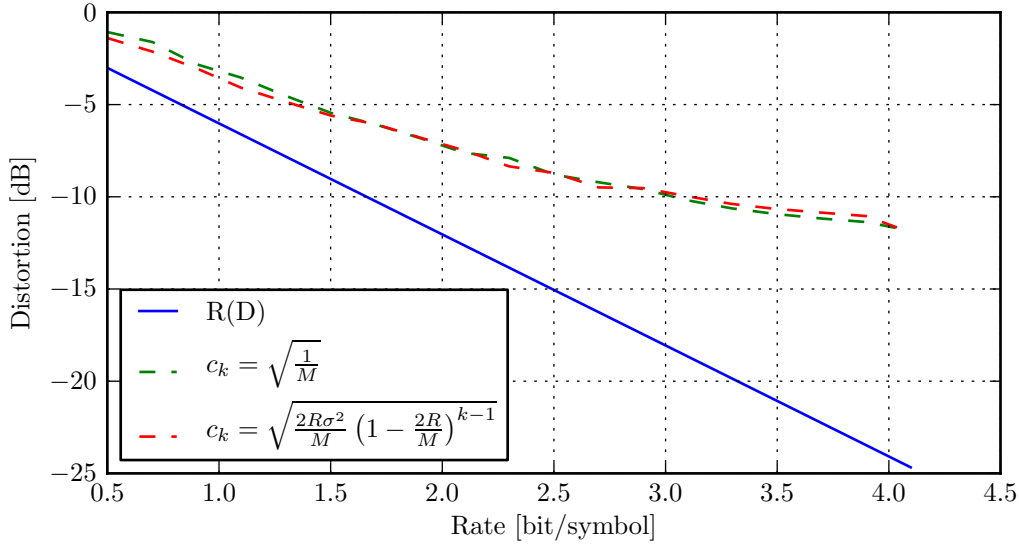
In [15], the authors found the scaling factor

$$c_k = \sqrt{\frac{2R\sigma^2}{M} \left(1 - \frac{2R}{M}\right)^{k-1}} \quad k = 1, \dots, M, \quad (3.9)$$

with  $R$  denoting the rate and  $M$  denoting the number of sections in  $\mathbf{\Gamma}$ . Figure 3.3 shows  $c_k$  from (3.6) and (3.9) compared. It shows that there is only a slight difference in performance when comparing these scaling factors when using Gaussian dictionaries and signals.



**Figure 3.2.** MP performed on a gaussian dictionary and on sparse regression codes.  
Code: `mprogression.py`.



**Figure 3.3.** Scaling factor  $c_k$  compared from equations (3.6) and (3.9).  
Code: `testspark.py`.

## 3.2 Homotopy

An alternative to matching pursuit is to reformulate the  $l_0$  relaxation problem defined in Equation (3.2)

$$\min_{\beta} \|\mathbf{x} - \mathbf{\Gamma}\beta\|_2^2 + \|\beta\|_0 \quad (3.10)$$

to

$$\min_{\beta} \max_{\lambda} \|\mathbf{x} - \mathbf{\Gamma}\beta\|_2^2 + \lambda \|\beta\|_1, \quad (3.11)$$

where  $\mathbf{\Gamma} \in \mathbb{R}^{N \times M}$ ,  $N \leq M$  and  $\lambda \geq 0$  [16]. This noisy version allows for a larger residual between the original data  $\mathbf{x}$  and the estimate  $\mathbf{\Gamma}\beta$  while requiring  $\beta$  to be more sparse.

The regularization parameter  $\lambda$  forces an amount of entries in  $\boldsymbol{\beta}$  to zero. The higher  $\lambda$  is, the sparser is  $\boldsymbol{\beta}$ .

When  $\lambda$  is fixed, the problem reduces to a quadratic programming problem, which can be solved analytically. The problem is to choose the parameter  $\lambda$  resulting in the desired sparsity in  $\boldsymbol{\beta}$  while reducing the noise [16]. The function is not differentiable whenever an entry  $\beta_i$  becomes zero since [16]

$$\|\boldsymbol{\beta}\|_1 = \sum_i |\beta_i|. \quad (3.12)$$

[16] proposes a method for solving this problem by solving a local problem for a limited range of  $\lambda$  whereafter the local solutions are set together to obtain solutions for all regions of  $\lambda$ .

### 3.2.1 The optimization problem

We define the cost-function

$$J(\boldsymbol{\beta}^*, \tilde{\lambda}) = \|\mathbf{x} - \mathbf{\Gamma}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1. \quad (3.13)$$

To find the solution minimizing this cost function, it is desired to find its gradient and set it equal to zero. Since  $\|\boldsymbol{\beta}\|_1$  is not a smooth function, it is not possible to differentiate it right away. It is, however possible to differentiate the cost function by using sub-differentials.

The vector  $\mathbf{v}$  is a sub-gradient of the function  $f$  at point  $\beta_0$  if [21]

$$f(\boldsymbol{\beta}) \geq f(\boldsymbol{\beta}_0) + \mathbf{v}(\boldsymbol{\beta} - \boldsymbol{\beta}_0) \quad (3.14)$$

is valid for any  $\boldsymbol{\beta}$ . The set of all these subgradients of  $f$  at  $\boldsymbol{\beta}_0$  is called the subdifferential of  $f$  at  $\boldsymbol{\beta}_0$ . In case the function  $f$  is convex and differential at a point  $\boldsymbol{\beta}$ , the subdifferential only consists of the gradient of  $f$  [21]. The subdifferential of  $|\beta|$  is given as the interval  $[-1, 1]$  if  $\beta = 0$ , and as  $-1$  or  $+1$  if  $\beta$  is less than or larger than 0 respectively. This gives the following expression for the sub-gradient of the  $l_1$  norm: [16]

$$\mathbf{u}(\boldsymbol{\beta}) \triangleq \partial\|\boldsymbol{\beta}\|_1 = \left\{ \mathbf{u} \in \mathbb{R}^N \left| \begin{array}{ll} u_i = 1 & \text{if } \beta_i > 0 \\ u_i = -1 & \text{if } \beta_i < 0 \\ u_i \in [-1 \cdots +1] & \text{if } \beta_i = 0 \end{array} \right. \right\} \quad (3.15)$$

If we fix  $\lambda$  as  $\tilde{\lambda}$ , the subdifferential of  $J(\boldsymbol{\beta}, \tilde{\lambda})$  becomes

$$\frac{\partial}{\partial \boldsymbol{\beta}} J(\boldsymbol{\beta}, \tilde{\lambda}) = 2\mathbf{\Gamma}^T(\mathbf{\Gamma}\boldsymbol{\beta} - \mathbf{x}) + \tilde{\lambda}\mathbf{u}(\boldsymbol{\beta}). \quad (3.16)$$

Setting the gradient to 0 for a  $\tilde{\boldsymbol{\beta}}$  that minimizes  $J(\boldsymbol{\beta}, \tilde{\lambda})$ , results in the solution

$$2\mathbf{\Gamma}^T\mathbf{\Gamma}\tilde{\boldsymbol{\beta}} + \tilde{\lambda}\tilde{\mathbf{u}} = 2\mathbf{\Gamma}^T\mathbf{x} \Leftrightarrow \mathbf{G}\tilde{\boldsymbol{\beta}} + \tilde{\lambda}\tilde{\mathbf{u}} = \mathbf{z} \quad (3.17)$$

for some  $\tilde{\mathbf{u}} \in \mathbf{u}(\tilde{\boldsymbol{\beta}})$ , where  $\mathbf{G} = 2\mathbf{\Gamma}^T\mathbf{\Gamma}$  and  $\mathbf{z} = 2\mathbf{\Gamma}^T\mathbf{x}$ . We now define a set  $I_{\text{on}}$  containing the support of  $\tilde{\boldsymbol{\beta}}$  for  $\tilde{\beta}_i \neq 0$ , whereas  $I_{\text{off}}$  contains the indexes where  $\beta_i$  equals 0 [16]. This makes  $\tilde{\boldsymbol{\beta}} = [\tilde{\boldsymbol{\beta}}_{\text{on}}^T, \tilde{\boldsymbol{\beta}}_{\text{off}}^T]^T$ . We similarly split the matrix  $\mathbf{G}$  and vector  $\mathbf{z}$  into

$$\mathbf{G}\tilde{\boldsymbol{\beta}} + \tilde{\lambda}\tilde{\mathbf{u}} = \mathbf{z} \Leftrightarrow \begin{bmatrix} \boldsymbol{\Phi} & \boldsymbol{\Psi} \\ \boldsymbol{\Psi}^T & \boldsymbol{\Upsilon} \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{\beta}}_{\text{on}} \\ \tilde{\mathbf{0}} \end{bmatrix} + \tilde{\lambda} \begin{bmatrix} \tilde{\mathbf{u}}_{\text{on}} \\ \tilde{\mathbf{u}}_{\text{off}} \end{bmatrix} = \begin{bmatrix} \mathbf{z}_{\text{on}} \\ \mathbf{z}_{\text{off}} \end{bmatrix}, \quad (3.18)$$

and write the following linear equations from this [16]

$$\boldsymbol{\Phi}\tilde{\boldsymbol{\beta}}_{\text{on}} + \tilde{\lambda}\tilde{\mathbf{u}}_{\text{on}} = \mathbf{z}_{\text{on}} \quad (3.19)$$

$$\boldsymbol{\Psi}^T\tilde{\boldsymbol{\beta}}_{\text{on}} + \tilde{\lambda}\tilde{\mathbf{u}}_{\text{off}} = \mathbf{z}_{\text{off}}. \quad (3.20)$$

We can neglect the right side of the matrix  $\mathbf{G}$ , since  $\tilde{\mathbf{x}}_{\text{off}}$  only contains zeros. The above equations can be solved for  $\tilde{\mathbf{x}}_{\text{on}}$  and  $\tilde{\mathbf{u}}_{\text{off}}$

$$\tilde{\boldsymbol{\beta}}_{\text{on}} = \boldsymbol{\Phi}^{-1} \left( \mathbf{z}_{\text{on}} - \tilde{\lambda}\tilde{\mathbf{u}}_{\text{on}} \right) \quad (3.21)$$

$$\tilde{\mathbf{u}}_{\text{off}} = \frac{1}{\tilde{\lambda}} \left( \mathbf{z}_{\text{off}} - \boldsymbol{\Psi}^T\tilde{\boldsymbol{\beta}}_{\text{on}} \right). \quad (3.22)$$

We start the algorithm by letting  $\tilde{\lambda} = \lambda_0$  begin at  $\infty$ , forcing all entries in  $\tilde{\boldsymbol{\beta}}$  to zero. While decreasing  $\lambda$ , entries in  $\tilde{\boldsymbol{\beta}}$  become non-zero one by one. Practically, we take the supremum of  $2\boldsymbol{\Gamma}^T\mathbf{x}$ , resulting in the  $\lambda_0$ , for which all larger values of  $\lambda$  will result in all  $\beta_i \in \tilde{\boldsymbol{\beta}}$  equaling zero.

$\tilde{\boldsymbol{\beta}}_{\text{on}}$  and  $\tilde{\mathbf{u}}_{\text{on}}$  are calculated. All entries in  $\tilde{\boldsymbol{\beta}}_{\text{on}}$  will be non-zero, and the vector  $\boldsymbol{\beta}$  is obtained, by putting the entries from  $\tilde{\boldsymbol{\beta}}_{\text{on}}$  in the entries in  $\boldsymbol{\beta}$ , where  $I_{\text{on}}$  contains the indexes in  $\boldsymbol{\beta}$ . [16] Afterwards the support for  $\boldsymbol{\beta}$  has to be recalculated. This is done by finding at which  $\lambda$  an entry in  $\boldsymbol{\beta}$  is forced to zero or to a non-zero value by solving (3.21) for  $\tilde{\lambda}$  [16]

$$\lambda_{\boldsymbol{\beta}} = \text{diag} \{ \tilde{\mathbf{u}}_{\text{on}} \}^{-1} \mathbf{z}_{\text{on}} \quad (3.23)$$

and equation (3.22) for  $\lambda$  where  $\mathbf{u}_{\text{off}}$  is set to  $\pm 1$ .

$$\lambda_{\mathbf{u}_{\text{pos}}} = \text{diag} \{ \mathbf{I} - \boldsymbol{\Psi}^T\boldsymbol{\Phi}^{-1}\tilde{\mathbf{u}}_{\text{on}}^T \}^{-1} (\mathbf{z}_{\text{off}} - \boldsymbol{\Psi}^T\boldsymbol{\Phi}^{-1}\mathbf{z}_{\text{on}}) \quad (3.24)$$

$$\lambda_{\mathbf{u}_{\text{neg}}} = \text{diag} \{ -\mathbf{I} - \boldsymbol{\Psi}^T\boldsymbol{\Phi}^{-1}\tilde{\mathbf{u}}_{\text{on}}^T \}^{-1} (\mathbf{z}_{\text{off}} - \boldsymbol{\Psi}^T\boldsymbol{\Phi}^{-1}\mathbf{z}_{\text{on}}), \quad (3.25)$$

where  $\mathbf{I}$  is the identity matrix.

We are interested in finding the largest  $\lambda$  lower than the current  $\lambda_0$  forcing either one component of  $\boldsymbol{\beta}$  to zero or to a non-zero value. This is done by discarding all values of  $\lambda$ , obtained in equations (3.23), (3.24) and (3.25), that are larger than or equal to  $\lambda_0$ , whereafter the largest value among the  $\lambda$ 's in equations (3.23), (3.24) and (3.25) is found, and the corresponding index is moved from  $I_{\text{on}}$  to  $I_{\text{off}}$  depending on whether the index in  $\boldsymbol{\beta}$  is forced to one or zero. The corresponding value in  $\mathbf{u}$  is set to  $\pm 1$  or zero as well. [16] This procedure is repeated either until a certain amount of indexes in  $\boldsymbol{\beta}$  become non-zero, a certain tolerance on  $\|\mathbf{x} - \boldsymbol{\Gamma}\boldsymbol{\beta}\|_2^2$  is reached, or  $\lambda$  reaches zero.



**Algorithm 3.3** Homotopy algorithm.

---

```

1:  $\mathbf{G} \leftarrow 2\mathbf{\Gamma}^T\mathbf{\Gamma}$  ▷ Init  $\mathbf{G}$ 
2:  $\mathbf{z} \leftarrow 2\mathbf{\Gamma}^T\mathbf{x}$  ▷ Init  $\mathbf{z}$ 
3:  $\lambda_0 \leftarrow \max_z |\mathbf{z}|$  ▷ Init  $\lambda_0$  with the infinity norm of  $\mathbf{z}$ 
4:  $\mathbf{I}_{\text{on}} \leftarrow \arg \max_z |\mathbf{z}|$  ▷ Set the index of  $\beta$  to be active
5:  $\mathbf{u} = \text{sign}(\mathbf{z})$  ▷ Set  $\mathbf{u}$  to correspond to the sign of the initial  $\mathbf{z}$ 
6: while not converged do
7:    $\mathbf{\Phi} \leftarrow \mathbf{G}_{\mathbf{I}_{\text{on}}, \mathbf{I}_{\text{on}}}$  ▷ Move active indexes from  $\mathbf{G}$  to  $\mathbf{\Phi}$  (eq. (3.18))
8:    $\mathbf{\Psi}^T \leftarrow \mathbf{G}_{\mathbf{I}_{\text{off}}, \mathbf{I}_{\text{on}}}$  ▷ Move indexes corresponding to eq. (3.18) to  $\mathbf{\Psi}^T$ 
9:    $\tilde{\beta}_{\text{on}} \leftarrow \mathbf{\Phi}^{-1}(\mathbf{z}_{\text{on}} - \lambda_0 \tilde{\mathbf{u}}_{\text{on}})$  ▷ Calculate  $\tilde{\beta}_{\text{on}}$  as eq. (3.21)
10:   $\tilde{\mathbf{u}}_{\text{off}} \leftarrow \frac{1}{\lambda_0}(\mathbf{z}_{\text{off}} - \mathbf{\Psi}^T \tilde{\beta}_{\text{on}})$  ▷ Calculate  $\tilde{\mathbf{u}}_{\text{off}}$  as eq. (3.22)
11:  Verify  $\tilde{\mathbf{u}}_{\text{off}} \in [-1, 1]$ 
12:   $\lambda_{\text{upos}} \leftarrow \text{diag}\{\mathbf{I} - \mathbf{\Psi}^T \mathbf{\Phi}^{-1} \tilde{\mathbf{u}}_{\text{on}}^T\}^{-1}(\mathbf{z}_{\text{off}} - \mathbf{\Psi}^T \mathbf{\Phi}^{-1} \mathbf{z}_{\text{on}})$  ▷  $\lambda$  forcing  $\beta$  positive
13:   $\lambda_{\text{uneg}} \leftarrow \text{diag}\{-\mathbf{I} - \mathbf{\Psi}^T \mathbf{\Phi}^{-1} \tilde{\mathbf{u}}_{\text{on}}^T\}^{-1}(\mathbf{z}_{\text{off}} - \mathbf{\Psi}^T \mathbf{\Phi}^{-1} \mathbf{z}_{\text{on}})$  ▷  $\lambda$  forcing  $\beta$  negative
14:   $\lambda_{\beta} \leftarrow \text{diag}\{\mathbf{z}_{\text{on}}^{-1}\} \tilde{\mathbf{u}}_{\text{on}}$  ▷  $\lambda$  forcing  $\beta$  to zero
15:   $\lambda_{\text{new}} \leftarrow \max_{\lambda} \left\{ \lambda_{\text{upos}} < \lambda_0, \lambda_{\text{uneg}} < \lambda_0, \lambda_{\beta} < \lambda_0 \right\}$  ▷ Select highest  $\lambda$  lower than  $\lambda_0$ 
16:  if  $\lambda_{\text{new}} \in \max_{\lambda} \lambda_{\beta}$  then ▷ Activate or deactivate indexes in  $\beta$ 
17:    Move index from  $\mathbf{I}_{\text{on}}$  to  $\mathbf{I}_{\text{off}}$ 
18:    Set corresponding index in  $\tilde{\mathbf{u}} = 0$ 
19:  end if
20:  if  $\lambda_{\text{new}} \in \max_{\lambda} \lambda_{\text{upos}}$  then
21:    Move index from  $\mathbf{I}_{\text{off}}$  to  $\mathbf{I}_{\text{on}}$ 
22:    Set corresponding index in  $\tilde{\mathbf{u}} = 1$ 
23:  end if
24:  if  $\lambda_{\text{new}} \in \max_{\lambda} \lambda_{\text{uneg}}$  then
25:    Move index from  $\mathbf{I}_{\text{off}}$  to  $\mathbf{I}_{\text{on}}$ 
26:    Set corresponding index in  $\tilde{\mathbf{u}} = -1$ 
27:  end if
28:   $\lambda_0 \leftarrow \lambda_{\text{new}}$ 
29: end while

```

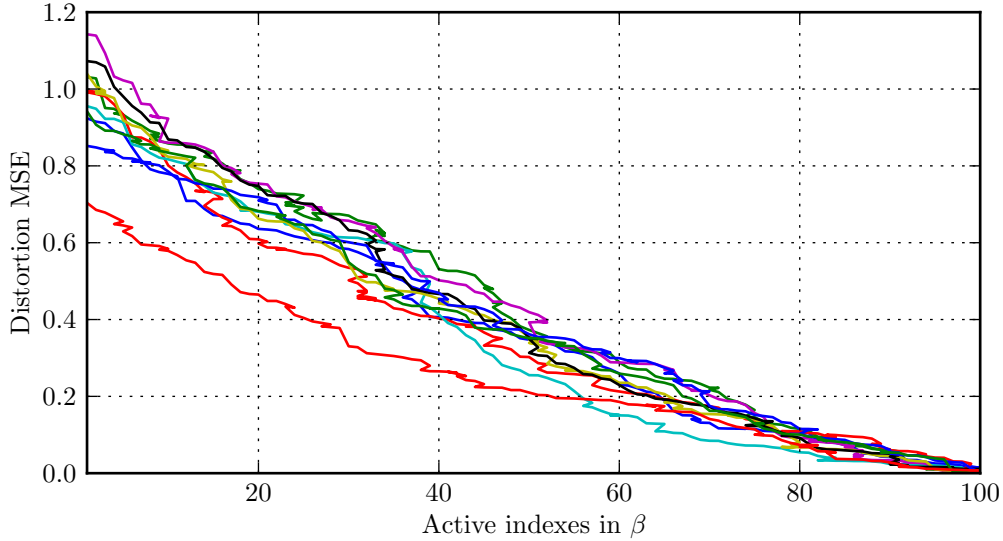
---

### 3.2.2 Algorithm

Algorithm 3.3 shows the algorithm for the homotopy implementation. The algorithm has to stop when it is not possible to find a value of  $\lambda$  that is lower than the current  $\lambda_0$ . It can additionally be stopped when the algorithm is considered converged, such that the norm of the residual of  $\mathbf{x} - \mathbf{\Gamma}\beta$  is below a certain threshold, or the length of  $\beta_{\text{on}}$  has reached a certain value.

### 3.2.3 Verification

The implementation of the homotopy continuation is verified by constructing a dictionary  $\Gamma \in \mathbb{R}^{M \times N}$  with  $M = 100$  and  $N = 500$  of random Gaussian distributed numbers and  $\mathbf{x} \in \mathbb{R}^M$ . The algorithm runs until  $\lambda$  reaches zero. The plots show the MSE compared to the number of non-zero indexes in  $v\beta$ .



**Figure 3.4.** Convergence paths for 10 simulations of the homotopy algorithm.  
Code: `homotopy-verification.py`.

Figure 3.4 shows the convergence of 10 simulations of the homotopy algorithm. It is clearly seen, that the algorithm activates and deactivates different  $\beta_i$  while converging.

### 3.2.4 Homotopy with a binary solution

Since we are interested in not only a sparse representation of  $\beta$  but also want  $\beta$  to be binary, containing only one or zero, we have modified the optimization problem to:

$$\begin{aligned} \min_{\beta} \quad & \|\mathbf{x} - \Gamma\beta\|_2^2 & (3.26) \\ \text{subject to} \quad & \|\beta\|_0 \leq L \\ & \beta_i = 1 \text{ for all } \beta_i \neq 0, \end{aligned}$$

such that all non-zero  $\beta \in \beta$  only contain 1. This results in a mixed integer programming problem, which is non-convex. The cost function can be relaxed into two convex sub-problems that approximate (3.26) and is formulated as

$$\begin{aligned} \mathcal{L}(\boldsymbol{\beta}^*, \lambda^*, \boldsymbol{\mu}^*) &= \mathbf{x}^T \mathbf{x} + \frac{1}{2} \begin{bmatrix} \beta_{\text{on}} & \beta_{\text{off}} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Phi} & \boldsymbol{\Psi} \\ \boldsymbol{\Psi}^T & \boldsymbol{\Upsilon} \end{bmatrix} \begin{bmatrix} \beta_{\text{on}} \\ \beta_{\text{off}} \end{bmatrix} + \lambda \left\| \begin{bmatrix} \beta_{\text{on}} \\ \beta_{\text{off}} \end{bmatrix} \right\|_1 \dots \\ &\quad + \begin{bmatrix} \boldsymbol{\mu}_{\text{on}} & \boldsymbol{\mu}_{\text{off}} \end{bmatrix} \begin{bmatrix} \beta_{\text{on}} - \mathbf{1} \\ \beta_{\text{off}} \end{bmatrix} - \begin{bmatrix} \beta_{\text{on}} & \beta_{\text{off}} \end{bmatrix} \begin{bmatrix} \mathbf{z}_{\text{on}} \\ \mathbf{z}_{\text{off}} \end{bmatrix}, \end{aligned} \quad (3.27)$$

with

$$\mathbf{G} = \begin{bmatrix} \boldsymbol{\Phi} & \boldsymbol{\Psi} \\ \boldsymbol{\Psi}^T & \boldsymbol{\Upsilon} \end{bmatrix} = 2\boldsymbol{\Gamma}^T \boldsymbol{\Gamma} \quad (3.28)$$

and

$$\mathbf{z} = \begin{bmatrix} \mathbf{z}_{\text{on}} \\ \mathbf{z}_{\text{off}} \end{bmatrix} = 2\boldsymbol{\Gamma}^T \mathbf{x}. \quad (3.29)$$

The gradient for som fixed  $\tilde{\lambda}$  is defined by

$$\frac{d}{d\boldsymbol{\beta}} \mathcal{L}(\boldsymbol{\beta}^*, \lambda^*, \boldsymbol{\mu}^*) = \begin{bmatrix} \boldsymbol{\Phi} & \boldsymbol{\Psi} \\ \boldsymbol{\Psi}^T & \boldsymbol{\Upsilon} \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{\beta}}_{\text{on}} \\ \tilde{\mathbf{0}} \end{bmatrix} + \tilde{\lambda} \begin{bmatrix} \tilde{\mathbf{u}}_{\text{on}} \\ \tilde{\mathbf{u}}_{\text{off}} \end{bmatrix} + \begin{bmatrix} \tilde{\boldsymbol{\mu}}_{\text{on}} \\ \mathbf{0} \end{bmatrix} - \begin{bmatrix} \mathbf{z}_{\text{on}} \\ \mathbf{z}_{\text{off}} \end{bmatrix}, \quad (3.30)$$

resulting in the linear equations given by

$$\boldsymbol{\Phi} \tilde{\boldsymbol{\beta}}_{\text{on}} + \tilde{\lambda} \tilde{\mathbf{u}}_{\text{on}} + \tilde{\boldsymbol{\mu}}_{\text{on}} = \mathbf{z}_{\text{on}} \quad (3.31)$$

$$\boldsymbol{\Psi}^T \tilde{\boldsymbol{\beta}}_{\text{on}} + \tilde{\lambda} \tilde{\mathbf{u}}_{\text{off}} = \mathbf{z}_{\text{off}}. \quad (3.32)$$

The gradient of (3.27) with respect to  $\boldsymbol{\mu}$  becomes

$$\frac{\partial}{\partial \boldsymbol{\mu}} \mathcal{L}(\boldsymbol{\beta}^*, \lambda^*, \boldsymbol{\mu}^*) = \begin{bmatrix} \beta_{\text{on}} \\ \beta_{\text{off}} \end{bmatrix} - \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \end{bmatrix}, \quad (3.33)$$

such that

$$\tilde{\boldsymbol{\mu}}_{\text{on}} = \tilde{\boldsymbol{\beta}}_{\text{on}} - \mathbf{1} \quad (3.34)$$

$$\tilde{\boldsymbol{\mu}}_{\text{off}} = \tilde{\boldsymbol{\beta}}_{\text{off}} = \mathbf{0}. \quad (3.35)$$

Since we are only interested in positive values in  $\boldsymbol{\beta}$ , all  $\lambda_{\text{uneg}}$  are neglected. This results in the following equations to calculate  $\tilde{\boldsymbol{\beta}}_{\text{on}}$  and  $\tilde{\mathbf{u}}_{\text{off}}$

$$\tilde{\boldsymbol{\beta}}_{\text{on}} = \boldsymbol{\Phi}^{-1} (\mathbf{z}_{\text{on}} - \lambda_0 \tilde{\mathbf{u}}_{\text{on}} - \tilde{\boldsymbol{\mu}}_{\text{on}}) \quad (3.36)$$

where

$$\begin{aligned} \tilde{\boldsymbol{\mu}}_{\text{on}} &= \tilde{\boldsymbol{\beta}}_{\text{on}} - \mathbf{1} \\ &\Downarrow \\ \tilde{\boldsymbol{\mu}}_{\text{on}} &= \boldsymbol{\Phi}^{-1} (\mathbf{z}_{\text{on}} - \lambda_0 \tilde{\mathbf{u}}_{\text{on}} - \tilde{\boldsymbol{\mu}}_{\text{on}}) - \mathbf{1} \\ &\Downarrow \\ \tilde{\boldsymbol{\mu}}_{\text{on}} &= (\boldsymbol{\Phi}^{-1} + \mathbf{1}) (\mathbf{z}_{\text{on}} - \lambda_0 \tilde{\mathbf{u}}_{\text{on}}) - \boldsymbol{\Phi} \mathbf{1} - \mathbf{1} \end{aligned} \quad (3.37)$$

and  $\tilde{\boldsymbol{\mu}}_{\text{on}}$  are the Lagrange multipliers. The vector  $\tilde{\boldsymbol{u}}_{\text{off}}$  is unchanged compared to (3.22).

The linear equations to find the critical values of  $\lambda$  when  $\boldsymbol{\beta}$  leaves the region are derived from (3.22) with all  $\tilde{\boldsymbol{u}}_{\text{off}}$  are set to 1

$$\boldsymbol{\lambda}_{\text{upos}} = \mathbf{I} \left( \mathbf{z}_{\text{off}} - \boldsymbol{\Psi}^T \tilde{\boldsymbol{\beta}}_{\text{on}} \right). \quad (3.38)$$

The  $\boldsymbol{\lambda}$  who lead to an entry in  $\boldsymbol{\beta}$  becoming negative are omitted, since the constraint will force these entries to +1, increasing the residual.

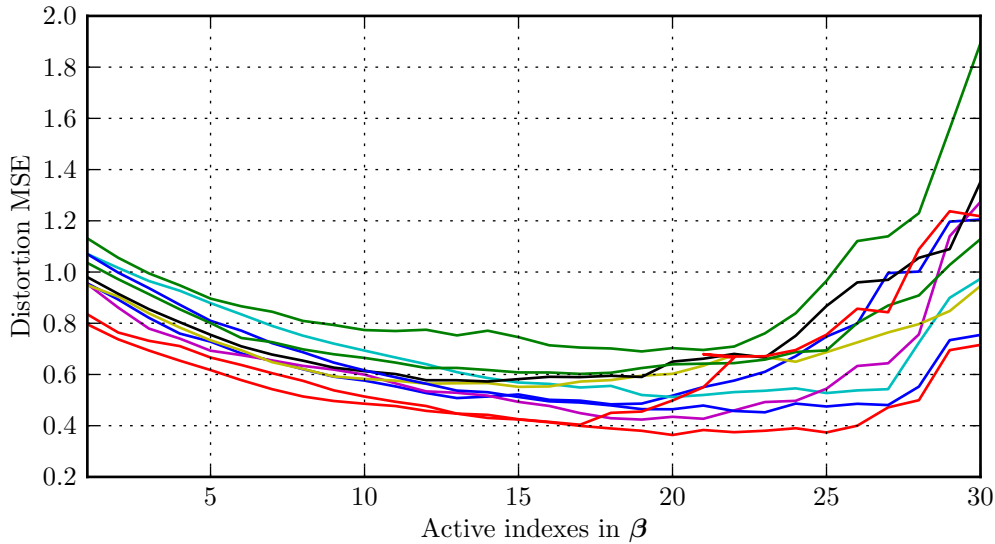
The  $\lambda$  forcing an entry in  $\boldsymbol{\beta}$  to zero are given by

$$\boldsymbol{\lambda}_{\boldsymbol{\beta}} = \text{diag} \{ \tilde{\boldsymbol{u}}_{\text{on}} \}^{-1} (\mathbf{z}_{\text{on}} - \tilde{\boldsymbol{\mu}}_{\text{on}}). \quad (3.39)$$

Finally the constraints for  $\tilde{\boldsymbol{\beta}}_{\text{on}}$  are verified. The rest of the algorithm is identical to Algorithm 3.3, with the expressions for the given variables replaced by those, given in this section. The calculation of  $\boldsymbol{\lambda}_{\text{uneg}}$  is omitted, and is not used.

### 3.2.5 Verification

The algorithm for the constrained homotopy is verified and the results for 10 simulations are plotted. The dictionary  $\boldsymbol{\Gamma} \in \mathbb{R}^{M \times N}$  is designed with  $M = 100$  and  $N = 500$  of random Gaussian distributed numbers and  $\boldsymbol{x} \in \mathbb{R}^M$ . We allow up to 30 entries in  $\boldsymbol{\beta}$  to become non-zero. The variance of each entry in  $\boldsymbol{\Gamma}$  is set to 1/30, while the variance of  $\boldsymbol{x}$  is 1.



**Figure 3.5.** Verification of the constrained homotopy algorithm with up to 30 non-zero values in  $\boldsymbol{\beta}$ .

Code: `constrainedhomotopy.py`.

Figure 3.5 shows the simulation results. The algorithm starts to converge, but the distortion tends to increase when more than 15 to 20 indexes in  $\boldsymbol{\beta}$  become non-zero. This can have various reasons. It can be the case, that there are no more vectors left in the dictionary, that reduce the MSE.

Another reason, that can explain, why the distortion increases is, that although the sub-problems are convex, since all  $\beta_{\mu_{\text{on}},i} \neq 0$  have the convex constraint  $\|\beta_{\mu_{\text{on}},i} - 1\| = 0$ , and  $\beta_{\mu_{\text{off}}} = 0$  have  $\|\beta_{\mu_{\text{off}},i}\| = 0$ , the main cost function (3.26) is not convex, since it forces each  $\beta_i$  to belong to a finite set, containing a 0 or 1.

### 3.2.6 SPARC in homotopy

Returning to the original problem, defined in Chapter 1, we have a dictionary  $\Gamma$  constructed as

$$\Gamma = \begin{array}{c} \leftarrow L \text{ columns} \rightarrow \leftarrow L \text{ columns} \rightarrow \quad \quad \quad \leftarrow L \text{ columns} \rightarrow \\ \left[ \begin{array}{ccc|ccc| \cdots | \cdots | \cdots} \gamma_{1,1} & \cdots & \gamma_{1,N} & \gamma_{1,N+1} & \cdots & \gamma_{1,2N} & \cdots & \gamma_{1,(L-1)N+1} & \cdots & \gamma_{1,LN} \\ \gamma_{2,1} & \cdots & \gamma_{2,N} & \gamma_{2,N+1} & \cdots & \gamma_{2,2N} & \cdots & \gamma_{2,(L-1)N+1} & \cdots & \gamma_{2,LN} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ \gamma_{M,1} & \cdots & \gamma_{M,N} & \gamma_{M,N+1} & \cdots & \gamma_{M,2N} & \cdots & \gamma_{M,(L-1)N+1} & \cdots & \gamma_{M,LN} \end{array} \right], \end{array} \quad (3.40)$$

where  $\beta$  only contains one non-zero entry on each Section. This entry must contain a 1.

To enforce this, we alter the homotopy implementation from section 3.2.4 to deactivate the region, as soon as one entry of  $\beta$  in that region becomes non-zero. If an entire section in  $\beta$  becomes zero, it will be activated again until a new entry becomes non-zero. This is done by introducing another list together with the already present  $\mathbf{I}_{\text{on}}$  and  $\mathbf{I}_{\text{off}}$ , called  $\mathbf{I}_{\text{pass}}$ , which contains all the elements which are passive. The consequence of this is that the entries in  $\mathbf{I}_{\text{pass}}$  are omitted in Equation (3.38), and thus never activated. The changes in algorithm 3.3 are shown in Algorithms 3.4 and 3.5, such that the sections are activated or deactivated, depending on whether there exists a 1 or 0 in the corresponding section of  $\beta$ .

---

**Algorithm 3.4** Modifications to Algorithm 3.3 line 16 to 19.

---

```

1: if  $\lambda_{\text{new}} \in \max_{\lambda} \lambda_{\beta}$  then
2:   Move index from  $\mathbf{I}_{\text{on}}$  to  $\mathbf{I}_{\text{off}}$ 
3:   for index in section of  $\lambda_{\text{new}}$  do
4:     Move index from  $\mathbf{I}_{\text{pass}}$  to  $\mathbf{I}_{\text{off}}$ 
5:   end for
6: end if
    
```

---



---

**Algorithm 3.5** Modifications to Algorithm 3.3 line 20 to 23.

---

```

1: if  $\lambda_{\text{new}} \in \max_{\lambda} \lambda_{\mathbf{u}_{\text{pos}}}$  then
2:   Move index from  $\mathbf{I}_{\text{off}}$  to  $\mathbf{I}_{\text{on}}$ 
3:   Set corresponding index in  $\mathbf{u} = 1$ 
4:   for index in section of  $\lambda_{\text{new}}$  do
5:     Move index from  $\mathbf{I}_{\text{off}}$  to  $\mathbf{I}_{\text{pass}}$ 
6:   end for
7: end if
    
```

---

### 3.2.7 Verification

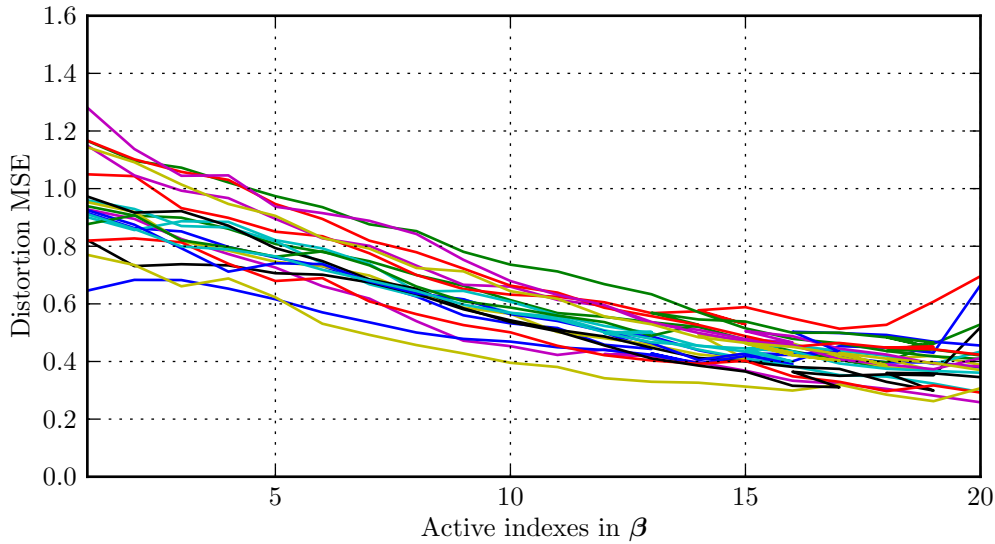
The verification of this algorithm is done by setting  $\mathbf{\Gamma} \in \mathbb{R}^{N \times ML}$  with  $N = 100$ ,  $L = 200$  and  $M = 20$ , such that we will obtain 20 non-zero entries in  $\beta$  which all contain 1. The  $\mathbf{\Gamma}$ -matrix is generated by scaled random Gaussian generated numbers, where each section in  $\mathbf{\Gamma}$  is scaled by

$$c_m = \sqrt{\frac{2R\sigma_x^2}{N} \left(1 - 2\frac{R}{N}\right)^{m-1}} \quad (3.41)$$

according to [15] where  $R$  is the rate given by

$$R = \frac{M \log_2(L)}{N} \quad (3.42)$$

and  $m \in [1, \dots, M]$  is the number of the section in  $\mathbf{\Gamma}$ .



**Figure 3.6.** Verification of the sectioned homotopy algorithm with 20 sections in  $\beta$ , which each has at most one non-zero value.  
Code: `section-homotopy.py`.

Figure 3.6 shows the convergence of the algorithm, and illustrates the occurrence of the same tendency already found for the constrained version of homotopy in Figure 3.5, where the algorithm converges at first until a sudden point, where the MSE starts to increase.

Another tendency is that the removal of an active  $\beta_i$  in  $\beta$  tends to increase the MSE instead of decreasing it, like homotopy normally does (Figure 3.4).

## 3.3 Summary

In this chapter, we have analyzed Matching Pursuit (MP) and homotopy algorithms. These are both suited to perform dictionary searches to estimate an input signal  $x$ , but

each uses different constraints. MP restricts the  $l_0$ -norm by limiting the number of non-zero entries in  $\beta$ , whereas homotopy limits the  $l_1$ -norm, such that the non-zero entries in  $\beta$  are not allowed to contain large numbers, depending on the weighting factor  $\lambda$ .

We have investigated to add additional constraints to the cost function used in the homotopy algorithm, forcing the non-zero entries in  $\beta$  to 1. The simulations showed, that this is not working as intended, with the MSE increasing after a few iterations. A possible explanation to this can be, that the cost-function is not convex, and has no analytic solution, even though the sub-problems, in which the cost-function is split, each are convex. It might be possible to solve the non-convex cost-function using other optimization algorithms and/or integer programming, but this is out of the scope of this thesis.

MP showed to perform well on sparse regression dictionaries, especially at lower bit-rates. Future optimization to the algorithm might include other scaling factors to the dictionary, and allowing the algorithm to select the vectors from the different sections in an arbitrary order instead of forcing it to select the first vector from the first section.





# 4 Control theory

---

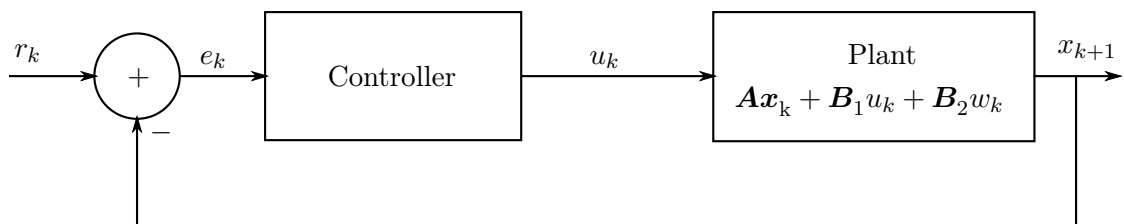
In this chapter, the motivation for feedback and control theory is described along with the basics of this theory. This is followed by an explanation of LQR control, which is a special type of feedback controller. The final sections describe finite horizon control to predict future control inputs and the introduction of packet loss to the network. The chapter concludes with simulation results. It should be noted that only discrete-time LTI systems are considered.

## 4.1 Control and feedback

Feedback control is applied in almost every modern system. Figure 4.1 shows a simplified basic diagram of a feedback controlled system. The simplifications that are used in this thesis are

- The output is identical to the variable we wish to control, which is not the case in the general setting.
- The signal  $u_k$  is scalar.

The plant is a not necessarily a stable system, which we wish to stabilize. This is done using feedback, by e.g. measuring the position, speed, temperature etc. from the plant. The controller uses the measurement to stabilize the plants state. Examples on feedback control are the thermostat controllers and cruise control in a vehicle.



**Figure 4.1.** General feedback loop with output  $x_{k+1}$ , input  $u_k$  and reference  $u_k$ .  $w_k$  is gaussian noise.

Figure 4.1 shows the plant, with the LTI state equation

$$\mathbf{A}\mathbf{x}_k + \mathbf{B}_1 u_k + \mathbf{B}_2 w_k, \quad (4.1)$$

where  $\mathbf{A}$  is the system matrix. This matrix has a direct relation to transfer functions as

$$H(z) = \det(z\mathbf{I} - \mathbf{A})^{-1} \mathbf{B}_1, \quad (4.2)$$

of which the poles are given by the eigenvalues of  $\mathbf{A}$  and  $z$  is the z-transform variable.

## 4.1.1 Stability

Consider a state-space equation

$$x_{k+1} = f(x_k) \quad (4.3)$$

with initial conditions  $x_{k_0}^0$  and  $x_{k_0}$ .

### Stability

The system in Equation (4.3) is stable if and only if for any given  $\epsilon > 0$  a  $\delta(\epsilon, k_0) > 0$  exists, such that all solutions with  $\|x_{k_0} - x_{k_0}^0\| < \delta$  imply that  $\|x_k - x_k^0\| < \epsilon$  for all  $k \geq k_0$ . [11].

### Asymptotic stability

A system is asymptotic stable if the system is stable and if  $\|x_k - x_k^0\| \rightarrow 0$  when  $k \rightarrow \infty$  provided that  $\|x_{k_0} - x_{k_0}^0\|$  is small enough [11]. For LTI systems the system is stable if and only if all eigenvalues of  $\mathbf{A}$  are strictly located within the unit circle [4],

### BIBO stability

A system is Bounded Input Bounded Output (BIBO) stable if a bounded input gives a bounded output for every initial value [11].

Asymptotical stability implies that a system is stable and BIBO stable[11].

The stability of an open-loop system can thus be checked by observing the eigenvalues of  $\mathbf{A}$ . The closed loop response is given by

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}_1 u_k \quad (4.4)$$

$\Downarrow$

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}_1 (-\mathbf{K}\mathbf{x}_k), \quad (4.5)$$

where  $\mathbf{K}$  is the controller gain. The system is again stable if and only if the eigenvalues of  $\mathbf{A} - \mathbf{B}_1\mathbf{K}$  are within the unit circle.

### Controllability

A system is said to be controllable if and only if [11]

$$\text{rank}\{\mathbf{W}_c\} = N, \quad (4.6)$$

where  $N$  is the dimension of  $\mathbf{A}$  and  $\mathbf{W}_c$  is the controllability matrix given by

$$\mathbf{W}_c = [\mathbf{B}_1 \mathbf{A}\mathbf{B}_1 \dots \mathbf{A}^{N-1}\mathbf{B}_1]. \quad (4.7)$$

If a system is controllable it is possible to find a control sequence such that the origin can be reached in finite time from any initial state. [11]

A system is reachable if it is possible to reach any state from any initial state in finite time [11]. Reachability and controllability are equivalent if  $\mathbf{A}$  is invertible [11].

## 4.2 Controller design

A system is controllable if the controllability matrix has full rank. This means, that it is possible to design a controller with gain  $\mathbf{K}$ , such that

$$u_k = -\mathbf{K}\mathbf{x}_k. \quad (4.8)$$

There are different approaches to design the controller e.g. by manually choosing the pole locations, and calculate the controller gain according to those. An efficient solution with guaranteed stability is the LQR method. Here a cost-function is designed and minimized to obtain the optimal controller gain.

The cost function is given as

$$J(u, \mathbf{x}_k) = \|\mathbf{x}_k\|_Q + \|u_k\|_R, \quad (4.9)$$

with the optimal control sequence to minimize the cost-function given in equation (4.8), where  $\mathbf{K}$  is given by

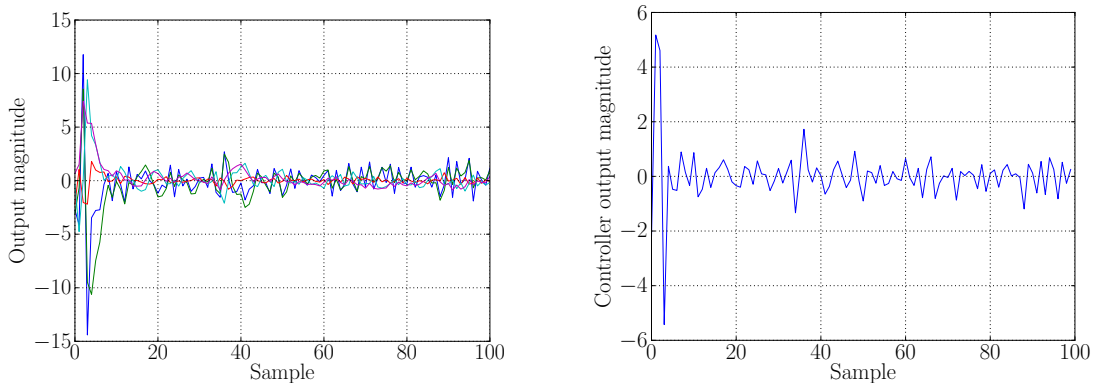
$$\mathbf{K} = (\mathbf{R} + \mathbf{B}_1^T \mathbf{P} \mathbf{B}_1)^{-1} \mathbf{B}_1^T \mathbf{P} \mathbf{A}. \quad (4.10)$$

Here  $\mathbf{P}$  is the solution to the discrete algebraic Ricatti equation [11]

$$\mathbf{P} = \mathbf{A}^T \mathbf{P} \mathbf{A} + \mathbf{Q} - (\mathbf{A}^T \mathbf{P} \mathbf{B}_1) (\lambda \mathbf{I} + \mathbf{B}_1^T \mathbf{P} \mathbf{B}_1)^{-1} (\mathbf{B}_1^T \mathbf{P} \mathbf{A}). \quad (4.11)$$

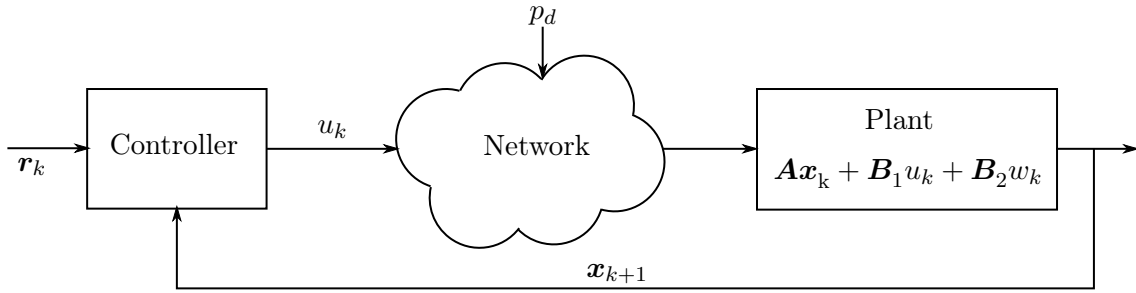
### 4.2.1 Simulations

The LQR controller is implemented for a randomly generated  $\mathbf{A} \in \mathbb{R}^{m \times m}$  and  $\mathbf{B}_1 = \mathbf{B}_2 = [1 \ 1 \ 1 \ 1]^T$ .  $\mathbf{Q} = \mathbf{I}$  and  $R = 1/2$ . The seed for the random generator is set to 111 in Python, such that all simulations are based on the same random numbers. Simulations with this controller with no packet dropout are shown in Figure 4.2. The noise is IID Gaussian distributed with variance  $\sigma_w^2 = 0.1$ .



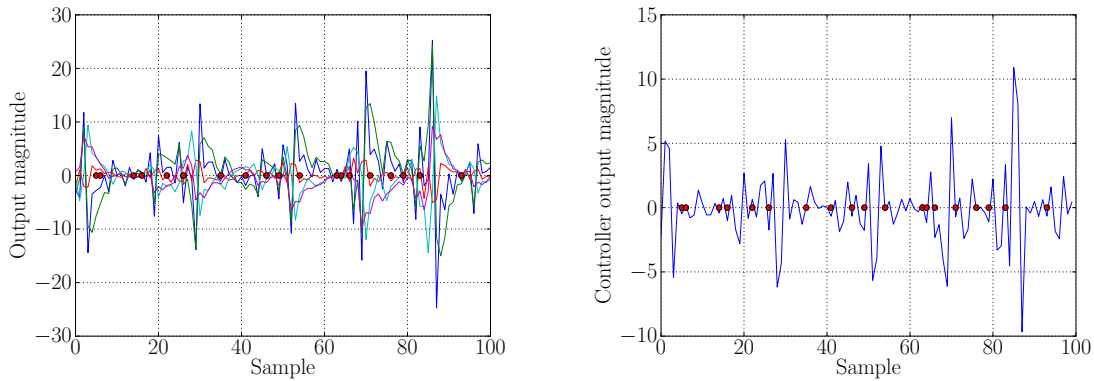
**Figure 4.2.** Left: output states of the plant, right: control signal  $u_k$ .  
Code: `lqr.py`.

The control network with a packet dropout introduced is shown in Figure 4.3. The first packet is set to always arrive at the plant.



**Figure 4.3.** Control system with a lossy network where packet dropout occurs.

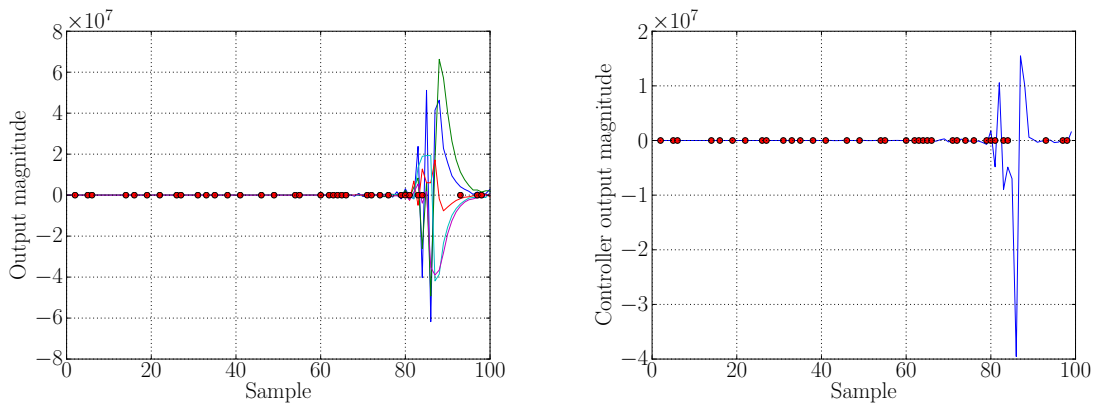
The network is simulated by a binomial process with the probability  $p$  for a packet dropout and probability  $1 - p$  for a packet passing through. The signals with a 20% packet dropout are shown in Figure 4.4. The red dots indicate when a packet is dropped. In this case  $u_k = 0$  is sent to the plant.



**Figure 4.4.** Controller with 20% packet dropout on the network. Red dots indicate a dropped packet. Left output states of the plant, right: control signal  $u_k$ .  
Code: `lqr-dropout.py`.

It can be seen, that the plant easily becomes unstable and begins to oscillate.

Figure 4.5 shows the same simulation with a 40% packet dropout, resulting in larger oscillations.

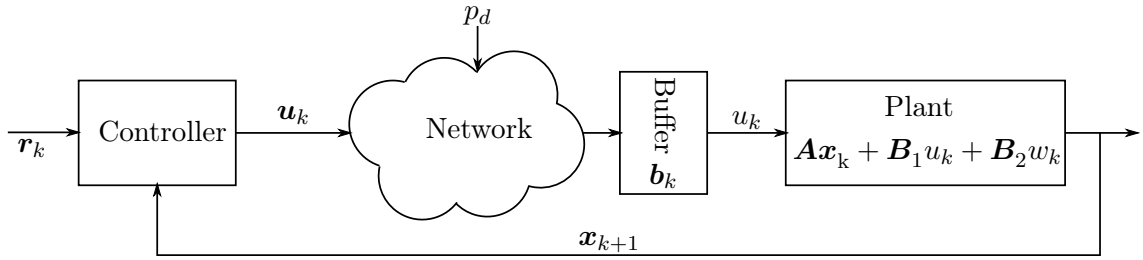


**Figure 4.5.** Controller with 40% packet dropout on the network. Red dots indicate a dropped packet. Left output states of the plant, right: control signal  $u_k$ .  
Code: `lqr-dropout40.py`.

As Figures 4.4 and 4.5 show, the plant can become unstable when packet dropouts occur. This corresponds to operating the plant in open-loop with no feedback.

## 4.3 Model predictive control

To prevent the plant from running completely in open-loop when a packet dropout occurs it is desired to predict a number of future control signals, which can be used if a packet dropout occurs at sample  $k$ . In this case the diagram from Figure 4.3 is modified to include a buffer, containing the predicted samples as shown in Figure 4.6.



**Figure 4.6.** Diagram of the model predictive control system, where packet dropout occurs.

The buffer  $b_k$  is operated as [4]

$$b_k = d_k M b_{k-1} + (1 - d_k) U_k, \quad (4.12)$$

where  $d_k = 1$  if a packet dropout occurs. The matrix  $M$  shifts the buffer one state if a packet dropout occurs, and is given by [4]. In this case the used  $u_k$  is deleted from the buffer and a zero is added to the end. It can also be chosen to make the buffer circular, repeating the used  $u_k$  in the end of the buffer.

$$M = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 \\ 0 & \cdots & \cdots & \cdots & 0 \end{bmatrix}. \quad (4.13)$$

The input to the controller  $u_k$  is updated as

$$u_k = e_1^T b_k, \quad (4.14)$$

with

$$e_1 = [1 \ 0 \ \cdots \ 0]^T. \quad (4.15)$$

It is in this case necessary to update the cost function of the controller to predict the future states of the plant. These are given as

$$x'_{l+1} = Ax'_l + B_1 u'_l \quad (4.16)$$

with  $l$  indicating the predicted sample number and  $\mathbf{x}'_0$  being the current plant state [4].

The cost function to minimize is given as [4]

$$J(\mathbf{u}_k, \mathbf{x}_k) = \|\mathbf{x}'_N\|_P^2 + \sum_{l=0}^{N-1} (\|\mathbf{x}'_l\|_Q^2 + \|u'_l\|_R^2), \quad (4.17)$$

which can be rewritten as

$$\begin{aligned} J(\mathbf{u}_k, \mathbf{x}_k) &= \|\Upsilon \mathbf{x}_k + \Phi \mathbf{u}_k\|_2^2 \\ &= \mathbf{x}_k^T \Upsilon^T \Upsilon \mathbf{x}_k + \mathbf{u}_k^T \Phi^T \Phi \mathbf{u}_k + 2\mathbf{x}_k^T \Upsilon^T \Phi \mathbf{u}_k, \end{aligned} \quad (4.18)$$

with

$$\Phi = \begin{bmatrix} B_1 & 0 & \cdots & 0 \\ AB_1 & B_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{N-1}B_1 & A^{N-2}B_1 & \cdots & B_1 \end{bmatrix} \quad (4.19)$$

and  $\Upsilon$  as

$$\Upsilon = \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^N \end{bmatrix}. \quad (4.20)$$

The gradient with respect to  $\mathbf{u}_k$  is

$$\begin{aligned} \frac{\partial}{\partial \mathbf{u}_k} J(\mathbf{u}_k, \mathbf{x}_k) &= 2\Phi^T \Phi \mathbf{u}_k + 2\Phi^T \Upsilon \mathbf{x}_k \\ &= \mathbf{G} \mathbf{u}_k + \mathbf{H} \mathbf{x}_k \end{aligned} \quad (4.21)$$

with

$$\mathbf{G} = \Phi^T \bar{\mathbf{Q}} \Phi + \bar{\mathbf{R}} \quad (4.22)$$

$$\mathbf{H} = \Phi^T \bar{\mathbf{Q}} \Upsilon \quad (4.23)$$

and

$$\bar{\mathbf{Q}} = \text{blockdiag}(\mathbf{Q}, \dots, \mathbf{Q}, \mathbf{P}). \quad (4.24)$$

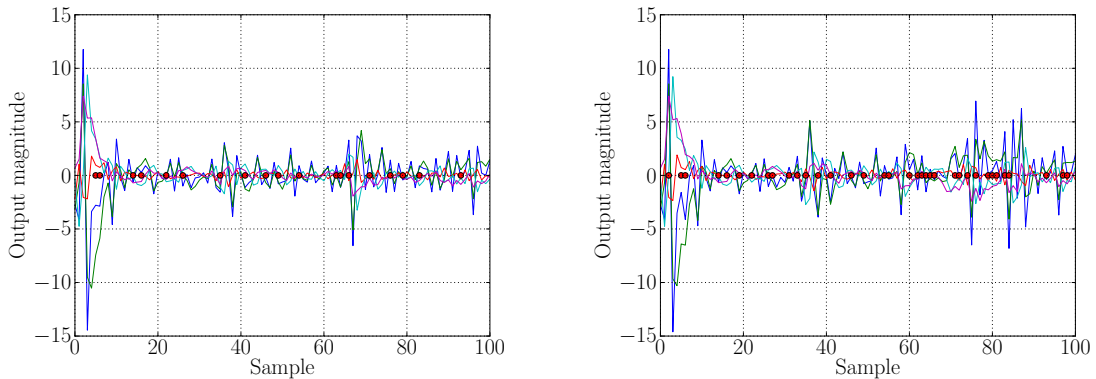
$$\bar{\mathbf{R}} = \text{diag}\{R, \dots, R\} \quad (4.25)$$

$\mathbf{P}$  is given by the discrete algebraic Ricatti equation (4.11) and  $\bar{\mathbf{R}}$  and  $\bar{\mathbf{Q}}$  are the weighting parameters to balance the penalty on the size of  $\mathbf{u}_k$  and of  $\mathbf{x}_{k+1}$ , respectively. This results in the control equation

$$\mathbf{u}_k = -\mathbf{G}^{-1} \mathbf{H} \mathbf{x}_k \quad (4.26)$$

### 4.3.1 Simulations

The MPC algorithm is simulated with a horizon length of 8 with the rest of the parameters remaining similar to those set in Section 4.2.1 and first sample set to always arrive at the buffer. The results are shown in Figure 4.7.

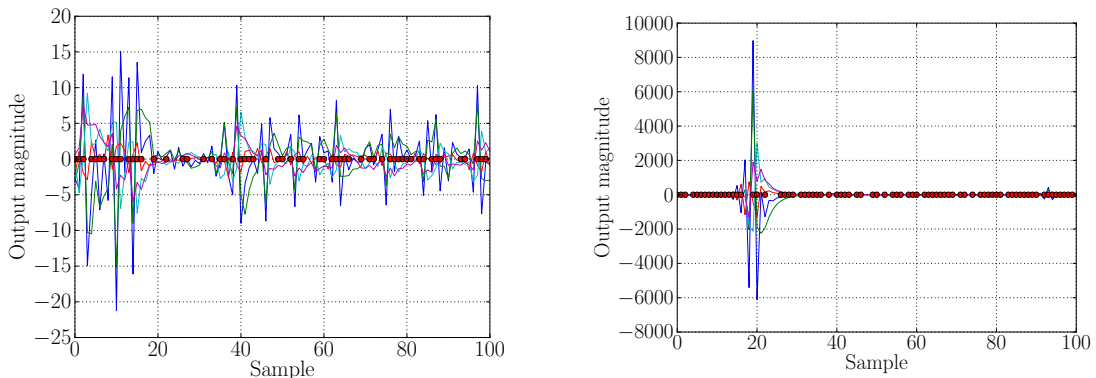


**Figure 4.7.** MPC with 20% (left) and 40% (right) packet dropout, horizon length  $N = 10$ .

Code: `lqr-horizon20.py` and `lqr-horizon40.py`.

When comparing these results to those obtained in Figure 4.2, it can be seen, that the packet dropout barely disturbs the plant states.

Figure 4.8 shows the control signals when 60% and 80% packet dropout occurs.



**Figure 4.8.** MPC with 60% and 80% packet dropout.

Code: `lqr-horizon60.py` and `lqr-horizon80.py`.

The curves on these figures change, since the packet dropout combined with the noise causes disturbances in the output signals, which the MPC does not take into account. The controllers also runs in open-loop when more than 8 consecutive packet dropouts occur, since the horizon length is set to 8 samples. This can also result in large disturbance in the plant, depending on the state and noise in the plant.

## 4.4 Summary

In this chapter we discussed the LQR controller and listed the stability criteria for these. It is shown how an LQR controller is designed, and this is simulated with a network with IID random dropouts occurring, resulting in large spikes in the state of the plant.

We finally showed the MPC, which is able to calculate future control signal for a finite horizon. These future predictions are stored in a buffer at the plant, and are used in case a packet dropout occurs. Simulations of this shows, that a MPC controller is able to maintain the state of the plant within lower values compared to the LQR, even with a high probability of packet dropouts.



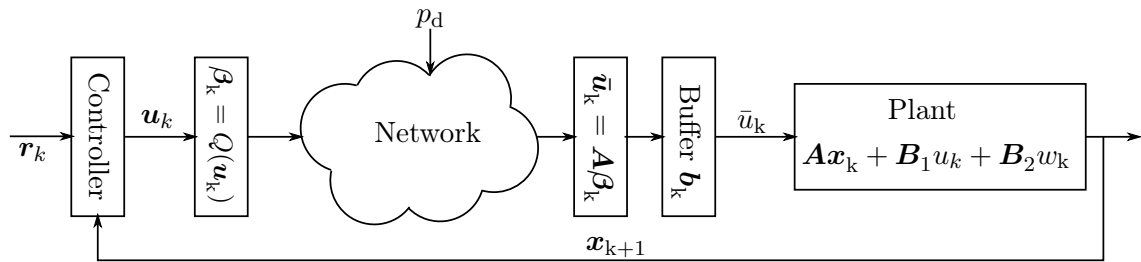
# 5 Quantized PPC

In this chapter the control loop from Chapter 4 is combined with a quantizer, reducing the bit rate of the control packets  $\mathbf{u}$  send through the network to the buffer  $\mathbf{b}_k$ .

The first section describes quantization added in the control loop and the issues related to this. The following sections investigate closed loop control, where the quantizer is applied directly to the cost function.

## 5.1 PPC with a quantizer

Quantization is regularly used to reduce the size of data, reducing the requirements to the bandwidth of the storage or channel. In this thesis, we want to reduce the size of the packet, generated at the controller, such that it can be transmitted through a network with limited bandwidth. Figure 5.1 shows the setup of the quantizer in the control loop.



**Figure 5.1.** The control network with the quantizer added.

In this case the controller calculates the control signal  $\mathbf{u}_k$  according to equation (4.26), which is repeated here

$$\mathbf{u}_k = -\mathbf{G}^{-1}\mathbf{H}\mathbf{x}_k. \quad (5.1)$$

The quantizer performs the operation

$$\beta_k = Q(\mathbf{u}_k) \quad (5.2)$$

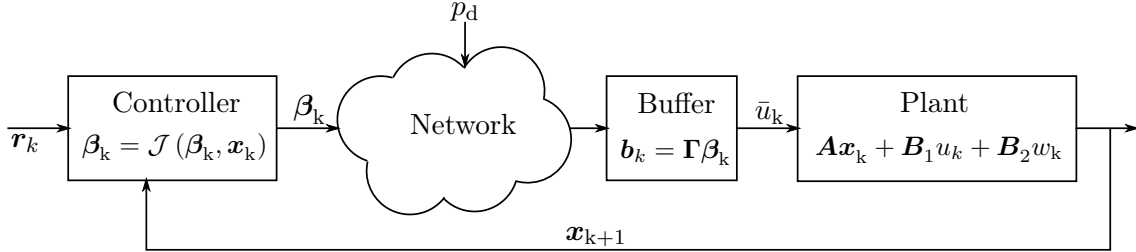
$$= \min_{\beta_k} \|\mathbf{u}_k - \mathbf{A}\beta_k\| + \|\beta_k\|_0, \quad (5.3)$$

which is transmitted. On the receiving side, the decoder reconstructs  $\bar{\mathbf{u}}_k = \mathbf{A}\beta_k$ , and feeds this to the buffer  $\mathbf{b}_k$ .

In this setup, the controller finds the control signal  $\mathbf{u}$ , minimizing the cost function (4.17). Afterwards, this signal is quantized using a finite set of numbers to approximate  $\mathbf{u}$ . The selected vectors are not necessarily the optimal ones to reduce the cost function. This issue can be avoided by using the finite set of numbers directly in the cost function.

## 5.2 Closed loop PPC

To address the problems mentioned in the previous section, it is chosen to implement the quantizer directly into the controller, such that the finite set of numbers created by the dictionary and  $\beta$  are used in the cost function. This setup is illustrated in Figure 5.2.



**Figure 5.2.** Closed loop PPC with the quantizer integrated in the cost function according to Equation (5.4).

The cost function is stated by

$$\hat{\mathcal{J}}(\beta_k, \mathbf{x}_k) = \min_{\bar{\mathbf{u}}_k} \mathbf{x}_k^T \Upsilon^T \Upsilon \mathbf{x}_k + \bar{\mathbf{u}}_k^T \Phi^T \Phi \bar{\mathbf{u}}_k + 2\mathbf{x}_k^T \Upsilon^T \Phi \bar{\mathbf{u}}_k, \quad \bar{\mathbf{u}}_k \in \mathbb{W}, \quad (5.4)$$

with  $\mathbb{W}$  denoting the set of vectors obtained by the linear combination

$$\bar{\mathbf{u}}_k = \Gamma\beta. \quad (5.5)$$

The remaining symbols are identical to the ones in Chapter 4.

The cost function is solved using a greedy method, selecting the vector from the first section  $M$  in the dictionary, reducing the cost function the most and repeats this for every section

$$i^* = \min_i \gamma_i^T \Phi^T \Phi \gamma_i + 2\mathbf{x}_k^T \Upsilon^T \Phi \gamma_i, \quad i \in ((m-1)L+1, \dots, Lm), \quad m \in (1, \dots, M) \quad (5.6)$$

$$\beta_{i^*} = 1 \quad (5.7)$$

$$\beta_j = 0, \quad \forall j \neq i^* \quad (5.8)$$

with  $L$  being the number of vectors in each part  $m$  in dictionary  $\Gamma$ . This is repeated for all sections  $M$  in the dictionary, such that

$$\bar{\mathbf{u}}_k = \Gamma\beta_k. \quad (5.9)$$

In this case, we chose to minimize the cost function for one section of  $\Gamma$  each iteration. This is shown in Algorithm 5.1.

It can also be chosen to provide all sections at every iteration, and remove the sections, when one vector in the section is chosen. This can, in some cases give better results, since the cost function (4.18) might get reduced further, although the computational cost is increased, since  $L(M-m)$  calculations have to be done for every iteration  $m$ .

---

**Algorithm 5.1** Algorithm to do a greedy search on the cost function.

---

```

1: Dictionary  $\Gamma$ 
2: Input signal  $\mathbf{x}_k$ 
3:  $\beta = \mathbf{0}$  ▷ Initialize  $\beta$  with zeros.
4: for  $m = 1 \rightarrow M$  do ▷ Repeat for every section.
5:    $\bar{\mathbf{u}} = \Gamma\beta$  ▷ Create  $\bar{\mathbf{u}}$  from the already selected vectors.
6:   for  $i = L(m - 1) + 1 \rightarrow Lm$  do
7:      $\mathbf{r} = \Gamma\beta + \gamma_i$ 
8:      $res_i = \mathbf{r}^T \Phi^T \Phi \mathbf{r} + 2\mathbf{x}_k^T \Upsilon^T \Phi \mathbf{r}$  ▷ Solve cost function for all  $\gamma$  in the section.
9:   end for
10:   $g = \arg \min(res)$  ▷ find the vector minimizing the cost function.
11:   $\beta_g = 1$ 
12: end for

```

---

## 5.3 Summary

This section described some basics regarding quantization on PPC systems. Two different methods to quantize a PPC are illustrated and described. It is chosen to continue working on the closed loop quantizer, since this integrates the controller and quantizer into one unit, allowing us to incorporate the quantizer directly into the cost function.

The performance of the quantizer is dependent on the design of the dictionary, which is covered in the next chapter.



# 6 Dictionary design

---

This chapter describes some considerations regarding the design of the SPARC dictionary for the closed loop PPC. We state three SPARC dictionaries in this chapter, two dictionaries consisting of IID Gaussian vectors. The first dictionary uses no scaling on the different sections (refinements) in the dictionary, whereas the second uses scaling based on the index of the refinement in the dictionary. The last dictionary is a lattice dictionary, with fixed size Voronoi cells. We finally introduce an optional variable gain, that can be added to the dictionaries.

The different ideas are described and simulated to verify whether they are able to keep the system stable at a rate of around 5 bit/symbol. The simulations are only performed to verify the dictionaries, and can not be used to perform a direct comparison between the different dictionary designs.

## 6.1 Gaussian dictionary

The SPARC dictionaries used in Chapter 3 are all based on Gaussian IID vectors, which results in a decent performance in the algorithms presented. It is for that reason chosen to test this kind of dictionary in PPC. The signals used Chapter 3 were Gaussian IID with unit variance, which were quantized by a dictionary created with the same parameters. The signals to quantize in PPC are more complex and the predictions in  $\mathbf{u}$  are correlated since they depend on each other. A possible dictionary can be made by simulating the PPC and use the calculated values for  $\mathbf{u}$ . An alternative is to calculate the parameters for the PPC and use these to generate Gaussian IID vectors for the dictionary. This requires the first and second moments of the system, which are found analogously to [1] by

$$\mathbb{E}\{\mathbf{u}\} = \mathbb{E}\{-\mathbf{G}^{-1}\mathbf{H}\mathbf{x}\} = -\mathbf{G}^{-1}\mathbf{H}\mathbb{E}\{\mathbf{x}\} \quad (6.1)$$

$$\mathbf{Q}_{\mathbf{u}} = \text{var}\{-\mathbf{G}^{-1}\mathbf{H}\mathbf{x}\} = \mathbf{G}^{-1}\mathbf{H}\mathbb{E}\{\mathbf{x}\mathbf{x}^T\}\mathbf{H}^T\mathbf{G}^{-T}. \quad (6.2)$$

In [1] the covariance of the PPC is based on the dropout probability  $p_d$ , which in this case is known. The covariance used to generate the dictionary is calculated analogously to [1] with the quantizer noise omitted, since it is not known. The covariance of the total system state is found to be

$$\mathbf{Q}_{\Theta, k+1} = \mathbb{E}\{\Theta_{k+1}\Theta_{k+1}^T\} = \mathcal{A}\mathbb{E}\{\Theta_k\Theta_k^T\}\mathcal{A}^T + p_d(1-p_d)\tilde{\mathcal{A}}\mathbb{E}\{\Theta_k\Theta_k^T\}\tilde{\mathcal{A}}^T + \mathcal{C}(p_d), \quad (6.3)$$

where

$$\Theta_k = \begin{bmatrix} \mathbf{x}_k \\ \mathbf{b}_{k-1} \end{bmatrix} \quad (6.4)$$

is the total state of the system and buffer, and

$$\mathcal{C}(p_d) = \sigma_w^2 \mathbf{B}_w \mathbf{B}_w^T \quad (6.5)$$

$$\mathcal{A} = \mathbb{E} \{ \bar{\mathbf{A}}(d_k) \} \quad (6.6)$$

$$\tilde{\mathcal{A}} = \bar{\mathbf{A}}(1) - \bar{\mathbf{A}}(0) \quad (6.7)$$

$$\mathcal{B} = \mathbb{E} \{ \bar{\mathbf{B}}(d_k) \}. \quad (6.8)$$

The matrices  $\bar{\mathbf{A}}(p_d)$  and  $\bar{\mathbf{B}}(p_d)$  are modified compared to [1] since we in this thesis are not operating in the  $\xi$ -domain and the quantizer noise is omitted. The matrices become

$$\bar{\mathbf{A}}(0) = \begin{bmatrix} \mathbf{A} - \mathbf{B}_1 \mathbf{e}_1^T \mathbf{K} & \mathbf{0} \\ -\mathbf{K} & \mathbf{0} \end{bmatrix} \quad \bar{\mathbf{A}}(1) = \begin{bmatrix} \mathbf{A} & \mathbf{B}_1 \mathbf{e}_1^T \mathbf{M} \\ \mathbf{0} & \mathbf{M} \end{bmatrix}$$

$$\bar{\mathbf{B}}(0) = \bar{\mathbf{B}}(1) = \begin{bmatrix} \mathbf{B}_2 \\ \mathbf{0} \end{bmatrix}$$

and  $\sigma_w^2$  denoting the plant noise. Since we are interested in the covariance of  $\mathbf{u}$ , which according to (6.2) depends on  $\mathbf{Q}_x$  since the covariance  $\mathbf{Q}_x$  is found as

$$\mathbf{Q}_\Theta = \begin{bmatrix} \mathbf{Q}_x & \mathbb{E} \{ \mathbf{x} \mathbf{b}^T \} \\ \mathbb{E} \{ \mathbf{b} \mathbf{x}^T \} & \mathbf{Q}_b \end{bmatrix}, \quad (6.9)$$

where the covariance of  $\mathbf{u}$  is given by

$$\mathbf{Q}_u = \mathbf{G}^{-1} \mathbf{H} \mathbf{Q}_x \mathbf{H}^T \mathbf{G}^{-T}. \quad (6.10)$$

The dictionary is created as  $N \times ML$  matrix with

$$\mathbf{\Gamma} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_u). \quad (6.11)$$

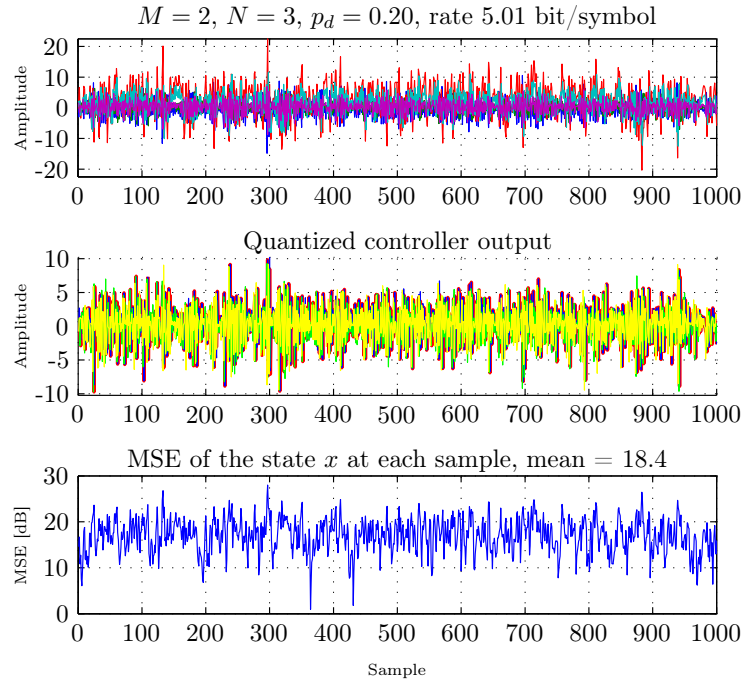
## 6.1.1 Verification of Gaussian dictionary

A simulation of the dictionary has been made to verify that it is working as desired. The controller is designed according to Algorithm 5.1, with a zero-mean normal distributed dictionary with covariance  $\mathbf{Q}_u$ .

The simulation of the controller is performed with horizon length  $N = 3$ , a dropout rate of  $p_d = 0, 20$ , and a system matrix  $\mathbf{A}$  randomly generated to be

$$\mathbf{A} = \begin{bmatrix} -0.7578 & -0.3245 & -0.085337 & 0.060403 & -2.2557 \\ 0.43212 & -0.35593 & 0.0024123 & 0.0071095 & -0.17091 \\ -0.17328 & 1.0627 & 0.36569 & 0.67106 & 0.93852 \\ 0.95123 & 0.66704 & 0.73738 & -0.43393 & 0.35231 \\ 1.0536 & 0.48356 & -0.15787 & 0.45405 & -0.26378 \end{bmatrix}, \quad (6.12)$$

with spectral radius  $r_\sigma = 1.66$  and vectors  $\mathbf{B}_1 = \mathbf{B}_2 = [1 \ 1 \ 1 \ 1 \ 1]^T$ . Figure 6.1 shows the plant state, controller output and whether a dropout occurred at each sample during the



**Figure 6.1.** Simulation of closed loop control with a Gaussian dictionary consisting of two sections and a rate of 5.01 bit/symbol, corresponding to  $L = 182$ . Code: `controllergaussian.m`.

simulation. It is seen, that the plant is kept stable, with a few large peaks, which are stabilized again. The plot at the top shows the amplitude of all five states in the plant at sample  $k$ , with the plot in the middle showing the quantized output and predictions of the controller at sample  $k$ . The bottom plot shows the MSE accumulated for all five states for each sample  $k$ . The bit rate is calculated per symbol (entry in  $\mathbf{u}_k$ ) as

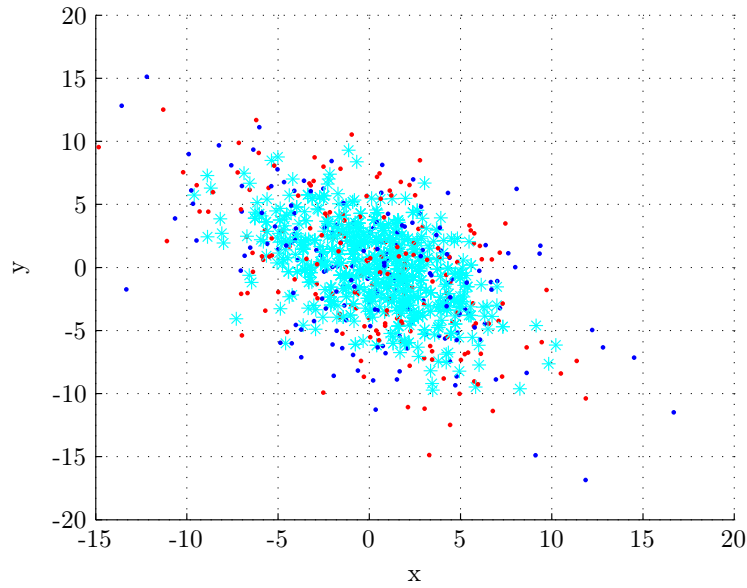
$$rate = M \frac{\log 2(L)}{N}. \quad (6.13)$$

A 2D view of the dictionary used in the simulation is shown in Figure 6.2. In this dictionary, both sections are created by different realizations of exactly the same distribution, with the covariance calculated by Equation (6.3) with the noise variance  $\sigma_w^2 = 0.1$ .

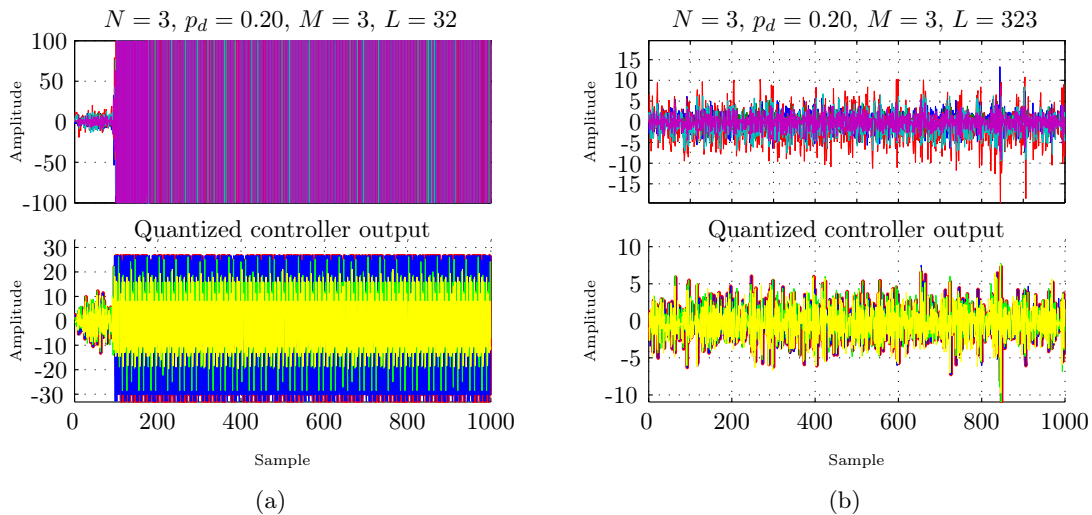
Figure 6.3 shows a Gaussian IID dictionary split into  $M = 3$  sections. Using a horizon length  $N = 3$  and  $L = 32$ , the system, as easily can be seen after a few samples in Figure 6.3(a). This is mainly due to the low amount of vectors in each section. Although the total number of combinations that can be obtained from the sections, is very close to the amount obtained when  $M = 2$  (32768 compared to 33124), the greedy method to select the vectors is limited to select one among only 32 vectors for every iteration.

By increasing the number of columns in each section by a factor ten leading to  $L = 323$  and a bit rate of 8.32 bit/symbol the system can be stabilized, as illustrated in Figure 6.3(b). The increase results in a total of  $33.7 \cdot 10^6$  possible linear combinations.

The performance of the quantizer, stability wise, can be improved by making the first search pick the best matching vector among all sections, and exclude the sections, for



**Figure 6.2.** A 2D view of the dictionary and values generated by the quantizer in the simulation shown in figure 6.1 the red and blue dots indicate the entries in the each section, while the cyan stars indicate the picked values generated by the controller.  
Code: `controllersimgaussian.m`.



**Figure 6.3.** Simulation run with  $N = 3$  with  $M = 3$  sections in the Gaussian IID dictionary and a rate of 5 bit/symbol (a) and 8.34 bit/symbol (b). The top figures show the plant state  $\mathbf{x}_k$ , whereas the bottom show  $\mathbf{u}_k$ . The remaining parameters are equal to the simulation shown in Figure 6.1. It is shown, that increasing the bit rate, leading to an increment in  $L$ , results in stabilizing the system.  
Code: `controllergaussian3secN3.m` and `controllergaussian3secN5.m`.



which a vector is already picked from the next iterations.

## 6.2 Gaussian dictionary with scaled refinements

Another method to design the SPARC dictionary is to scale each section of the dictionary, such that the first section contains the most energy, which is reduced for every section. This scaling can be done using Equation (3.9), which is repeated here

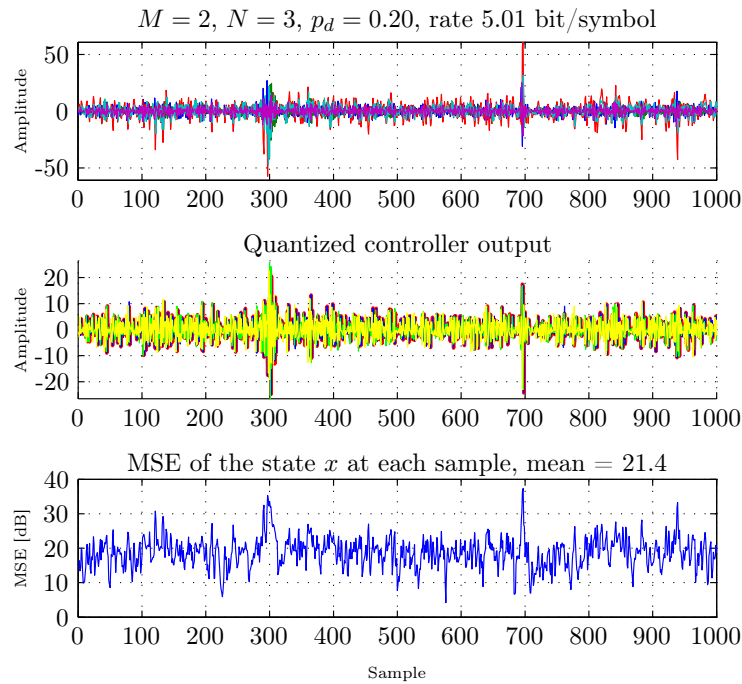
$$c_k = \sqrt{\frac{2R\sigma^2}{M} \left(1 - \frac{2R}{M}\right)^{k-1}} \quad k = 1, \dots, M, \quad (6.14)$$

although it is not suited for dictionaries with relatively few sections, since the negative part gets too large, and results in imaginary numbers. For this reason, the scaling has been changed to

$$c_k = \sqrt{\frac{2R\sigma^2}{M} \left(1 - \frac{0.9R}{M}\right)^{k-1}} \quad k = 1, \dots, M. \quad (6.15)$$

### 6.2.1 Verification

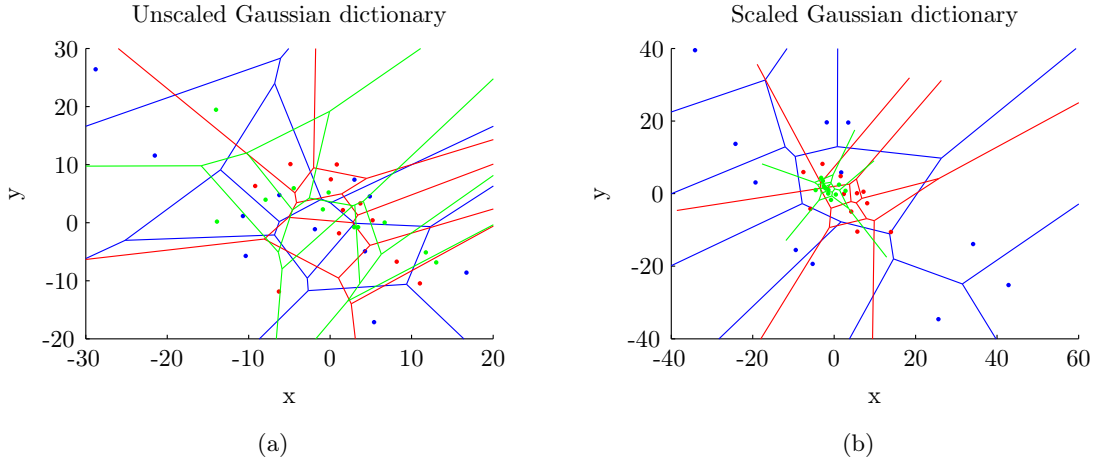
The performance of this dictionary is shown in Figure 6.4, and seems to contain more



**Figure 6.4.** Simulation of closed loop PPC with a scaled Gaussian dictionary with 2 sections and a rate of 5.01 bit/symbol, corresponding to  $L = 182$ .  
Code: `controlsimgaussianscale.m`.

spikes than the Gaussian dictionary without scaled refinements when operating with a

horizon length of  $N = 3$ . This dictionary also results in a slightly higher MSE than the unrefined version illustrated in Figure 6.1.



**Figure 6.5.** Voronoi cells of the Gaussian quantizer without scaling (a) and with scaling (b). The colors indicate the different sections  $M$  in the dictionary. The horizon length is  $N = 2$  with a rate of 5 bit/symbol and  $M = 3$  sections.  
Code: `voronoiplot.m`.

Figure 6.5 depicts the Voronoi cells for the Gaussian dictionary with and without scaled refinements for a horizon length  $N = 2$ , and  $L = 11$ . When using the scaled Gaussian dictionary, shown in Figure 6.5(b), the inner Voronoi cells have a smaller volume. The disadvantage in this case is that the dictionary is overloaded earlier, since the total energy of the linear combination of the scaled dictionary is lower, than the non-scaled. This can be resolved by using a larger scaling for the first section

## 6.3 Lattice dictionary

The dictionaries presented in Sections 6.1 and 6.2 both use a randomly generated dictionary. This means that the Voronoi cells have different volumes, and are placed randomly with the possibility of a few outliers, which is shown in Figure 6.5. An alternative method to design a dictionary is to use lattices, fixing the volumes of the Voronoi cells.

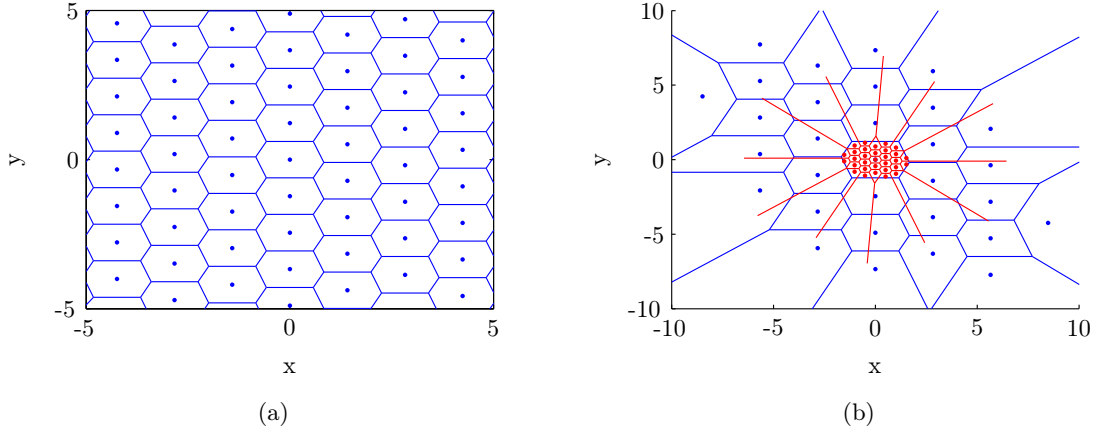
The root lattice A2 [22] is shown in figure 6.6(a), which uses the gram matrix

$$\mathbf{U}_{\text{gram,A2}} = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}, \quad (6.16)$$

to create the generator matrix

$$\mathbf{U}_{\text{A2}} = \text{chol}(\mathbf{U}_{\text{gram,A2}}) = \begin{bmatrix} \sqrt{2} & 0 \\ -\frac{\sqrt{2}}{2} & 1.2247 \end{bmatrix}. \quad (6.17)$$

Figure 6.6(b) shows the dictionary shaped by the pdf of the distribution of the system based on Equation (6.3) with one refinement. The refinement is created by scaling the



**Figure 6.6.** The Voronoi cells of a 2D lattice dictionary (a) and the  $L$  lattices formed as the pdf of the system with one refinement (b). The horizon length is chosen to be 3, with  $L = 31$  and  $M = 2$  sections, resulting in 4.95 bit/symbol. The system matrix  $\mathbf{A}$  is taken from Equation 6.12.

Code: `lattice2dtest.m`.

lattice until  $L$  Voronoi cells are located within the first Voronoi cell in its parent lattice, by measuring the distance to the center of the cell. If more than  $L$  Voronoi cells of the sublattice are located within the parent Voronoi cell, the first  $L$  sublattice Voronoi cells are chosen. This can in some cases lead to larger Voronoi cells at the corner of the parent lattice, which is not optimal. More sophisticated and optimal methods to create sublattices exist, and are described in [23]. This is though out of the scope of this thesis.

## 6.3.1 Dithering

The pdf of the system is, as mentioned before, based on Equation (6.3), which includes the quantizer noise,  $\sigma_n^2$ . This is based on the dithering used in a lattice quantizer [23], where some (pseudo-) random noise is added to the signal prior to the quantization, which again is subtracted from the reconstructed signal. Therefore, it requires that the dither signal is known at both the quantizer and decoder. The reconstructed signal with dither is described by

$$\bar{\mathbf{u}}_k = Q(\mathbf{u}_k + \eta_k) - \eta_k, \quad (6.18)$$

where  $\eta_k$  is the dither noise. This dithering noise adds distortion to the quantizer, which can be measured by

$$D_N \approx G(\Lambda)v^{\frac{2}{N}}, \quad (6.19)$$

where  $G(\Lambda)$  is the dimensionless second moment based on the lattice shape and  $v$  is the volume of a Voronoi cell in the lattice of dimension  $N$ , and is found as in [23] to be

$$v = \sqrt{\det(\mathbf{U}_{\text{gram}})} = \det(\mathbf{U}), \quad (6.20)$$

with  $\mathbf{U}$  being the generator matrix and  $\mathbf{U}_{\text{gram}}$  its Gram matrix, defined as  $\mathbf{U}_{\text{gram}} = \mathbf{U}\mathbf{U}^T$ . The more sphere-shaped the lattice becomes and the lower  $G(\Lambda)$  becomes [23]. The  $G(\Lambda)$  values used in this thesis are found in Table 3.1 in [23].

Since

$$Q(\mathbf{u}_k + \eta_k) - \eta_k = \mathbf{u}_k + n_k, \quad (6.21)$$

the noise variance of the quantizer is given by

$$\sigma_n^2 = G(\Lambda)v^{\frac{2}{N}}. \quad (6.22)$$

## 6.3.2 Verification

Figure 6.7 shows a simulation using a lattice dictionary with one refinement. The parameters used in the simulation are identical to those used in Section 6.1.1. A few spikes can be seen in the simulation, but the system stabilises quickly afterwards. The horizon length in this simulation is set to  $N = 3$ , requiring a 3D dictionary. The dictionary is designed using the  $\tilde{A}3$  lattice [22] with Gram matrix

$$\mathbf{U}_{\text{gram},\tilde{A}3} = \begin{bmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 3 \end{bmatrix}, \quad (6.23)$$

with a possible generator matrix

$$\mathbf{U}_{\tilde{A}3} = \text{chol}\left(\mathbf{U}_{\text{gram},\tilde{A}3}\right) = \begin{bmatrix} 1.7321 & -0.5774 & -0.5774 \\ 0 & 1.6330 & -0.8165 \\ 0 & 0 & 1.4142 \end{bmatrix}, \quad (6.24)$$

with [23]

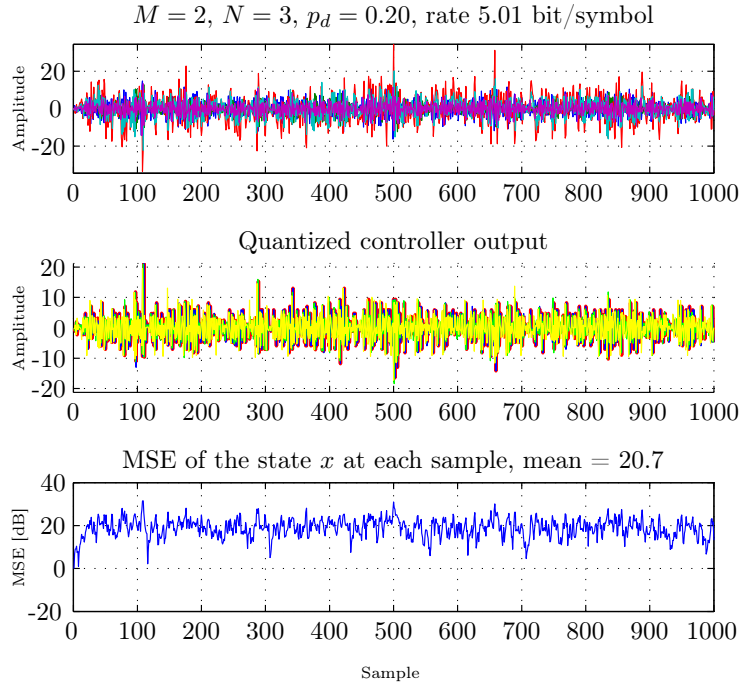
$$G(\Lambda) = 0.0787. \quad (6.25)$$

The Gram matrices and  $G(\Lambda)$  values used in this thesis are found in appendix A.

## 6.4 Dictionaries with variable scaling

In some cases it happens, that the state of the plant obtains high values, which can overload the quantizers with finite support. In this case the plant will keep oscillating and the controller is unable to stabilize the system. This can be avoided partially by adding a fixed gain to the quantizer, increasing the volume of the Voronoi cells, resulting in an increment in the amplitude of the spikes in the state of the plant, which might not be desired. A better solution to this is to add a variable gain to the quantizer. This requires that additional bits to be sent to the decoder (buffer), since the decoder needs to know the added gain. To limit the additional bits needed, it is chosen to only add a total gain to the entire quantizer, not to every vector picked in the quantizer.

Since the gain is variable, it is not known on the decoder and should therefore be transmitted. The scaling used for the dictionary is stored in an integer, calculated from



**Figure 6.7.** Simulation of the lattice dictionary using the  $\tilde{A}3$  lattice and parameters as in Section 6.1.1

Code: `controllerlattice.m`.

logarithmic scaling. The number of bits allocated to store the gain differs, and is simulated in Chapter 7. The gain is stored as

$$G_{\text{dict}} = \frac{\log_2(G_{\text{value}})}{\log_2(G_{\text{base}})}, \quad (6.26)$$

where  $G_{\text{value}}$  is the factor of the gain and  $G_{\text{base}}$  the base of the logarithm. Provided with the number of bits allocated,  $G_{\text{value}}$  is calculated as

$$G_{\text{value}} = G_{\text{base}}^k, \quad k \in \{0, 1, \dots, 2^b - 1\}, \quad (6.27)$$

with  $b$  denoting the number of bits allocated to store the gain. The rate in bit/symbol for the dictionaries using variable gain is given by

$$R = \frac{bM \log_2(L)}{N}. \quad (6.28)$$

The controller Algorithm 5.1 is modified, resulting in Algorithm 6.1, such that the combination of vectors and gains is selected, that minimizes the cost function.

Figure 6.8 shows the first section of a scaled lattice dictionary with 2 bit scaling. Only the first section is shown to illustrate the effect of the scaling, although it is applied to all sections in the dictionary. The lattice is created with 2 sections with 3.9 bit/symbol resulting in  $L = 15$ . This results in 4.9 bit/symbol for the dictionary.

When comparing the dictionary in Figure 6.8 to the dictionary in Figure 6.6(b), it can be seen, that although the lattice consists of fewer points ( $L = 15$  compared to  $L = 31$ ), a larger area is covered by the lattice with  $gain = 8$ . The volume of the Voronoi cells at the higher gain is larger, introducing a larger distortion, but reducing the risk of overloading the dictionary.

---

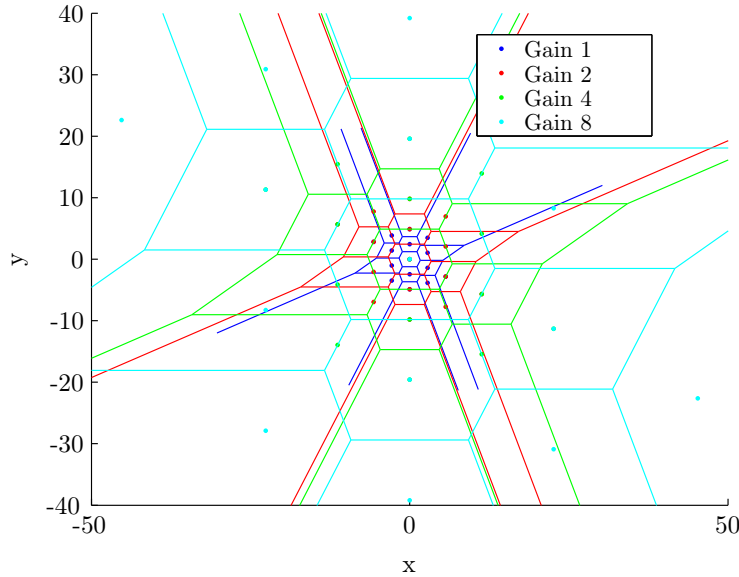
**Algorithm 6.1** Algorithm to do a greedy search on the cost function with different gains.

---

```

1: Dictionary  $\Gamma$ 
2: Input signal  $\mathbf{x}_k$ 
3:  $\beta = \mathbf{0}$  ▷ Initialize  $\beta$  with zeros.
4:  $gains = 2^i, i \in \{0, 1, \dots, 2^{gainBits} - 1\}$ 
5: for  $gain \in gains$  do ▷ Repeat for every gain.
6:   for  $m = 1 \rightarrow M$  do ▷ Repeat for every section.
7:      $\bar{\mathbf{u}} = gain\Gamma\beta$  ▷ Create  $\bar{\mathbf{u}}$  from the already selected vectors.
8:     for  $i = L(m - 1) + 1 \rightarrow Lm$  do
9:        $\mathbf{r} = gain\Gamma\beta + \gamma_i$ 
10:       $res_{gain,i} = \mathbf{r}^T\Phi^T\Phi\mathbf{r} + 2\mathbf{x}_k^T\Upsilon^T\Phi\mathbf{r}$  ▷ Solve cost function for all  $\gamma$  in the
      section.
11:     end for
12:      $g = \arg \min(res)$  ▷ find the vector minimizing the cost function.
13:      $\beta_{gain,g} = 1$ 
14:   end for
15:   Select  $\beta_{\min res}$  ▷ Select the  $\beta$  with smallest resulting cost.
16: end for
    
```

---

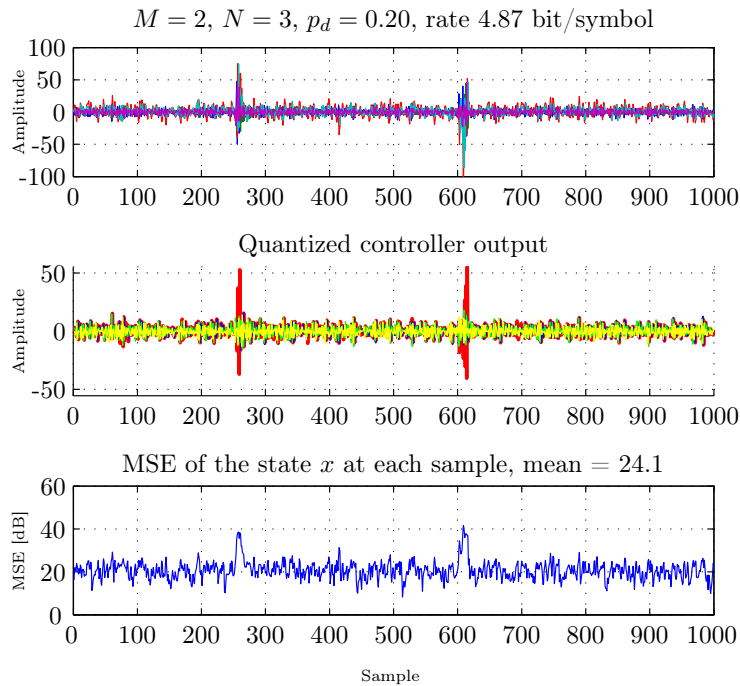


**Figure 6.8.** Scaling of the lattice with 2 bit used to store the gain. The different colors show the gain used. Only the first section is plotted with each gain. The lattice is 3.9 bit/symbol with  $L = 15$ .

Code: `lattice2dtestgain.m`.

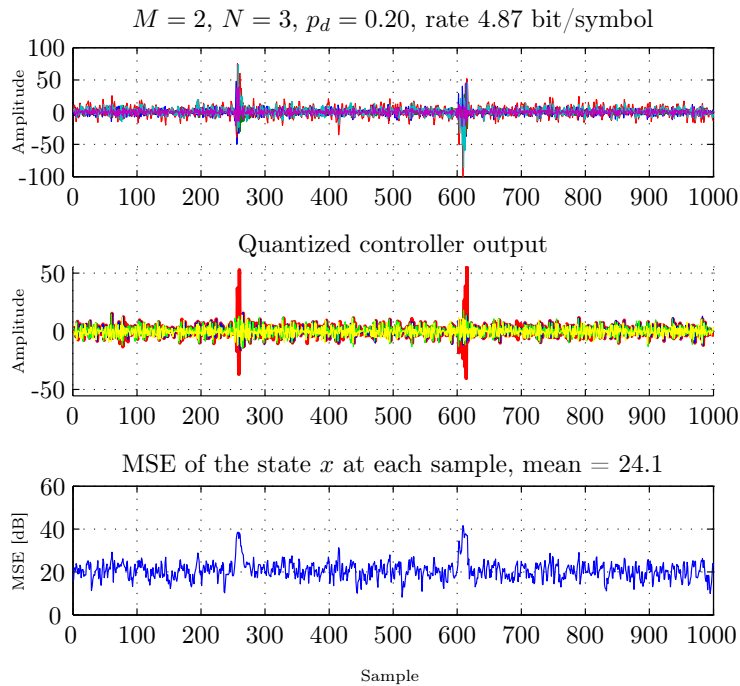
## 6.4.1 Verification

The scaling can be used on both a Gaussian and a lattice dictionary. Figure 6.9 shows the simulation of the Gaussian dictionary with unscaled refinements and a 2 bit gain with a base of 2. Figure 6.10 shows the simulation when using a Gaussian dictionary with scaled refinements as described in Section 6.2 and 2 bit scaling.

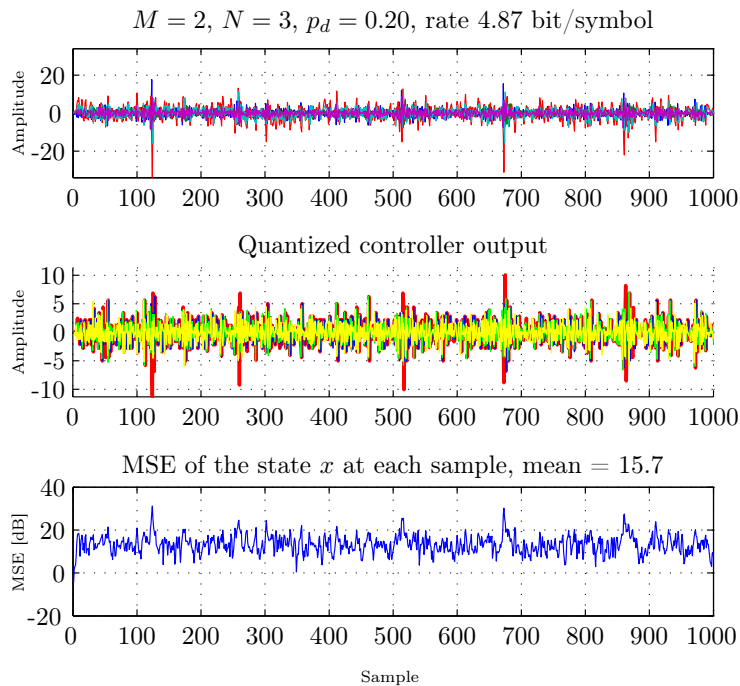


**Figure 6.9.** Simulation of the Gaussian dictionary without scaled refinements and 2 bit gain. The parameters are identical to the simulation in Section 6.1.1, with the rate of the lattice being 4.2 bit/symbol and the gain 2 bit, resulting in a total of 4.87 bit/symbol. Code: `controllergaussiangain.m`.

Figure 6.11 shows a simulation with a lattice dictionary, using the A3 lattice with 2 bit scaling, resulting in 0.67 bit/symbol. The bit rates for the dictionaries are lowered slightly (4.2 bit/symbol compared to 5 bit/symbol) compared to unscaled dictionaries so the total bit rate is maintained, resulting in  $L = 79$  compared to  $L = 182$  for a dictionary with 2 sections and a horizon length of  $N = 3$ . Both dictionaries are able to keep the system stable at the selected bit-rates with a dropout probability of  $p_d = 0.2$ .



**Figure 6.10.** Simulation of the Gaussian dictionary with scaled refinements and 2 bit gain. The parameters are identical to the simulation in Section 6.1.1, with the rate of the lattice being 4.2 bit/symbol and the gain 2 bit, resulting in a total of 4.87 bit/symbol. Code: `controllersectionscalegain.m`.



**Figure 6.11.** Simulation of the lattice dictionary with a 2 bit gain. The parameters are similar to the simulation in section 6.1.1, with the rate of the lattice being 4.2 bit/symbol and the gain 2 bit, resulting in a total of 4.87 bit/symbol. Code: `controllerlatticegain.m`.



## 6.5 Summary

Different methods to design dictionaries have been shown and described in this chapter. The methods are based on either IID Gaussian samples or lattices, that have been shaped according the covariance of the closed loop system. All methods have been simulated with a bit rate around 5 bit/symbol to verify whether the quantizer is able to keep the system stable for 1000 samples.

So far the performance of the quantizer has not been considered, i.e. which quantizer has the best performance while keeping the system stable at the lowest bit rate. This is investigated in Chapter 7.
















# 7 Simulations

---

In this chapter we simulate the different SPARC dictionaries that were presented in Chapter 6. The first simulations focus on optimizing the scaling of the dictionaries in the form of a fixed gain, that is known by both the quantizer and the decoder, and as a variable gain that has to be transmitted. The parameters used in the dictionaries are shown in Appendix B. The results are plotted using MSE, averaged over 3000 samples, in dB on the y-axis, and the non operational bit rate, calculated as in Equation (2.6) on the x-axis in bit/symbol.

## 7.1 Dictionary scaling - fixed gain

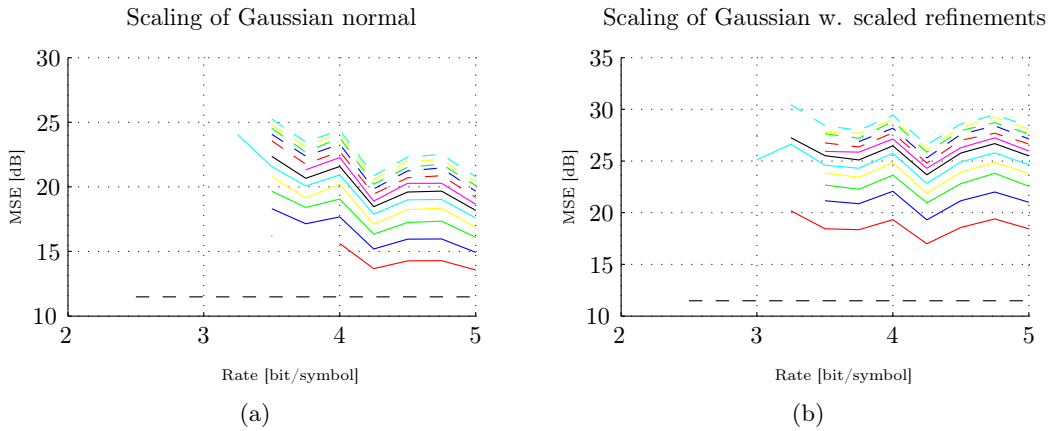
The simulations in this section are used to decide how much the entire dictionary should be scaled. The scaling of the dictionary is applied differently, depending on the dictionary used. This is tested for the three dictionaries described in Chapter 6. In the Gaussian dictionaries, the scaling is applied by increasing the variance  $\sigma_n$  in Equation (6.3), affecting the total system variance and thereby generating IID Gaussian samples with higher variance. In the lattice dictionary, the volume of the Voronoi cells is scaled. The dropout rate  $p_d$  is set to 0.20, and the remaining parameters used in the simulation are described and listed in appendix B. The legend for the figures is explained in table 7.1.

Line color and style	Scale factor
	0.5
	1
	1.5
	2
	2.5
	3
	3.5
	4
	4.5
	5
	5.5
	6
	Reference plant without quantizer

**Table 7.1.** Values used in the simulations with different scaling of the dictionary.

Figures 7.1(a) shows the distortion for different rates for the normal Gaussian dictionary (without scaling on the refinements), which is explained in Section 6.1. The MSE is lowest when the scaling factor is 0.5, but in this case, the system is not stable for bit rates lower than 4 bit/symbol. Increasing the scaling to 1, results in a slightly

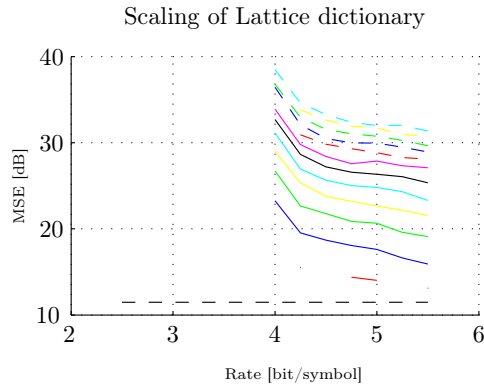
higher MSE, but a stable system at bit rates down to 3.5 bit/symbol. When the scaling is increased to 2.5, the system is stable at 3.25 bit/symbol at the cost of a higher MSE.



**Figure 7.1.** The performance of the Gaussian dictionaries with scaling applied to the entire dictionary. Figure (a) shows the normal Gaussian dictionary with similar scaling in all refinements, whereas Figure (b) shows the Gaussian dictionary with scaled refinements. The legend is explained in Table 7.1.

Code: `plotvardict.m,msemeasvardict.m`.

Figure 7.1(b) shows the Gaussian dictionary with the refinements being scaled as described in Section 6.2. Here the MSE is lowest with a scaling of 0.5 results in the lowest MSE and results in a stable system at bit rates down to 3.25 bit/symbol. A scaling of 2.5 here results in a stable system at 3 bit/symbol as well.



**Figure 7.2.** The performance of the Lattice dictionary with scaling applied to the entire dictionary. The legend is explained in Table 7.1.

Code: `plotvardict.m,msemeasvardict.m`.

Figure 7.2 shows the Lattice dictionary with scaling. The system is maintained stable at a bit rate of 3.5 bit/symbol with a scaling of 1 or higher. The MSE decreases as the bit rate is increased towards 4.25 bit/symbol, after which the system becomes unstable. This behavior is unexpected, and thought to be the result of the very simple algorithm used to create the lattice refinements, which is explained in Section 6.3.

## 7.2 Dictionary scaling - variable gain

In this section we simulate the dictionary with gain, which is explained in Section 6.4. The gain in the dictionary is stored as integers, which are scaled logarithmic to create the gain. We simulate the dictionaries with different logarithmic bases and a different amounts of bits allocated to store the gain, where we will find the combination that results in the best performance. The gain is calculated according to Equation (6.27) in Section 6.4, and is restated here

$$G_{\text{value}} = G_{\text{base}}^k, \quad k \in \{0, 1, \dots, 2^b - 1\}. \quad (7.1)$$

In this simulation the base  $G_{\text{base}}$  and number of bits  $b$  is varied using different bit rates for the quantizers. The bit rate listed on the x-axis is the total bit rate per symbol, which is calculated as

$$R = \frac{bM \log_2(L)}{N}. \quad (7.2)$$

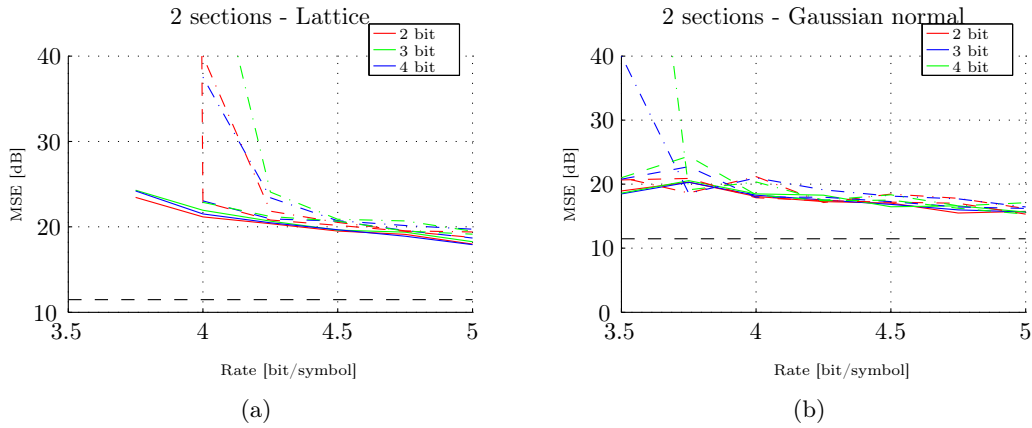
Figure 7.3(a) shows a lattice dictionary using gain. The legend is explained in Table 7.2 and dropout probability  $p_d = 0.20$ . The remaining parameters can be found in Appendix B.

Line color and style	Bits	Base
—	2	2
- -	2	2.5
-.-.	2	3
—	3	2
- -	3	2.5
-.-.	3	3
—	4	2
- -	4	2.5
-.-.	4	3
- -	Reference plant without quantizer	

**Table 7.2.** Values used in the simulations with varying gain.

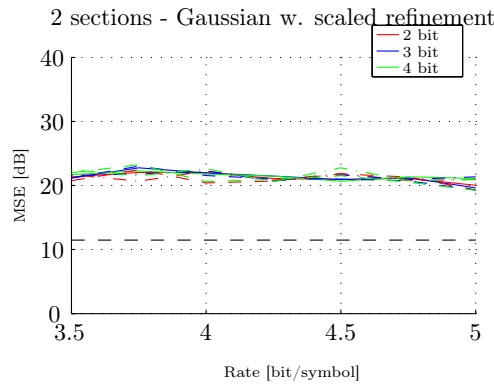
Figure 7.3(b) shows the Gaussian dictionary without scaled refinements, whereas Figure 7.4 shows the Gaussian dictionary with scaled refinements. The lattice dictionary has the best performance with 2 bit designated for the gain and a base of 2. The Gaussian dictionary without refinements in Figure 7.3(b) shows similar results, although the overall MSE is lower. The Gaussian dictionary with scaled refinements, shown in Figure 7.4 shows a mostly similar performance on all parameter combinations. The overall MSE is slightly higher than the unrefined counterpart though.

Figures 7.5(a) and 7.5(b) show the simulations performed on the Gaussian dictionary with, and without scaled refinements, respectively, with  $M = 3$  sections. The lattice dictionary was unable to maintain the system stable in any of the provided parameter sets when  $M = 3$ , and is therefore not shown. Figures 7.5(a) and 7.5(b) illustrate, that



**Figure 7.3.** Simulation of lattice dictionary 7.3(a) and Gaussian dictionary 7.3(b) without scaling on the refinements with varying gain. The full lines show the MSE for a base of 2, the dashed 2.5 and dot-dash 3.

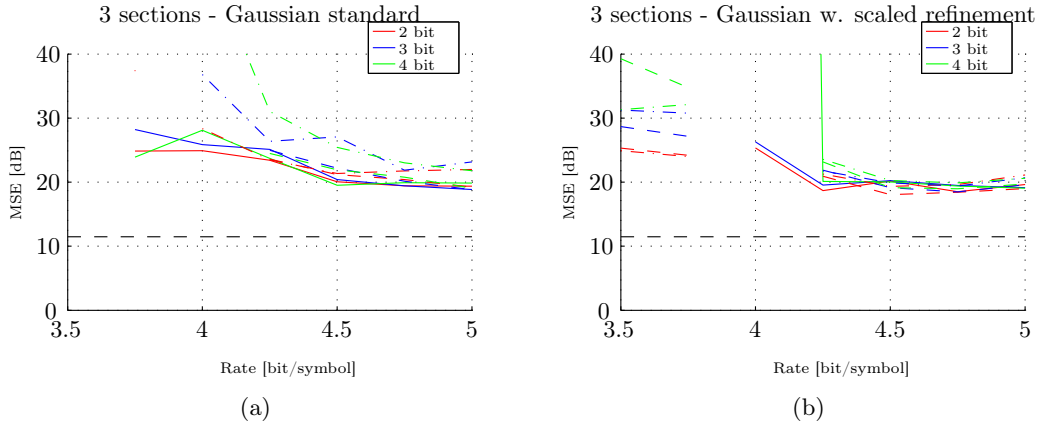
Code: `plotgainbasebits.m,msemeasbit.m`.



**Figure 7.4.** Simulation of the Gaussian dictionary with scaled refinements and varying gain. The full lines show the MSE for a base of 2, the dashed 2.5 and dot-dash 3.

Code: `plotgainbasebits.m,msemeasbit.m`.

the amount of bits used for the gain are not affecting the performance too much for the Gaussian case without scaling (Figure 7.5(a)), as long as the base is 2. In the Gaussian case with refinements (Figure 7.5(b)) the 2 bit and 3 bit gain result in the best performance.



**Figure 7.5.** Simulation of the Gaussian dictionary without 7.5(a) and with 7.5(b) refinements and varying gain. The full lines show the MSE for a base of 2, the dashed 2.5 and dot-dash 3.

Code: `plotgainbasebits.m,msemeasbit.m`.

## 7.3 Dropout probabilities

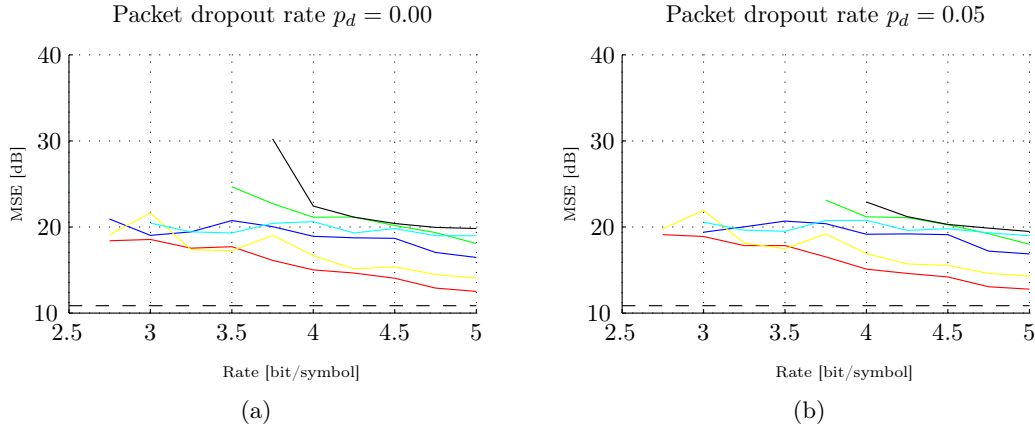
In this section, we run the quantizers with the optimal parameters and test with different dropout probabilities. The legend for the figures in this section are explained in Table 7.3. All simulations are performed with  $M = 2$  sections, a horizon length  $N = 5$ , and the remaining parameters defined as stated in Appendix B.

Line color and style	Dictionary used
— (red)	Normal Gaussian
— (blue)	Gaussian with scaled refinements
— (green)	Lattice
— (yellow)	Normal Gaussian with 2 bit gain
— (cyan)	Gaussian scaled refinements and 2 bit gain
— (black)	Lattice with 2 bit gain
- - (black)	Reference plant without quantizer

**Table 7.3.** Values used in the simulations with different packet dropout rates.

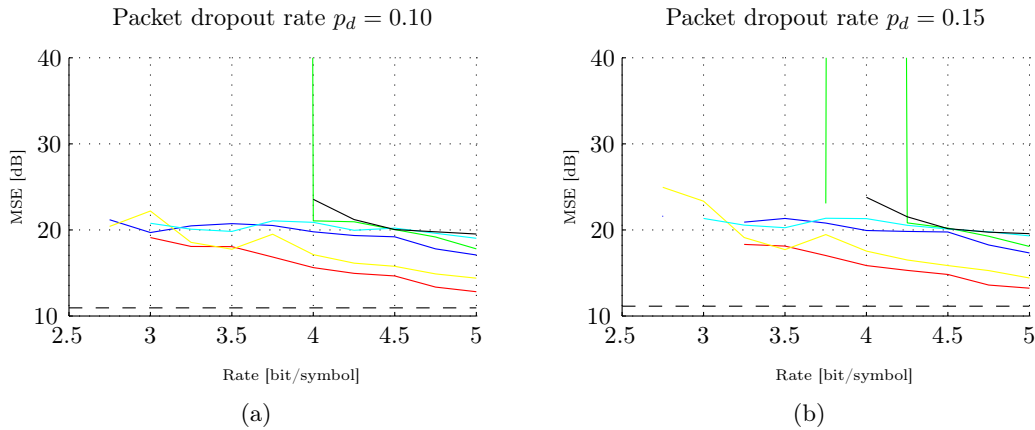
Figures 7.6(a) and 7.6(b) show the MSE for a NCS with no packet dropouts  $p_d = 0$ , and  $p_d = 0.05$ . It is shown, that although all dictionaries are able to maintain the system stable for bit rates of 3.75 bit/symbol at  $p_d = 0$ , the Gaussian dictionaries produce the best rate compared to the distortion. As the rate increases, the distortion of the normal unscaled Gaussian only differs a few dB from the plant without quantizer. When  $p_d = 0.05$  in Figure 7.6(b), the performance is very similar. This is mainly due to the length of the horizon ( $N = 5$ ), which results in a probability of  $3.125 \cdot 10^{-7}$  for five consecutive dropouts occurring.

In Figures 7.7(a) and 7.7(b) the dropout probabilities are increased to  $p_d = 0.10$  and  $p_d = 0.15$ , respectively. While the performance of the Gaussian dictionaries is similar to the results shown in Figures 7.6(a) and 7.6(b), the Lattice dictionaries are unable to



**Figure 7.6.** Simulation of the NCS with  $p_d = 0.00$  (a) and  $p_d = 0.05$  (b) for the different quantizers shown in Table 7.3. The black line illustrates the MSE of the NCS without quantizer. Code: `plotmsedropout.m,msemeasdropout.m`.

stabilize the system at bit rates lower than 4 bit/symbol.

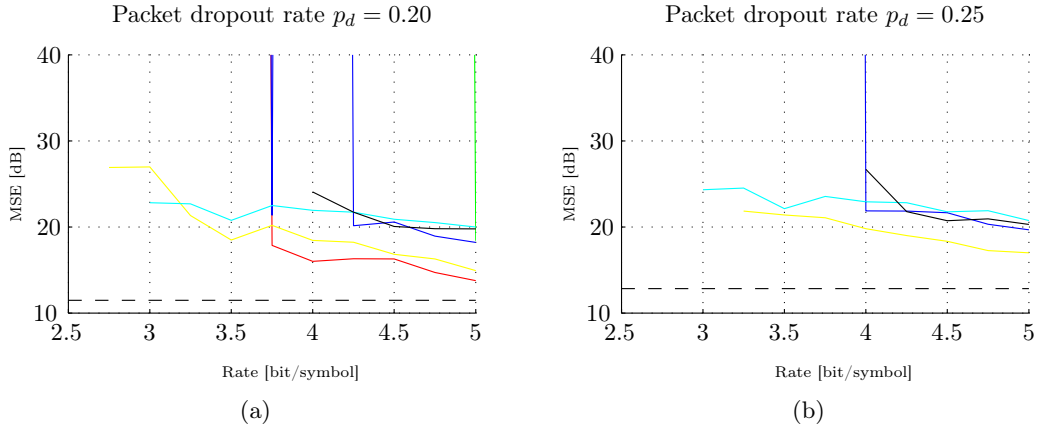


**Figure 7.7.** Simulation of the NCS with  $p_d = 0.10$  (a) and  $p_d = 0.15$  (b) for the different quantizers shown in Table 7.3. The black line illustrates the MSE of the NCS without quantizer. Code: `plotmsedropout.m,msemeasdropout.m`.

By further increasing  $p_d$  to 0.20 (Figure 7.8(a)) and  $p_d = 0.25$  (Figure 7.8(b)), the MSE increases slightly for the normal Gaussian dictionaries, both with and without variable gain, but these are still able to stabilize the PPC at bit rates equal to or higher than 3 bit/symbol. The Gaussian dictionary with variable gain is even stable at 2.75 bit/symbol with  $p_d = 0.20$ . The other dictionaries are unable to stabilize the PPC for bit rates lower than 3.75 bit/symbol. The Lattice dictionaries require yet higher bit rates before they are able to stabilize the system.

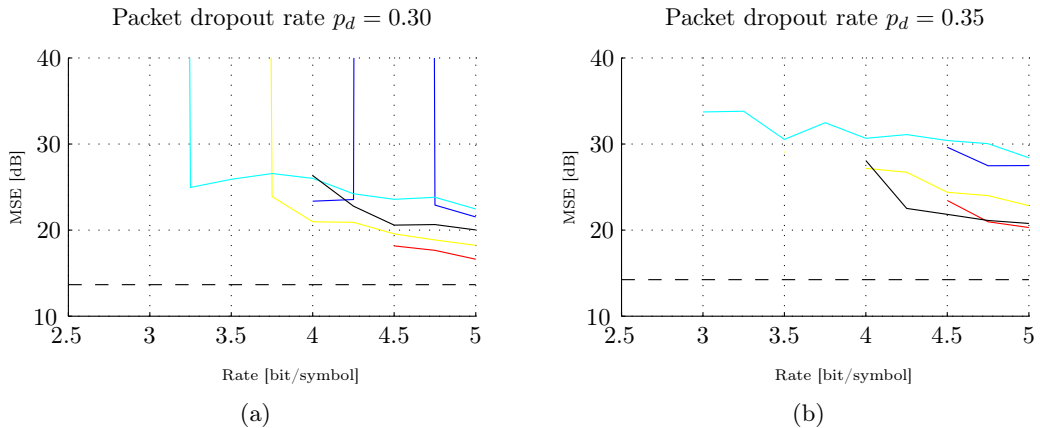
Figures 7.9(a) and 7.9(b) show the results for  $p_d = 0.3$  and  $p_d = 0.35$ , respectively. Here the normal Gaussian dictionary without scaled refinements, with 2 bit variable gain can stabilize the system with 3.25 bit/symbol at  $p_d = 0.3$  and even down to 3 bit/symbol for  $p_d = 0.35$ . At this rate, the probability for 5 consecutive dropouts occurring is 0.0053. The MSE increases with 8 dB compared to the results shown in Figure 7.9(a). When simulating over more than 3000 samples, as is the case in this simulation, the results in





**Figure 7.8.** Simulation of the NCS with  $p_d = 0.20$  (a) and  $p_d = 0.25$  (b) for the different quantizers shown in Table 7.3. The black line illustrates the MSE of the NCS without quantizer. Code: `plotmsedropout.m,msemeasdropout.m`.

Figure 7.9(b) would not be expected to be able to stabilize the system at lower bit rates than in Figure 7.9(a).



**Figure 7.9.** Simulation of the NCS with  $p_d = 0.30$  (a) and  $p_d = 0.35$  (b) for the different quantizers shown in Table 7.3. The black line illustrates the MSE of the NCS without quantizer. Code: `plotmsedropout.m,msemeasdropout.m`.

In general, the Gaussian dictionaries, both with and without scaling applied on the refinements and a 2 bit variable gain had the best performance. These were able to stabilize the NCS with a dropout probability  $p_d$  reaching up to 0.35.

The lattice dictionary resulted in worse performance, which is caused by the fixed size of the Voronoi cells. Since the distribution of the NCS is Gaussian, the best results are obtained when using a dictionary resembling these. The performance of the lattice dictionary can possibly be improved by increasing the volume of the Voronoi cells gradually as they are placed further away from the origin in the dictionary, since the likelihood for the Voronoi cell get selected reduces, the further it is away from the origin. This can for example be done using companding, which is described in [24], but is out of the scope of this thesis and left for future research.

## 7.4 Summary

In this chapter we simulated the different SPARC dictionaries, described in Chapter 6 with varying dropout probabilities and both a fixed and variable gain.

In the first simulations the scaling of the entire dictionary is varied to find the optimal scaling. The gain is fixed, and therefore known for both the quantizer and decoder (buffer), and does therefore not require any information to be sent over the network. The results in general showed, that with  $p_d = 0.20$ , the lower scaling values which resulted in smaller Voronoi cells gave a lower MSE, but were in some cases unable to stabilize the NCS. Since all dictionaries had a good performance with a fixed gain factor of 1, it is chosen not to scale the dictionaries in the remaining simulations performed.

Section 7.2 shows results for the different dictionaries scaled with a variable gain. Since this gain varies, it has to be transmitted to the decoder. We simulated different amounts of bits allocated for the gain which, since the overall bit rate is maintained, reduced the bits allocated for  $\beta$ . This results in a smaller dictionary. Here the Gaussian dictionary with scaled refinements showed similar performance with all gain parameters. The Gaussian dictionary with equal scaling on the refinements and the lattice dictionary showed the best performance with 2 bit allocated for the gain using a base of 2. The dictionaries featuring gain simulated in the next sections therefore use these parameters.

We finally simulated all dictionaries with and without gain applied, using the optimal parameters, found in Sections 7.1 and 7.2, on a NCS with different dropout probabilities. The Gaussian dictionaries featuring gain, showed to provide the best overall performance at higher packet dropout rates. This is due to the increased size of the Voronoi cells when the gain is applied to the dictionary. The performance of the lattice dictionary is worse than with the Gaussian dictionaries, and resulted in a higher distortion compared to the Gaussian dictionaries. It was in addition also was unable to stabilize the system at bit rates lower than 3.5 to 4 bit/symbol. The reason for this is that fixed size of the Voronoi cells in a lattice quantizer do not resemble the Gaussian distribution of the NCS as well as a dictionary containing Gaussian IID vectors.

The performance of the Lattice quantizer can be increased by increasing the volume of the Voronoi cells based on their distance to the origin. This makes a better fit of the Voronoi cells to a Gaussian distribution. Methods for this, such as companding explained in [24], exist and can eventually be applied in future work.

# 8 Alternative dropout scenarios

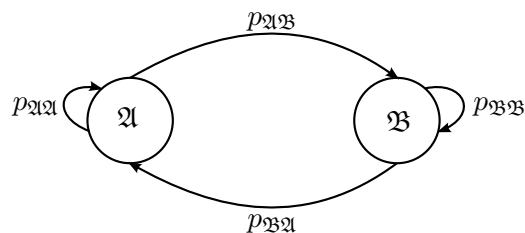
---

We have so far covered a network with a fixed dropout probability  $p_d$ . As soon as this dropout probability gets too high based on the horizon length  $N$  and bit rate, the system gets unstable. This situation does not necessarily describe, what happens in real world applications. In a practical setting a network might operate stable and only loose a few packets most of the time, while the dropout probability increases significantly for short periods due to various reasons, such as bandwidth limitations, busy periods, etc. The scenario is modeled in this chapter after which the design of the dictionaries is described briefly. We analyze the stability of this scenario using MJLS following the basics from [25] and [4]. The following section expands the theory to include systems with different dropout scenarios, that can occur in the network. It is throughout the entire chapter assumed, that the entire system knows, in which state it is. We finally show simulations performed on this network model.

## 8.1 Scenario with two states

The system considered in previous chapters has one state dropout state with a fixed probability  $p_d$ . In this state a dropout occurs with probability  $p_d$  or a packet is received with probability  $1 - p_d$ . When  $p_d$  is increased above a certain probability, the system becomes unstable, which is simulated in Chapter 7. This occurs when multiple dropouts occur consecutively and the number of consecutive dropouts exceed the horizon length  $N$  for longer periods, causing the system to oscillate.

In this section a new model is proposed, which better suits a real-world network with a relatively low dropout probability for most of the time, while for short periods the probability increases drastically, causing the system to become unstable. This system is modelled as a MJLS and is shown in Figure 8.1, with dropout scenarios  $\mathfrak{A}$  and  $\mathfrak{B}$ . The



**Figure 8.1.** Model of a network with two dropout states.

state transition matrix for this system, containing the probabilities to change to the other state, is given as

$$\mathbf{P} = \begin{bmatrix} p_{\mathfrak{A}\mathfrak{A}} & p_{\mathfrak{A}\mathfrak{B}} \\ p_{\mathfrak{B}\mathfrak{A}} & p_{\mathfrak{B}\mathfrak{B}} \end{bmatrix}. \quad (8.1)$$

Each state in figure 8.1 has its own dropout probability  $p_{d,\mathfrak{A}}$  or  $p_{d,\mathfrak{B}}$ .

## 8.2 Dictionary considerations

The dictionary design in Chapter 6 is based on the state distribution of the closed loop system. As found in Section 6.1, the distribution has zero mean and its covariance is repeated here from equation (6.3)

$$\mathbf{Q}_{k+1} = \mathbb{E} \{ \boldsymbol{\Theta}_{k+1} \boldsymbol{\Theta}_{k+1}^T \} = \mathcal{A} \mathbb{E} \{ \boldsymbol{\Theta}_k \boldsymbol{\Theta}_k^T \} \mathcal{A}^T + p_d(1 - p_d) \tilde{\mathcal{A}} \mathbb{E} \{ \boldsymbol{\Theta}_k \boldsymbol{\Theta}_k^T \} \tilde{\mathcal{A}}^T + \mathcal{C}(p_d), \quad (8.2)$$

where the parameter  $\mathcal{C}(p_d)$  is given in Section 6.1. It is clear from  $\mathbf{Q}$  that the behavior of the system depends on the probability  $p_d$  for a packet to be lost.

Under the assumption, that both the controller and the buffer know, whether the network is in state  $\mathfrak{A}$  or  $\mathfrak{B}$ , it is investigated whether the system performs better when there is a different dictionary for each state. The dictionaries differ according to  $\mathbf{Q}$ , which is based on  $p_d$  in the current state, and eventually a different scaling.

In this thesis we assume that both the quantizer and the decoder know in which state the network is, hence no extra information has to be transmitted over the network to indicate this. When the network goes from one state to the other, the quantizer and decoder change to the dictionary generated according to the state of the network.

## 8.3 Stability of Markov Jump Linear Systems

In this section we summarize the basics on Mean Square Stable (MSS) of MJLSs mentioned in [25] and [4].

Consider the model shown in Figure 4.6, with dropout rate  $p_d$ , where the next state described by

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}_1 u_k + \mathbf{B}_2 \omega_k \quad (8.3)$$

$$\mathbf{b}_k = d_k \mathbf{M} \mathbf{b}_{k-1} + (1 - d_k) \mathbf{u}_k, \quad (8.4)$$

with  $d_k$  indicating the state of the system at sample  $k$ . In Figure 4.6,  $d_k = 1$  when a dropout occurred.

For the stability analysis in this section, the system is described as

$$\boldsymbol{\Theta}_{k+1} = \bar{\mathbf{A}}(d_k) \boldsymbol{\Theta}_k + \bar{\mathbf{B}}(d_k) \mathbf{v}_k, \quad (8.5)$$

with

$$\begin{aligned}\bar{\mathbf{A}}(0) &= \begin{bmatrix} \mathbf{A} - \mathbf{B}_1 \mathbf{e}_1^T \mathbf{K} & 0 \\ -\mathbf{K} & 0 \end{bmatrix} & \bar{\mathbf{A}}(1) &= \begin{bmatrix} \mathbf{A} & \mathbf{B}_1 \mathbf{e}_1^T \mathbf{M} \\ 0 & \mathbf{M} \end{bmatrix} \\ \bar{\mathbf{B}}(0) &= \begin{bmatrix} \mathbf{B}_2 & \mathbf{B}_1 \mathbf{e}_1^T \\ 0 & \mathbf{I} \end{bmatrix} & \bar{\mathbf{B}}(1) &= \begin{bmatrix} \mathbf{B}_2 & 0 \\ 0 & 0 \end{bmatrix}\end{aligned}$$

and

$$\Theta_{\mathbf{k}} = \begin{bmatrix} \mathbf{x}_{\mathbf{k}} \\ \mathbf{b}_{\mathbf{k}-1} \end{bmatrix} \quad \mathbf{v}_{\mathbf{k}} = \begin{bmatrix} \omega_{\mathbf{k}} \\ n_{\mathbf{k}} \end{bmatrix}.$$

The lattice quantizer is dithered, as mentioned in Section 6.3.1, such that

$$\bar{\mathbf{u}}_{\mathbf{k}} = Q(\mathbf{u}_{\mathbf{k}} + \eta_{\mathbf{k}}) - \eta_{\mathbf{k}}, \quad (8.6)$$

where  $\eta_{\mathbf{k}}$  is uniformly distributed over the Voronoi cell, and is independent on current and past values of the input signal. In this analysis we assume, that the following linear additive noise model holds as in [4]

$$Q(\mathbf{u}_{\mathbf{k}} + \eta_{\mathbf{k}}) - \eta_{\mathbf{k}} = \mathbf{u}_{\mathbf{k}} + n_{\mathbf{k}}, \quad (8.7)$$

with  $n_{\mathbf{k}}$  distributed as  $-\eta_{\mathbf{k}}$ , such that  $\mathbb{E}\{\|n_{\mathbf{k}}\|\} = \mathbb{E}\{\|\eta_{\mathbf{k}}\|\}$ , and each  $n_{\mathbf{k}}$  is white with zero mean, such that it is independent on  $\mathbf{u}_{\mathbf{k}-1}$ ,  $\forall l \geq 0$ .

When the plant noise  $\omega_{\mathbf{k}}$  and quantizer noise  $v_{\mathbf{k}}$  is white and has bounded variance, it is sufficient to determine whether the system is MSS by observing  $\bar{\mathbf{A}}$ , reducing the model to analyse to

$$\tilde{\Theta}_{\mathbf{k}+1} = \bar{\mathbf{A}}(d_{\mathbf{k}}) \tilde{\Theta}_{\mathbf{k}}. \quad (8.8)$$

In this case the MJLS is MSS according to [25] and [4] if  $\mathcal{E}\{\Theta_{\mathbf{k}} \Theta_{\mathbf{k}}^T\}$  converges as  $\mathbf{k} \rightarrow \infty$  such that, for any initial condition  $\tilde{\Theta}_0 \in \mathbb{R}^n$ ,  $\bar{\mathbf{d}}_{\mathbf{k}0} \in \mathbf{d}_{\mathbf{k}0}$ , there exists a mean  $\boldsymbol{\mu}$  and covariance  $\mathbb{Q}$ , such that

$$\|\boldsymbol{\mu}(k) - \boldsymbol{\mu}\|_2 \rightarrow 0 \text{ as } k \rightarrow \infty \quad (8.9)$$

$$\|\mathbb{Q}(k) - \mathbb{Q}\|_2 \rightarrow 0 \text{ as } k \rightarrow \infty, \quad (8.10)$$

with the covariance

$$\mathbb{Q}(k) = \sum_{i=1}^N \mathbf{Q}_i(k). \quad (8.11)$$

Theorem 1 from [25] shows the necessary and sufficient conditions a system has fulfill to be MSS.

**Theorem 1** [25, Theorem 3.9] *The following assertions are equivalent and contain necessary and sufficient conditions for MSS:*

1. The system (8.8) is MSS.
2. The spectral radius of  $\mathcal{A}$ ,  $r_\sigma(\mathcal{A}) < 1$ .
3. For any positive definite  $\mathbf{S}$  in the positive definite Hilbert  $\mathbb{H}^{n^+}$ , there exists a unique positive definite  $\mathbf{V} \in \mathbb{H}^{n^+}$ , such that

$$\mathbf{V} - \mathcal{T}(\mathbf{V}) = \mathbf{S} \quad (8.12)$$

4. For some positive definite  $\mathbf{V} \in \mathbb{H}^{n^+}$ , we have

$$\mathbf{V} - \mathcal{T}(\mathbf{V}) \succ 0 \quad (8.13)$$

5. For all initial conditions  $d_0 \in \{0, 1\}$  and  $\tilde{\Theta}_0$  with bounded variance, the following holds:

$$\sum_{k=0}^{\infty} E \left\{ \|\tilde{\Theta}_k\|^2 \right\} < \infty \quad (8.14)$$

The proof is found in [25].

The matrix  $\mathcal{A}$  is defined as

$$\mathcal{A} = \begin{bmatrix} \bar{\mathbf{A}}_0 \otimes \bar{\mathbf{A}}_0 & & & \\ & \bar{\mathbf{A}}_1 \otimes \bar{\mathbf{A}}_1 & & \\ & & \ddots & \\ & & & \bar{\mathbf{A}}_{N-1} \otimes \bar{\mathbf{A}}_{N-1} \end{bmatrix} (\mathbf{P}^T \otimes \mathbf{I}_{n^2}), \quad (8.15)$$

which is stable if the spectral radius  $r_\sigma < 1$ .  $\mathbf{P}$  denotes the transition matrix of the Markov chain and  $\bar{\mathbf{A}}_i$  denoting the system state matrix at state  $i$ .

The linear operator  $\mathcal{T}$  is defined by

$$\mathcal{T}_j(\mathbf{V}) = \sum_{i=0}^{N-1} p_{ij} \bar{\mathbf{A}}_i \mathbf{V}_i \bar{\mathbf{A}}_i^T, \quad (8.16)$$

with  $p_{ij}$  denoting the transition probability from state  $i$  to  $j$ .

The covariance and the operator  $\mathcal{T}$  are linked as [25]

$$\mathbf{Q}(k+1) = \mathcal{T}(\mathbf{Q}(k)), \quad (8.17)$$

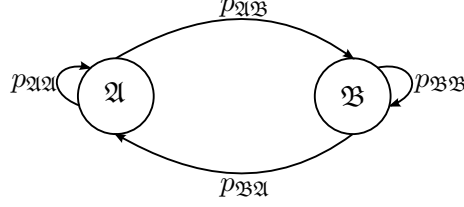
with the total covariance [25]

$$\mathbb{Q}(k) = \sum_{i=1}^N \mathbf{Q}_i(k) = \sum_{i=1}^N \mathcal{T}_i^k(\mathbf{Q}(0)). \quad (8.18)$$

## 8.4 Stability for MJLS with different dropout scenarios

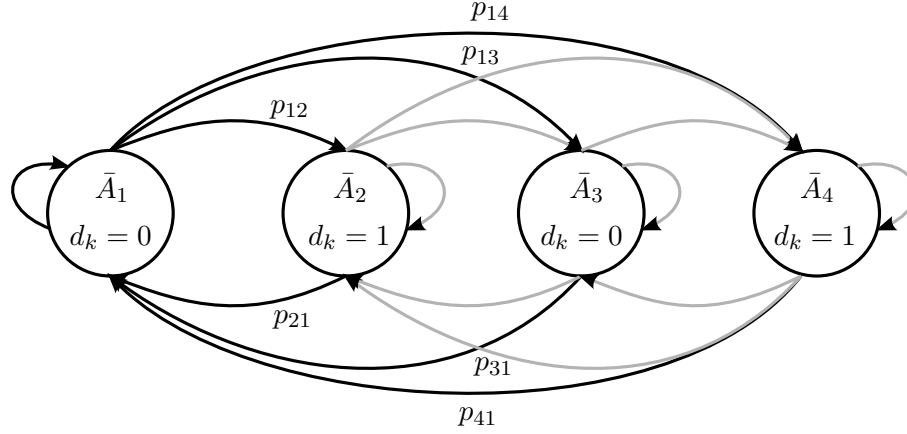
In this section we consider a MJLS with different dropout scenarios. Initially we show a two state dropout scenario where one state models a stable network with a low dropout

probability and one state modeling an unstable network with a high dropout probability. The probability to switch from the stable to the unstable system is larger than zero. The system model is shown in Figure 8.2, where state  $\mathfrak{A}$  and  $\mathfrak{B}$  have dropout probabilities  $p_{d1}$  and  $p_{d2}$ , respectively.



**Figure 8.2.** Markov model showing the two dropout scenarios.

Figure 8.3 includes the states, where a dropout occurs together with the network state. The Markov transition matrix entries are dependent on the dropout rate in the



**Figure 8.3.** Markov model showing the two dropout scenarios with the dropout states. The arrows not going to or from node  $\bar{A}_1$  are colored gray for easier viewing.

current state as well as the probability to switch to the other state and are calculated using normal probability theory as

$$\mathbf{P} = \begin{bmatrix} p_{\mathfrak{A}\mathfrak{A}}(1 - p_{d1}) & p_{\mathfrak{A}\mathfrak{A}}p_{d1} & p_{\mathfrak{A}\mathfrak{B}}(1 - p_{d2}) & p_{\mathfrak{A}\mathfrak{B}}p_{d2} \\ p_{\mathfrak{A}\mathfrak{A}}(1 - p_{d1}) & p_{\mathfrak{A}\mathfrak{A}}p_{d1} & p_{\mathfrak{A}\mathfrak{B}}(1 - p_{d2}) & p_{\mathfrak{A}\mathfrak{B}}p_{d2} \\ p_{\mathfrak{B}\mathfrak{A}}(1 - p_{d1}) & p_{\mathfrak{B}\mathfrak{A}}p_{d1} & p_{\mathfrak{B}\mathfrak{B}}(1 - p_{d2}) & p_{\mathfrak{B}\mathfrak{B}}p_{d2} \\ p_{\mathfrak{B}\mathfrak{A}}(1 - p_{d1}) & p_{\mathfrak{B}\mathfrak{A}}p_{d1} & p_{\mathfrak{B}\mathfrak{B}}(1 - p_{d2}) & p_{\mathfrak{B}\mathfrak{B}}p_{d2} \end{bmatrix}. \quad (8.19)$$

The rows in  $\mathbf{P}$  sum up to 1, since the total probability of leaving a state is equal to 1. The easiest method to check for stability is by using (2) and verify that the spectral radius of  $\mathcal{A} < 1$ , where  $\mathcal{A}$  is defined according to Equation (8.15).

Checking whether point 3 and 4 in Theorem 1 are fulfilled requires solving a set of coupled Lyapunov equations given by point 3 and Equation (8.16), where we solve for each  $j$

$$\mathbf{V}_j = \mathbf{S}_j + p_{1j}\bar{\mathbf{A}}_1\mathbf{V}_1\bar{\mathbf{A}}_1^T + p_{2j}\bar{\mathbf{A}}_2\mathbf{V}_2\bar{\mathbf{A}}_2^T + p_{3j}\bar{\mathbf{A}}_3\mathbf{V}_3\bar{\mathbf{A}}_3^T + p_{4j}\bar{\mathbf{A}}_4\mathbf{V}_4\bar{\mathbf{A}}_4^T, \quad (8.20)$$

with  $\mathbf{S}_j$  being a unique positive definite matrix.

The system matrix  $\mathbf{A}$  and buffer  $\mathbf{b}_k$  do not change in the different dropout scenarios  $\mathfrak{A}$  and  $\mathfrak{B}$ , such that  $\bar{\mathbf{A}}_1 = \bar{\mathbf{A}}_3 = \bar{\mathbf{A}}(0)$  and  $\bar{\mathbf{A}}_2 = \bar{\mathbf{A}}_4 = \bar{\mathbf{A}}(1)$ , reducing Equation (8.20) to

$$\mathbf{V}_j = \mathbf{S}_j + \bar{\mathbf{A}}(0) (p_{1j} \mathbf{V}_1 + p_{3j} \mathbf{V}_3) \bar{\mathbf{A}}^T(0) + \bar{\mathbf{A}}(1) (p_{2j} \mathbf{V}_2 + p_{4j} \mathbf{V}_4) \bar{\mathbf{A}}^T(1), \quad (8.21)$$

which can be solved iteratively, and results in positive definite matrices  $\mathbf{V}_j$  if the system is stable.

Equation (8.18) is used to check for point 5 in Theorem 1. This can be done by using the  $\mathbf{V}_j$  obtained in Equation (8.21) and sum these up. If  $\mathbb{Q}$  is bounded, which requires  $\mathbf{V}_j$  to be bounded for all  $j$ , the system is stable.

If these conditions are satisfied, the MJLS with transition matrix  $\mathbf{P}$  is MSS, such that the state system converges to the desired state.

## 8.5 Verification

In this section we verify the network model with two states. No thorough benchmark is done to fine tune the parameters in this situation. We instead focus on the stability criteria mentioned in Section 8.4 and verify these results on a NCS. The NCS used in the simulations uses the fixed rate quantizer, designed in Chapter 5, with a Gaussian SPARC dictionary containing zero-mean IID vectors with the covariance calculated as in Equation (8.2). We simulate a network with parameters identical to those, listed in Appendix B. Only two simulations are shown due to time constraints while performing these.

### Simulation 1

In this simulation we have the probabilities for a packet dropout in both states  $p_{d\mathfrak{A}} = 0.05$  and  $p_{d\mathfrak{B}} = 0.4$ . The network transition probabilities are  $p_{12} = 0.05$  and  $p_{21} = 0.1$ , which using the transition matrix in Equation (8.19), result in the transition matrix

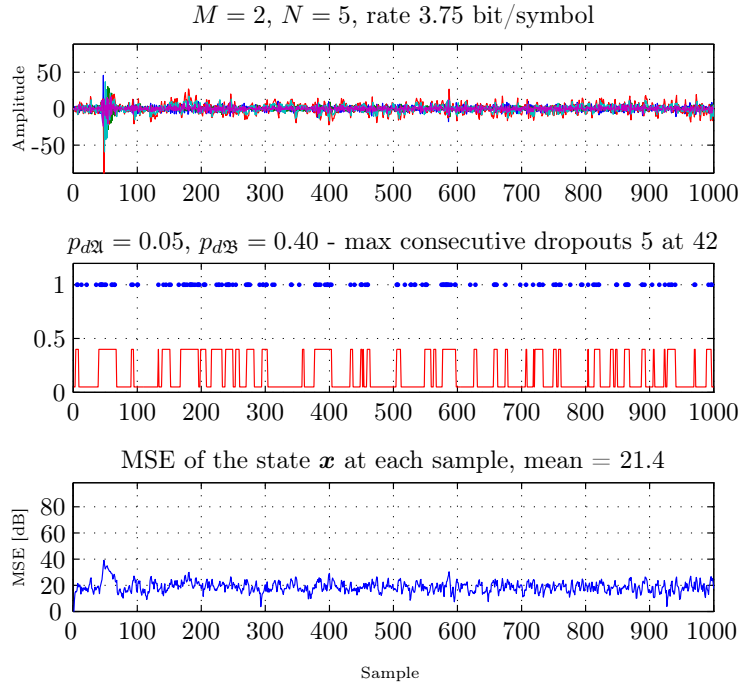
$$\mathbf{P}_1 = \begin{bmatrix} 0.902 & 0.048 & 0.030 & 0.020 \\ 0.902 & 0.048 & 0.030 & 0.020 \\ 0.095 & 0.005 & 0.540 & 0.360 \\ 0.095 & 0.005 & 0.540 & 0.360 \end{bmatrix}. \quad (8.22)$$

The condition from Theorem 1 are verified using the code `stabilityMJLS1.m`. This results in spectral radius of  $\mathcal{A}$ ,  $r_\sigma(\mathcal{A}) = 0.993$ , which is less than 1 and thus satisfying condition 2. The remaining conditions are also satisfied.

Figure 8.4 shows the simulation of the NCS with the network modeled as a two-state MJLS with transition matrix  $\mathbf{P}_1$ . The quantizer used is the Gaussian dictionary where the refinements are scaled, which is explained in Section 6.2, operating at 3.75 bit/symbol.

Only 1000 samples are shown in the simulation for easier viewing. The NCS has been simulated up to 10 000 samples, where the NCS also showed to be stable.





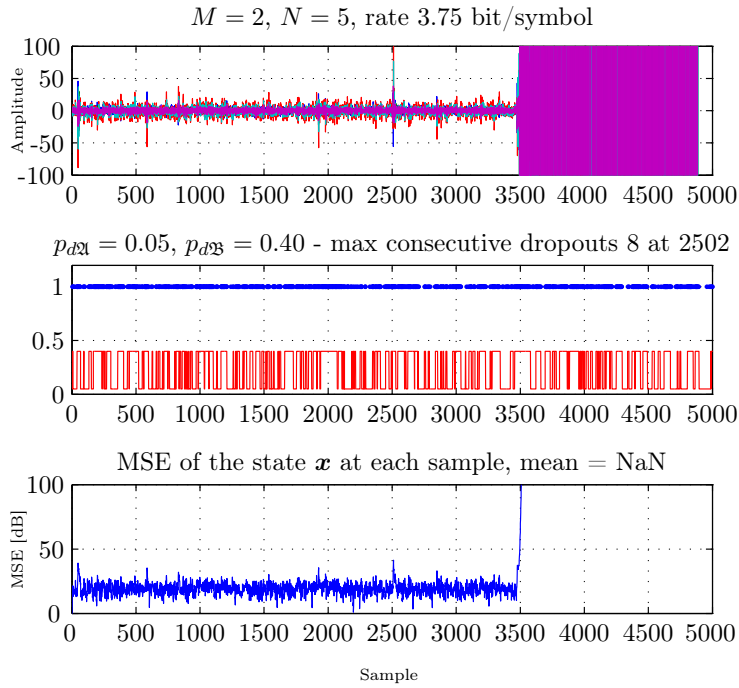
**Figure 8.4.** Simulation of the NCS with the network modeled as a 2-state MJLS with transition matrix  $\mathbf{P}_1$ . The first graph shows the plant state  $\mathbf{x}$  at each sample  $k$ . The second plot the dropout probability  $p_d$  at each sample  $k$ , indicating in which state the network is. The blue dots indicate that a packet is lost at  $k$ . The bottom plot show the MSE. Code: `controller2state1.m`.

#### Simulation 2

In this simulation we have similar parameters as in Simulation 8.5, but change  $p_{21}$  from 0.1 to 0.05, which using the transition matrix in Equation (8.19), result in the transition matrix

$$\mathbf{P}_2 = \begin{bmatrix} 0.902 & 0.048 & 0.030 & 0.020 \\ 0.902 & 0.048 & 0.030 & 0.020 \\ 0.048 & 0.003 & 0.570 & 0.380 \\ 0.048 & 0.003 & 0.570 & 0.380 \end{bmatrix}. \quad (8.23)$$

The condition from Theorem 1 are verified using the code `stabilityMJLS2.m`. This results in spectral radius of  $\mathcal{A}$ ,  $r_\sigma(\mathcal{A}) = 1.047$ , which is not lower than 1. The remaining conditions are neither fulfilled, thus the system should not be stable. To verify this, we simulated 10 000 samples. The results are shown in Figure 8.5 shows the stability results for the MJLS, which according to the theory should not be stable.



**Figure 8.5.** Simulation of the NCS with the network modeled as a 2-state MJLS with transition matrix  $\mathbf{P}_2$ , which is unstable. The first graph shows the plant state  $\mathbf{x}$  at each sample  $k$ . The second plot the dropout probability  $p_d$  at each sample  $k$ , indicating in which state the network is. The blue dots indicate that a packet is lost at  $k$ . The bottom plot show the MSE.

Code: `controller2state2.m`.

## 8.6 Summary

In this chapter we illustrated an alternative network setup compared to the default setup that is used in previous chapters. This setup gives a more real-world impression of the network, which, most of the time, is operating stable featuring low packet dropouts, while it for short periods can feature high packet dropout rates due to e.g. overload or instability.

We proposed to design two dictionaries, one for each state of the network. This is based on the assumption that both the controller and the plant know the network state.

Stability criteria based on MJLS has been analyzed and modified to fit the NCS. These have briefly been verified in two simulations, showing that the system also is stable in the simulation, when the theory tells it is stable. This theory can be used to verify systems in NCS situations, where busy periods or instabilities on the network are expected for short time periods, and can verify if the system will be stable without the need for simulations.

# 9 Conclusions

---

In this thesis we implemented Sparse Regression Code (SPARC) for Networked Control System (NCS)s using fixed rate vector quantizers based on previous work in [4].

Two different methods to perform dictionary search on Gaussian dictionaries have been investigated and modified to operate with SPARC. Since no optimal solvers exist to perform a dictionary search sub-optimal solvers, based on greedy methods have been applied. The MP algorithm showed good performance on SPARC and low bit rates match a regular Gaussian dictionary. The homotopy continuation algorithm showed to perform well on Gaussian dictionaries. This algorithm was modified to solve the Integer Programming (IP) problem using a relaxation approximation on the cost function. Unfortunately it did not provide the desired results, which is most likely due to the non-convexity of the cost function.

The receding horizon controller is explained, and combined with the quantizer. After discussions on the advantages and disadvantages of integrating the quantizer directly into the controller, it showed to be advantageous to integrate the quantizer and solve the cost function directly from the finite set using a greedy algorithm.

In this thesis we consider two Gaussian SPARC dictionaries, containing IID random generated vectors, where the sections one dictionary are of are scaled versions of the previous sections as well as a Lattice SPARC dictionary where the sections consists of a sub-lattice of the Voronoi cells. During the design of the dictionaries for fixed rate vector quantizers caution has to be exercised that the state of the NCS can not get out of reach of the quantizer, which would cause it to overload. Gaussian dictionaries resembling the distribution of the NCS were expected to provide good results. Lattice dictionaries were expected to perform better, since the Voronoi cells have the same volume, making the quantizer able to quantize values within the set, with lower MSE than the Gaussian dictionaries.

Simulations of the dictionaries with fixed and variable gain on a NCS with different dropout rates showed that the Gaussian dictionaries perform better than the lattice dictionary, and were able to stabilize the NCS at bit rates down to 2.75 bit/symbol for dropout rates up to 0.20. The poor performance of the lattice dictionary is due to the shape of the Voronoi cells.

After simulations of the different dictionaries and parameters, the network model is altered such that it can switch between two states, featuring different dropout rates. The network is modeled using Markov Jump Linear System (MJLS), and criteria have been described to verify whether the system with the given parameters is Mean Square Stable (MSS). The system has been simulated with a different quantizer for each state, and has been compared to the theory. With this model we are able to simulate more realistic networks, with high packet dropout rates for short periods. We design the quantizers individually, which results in a stable system with a low MSE during the stable periods

and a higher MSE in the periods with high dropout rates.

Throughout this thesis we have designed a fixed rate vector quantizer based on SPARC dictionaries, and have shown that these can maintain the NCS used in the thesis, stable with a bit rate of 2.75 bit/symbol, while the MSE converges towards the MSE of the unquantized NCS as the bit rate increases.

# 10 Future research

---

In this thesis a quantizer using SPARC has been designed, for which multiple dictionaries have been implemented and tested. To stay within the scope of this thesis, the performance of the dictionaries is only tested using a single system dependent system matrix  $\mathbf{A}$ . Since increasing the spectral radius of the system matrix results in a more unstable system which oscillates faster when operating in open loop, it would affect the design, scaling and bit rate of the dictionary significantly. The reason for this is that faster oscillations overload the dictionary faster. When continuing work on this, it is relevant to specify conditions for the dictionaries and bit rates under which the system with a given spectral radius and dropout rate can be maintained stable.

The lattice dictionary designed has a decent performance. One weakness of this dictionary is, that all Voronoi cells have the same volume. The system is, depending on the packet dropout rate, mostly operating with small controller outputs, thus operating in the center of the dictionary. Since the lattice Voronoi cells as well as the refinements all are of equal size, the quantization noise is fixed, and results in a higher MSE when operating in closed loop (without dropouts). It would here be of interest to scale the size of the Voronoi cells, making them grow, the further they are away from the origin. This can for example be done using companding, which is described in [24]. Companding uses a mapping on the input signal before it is quantized, after which the inverse mapping is applied in the decoder. This results in smaller Voronoi cells in the center of the dictionary, reducing the MSE, while the outer Voronoi cells are larger, reducing the likelihood to overload the dictionary.

We have investigated networks which can switch between a good state featuring low dropout rates and a bad state with a high number of dropouts occurring. Theory to verify whether the system with the given transition probabilities and dropout probabilities is MSS has been described. In future work, this could be expanded to analyze an arbitrary number of network states.

---



# Bibliography

---

- [1] D. Quevedo, J. Ostergaard, E. Silva, and D. Netic, "Correction to "packetized predictive control of stochastic systems over bit-rate limited channels with packet loss"," *Automatic Control, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2013.
- [2] M. Nagahara, D. Quevedo, and J. Ostergaard, "Packetized predictive control for rate-limited networks via sparse representation," *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pp. 1362–1367, 2012.
- [3] D. E. Quevedo and D. Netic, "Robust stability of packetized predictive control of nonlinear systems with disturbances and markovian packet losses," *Automatica*, vol. 48, no. 8, pp. 1803 – 1811, 2012. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0005109812002191>
- [4] D. Quevedo, J. Ostergaard, and D. Netic, "Packetized predictive control of stochastic systems over bit-rate limited channels with packet loss," *Automatic Control, IEEE Transactions on*, vol. 56, no. 12, pp. 2854 –2868, dec. 2011.
- [5] J. Ostergaard and D. Quevedo, "Multiple descriptions for packetized predictive control over erasure channels," in *Control and Automation (ICCA), 2011 9th IEEE International Conference on*, 2011, pp. 165–170.
- [6] D. Quevedo and D. Netic, "Input-to-state stability of packetized predictive control over unreliable networks affected by packet-dropouts," *Automatic Control, IEEE Transactions on*, vol. 56, no. 2, pp. 370–375, 2011.
- [7] G. Pin and T. Parisini, "Networked predictive control of uncertain constrained nonlinear systems: Recursive feasibility and input-to-state stability analysis," *Automatic Control, IEEE Transactions on*, vol. 56, no. 1, pp. 72–87, 2011.
- [8] D. E. Quevedo, G. C. Goodwin, and J. A. De Doná, "Finite constraint set receding horizon quadratic control," *International Journal of Robust and Nonlinear Control*, vol. 14, no. 4, pp. 355–377, 2004. [Online]. Available: <http://dx.doi.org/10.1002/rnc.887>
- [9] W. Zhang, M. Branicky, and S. Phillips, "Stability of networked control systems," *Control Systems, IEEE*, vol. 21, no. 1, pp. 84–99, 2001.
- [10] M. J.M., *Predictive Control with Constraints*. Prentice-Hall, 2002, japanese translation published by Tokyo Denki University Press, 2005. "Best Control Engineering Textbook" prize, SICE, 2007.
- [11] K. J. Åström & Björn Wittenmark, *Computer Controlled Systems - Theory and Design*. Prentice-Hall, 1984.

- [12] K. J. Åström and Richard M. Murray, *Feedback Systems - An introduction for scientists and Engineers*. Princeton University Press, 2008.
- [13] J. Rawlings and D. Mayne, *Model Predictive Control: Theory and Design*. Nob Hill Publishing, 2009. [Online]. Available: [http://books.google.dk/books?id=3\\_rfQQAACAAJ](http://books.google.dk/books?id=3_rfQQAACAAJ)
- [14] R. Venkataramanan, A. Joseph, and S. Tatikonda, "Gaussian rate-distortion via sparse linear regression over compact dictionaries," in *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, July 2012, pp. 368–372.
- [15] R. Venkataramanan, T. Sarkar, and S. Tatikonda, "Lossy compression via sparse linear regression: Computationally efficient encoding and decoding," *CoRR*, vol. abs/1212.1707, 2012.
- [16] D. Malioutov, M. Cetin, and A. Willsky, "Homotopy continuation for sparse signal representation," in *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, vol. 5, March 2005, pp. v/733–v/736 Vol. 5.
- [17] R. Veldhuis and M. Breeuwer, *An introduction to source coding*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.
- [18] B. Girod. (2001) Rate-distortion theory. Downloaded 04-06.2013. [Online]. Available: <http://www.stanford.edu/class/ee368b/Handouts/04-RateDistortionTheory.pdf>
- [19] Mathworks. (2013) Matching pursuit algorithms. Accessed 15-05.2013. [Online]. Available: <http://www.mathworks.se/help/wavelet/ug/matching-pursuit-algorithms.html>
- [20] Wikipedia. (2013) Matching pursuit. Accessed 15-05.2013. [Online]. Available: [http://en.wikipedia.org/wiki/Matching\\_pursuit](http://en.wikipedia.org/wiki/Matching_pursuit)
- [21] (2007) Accessed 03-06.2013. [Online]. Available: <http://www.stanford.edu/class/ee392o/subgrad.pdf>
- [22] G. Nebe and N. Sloane. Lattices, a catalogue of lattices. Accessed 30-05.2013. [Online]. Available: <http://www.math.rwth-aachen.de/~Gabriele.Nebe/LATTICES/index.html>
- [23] J. Østergaard, "Multiple-description lattice vector quantization," *CoRR*, vol. abs/0707.2482, 2007.
- [24] T. Linder, R. Zamir, and K. Zeger, "High-resolution source coding for non-difference distortion measures: multidimensional companding," *Information Theory, IEEE Transactions on*, vol. 45, no. 2, pp. 548–561, 1999.



- [25] O. Costa, M. Fragoso, and R. Marques, *Discrete-Time Markov Jump Linear Systems*, C. H. J. Gani, P.Jagers, and T. Kurtz, Eds. Springer, 2005.
- [26] W. Jagy. (2012) Accessed 03-06.2013. [Online]. Available: <http://math.stackexchange.com/questions/259069/how-do-you-construct-a-lattice-from-its-basis-or-its-gram-matrix>



# Appendix



# A Lattice Gram matrices

---

This appendix lists the Gram matrices used to construct the lattice dictionary for different horizon lengths  $N$ . All lattices are found in [22] and the  $G(\Lambda)$  values are found in Table 3,3 in [23]. The Gram matrices are defined as

$$\mathbf{U}_{\text{gram}} = \mathbf{U}^T \mathbf{U}. \quad (\text{A.1})$$

One of many generator matrices can be found using the Cholesky factorization or the matrix square root [26]. In this thesis the Cholesky decomposition is used, since it results in sparse matrices featuring fewer decimals.

A2 lattice

$$\mathbf{U}_{\text{gram,A2}} = \begin{bmatrix} 2.0 & -1.0 \\ -1.0 & 2.0 \end{bmatrix} \quad (\text{A.2})$$

$$G(\Lambda) = 0.0802 \quad (\text{A.3})$$

$\tilde{A}3$  lattice

$$\mathbf{U}_{\text{gram,\tilde{A}3}} = \begin{bmatrix} 3.0 & -1.0 & -1.0 \\ -1.0 & 3.0 & -1.0 \\ -1.0 & -1.0 & 3.0 \end{bmatrix} \quad (\text{A.4})$$

$$G(\Lambda) = 0.0787 \quad (\text{A.5})$$

D4 lattice

$$\mathbf{U}_{\text{gram,D4}} = \begin{bmatrix} 2.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 2.0 & -1.0 & 0.0 \\ 1.0 & -1.0 & 2.0 & -1.0 \\ 0.0 & 0.0 & -1.0 & 2.0 \end{bmatrix} \quad (\text{A.6})$$

$$G(\Lambda) = 0.0766 \quad (\text{A.7})$$

$\tilde{D}5$  lattice

$$\mathbf{U}_{\text{gram,\tilde{D}5}} = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.5 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.5 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 & 1.2 \end{bmatrix} \quad (\text{A.8})$$

$$G(\Lambda) = 0.0756 \quad (\text{A.9})$$

E6 lattice

$$\mathbf{U}_{\text{gram,E6}} = \begin{bmatrix} 2.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ -1.0 & 2.0 & -1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -1.0 & 2.0 & -1.0 & 0.0 & -1.0 \\ 0.0 & 0.0 & -1.0 & 2.0 & -1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & -1.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & -1.0 & 0.0 & 0.0 & 2.0 \end{bmatrix} \quad (\text{A.10})$$

$$G(\Lambda) = 0.0743 \tag{A.11}$$

$\tilde{E}7$  lattice

$$\mathbf{U}_{\text{gram}, \tilde{E}7} = \begin{bmatrix} 3.0 & 4.0 & 5.0 & 6.0 & 4.0 & 2.0 & 3.0 \\ 4.0 & 8.0 & 10.0 & 12.0 & 8.0 & 4.0 & 6.0 \\ 5.0 & 10.0 & 15.0 & 18.0 & 12.0 & 6.0 & 9.0 \\ 6.0 & 12.0 & 18.0 & 24.0 & 16.0 & 8.0 & 12.0 \\ 4.0 & 8.0 & 12.0 & 16.0 & 12.0 & 6.0 & 8.0 \\ 2.0 & 4.0 & 6.0 & 8.0 & 6.0 & 4.0 & 4.0 \\ 3.0 & 6.0 & 9.0 & 12.0 & 8.0 & 4.0 & 7.0 \end{bmatrix} \tag{A.12}$$

$$G(\Lambda) = 0.0731 \tag{A.13}$$

$E8$  lattice

$$\mathbf{U}_{\text{gram}, E8} = \begin{bmatrix} 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ -1.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -1.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -1.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & -1.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 1.0 & 0.0 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{bmatrix} \tag{A.14}$$

$$G(\Lambda) = 0.0717 \tag{A.15}$$

# B Parameters for simulation

---

This appendix lists the parameters used for the simulations in this thesis.

The system matrix  $\mathbf{A} \in \mathbb{R}^{5 \times 5}$  is randomly generated to be

$$\mathbf{A} = \begin{bmatrix} -0.7578 & -0.3245 & -0.085337 & 0.060403 & -2.2557 \\ 0.43212 & -0.35593 & 0.0024123 & 0.0071095 & -0.17091 \\ -0.17328 & 1.0627 & 0.36569 & 0.67106 & 0.93852 \\ 0.95123 & 0.66704 & 0.73738 & -0.43393 & 0.35231 \\ 1.0536 & 0.48356 & -0.15787 & 0.45405 & -0.26378 \end{bmatrix}, \quad (\text{B.1})$$

and the input matrix, as well as the noise matrices are given by

$$\mathbf{B}_1 = \mathbf{B}_2 = [1 \ 1 \ 1 \ 1 \ 1]^T \quad (\text{B.2})$$

The LQR weighting parameter  $\mathbf{Q} \in \mathbb{R}^{5 \times 5}$  is set to be

$$\mathbf{Q} = \text{diag}\{1, \dots, 1\}. \quad (\text{B.3})$$

The parameter  $R = 1$ .