

NORTHWESTERN UNIVERSITY

MIMO Communications with Reduced-Rank Filter Estimation and  
Bi-Directional Training

A PROJECT REPORT

SUBMITTED TO THE GRADUATE SCHOOL  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

MASTER OF SCIENCE

Field of Electrical Engineering

By

Zikun Tan

Advisor: Prof. Michael Honig

EVANSTON, ILLINOIS

March 2021

## **Abstract**

Due to the rapid development of Information and Communications Technology (ICT), the research of optimizing the performance of communications systems has gained much popularity. Suppression of interference and noise in communications systems is one such subject of intense research. Many scholars apply conventional signal processing at receivers to suppress interference and noise, and extract the desired signal. In this research project, two signal processing methods, reduced-rank filtering and bi-directional training, are used to meet this requirement.

Reduced-rank filtering constrains the received signal in a lower dimensional subspace to reduce the feedback requirement. Under some situations, it outperforms full-rank filtering when the training length is limited to a certain range, in terms of the Mean Square Error (MSE) or Signal-to-Interference-plus-Noise Ratio (SINR).

Bi-directional training is for updating the precoder and the receive filter. In this process, training symbol sequences are transmitted between the transmitter and the receiver backwards and forwards, therefore both the precoder and the receive filter are adapted iteratively.

This research project sits between the two aforementioned methods to design a Multiple-Input Multiple-Output (MIMO) communications system. Advantages are demonstrated via experimental simulation.

# Acknowledgement

An unforgettable stage in my life finally walks to the end. When I look back, I just realize how many people I have met and how much I have been helped. Hereby I would like to express my sincere gratitude to all those who were with me over such one year and a half.

First of all, I would like to give my special appreciation to my advisor, Prof. Michael Honig, for his conscientious academic guidance on my project research and report writing. He was always able to make time for me whenever I needed for commenting and discussing the project work, even in difficult times during the pandemic. Also, I feel deeply indebted for his valuable recommendations and suggestions for my further education in the realm of signal processing and communications. It is undoubtedly a great pleasure to do MS study under his supervision.

Meanwhile, I am extremely grateful to my father, Jianbo Tan, and my mother, Hui Fan, for their unsparing support on my MS study abroad. Despite the long distance that separates us across the whole Pacific Ocean, their unconditional love has been an indispensable source of motivation and strength for me to go always on.

Additionally, I am very thankful to all of my course instructors and classmates, for the nice academic atmosphere they create. Learning in Northwestern University and living in the City of Evanston for one year and a half would certainly be an indelible life experience for me.

# Contents

<b>List of Figures</b>	<b>iii</b>
<b>List of Abbreviations</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
1.1 General Introduction . . . . .	1
1.2 Literature Review . . . . .	3
1.3 Industrial Relevance . . . . .	4
1.3.1 Satellite Communications . . . . .	5
1.3.2 Imaging Radars . . . . .	5
<b>2 Theoretical Background</b>	<b>6</b>
2.1 Basic Models of Communications Systems . . . . .	6
2.2 Interference and Noise . . . . .	8
2.2.1 Interference . . . . .	8
2.2.2 Noise . . . . .	8
2.3 Beamforming Techniques . . . . .	8
2.4 Multi-Antenna Systems . . . . .	10
2.5 Transmission Control Schemes . . . . .	13
2.5.1 FDD . . . . .	13
2.5.2 TDD . . . . .	14
2.6 Coherence Time . . . . .	15
2.7 Centralized and Distributed Configurations . . . . .	15
2.7.1 Centralized Configurations . . . . .	15
2.7.2 Distributed Configurations . . . . .	16
2.8 Channel Capacity . . . . .	16
<b>3 Methodology and Algorithms</b>	<b>18</b>
3.1 Wiener (optimal) Estimation . . . . .	18
3.2 Least Squares (LS) Estimation . . . . .	26
3.3 Reduced-Rank Filtering . . . . .	29

3.4	Bi-Directional Optimization and Training . . . . .	33
3.4.1	Bi-Directional Optimization . . . . .	35
3.4.2	Bi-Directional Training . . . . .	37
<b>4</b>	<b>Experiments and Results</b>	<b>40</b>
4.1	Experimental Platform . . . . .	41
4.2	Uni-Directional Optimization and Training . . . . .	41
4.3	Bi-Directional Optimization and Training . . . . .	57
<b>5</b>	<b>Discussion</b>	<b>74</b>
5.1	General Discussion . . . . .	74
5.2	Future Works . . . . .	75
<b>6</b>	<b>Conclusion</b>	<b>76</b>
	<b>References</b>	<b>77</b>
	<b>Appendices</b>	<b>82</b>
<b>A</b>	<b>Main functions</b>	<b>83</b>
<b>B</b>	<b>Self-defined functions</b>	<b>103</b>

# List of Figures

2.1	Basic model of communications systems . . . . .	6
2.2	Model of electronic communications systems . . . . .	7
2.3	Conventional omni-directional antenna array vs Beamformed antenna array . . . . .	9
2.4	Single-input single-output systems . . . . .	11
2.5	Single-input multi-output systems . . . . .	11
2.6	Multi-input single-output systems . . . . .	12
2.7	Multi-input multi-output systems . . . . .	12
2.8	Frequency division duplex . . . . .	14
2.9	Time division duplex . . . . .	14
2.10	Centralized configuration . . . . .	15
2.11	Distributed configuration . . . . .	16
3.1	Digital estimator . . . . .	19
3.2	MIMO communications systems with precoding and receive filtering . . . . .	20
3.3	Multi-Stage Wiener Filter [1] . . . . .	30
3.4	TDD data frames in bi-directional training . . . . .	34
4.1	Transmitted training symbol sequence . . . . .	41
4.2	MMSE vs background SNR with the full-rank receiver, with the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8 and the number of transmitted signal stream $N_s$ 1 . . . . .	42
4.3	SINR vs background SNR with the full-rank receiver, with the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8 and the number of transmitted signal stream $N_s$ 1 . . . . .	43
4.4	MMSE vs background SNR with the full-rank receiver, with the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8 and the number of transmitted signal streams $N_s$ 8 . . . . .	44

4.5	SINR vs background SNR with the full-rank receiver, with the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8 and the number of transmitted signal streams $N_s$ 8 . . . . .	45
4.6	MMSE vs background SNR with reduced-rank receivers, with the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8, the number of transmitted signal streams $N_s$ 8 and ranks 2 and 4 . . . . .	46
4.7	SINR vs background SNR with reduced-rank receivers, with the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8, the number of transmitted signal streams $N_s$ 8 and ranks 2 and 4 . . . . .	47
4.8	MSE vs training length with the full-rank receiver, with the transmit power of each transmit antenna $P$ 100, the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8, the number of transmitted training symbol sequence $N_s$ 1, the length of each training symbol sequence $L$ from 8 to 400 with increment 8 and the number of trials for plotting the averaged training result $N_a$ 20 . . . . .	49
4.9	SINR vs training length with the full-rank receiver, with the transmit power of each transmit antenna $P$ 100, the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8, the number of transmitted training symbol sequence $N_s$ 1, the length of each training symbol sequence $L$ from 8 to 400 with increment 8 and the number of trials for plotting the averaged training result $N_a$ 20 . . . . .	50
4.10	MSE vs training length with the full-rank receiver, with the transmit power of each transmit antenna $P$ 100, the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8, the number of transmitted training symbol sequences $N_s$ 8, the length of each training symbol sequence $L$ from 8 to 400 with increment 8 and the number of trials for plotting the averaged training result $N_a$ 20 . . . . .	51
4.11	SINR vs training length with the full-rank receiver, with the transmit power of each transmit antenna $P$ 100, the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8, the number of transmitted training symbol sequences $N_s$ 8, the length of each training symbol sequence $L$ from 8 to 400 with increment 8 and the number of trials for plotting the averaged training result $N_a$ 20 . . . . .	52

4.12	MSE vs training length with reduced-rank receivers, with the transmit power of each transmit antenna $P$ 100, the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8, the number of transmitted training symbol sequences $N_s$ 8, the length of each training symbol sequence $L$ from 8 to 400 with increment 8, the number of trials for plotting the averaged training result $N_a$ 20 and ranks 2 and 4 . . . . .	53
4.13	SINR vs training length with reduced-rank receiver, with the transmit power of each transmit antenna $P$ 100, the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8, the number of transmitted training symbol sequences $N_s$ 8, the length of each training symbol sequence $L$ from 8 to 400 with increment 8, the number of trials for plotting the averaged training result $N_a$ 20 and ranks 2 and 4 . . . . .	53
4.14	MSE vs training length with reduced-rank receivers, with the transmit power of each transmit antenna $P$ 100, the number of transmit antennas $N_t$ 64, the number of receive antennas $N_r$ 64, the number of transmitted training symbol sequences $N_s$ 8, the length of each training symbol sequence $L$ from 8 to 400 with increment 8, the number of trials for plotting the averaged training result $N_a$ 20 and ranks 2 and 4 . . . . .	54
4.15	SINR vs training length with reduced-rank receivers, with the transmit power of each transmit antenna $P$ 100, the number of transmit antennas $N_t$ 64, the number of receive antennas $N_r$ 64, the number of transmitted training symbol sequences $N_s$ 8, the length of each training symbol sequence $L$ from 8 to 400 with increment 8, the number of trials for plotting the averaged training result $N_a$ 20 and ranks 2 and 4 . . . . .	55
4.16	MSE vs training length with reduced-rank receivers, with the transmit power of each transmit antenna $P$ 100, the number of transmit antennas $N_t$ 64, the number of receive antennas $N_r$ 64, the number of transmitted training symbol sequences $N_s$ 64, the length of each transmitted training symbol sequence $L$ from 8 to 400 with increment 8, the number of trials for plotting the averaged training result $N_a$ 20 and ranks 2 and 4 . . . . .	56



4.17	SINR vs training length with reduced-rank receivers, with the transmit power of each transmit antenna $P$ 100, the number of transmit antennas $N_t$ 64, the number of receive antennas $N_r$ 64, the number of transmitted training symbol sequences $N_s$ 64, the length of each training symbol sequence $L$ from 8 to 400 with increment 8, the number of trials for plotting the averaged training result $N_a$ 20 and ranks 2 and 4 . . . . .	56
4.18	MSE vs training length with a full-rank transmitter-receiver pair, with the transmit power of each transmit or receive antenna $P$ 100, the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8, the number of transmitted training symbol sequences $N_s$ 8, the length of each transmitted training symbol sequence $L$ from 8 to 400 with increment 8, the fixed number of bi-directional optimization or training iterations $N_b$ 20 and the number of trials for plotting the averaged training result $N_a$ 20 . . . . .	58
4.19	SINR vs training length with a full-rank transmitter-receiver pair, with the transmit power of each transmit or receive antenna $P$ 100, the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8, the number of transmitted training symbol sequences $N_s$ 8, the length of each transmitted training symbol sequence $L$ from 8 to 400 with increment 8, the fixed number of bi-directional optimization or training iterations $N_b$ 20 and the number of trials for plotting the averaged training result $N_a$ 20 . . . . .	59
4.20	MSE vs number of bi-directional optimization or training iterations with a full-rank transmitter-receiver pair, with the transmit power of each transmit or receive antenna $P$ 100, the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8, the number of transmitted training symbol sequences $N_s$ 8, the number of bi-directional optimization or training iterations $N_b$ from 1 to 20 with increment 1, the fixed length of each transmitted training symbol sequence $L$ 40 and the number of trials for plotting the averaged training result $N_a$ 20 . . . . .	60

4.21	SINR vs number of bi-directional optimization or training iterations with a full-rank transmitter-receiver pair, with the transmit power of each transmit or receive antenna $P$ 100, the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8, the number of transmitted training symbol sequences $N_s$ 8, the number of bi-directional optimization or training iterations $N_b$ from 1 to 20 with increment 1, the fixed length of each training symbol sequence $L$ 40 and the number of trials for plotting the averaged training result $N_a$ 20 . . . .	61
4.22	MSE vs training length with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna $P$ 100, the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8, the number of transmitted training symbol sequences $N_s$ 8, the length of each transmitted training symbol sequence $L$ from 8 to 400 with increment 8, the fixed number of bi-directional optimization or training iterations $N_b$ 20, the number of trials for plotting the averaged training result $N_a$ 20 and ranks 2 and 4 . . . .	62
4.23	SINR vs training length with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna $P$ 100, the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8, the number of transmitted training symbol sequences $N_s$ 8, the length of each transmitted training symbol sequence $L$ from 8 to 400 with increment 8, the fixed number of bi-directional optimization or training iterations $N_b$ 20, the number of trials for plotting the averaged training result $N_a$ 20 and ranks 2 and 4 . . . .	63
4.24	MSE vs number of bi-directional optimization or training iterations with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna $P$ 100, the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8, the number of transmitted training symbol sequences $N_s$ 8, the number of bi-directional optimization or training iterations $N_b$ from 1 to 20 with increment 1, the fixed length of each transmitted training symbol sequence $L$ 40, the number of trials for plotting the averaged training result $N_a$ 20 and ranks 2 and 4 . . . . .	64

4.25	SINR vs number of bi-directional optimization or training iteration with two transmitter-receiver pairs, with the transmit power of each transmit or receive antenna $P$ 100, the number of transmit antennas $N_t$ 8, the number of receive antennas $N_r$ 8, the number of transmitted training symbol sequences $N_s$ 8, the number of bi-directional optimization or training iterations $N_b$ from 1 to 20 with increment 1, the fixed length of each transmitted training symbol sequence $L$ 40, the number of trials for plotting the averaged training result $N_a$ 20 and ranks 2 and 4 . . . . .	65
4.26	MSE vs training length with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna $P$ 100, the number of transmit antennas $N_t$ 64, the number of receive antennas $N_r$ 64, the number of transmitted training symbol sequences $N_s$ 64, the length of each transmitted training symbol sequence $L$ from 8 to 400 with increment 8, the fixed number of bi-directional optimization or training iterations $N_b$ 20, the number of trials for plotting the averaged training result $N_a$ 20 and ranks 2 and 4 . . . . .	66
4.27	SINR vs training length with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna $P$ 100, the number of transmit antennas $N_t$ 64, the number of receive antennas $N_r$ 64, the number of transmitted training symbol sequences $N_s$ 64, the length of each transmitted training symbol sequence $L$ from 8 to 400 with increment 8, the fixed number of bi-directional optimization or training iterations $N_b$ 20, the number of trials for plotting the averaged training result $N_a$ 20 and ranks 2 and 4 . . . . .	67
4.28	MSE vs number of bi-directional optimization or training iterations with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna $P$ 100, the number of transmit antennas $N_t$ 64, the number of receive antennas $N_r$ 64, the number of transmitted training symbol sequences $N_s$ 64, the number of bi-directional optimization or training iterations $N_b$ from 1 to 20 with increment 1, the fixed length of each transmitted training symbol sequence $L$ 40, the number of trials for plotting the averaged training result $N_a$ 20 and ranks 2 and 4 . . . . .	68

4.29	SINR vs number of bi-directional optimization or training iterations with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna $P$ 100, the number of transmit antennas $N_t$ 64, the number of receive antennas $N_r$ 64, the number of transmitted training symbol sequences $N_s$ 64, the number of bi-directional optimization or training iterations $N_b$ from 1 to 20 with increment 1, the fixed length of each transmitted training symbol sequence $L$ 40, the number of trials for plotting the averaged training result $N_a$ 20 and ranks 2 and 4 . . . . .	69
4.30	Model of interference networks . . . . .	70
4.31	Sum rate vs training length with reduced-rank transmitters and receivers in an interference network, with the transmit power of each transmit or receive antenna $P$ 100, the number of transmitter-receiver pairs $N_p$ 3, the number of transmit antennas on each transmitter $N_t$ 64, the number of receive antennas on each receiver $N_r$ 64, the number of transmitted training symbol sequences for each transmitter-receiver pair $N_s$ 64, the length of each transmitted training symbol sequence $L$ from 8 to 240 with increment 8, the fixed number of bi-directional optimization or training iterations $N_b$ 10, the number of trials for plotting the averaged training result $N_a$ 10 and ranks 2 and 4 . . . . .	71
4.32	Sum rate vs number of bi-directional optimization or training iterations with reduced-rank transmitters and receivers in an interference network, with the transmit power of each transmit or receive antenna $P$ 100, the number of transmit antennas on each transmitter $N_t$ 64, the number of receive antennas on each receiver $N_r$ 64, the number of transmitted training symbol sequences for each transmitter-receiver pair $N_s$ 64, the number of bi-directional optimization or training iterations $N_b$ from 1 to 20 with increment 1, the fixed length of each transmitted training symbol sequence $L$ 40, the number of trials for plotting the averaged training result $N_a$ 10 and ranks 2 and 4 . . . . .	72

# List of Abbreviations

**4G** Fourth-Generation.

**5G** Fifth-Generation.

**ACI** Adjacent-Channel Interference.

**ADAS** Advanced Driver-Assistance Systems.

**ADC** Analog-to-Digital Converters.

**AWGN** Additive White Gaussian Noise.

**BPSK** Binary Phase-Shift Keying.

**CCI** Co-Channel Interference.

**CSI** Channel State Information.

**CTIA** Cellular Telecommunications and Internet Association.

**DAC** Digital-to-Analog Converters.

**DoFs** Degrees of Freedom.

**DS-CDMA** Direct-Sequence Code Division Multiple Access.

**FCC** Federal Communications Commission.

**FDD** Frequency Division Duplex.

**FIR** Finite Impulse Response.

**FPGAs** Field Programmable Gate Arrays.

**GSC** Generalized Side-lobe Canceller.

**HiCapS** High Capacity Satellite.

**HTS** High Throughput Satellite.

**ICI** Inter-Carrier Interference.

**ICT** Information and Communications Technology.

**IIR** Infinite Impulse Response.

**ISI** Inter-Symbol Interference.

**LOS** Line of Sight.

**LS** Least Squares.

**LTE** Long Term Evolution.

**LTI** Linear Time-Invariant.

**MATLAB** Matrix Laboratory.

**max-SINR** maximum Signal-to-Interference-plus-Noise Ratio.

**MIMO** Multiple-Input Multiple-Output.

**MISO** Multi-Input Single-Output.

**mMIMO** massive Multiple-Input Multiple-Output.

**MMSE** Minimum Mean Square Error.

**MSE** Mean Square Error.

**MSWF** Multi-Stage Wiener Filter.

**MUSIC** Multiple Signal Classification.

**NTIA** National Telecommunications and Information Administration.

**PC** Principal Components.

**PSK** Phase-Shift Keying.

**QPSK** Quadrature Phase-Shift Keying.

**RF** Radio Frequency.

**SAMV** Sparse Asymptotic Minimum Variance.

**SIC** Successive Interference Cancellation.

**SIMO** Single-Input Multi-Output.

**SINR** Signal-to-Interference-plus-Noise Ratio.

**SISO** Single-Input Single-Output.

**sum-MSE** sum-Mean Squared Error.

**TDD** Time Division Duplex.

**UMTS** Universal Mobile Telecommunications Service.

**USCB** United States Census Bureau.

**WLAN** Wireless Local Area Network.

# Chapter 1

## Introduction

### 1.1 General Introduction

With the unprecedented development of electronic and computer technology starting from the second half of the twentieth century, the widespread adoption of wireless communications devices has profoundly revolutionized people's patterns of production and life. According to the statistical data surveyed by the Cellular Telecommunications and Internet Association (CTIA), up to the year of 2019, the number of wireless cell sites and mobile Internet users in the United States had become 395,562 and 267,600,000, respectively [2, 3]. Many kinds of resources in communications networks, such as bandwidth and power, are extremely valuable and expensive, thus improving the rate of utilization of these resources and reducing the cost are necessary. Additionally, in real systems design, there are usually tradeoffs between these important technical and financial factors. Therefore, it is vital to design wireless communications systems with an optimized solution. In this research project, instead of using conventional signal processing methods, like Successive Interference Cancellation (SIC), two different techniques, reduced-rank filtering and bi-directional training, are adopted to design MIMO communications systems.

Reduced-rank filtering is widely used in many different signal processing applications, such as radar signal processing and array signal processing (e.g., see [4–7]) and has recently been proposed for interference suppression in Direct-Sequence Code Division Multiple Access (DS-CDMA) [8–13]. It differs from full-rank filtering: in a reduced-rank filter, the received signal is projected onto a lower dimensional subspace, then the design of the filter structure occurs within this subspace [14]. Reducing the originally received



signal dimension to a lower subspace dimension reduces the number of parameters to estimate, and accordingly reduces the feedback requirement and system complexity, but meanwhile limits the Degrees of Freedom (DoFs) available for suppressing interference and noise [15]. The Minimum Mean Square Error (MMSE) achieved by reduced-rank filtering might be higher than the counterpart achieved by full-rank filtering [14]. In the training process of communications systems, when the available training length resource is limited, reduced-rank filtering can achieve a better system performance than full-rank filtering. The detailed implementation method of reduced-rank filtering is demonstrated in Section 3.3.

Bi-directional training is a kind of distributed mechanism to update the precoder and the receive filter iteratively. It differs from bi-directional optimization, where the global Channel State Information (CSI) is known *a priori*. In bi-directional training, there is no such centralized controller that stores the CSI [16]. The precoder and the receive filter can only be adapted via backward training and forward training, respectively, without explicit exchange of the CSI. Bi-directional training consists of five main steps: 1) the transmitter randomly initializes its precoder matrix; 2) the transmitter sends training symbol sequences to the receiver, then the receiver updates its receive filter matrix by Least Squares (LS) algorithm using these sequences; 3) the receiver transmits training symbol sequences back to the transmitter by the principle of channel reciprocity, then the transmitter updates its precoder matrix also by LS algorithm; 4) repeat 2) and 3) up to the maximally allowed time or until the pre-set convergence requirement is met. This method is different from two-way channel estimation, as presented in [17–22]. The detailed implementation procedure of bi-directional training is demonstrated in Section 3.4.2.

The MIMO communications system is a novel configuration for multiplying the capacity of radio links using multiple antennas to exploit multipath propagation [23]. MIMO has become an essential element of many wireless communications standards including IEEE 802.11n (Wi-Fi), IEEE 802.11ac (Wi-Fi), HSPA+ (3G), WiMAX, and Long Term Evolution (4G LTE) [23]. In MIMO communications systems, there are multiple transmit antennas in the transmitter and multiple receive antennas in the receiver, and usually there are multiple signal streams back and forth between the transmitter and the receiver. Compared with conventional antenna configurations, MIMO has a number of technical advantages, such as increased capacity, enhanced reliability, higher spectrum efficiency, enhanced energy efficiency and more guaranteed system security [24, 25]. Combined with adaptive array beamforming

techniques, MIMO can guide transmitted Radio Frequency (RF) signals to some certain planned directions to reduce the interference between different signal streams and enhance the trackability of the desired signal, thus the received SINR increases. MIMO can also be extended to become massive Multiple-Input Multiple-Output (mMIMO) with larger antenna arrays. The premise behind mMIMO technology is to reap all benefits of conventional MIMO, but on a greater scale [26]. The formulation of a MIMO system is presented in Section 2.4.

The aim of this research project is to design a MIMO communications system using reduced-rank filtering and bi-directional training, then evaluate the system performance under different parameter settings. The research project work consists of both theoretical derivations and experimental simulation. Chapter 2 demonstrates several key technical concepts involved in the design; Chapter 3 illustrates four crucial algorithms by mathematical expressions; Chapter 4 describes experimental steps and presents simulation results; possible problems and future works are discussed in Chapter 5, then Chapter 6 draws conclusions for the research project.

## 1.2 Literature Review

In recent decades, many scholars have done much research about reduced-rank filtering, bi-directional training, MIMO communications systems and some other relevant topics. The work of this research project is an extension based on existing research achievements.

Sun and Honig (2006) proposed the signature-receiver adaptation with reduced-rank filtering in [27]. In their paper, they present an iterative algorithm based on the Multi-Stage Wiener Filter (MSWF) reduced-rank filter only at receivers, under the system of CDMA, to suppress interference and noise. In each training iteration, transmitters transmit training symbol sequences to receivers, then receivers update their receive filters within the lower dimensional subspace and relay normalized receive filters back to transmitters as the signatures for the next training iteration. This process is repeated for the pre-set time or until the certain convergence requirement is satisfied. Simulation results from this paper suggest that if signatures and receive filters are estimated from training, it is the signature adaptation with reduced-rank receive filters that provides a better system performance over that with full-rank receive filters, here the received SINR is used to measure the system performance. The reduced-rank filtering algorithm adopted in

this research project is the same with the one from their paper.

Bi-directional training in MIMO interference networks is demonstrated by Shi, Berry and Honig (2014) in [28]. In their paper, both bi-directional optimization with the maximum Signal-to-Interference-plus-Noise Ratio (max-SINR) algorithm and bi-directional training using LS estimation are presented, and the comparison between them is made. In the bi-directional training from their paper, for forward training, transmitters send packets containing training symbol sequences to receivers, whereafter receivers update receive filters by these symbol sequences with LS algorithm; inversely, for backward training, the original receivers act as transmitters, then send training symbol sequences back, thus precoders are updated using these symbol sequences also by LS algorithm. After repeating this procedure for an enough time, trained precoders and receive filters are obtained and the transmission of useful data information therewith can be started. Adaptive power control is also considered in their paper, but it is beyond the scope of this research project.

A point in common for the two aforementioned papers is that both of them are based on Time Division Duplex (TDD) within one certain coherence time rather than Frequency Division Duplex (FDD), otherwise the principle of channel reciprocity does not hold. These two papers lay the groundwork of this research project. Bi-directional training under FDD systems is discussed by Zhou, Honig, Liu and Xiao (2019) in [16].

Some further topics are also studied. Zhuang, Berry and Honig (2011) researched suppression of interference in MIMO cellular networks in [29]. A systematic comparison among several distributed beamforming optimization techniques, including the max-SINR, distributed interference pricing and weighted sum-Mean Squared Error (sum-MSE), is made by Schmidt, Shi, Berry, Honig and Utschick (2013) in [30].

### 1.3 Industrial Relevance

The advanced research of communications systems is always firmly associated with telecommunications equipment vendors, including Cisco Systems (United States), Qualcomm (United States), Motorola Solutions (United States), Huawei (China), ZTE (China), Samsung (South Korea), Nokia (Finland) and Ericsson (Sweden). There is a huge market for telecommunications equipment in the United States. According to the statistical data released

by the United States Census Bureau (USCB), by the year of 2018, the volume of imports of telecommunications equipment in the United States was 73,980,000 dollars [31]. In recent years, Fifth-Generation (5G) communications, the latest communications technology standard for broadband cellular networks and the successor of Fourth-Generation (4G) communications, comes with spotlights. 5G technology is expected to provide the increased throughput with advanced transmitting and receiving techniques [32].

Two examples of industrial applications about MIMO communications are briefly presented below.

### **1.3.1 Satellite Communications**

MIMO communications has important applications in satellite communications. For a satellite communications channel, having a strong Line of Sight (LOS) between the satellite and the earth ground is common [33]. High Throughput Satellite (HTS) and High Capacity Satellite (HiCapS) are the two satellite systems that have emerged, with the first one is to increase the overall throughput of a satellite and the second one is to increase the satellite's capacity in a given region [33]. In both systems, the applicability of MIMO is beneficial to improve the system performance [33].

### **1.3.2 Imaging Radars**

The imaging radar is an important component within Advanced Driver-Assistance Systems (ADAS). Combining MIMO beamforming with imaging radar systems is helpful for improving angular resolution, therefore the usability of ADAS is accordingly enhanced. It is especially useful in short-range applications, in which there is enough time to switch between antenna elements as an alternative to more sophisticated implementations [34].

Generally, besides the two aforesaid examples of applications, MIMO is also broadly applied in some other scientific fields, including optical communications, adaptive underwater sonar systems and space telescope design.

# Chapter 2

## Theoretical Background

### 2.1 Basic Models of Communications Systems

A communications system is an integration of signals, input and output devices, communications channels, electronic processors, controllers, etc. The aim of communications systems is to convey messages from one terminal to the other as accurately as possible by any means, including encoding and decoding.

The most straightforward model of communications systems consists of only transmitters, communications channels, additive noise and receivers. The model is depicted in Figure 2.1.

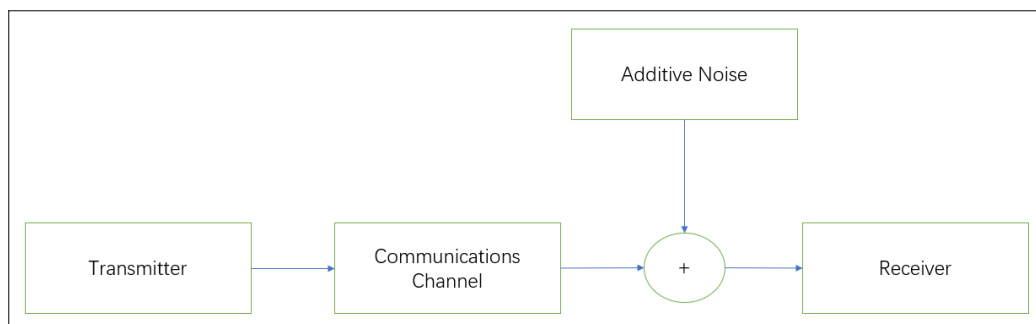


Figure 2.1: Basic model of communications systems

In Figure 2.1, the message is initially sent from the transmitter, there-with passes through the communications channel, which is usually modeled as Gaussian distribution; in the end, the receiver receives the noisy transmitted message and tries to reconstruct the originally transmitted information.

However, in practice, classifying system components into such four parts is oversimplified. For instance, in electronics, a more detailed model of communications systems is pictorially demonstrated in Figure 2.2.

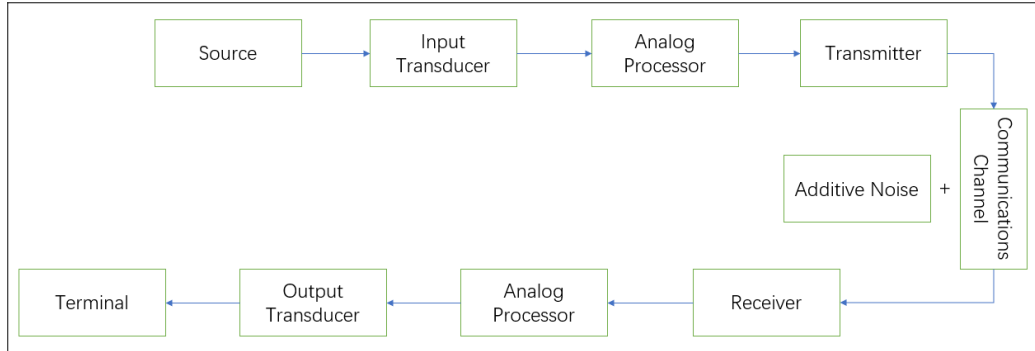


Figure 2.2: Model of electronic communications systems

In Figure 2.2, the source transmits the original signal to the transducer, in which the original signal is transduced into the electronic form; in the analog processor, electronic signals are pre-processed in some ways, such as amplifying or noise filtering. The transmitter transmits the pre-processed signal via the communications channel, usually modulation or Analog-to-Digital Converters (ADC) are used within the transmitter to heighten the electromagnetic interference immunity of the signal.

Communications channels can be generally classified as guided channels and unguided channels. For example, in optical-fiber communications, optical fiber, acts as the medium, owns the ability of guiding the transmitted signal to the other terminal; on the contrary, in RF communications, there is no guided medium, RF signals are transmitted in the air.

What is the next is merely the reverse engineering of those occurred before. The receiver demodulates or converts the received signal by Digital-to-Analog Converters (DAC), then the analog processor post-processes it and eventually the output transducer transforms it into an appropriate form to the terminal.

## 2.2 Interference and Noise

### 2.2.1 Interference

In communications theory, interference is those other signals come along with the desired signal between the transmitter side and the receiver side. Frequently encountered examples include Co-Channel Interference (CCI), Adjacent-Channel Interference (ACI), Inter-Symbol Interference (ISI) and Inter-Carrier Interference (ICI).

Interference mitigation is the research to reduce the negative influence made by interference. This topic is closely related with the aim of this research project. In the experimental simulation of an interference network containing multiple transmitter-receiver pairs presented in Section 4.3, interference consists of both signal streams from the same transmitter and other transmitters within the network.

### 2.2.2 Noise

In communications systems, noise is the undesired signal possibly comes from the external environment or internal devices. It has nothing related with the desired information. The influence of noise can be reduced, but is impossible to be eliminated completely.

Noise can be coarsely classified as man-made noise and natural noise. The former one includes noise coming from poor filtering and electronic apparatuses, whereas the later one includes noise originating from antenna noise and thermal noise.

Noise is inherently sophisticated to analyze, for the convenience of modeling, Additive White Gaussian Noise (AWGN) model is usually applied in experimental noise modeling.

## 2.3 Beamforming Techniques

Beamforming, sometimes named as spatial filtering, is the signal processing technique used in sensor arrays for directional signal transmission or reception [35]. Beamforming technology has widespread applications in many subjects of engineering, including communications, optics, astronomy, radar, seismology, acoustics and remote sensing.

In wireless communications, generally, for an antenna array, beamforming effect is performed by enhancing the directivity, trackability and strength of electromagnetic waves towards the desired angle, meanwhile suppressing signals from other angles, to mitigate the adverse influence of interference and noise and increase the rate of utilization of transmitted power. It can be applied in both the transmitter terminal and the receiver terminal. In the transmitter terminal, the beamforming controller adjusts relative phases between antenna elements, to create a special pattern of radiation; in the receiver terminal, beamforming is made by deploying the receive antenna array configuration to capture the desired pattern of transmission and receive it.

The beamforming pattern is pictorially illustrated in Figure 2.3.

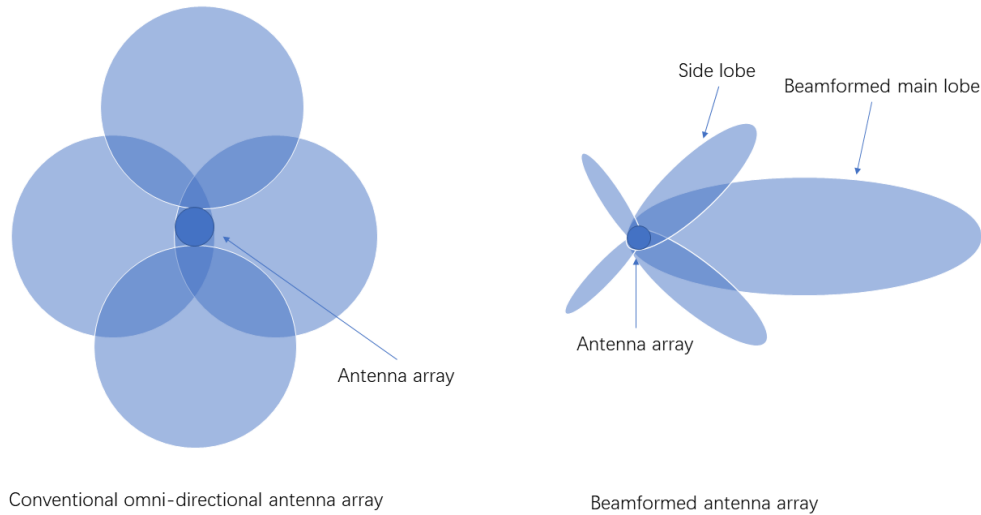


Figure 2.3: Conventional omni-directional antenna array vs Beamformed antenna array

For the radiation pattern on the left of Figure 2.3, this is the typical effect of a conventional omni-directional antenna array, in which the array radiates electromagnetic waves uniformly into all angles across the whole 360 degree. The figure simplifies the radiation pattern by only depicting four main directions whereas in practice, omni-directional antenna can transmit waves into more directions. In such an unplanned radiation pattern, much energy is wasted. Inversely, for the pattern on the right of this figure, most radiated energy is transmitted into only one concentrated direction, which is the beamformed main lobe; several side lobes have much less energy.



For the algorithmic aspect, beamforming can be classified as conventional beamforming and adaptive beamforming.

Conventional beamformers use antenna arrays with fixed weights and fixed relative phases to transmit electromagnetic waves into certain directions, Butler matrix is one of the examples of conventional beamforming method; whereas in adaptive beamforming, such as Multiple Signal Classification (MUSIC) algorithm and Sparse Asymptotic Minimum Variance (SAMV) algorithm, beamformers always update their array matrices by the newly received signal using adaptive filtering and estimation approaches, thus adaptive beamforming is more intelligent and more capable for practical applications. What needs to be pointed out is that, as the computational requirement of adaptive beamforming systems is becoming increasingly intensive, digital hardware is introduced, such as Field Programmable Gate Arrays (FPGAs), instead of conventional computer software. FPGAs can provide higher levels of computational performance, hence they are specially suitable for computationally intensive beamforming applications [36]. The application of FPGAs in beamforming is helpful for reinforcing the computational flexibility and might overcome many existing drawbacks in conventional analog electronic systems [37]. In this research project, adaptive beamforming is adopted rather than the conventional counterpart.

## 2.4 Multi-Antenna Systems

The multi-antenna system is one of the smart antenna systems used in the research of antenna arrays. Regarded as a crucial technical breakthrough in wireless communications, multi-antenna systems fuel the increasing requirement of data rate for some advanced techniques, such as Universal Mobile Telecommunications Service (UMTS), Long Term Evolution (LTE) and Wireless Local Area Network (WLAN) [38]. Diversity is an important characteristic of multi-antenna systems, including time diversity, frequency diversity, multi-user diversity, spatial diversity, polarization diversity and pattern diversity [39]. Diversity techniques, by exploiting the characteristic of channel variation, are employed to make wireless communications systems more reliable and interference immune [39].

There are several different configurations for multi-antenna systems, including Single-Input Single-Output (SISO), Single-Input Multi-Output (SIMO), Multi-Input Single-Output (MISO) and MIMO. The choice of the certain

configuration depends on the practical application scenario. Pictorial demonstrations of SISO, SIMO, MISO and MIMO systems are in Figures 2.4, 2.5, 2.6 and 2.7, respectively.

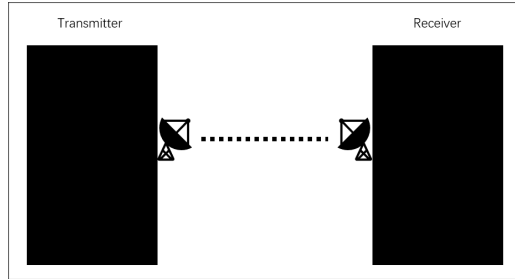


Figure 2.4: Single-input single-output systems

In Figure 2.4, there is only one antenna attached on both the transmitter and the receiver, which is the simplest form of the multi-antenna system. In this configuration, there is no signal processing involved with diversity characteristics and it is relatively easy to construct. However, SISO systems are fragile when being disrupted by external interference and noise, and their throughput is severely limited by the scale of the antenna array.

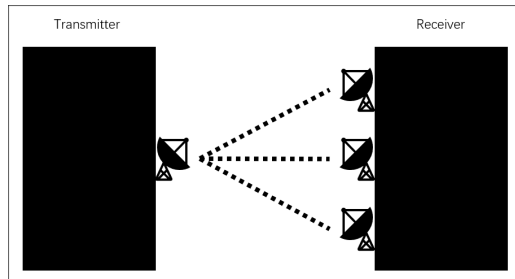


Figure 2.5: Single-input multi-output systems

In Figure 2.5, there is one antenna at the transmitter and several antennas at the receiver, in which receive diversity is exploited. Signal processing techniques are necessary within the receiver terminal.

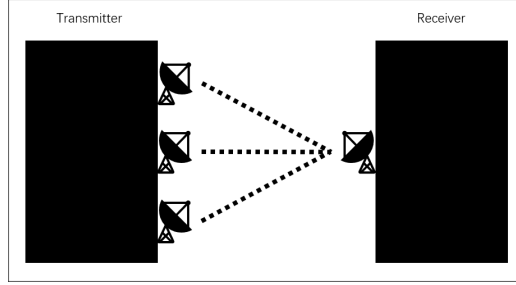


Figure 2.6: Multi-input single-output systems

Several antennas at the transmitter and one antenna at the receiver are drawn in Figure 2.6. To exploit transmit diversity, several same data streams are sent from each transmit antenna using some coding schemes, such as the repetition code scheme.

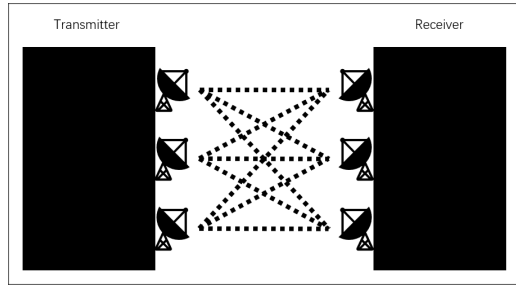


Figure 2.7: Multi-input multi-output systems

Figure 2.7 depicts a MIMO system, in which several antennas are in both the transmitter and the receiver. Combining with both transmit diversity and receive diversity, MIMO systems can effectively improve interference immunity, signal directivity and signal trackability, compared with the three aforesaid antenna configurations. By the Shannon-Hartley Theorem, MIMO configurations can increase the channel throughput of communications systems.

The MIMO system can be mathematically expressed as:

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{n} \quad (2.1)$$

Where received signal  $\mathbf{y} = [y_1 \ y_2 \ y_3 \ \dots \ y_{Nr}]^T$ , transmitted signal  $\mathbf{x} = [x_1 \ x_2 \ x_3 \ \dots \ x_{Nt}]^T$ , where the superscript  $T$  refers to matrix transpose,  $Nr$  is the number of receive antennas at the receiver and  $Nt$  is

the number of transmit antennas at the transmitter.  $\mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & \dots \\ \vdots & \ddots & \\ h_{Nr1} & & h_{NrNt} \end{bmatrix}$  is the MIMO channel matrix, in which each element  $h_{ij}$  refers to the channel gain from the transmit antenna  $j$  to the receive antenna  $i$ .  $\mathbf{n} = [n_1 \ n_2 \ n_3 \ \dots \ \dots \ n_{Nr}]^T$  is the additive noise signal. This mathematical expression is the basis for further theoretical derivations.

## 2.5 Transmission Control Schemes

There are several different transmission control schemes in communications networks. Classified by the directivity of transmission, there are three kinds of control schemes exist, they are simplex, half-duplex and full-duplex. In simplex networks, transmission only occurs in one direction, which is from the transmitter to the receiver, such as remote monitoring systems; in half-duplex systems, there is bi-directional transmission, but forward transmission and backward transmission do not occur simultaneously, such as interphones; full-duplex is the scheme whereby each communications terminal can transmit data to the other simultaneously.

FDD and TDD are the two standards for the full-duplex scheme, the following is an illustration of them.

### 2.5.1 FDD

FDD is the acronym of Frequency Division Duplex. In the FDD scheme, the transmitter and the receiver simultaneously send information to each other by modulating their information under different frequency channels, as depicted in Figure 2.8. In Figure 2.8, there is a gap between the two channels on the frequency axis, which is called guard frequency band, to mitigate the mutual interference of the two adjacently distributed frequency bands.

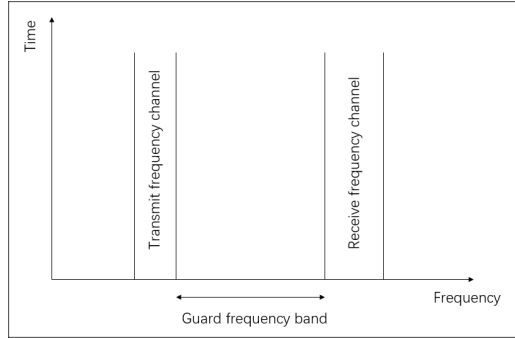


Figure 2.8: Frequency division duplex

In the United States, main frequency allocation authorities are the National Telecommunications and Information Administration (NTIA) for the federal government and the Federal Communications Commission (FCC) for non-federal governmental agencies, respectively.

### 2.5.2 TDD

TDD is the initialism standing for Time Division Duplex. In the TDD mechanism, both the transmitter and the receiver send data using signals of small bursts in the time domain, as suggested in Figure 2.9. The bursts are too short to affect the simultaneousness of mutual communications.

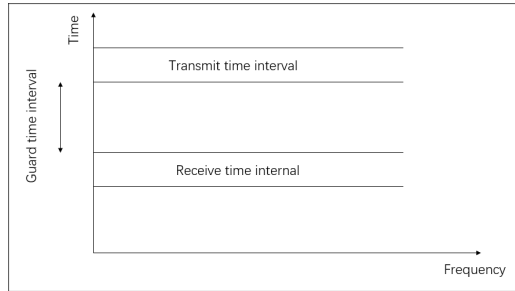


Figure 2.9: Time division duplex

In Figure 2.9, similarly with FDD, there is a guard time interval between the transmit interval and the receive interval, to alleviate the mutual interference between them. A technical characteristic of TDD over FDD is the principle of channel reciprocity. When this principle holds, the downlink channel matrix and the uplink channel matrix are algebraically matchable, the later one is just the complex conjugated transposed form of the former

one. In this research project, the TDD scheme is applied to exploit this important characteristic.

## 2.6 Coherence Time

On account of the Doppler effect, channel variation usually occurs in wireless communications systems, accordingly, the matrix used for expressing the communications channel also varies from time to time. To reduce the sophistication, in the simulation of wireless networks, coherence time is defined as the time interval, during which the communications channel is considered to be unvaried. For the convenience of analysis, in this research project, the simulation is assumed to occur within only one certain coherence time, therefore, the MIMO channel matrix is considered to be time-invariant.

## 2.7 Centralized and Distributed Configurations

There are two kinds of configurations in the design of communications systems, they are centralized configurations and distributed configurations. Characteristics of each of them are shortly discussed.

### 2.7.1 Centralized Configurations

The centralized configuration is the one in which there is only one server and several clients. Figure 2.10 provides the visual graph.

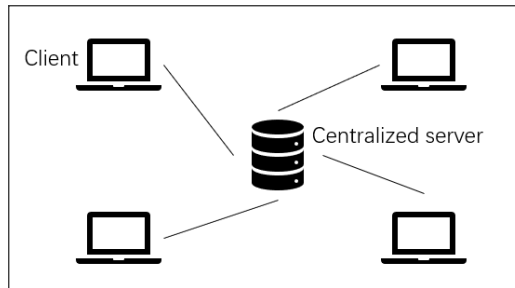


Figure 2.10: Centralized configuration

In Figure 2.10, each client is connected with the central server, which is the only exchange information provider for the whole system. In the scenario of MIMO communications networks, the so-called centralized server is

the controller which undertakes filtering computation missions and updates both precoders and receive filters in the real time. Most precoder designs are centralized, with the prerequisite that the controller owns the CSI [16]. Examples of communications systems with the centralized configuration are presented in [40] and [41]. The centralized architecture is efficient for small-scaled systems, but as systems grow larger, its bottleneck appears due to its limitation of the system capability.

### 2.7.2 Distributed Configurations

The distributed configuration is entirely different with the aforementioned centralized one. Figure 2.11 displays it visually.

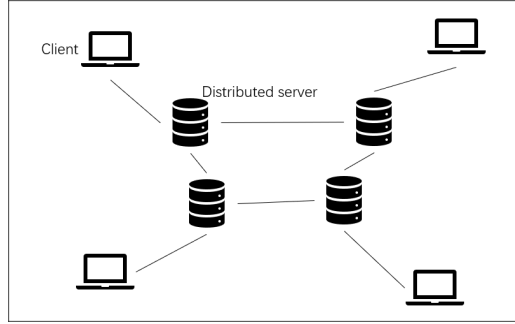


Figure 2.11: Distributed configuration

In Figure 2.11, each client is associated with its own server, without a global controller to coordinate activities of the whole system. Low latency is one of the technical advantages owned by the distributed system over the centralized counterpart, but the distributed system is relatively difficult to design. Distributed configurations are widely applied in some advanced computing systems, such as cloud computing, grid computing and cluster computing.

In this research project, the distributed manner is adopted to compute and update precoders and receive filters in the process of bi-directional training, without knowing the CSI of the communications system *a priori*.

## 2.8 Channel Capacity

In signal processing and communications, channel capacity, sometimes called the maximum data rate, is the maximum rate at which the data can be trans-

mitted with a relatively negligible error rate. For the aspect of information theory, it is the amount of information the output conveys about the input, which is calculated by the mutual information [42]. As stated by the Shannon's Noisy Channel Coding Theorem, in each memoryless channel, for any error probability  $\epsilon$ , for any transmission rate  $\mathbf{R}$  smaller than or equal to the channel capacity  $\mathbf{C}$ , an encoding-decoding mechanism always exists, whose error probability is smaller than or equal to  $\epsilon$  [42].

Channel capacity can be expressed as the Shannon-Hartley Theorem in AWGN communications channels:

$$\mathbf{C} = \mathbf{B} \log_2 \left( 1 + \frac{\mathbf{S}}{\mathbf{I} + \mathbf{N}} \right) \quad (2.2)$$

In which,  $\mathbf{C}$  is the channel capacity measured in bits per second,  $\mathbf{B}$  is the bandwidth expressed in Hertz,  $\mathbf{S}$  is the power of the desired signal,  $\mathbf{I}$  is the power of interference,  $\mathbf{N}$  is the power of noise, thus the ratio  $\frac{\mathbf{S}}{\mathbf{I} + \mathbf{N}}$  is the value of SINR. In this research project, this formula is applied to compute the value of channel capacity versus training length and the number of bi-directional optimization or training iterations in the experimental simulation.



## Chapter 3

# Methodology and Algorithms

In this research project, adaptive filtering and estimation algorithms are the central topics. In signal processing, adaptive filtering is the algorithm with a transfer function containing variable parameters, which are updated by some certain means [43]. Due to the sophistication of analog signal processing, adaptive filtering is usually based on digital signal processing. In many real applications, it is unrealistic to compute parameters *a priori*, therefore, adaptive filtering is adopted to compute these parameters intelligently. The utilization of adaptive filtering is common in various fields of research, such as radar signal processing, audio signal processing, image and video processing and pattern recognition. In this chapter, two adaptive filtering approaches, Wiener estimation and LS estimation, are discussed in details along with reduced-rank filtering, bi-directional optimization and training.

### 3.1 Wiener (optimal) Estimation

In adaptive filtering and estimation, the so-called Wiener filter is actually the generalization of discrete-time linear optimal filters [44]. The design aim of such a filter is to minimize the MSE between the estimated signal and the desired signal. Parameters of adaptive filters are normally based on complex values, since in many practical situations, observables are presented in the baseband form. The term, baseband form, refers to the frequency band representing the original signal [44]. Figure 3.1 visualizes the process of digital estimation.

In Figure 3.1, the original signal is input into the digital Linear Time-Invariant (LTI) filter, which outputs the estimated signal. A subtraction is made between the estimated signal and the desired signal, their difference is

the estimated error. From the aspect of statistics, the smaller the estimated error, the better the filter design.

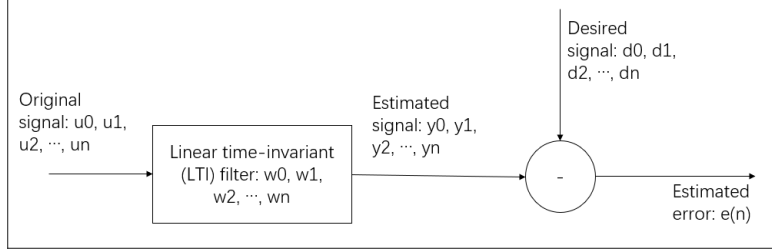


Figure 3.1: Digital estimator

It is worth pointing out that, classified by whether there are feedback signal loops involved, there are two sorts of digital filters, Infinite Impulse Response (IIR) filters and Finite Impulse Response (FIR) filters. IIR filters, as the name suggests, contain impulse responses which never fade away to precisely zero along the time axis and feedback signal loops are within them; conversely, in FIR filters, there are only feedforward signal loops involved. Technically speaking, adaptive filters can be designed using IIR filters, but they usually bring inherent instability, which leads to the negative effect of oscillation in the use; whereas, designing with FIR filters does not have problems with instability. However, the computational requirement and storage overhead of IIR filters are usually less than the counterparts of FIR filters [45]. By the tradeoff between those aforesaid factors, the FIR filter is adopted in this research project.

The mathematical derivation of Wiener estimation from the most fundamental linear convolution between the input signal and the impulse response of the filter can be referred in [44]. It is not going to be presented in this section. What is going to be given is the extension of some expressions in MIMO communications systems.

First of all, the system model of a MIMO communications system should be demonstrated. Figure 3.2 pictorially depicts a MIMO communications system with precoding at the transmitter terminal and receive filtering at the receiver terminal.

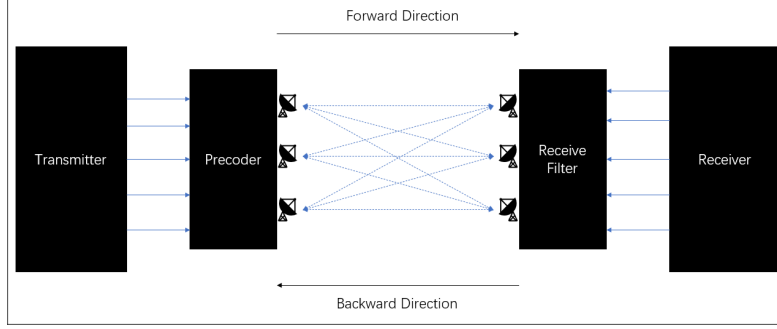


Figure 3.2: MIMO communications systems with precoding and receive filtering

In Figure 3.2, looking at the transmitter terminal, a precoder is attached on the output port of the transmitter as a beamforming controller to guide the transmitted signals; similarly, at the receiver terminal, a receive filter is with the receiver to filter unwanted interference and noise and extract the desired signal. The direction pointing from the transmitter to the receiver is the forward direction and the inverse direction is the backward direction. In this research project, bi-directional training is used to update the precoder and the receive filter adaptively and iteratively.

This model can also be demonstrated by mathematical expressions. As Figure 3.2 suggests, it is assumed that there are one transmitter and one receiver, with  $N_t$  transmit antennas at the transmitter side,  $N_r$  receive antennas at the receiver side, thus the size of the MIMO channel matrix is  $N_r$ -by- $N_t$ . It is set that there are  $N_s$  transmitted signal streams iterated between the transmitter and the receiver. At time  $i$ , for the receiver, the  $N_r$ -by-1 input signal vector  $\mathbf{u}(i)$  is presented as:

$$\mathbf{u}(i) = \sqrt{P}\mathbf{H}\mathbf{v}\mathbf{b}(i) + \mathbf{n}(i) \quad (3.1)$$

$$= \mathbf{M}\mathbf{b}(i) + \mathbf{n}(i) \quad (3.2)$$

Where  $P$  is the transmit power of each transmit antenna,  $\mathbf{H}$  is the  $N_r$ -by- $N_t$  MIMO channel matrix,  $\mathbf{v}$  is the  $N_t$ -by- $N_s$  transmitter precoder matrix,  $\mathbf{b}(i)$  is the  $N_s$ -by-1 transmitted signal vector at time  $i$  and  $\mathbf{n}(i)$  is the  $N_r$ -by-1 additive noise vector at time  $i$ .  $\mathbf{M}$  is the  $N_r$ -by- $N_s$  mixing matrix,  $\mathbf{M} = \sqrt{P}\mathbf{H}\mathbf{v}$ .

Using the conclusion directly from [44], in digital adaptive filtering, the filter  $\mathbf{g}_k$  for the  $k$ th transmitted signal symbol  $b_k(i)$ , which is the  $k$ th element

of the vector  $\mathbf{b}(i)$ , is expressed as:

$$\mathbf{g}_k = \mathbf{R}^{-1} \mathbf{p}_k \quad (3.3)$$

Where, in Wiener estimation,  $\mathbf{R}$  and  $\mathbf{p}_k$  are the statistical autocorrelation function of the input signal vector and the statistical cross-correlation function between the input signal vector and the desired signal symbol, respectively.

Thus, the reconstructed  $k$ th transmitted signal symbol is given as:

$$\hat{b}_k(i) = \mathbf{g}_k^H \mathbf{u}(i) \quad (3.4)$$

The estimated error for the  $k$ th transmitted signal symbol is:

$$e_k(i) = b_k(i) - \hat{b}_k(i) \quad (3.5)$$

$$= b_k(i) - \mathbf{g}_k^H \mathbf{u}(i) \quad (3.6)$$

What needs to be pointed out in this stage is that, in this research project, both the transmitted signal and the additive noise are statistically generated by two mutually independent Gaussian distributions, both with means zero, the former one has the variance one and the later one has the variance 0.01. Accordingly, some conclusions can be given beforehand for the later use:

$$E[\mathbf{b}(i)\mathbf{b}^H(i)] = \mathbf{I}_{N_s} \quad (3.7)$$

$$E[\mathbf{b}(i)\mathbf{n}^H(i)] = \mathbf{O}_{N_s, N_r} \quad (3.8)$$

$$E[\mathbf{n}(i)\mathbf{b}^H(i)] = \mathbf{O}_{N_r, N_s} \quad (3.9)$$

$$E[\mathbf{n}(i)\mathbf{n}^H(i)] = 0.01\mathbf{I}_{N_r} \quad (3.10)$$

Where  $E$  refers to the statistical expectation of the expression within the bracket.  $\mathbf{I}_{N_s}$  is a  $N_s$ -by- $N_s$  identity matrix,  $\mathbf{O}_{N_s, N_r}$  is a  $N_s$ -by- $N_r$  null matrix,  $\mathbf{O}_{N_r, N_s}$  is a  $N_r$ -by- $N_s$  null matrix and  $\mathbf{I}_{N_r}$  is a  $N_r$ -by- $N_r$  identity matrix.

Therefore,  $\mathbf{R}_{wiener}$ , the statistical autocorrelation function of the input signal vector  $\mathbf{u}(i)$  is expressed as:

$$\mathbf{R}_{wiener} = E[\mathbf{u}(i)\mathbf{u}^H(i)] \quad (3.11)$$

$$= E[[\sqrt{P}\mathbf{H}\mathbf{v}\mathbf{b}(i) + \mathbf{n}(i)][\sqrt{P}\mathbf{H}\mathbf{v}\mathbf{b}(i) + \mathbf{n}(i)]^H] \quad (3.12)$$

$$= E[[\mathbf{M}\mathbf{b}(i) + \mathbf{n}(i)][\mathbf{M}\mathbf{b}(i) + \mathbf{n}(i)]^H] \quad (3.13)$$

$$= E[[\mathbf{M}\mathbf{b}(i) + \mathbf{n}(i)][\mathbf{b}^H(i)\mathbf{M}^H + \mathbf{n}^H(i)]] \quad (3.14)$$

$$= E[\mathbf{M}\mathbf{b}(i)\mathbf{b}^H(i)\mathbf{M}^H] + E[\mathbf{M}\mathbf{b}(i)\mathbf{n}^H(i)] + E[\mathbf{n}(i)\mathbf{b}^H(i)\mathbf{M}^H] + E[\mathbf{n}(i)\mathbf{n}^H(i)] \quad (3.15)$$

$$= \mathbf{M}E[\mathbf{b}(i)\mathbf{b}^H(i)]\mathbf{M}^H + \mathbf{M}E[\mathbf{b}(i)\mathbf{n}^H(i)] + E[\mathbf{n}(i)\mathbf{b}^H(i)]\mathbf{M}^H + E[\mathbf{n}(i)\mathbf{n}^H(i)] \quad (3.16)$$

$$= \mathbf{M}\mathbf{M}^H + 0.01\mathbf{I}_{N_r} \quad (3.17)$$

Next, for  $\mathbf{p}_{wiener_k}$ , the cross-correlation function between the input signal vector  $\mathbf{u}(i)$  and the desired  $k$ th transmitted signal symbol  $b_k(i)$ , which is the  $k$ th element of the transmitted signal vector  $\mathbf{b}(i)$ , is presented as:

$$\mathbf{p}_{wiener_k} = E[\mathbf{u}(i)b_k^*(i)] \quad (3.18)$$

$$= E[[\sqrt{P}\mathbf{H}\mathbf{v}\mathbf{b}(i) + \mathbf{n}(i)]b_k^*(i)] \quad (3.19)$$

$$= E[[\mathbf{M}\mathbf{b}(i) + \mathbf{n}(i)]b_k^*(i)] \quad (3.20)$$

$$= E[\mathbf{M}\mathbf{b}(i)b_k^*(i)] + E[\mathbf{n}(i)b_k^*(i)] \quad (3.21)$$

$$= \mathbf{M}E[\mathbf{b}(i)b_k^*(i)] + E[\mathbf{n}(i)b_k^*(i)] \quad (3.22)$$

Where  $E[\mathbf{b}(i)b_k^*(i)]$  is a  $N_s$ -by-1 vertical vector, in which only the  $k$ th element is one, all other elements in this vector are zeros;  $E[\mathbf{n}(i)b_k^*(i)]$  is a  $N_r$ -by-1 vertical null vector. Therefore:

$$\mathbf{p}_{wiener_k} = \mathbf{M}_k \quad (3.23)$$

Where the  $N_r$ -by-1 vector  $\mathbf{M}_k$  is the  $k$ th column of the mixing matrix  $\mathbf{M}$ .

Now, both of the statistical autocorrelation function  $\mathbf{R}_{wiener}$  between the input signal vector  $\mathbf{u}(i)$ , and the statistical cross-correlation function  $\mathbf{p}_{wiener_k}$  between the input signal vector  $\mathbf{u}(i)$  and the desired  $k$ th transmitted signal symbol  $b_k(i)$ , under MIMO channel communications systems, are derived out. Therefore, the  $N_r$ -by-1 Wiener estimated receive filter  $\mathbf{g}_{wiener_k}$  for the desired  $k$ th transmitted signal symbol  $b_k(i)$  is expressed as:

$$\mathbf{g}_{wiener_k} = \mathbf{R}_{wiener}^{-1}\mathbf{p}_{wiener_k} \quad (3.24)$$

Thus the reconstructed  $k$ th transmitted signal symbol at time  $i$  under Wiener estimation is:

$$\hat{b}_{wiener_k}(i) = \mathbf{g}_{wiener_k}^H \mathbf{u}(i) \quad (3.25)$$

The estimated error for the desired  $kth$  transmitted signal symbol at time  $i$  under Wiener estimation is:

$$e_{wiener_k}(i) = b_k(i) - \hat{b}_{wiener_k}(i) \quad (3.26)$$

$$= b_k(i) - \mathbf{g}_{wiener_k}^H \mathbf{u}(i) \quad (3.27)$$

For the convenience for the later experimental analysis, here the expressions of MMSE and SINR under Wiener estimation are derived.

Firstly, derive the expression of the MMSE in terms of the desired  $kth$  transmitted signal symbol.

$$MMSE_{wiener_k} = E[e_{wiener_k}(i)e_{wiener_k}^*(i)] \quad (3.28)$$

$$= E[(b_k(i) - \mathbf{g}_{wiener_k}^H \mathbf{u}(i))(b_k(i) - \mathbf{g}_{wiener_k}^H \mathbf{u}(i))^*] \quad (3.29)$$

$$= E[(b_k(i) - \mathbf{g}_{wiener_k}^H \mathbf{u}(i))(b_k^*(i) - \mathbf{u}^H(i)\mathbf{g}_{wiener_k})] \quad (3.30)$$

$$= E[b_k(i)b_k^*(i)] - E[b_k(i)\mathbf{u}^H(i)\mathbf{g}_{wiener_k}] - E[\mathbf{g}_{wiener_k}^H \mathbf{u}(i)b_k^*(i)] + E[\mathbf{g}_{wiener_k}^H \mathbf{u}(i)\mathbf{u}^H(i)\mathbf{g}_{wiener_k}] \quad (3.32)$$

$$= E[b_k(i)b_k^*(i)] - E[b_k(i)\mathbf{u}^H(i)]\mathbf{g}_{wiener_k} - \mathbf{g}_{wiener_k}^H E[\mathbf{u}(i)b_k^*(i)] + \mathbf{g}_{wiener_k}^H E[\mathbf{u}(i)\mathbf{u}^H(i)]\mathbf{g}_{wiener_k} \quad (3.34)$$

$$= 1 - \mathbf{p}_{wiener_k}^H \mathbf{g}_{wiener_k} - \mathbf{g}_{wiener_k}^H \mathbf{p}_{wiener_k} \quad (3.35)$$

$$+ \mathbf{g}_{wiener_k}^H \mathbf{R}_{wiener} \mathbf{g}_{wiener_k} \quad (3.36)$$

$$= 1 - \mathbf{p}_{wiener_k}^H \mathbf{g}_{wiener_k} - \mathbf{g}_{wiener_k}^H \mathbf{p}_{wiener_k} + \mathbf{p}_{wiener_k}^H \mathbf{g}_{wiener_k} \quad (3.37)$$

$$= 1 - \mathbf{g}_{wiener_k}^H \mathbf{p}_{wiener_k} \quad (3.38)$$

$$= 1 - \mathbf{g}_{wiener_k}^H \mathbf{R}_{wiener} \mathbf{g}_{wiener_k} \quad (3.39)$$

$$= 1 - \mathbf{p}_{wiener_k}^H \mathbf{g}_{wiener_k} \quad (3.40)$$

$$= 1 - \mathbf{p}_{wiener_k}^H \mathbf{R}_{wiener}^{-1} \mathbf{p}_{wiener_k} \quad (3.41)$$

Next, derive the expression of the SINR in terms of the desired  $kth$  transmitted signal symbol.

Considering at time  $i$ , the desired  $kth$  transmitted signal symbol  $b_k(i)$ , the interference symbols  $\sum_{j \neq k} b_j(i)$  and the additive noise vector  $\mathbf{n}(i)$ , in a

MIMO communications system, for the receiver, the input signal vector  $\mathbf{u}(i)$  can also be rearranged as:

$$\mathbf{u}(i) = \mathbf{M}_k b_k(i) + \sum_{j \neq k} \mathbf{M}_j b_j(i) + \mathbf{n}(i) \quad (3.42)$$

Where the term  $\mathbf{M}_k b_k(i)$  is the component for the desired  $k$ th transmitted signal symbol, the accumulation term  $\sum_{j \neq k} \mathbf{M}_j b_j(i)$  is the component for the interference symbols,  $\mathbf{M}_j$  is the  $j$ th column of the mixing matrix  $\mathbf{M}$  while  $b_j(i)$  is the  $j$ th element of the transmitted signal vector  $\mathbf{b}(i)$ ; the last term  $\mathbf{n}(i)$  is the additive noise component.

SINR, as this name suggests, is the ratio between the desired signal power and the sum of interference power and noise power. Looking at the receiver side, the statistical mean of the desired signal power is expressed as  $E[[\mathbf{g}_{wiener_k}^H \mathbf{M}_k b_k(i)][\mathbf{g}_{wiener_k}^H \mathbf{M}_k b_k(i)]^*]$  and the statistical mean of the total received signal power is expressed to be  $E[[\mathbf{g}_{wiener_k}^H \mathbf{u}(i)][\mathbf{g}_{wiener_k}^H \mathbf{u}(i)]^*]$ , so the statistical mean of the sum of interference power and noise power can be directly presented by a subtraction, as  $E[[\mathbf{g}_{wiener_k}^H \mathbf{u}(i)][\mathbf{g}_{wiener_k}^H \mathbf{u}(i)]^*] - E[[\mathbf{g}_{wiener_k}^H \mathbf{M}_k b_k(i)][\mathbf{g}_{wiener_k}^H \mathbf{M}_k b_k(i)]^*]$ .

Define:

$$\mathbf{M}_{-k} b_{-k}(i) = \sum_{j \neq k} \mathbf{M}_j b_j(i) \quad (3.43)$$

Correspondingly, the SINR can be expressed as:

$$SINR_{wiener_k} = \frac{E[[\mathbf{g}_{wiener_k}^H \mathbf{M}_k b_k(i)][\mathbf{g}_{wiener_k}^H \mathbf{M}_k b_k(i)]^*]}{E[[\mathbf{g}_{wiener_k}^H \mathbf{u}(i)][\mathbf{g}_{wiener_k}^H \mathbf{u}(i)]^*] - E[[\mathbf{g}_{wiener_k}^H \mathbf{M}_k b_k(i)][\mathbf{g}_{wiener_k}^H \mathbf{M}_k b_k(i)]^*]} \quad (3.44)$$

$$= \frac{E[[\mathbf{g}_{wiener_k}^H \mathbf{M}_k b_k(i)][b_k^*(i) \mathbf{M}_k^H \mathbf{g}_{wiener_k}]]}{E[[\mathbf{g}_{wiener_k}^H \mathbf{u}(i)][\mathbf{u}^H(i) \mathbf{g}_{wiener_k}]] - E[[\mathbf{g}_{wiener_k}^H \mathbf{M}_k b_k(i)][b_k^*(i) \mathbf{M}_k^H \mathbf{g}_{wiener_k}]]} \quad (3.45)$$

$$= \frac{\mathbf{g}_{wiener_k}^H \mathbf{M}_k E[b_k(i) b_k^*(i)] \mathbf{M}_k^H \mathbf{g}_{wiener_k}}{\mathbf{g}_{wiener_k}^H E[\mathbf{u}(i) \mathbf{u}^H(i)] \mathbf{g}_{wiener_k} - \mathbf{g}_{wiener_k}^H \mathbf{M}_k E[b_k(i) b_k^*(i)] \mathbf{M}_k^H \mathbf{g}_{wiener_k}} \quad (3.46)$$

$$= \frac{\mathbf{g}_{wiener_k}^H \mathbf{M}_k \mathbf{M}_k^H \mathbf{g}_{wiener_k}}{\mathbf{g}_{wiener_k}^H \mathbf{R}_{wiener} \mathbf{g}_{wiener_k} - \mathbf{g}_{wiener_k}^H \mathbf{M}_k \mathbf{M}_k^H \mathbf{g}_{wiener_k}} \quad (3.47)$$

Recall Equation 3.24, get:

$$SINR_{wiener_k} = \frac{\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k \mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k}{\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{R}_{wiener} \mathbf{R}_{wiener}^{-1} \mathbf{M}_k - \mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k \mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k} \quad (3.48)$$

$$= \frac{[\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k][\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k]}{\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k - [\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k][\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k]} \quad (3.49)$$

$$= \frac{|\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k|^2}{\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k - |\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k|^2} \quad (3.50)$$

$$= \frac{1}{1 - \mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k} \mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k \quad (3.51)$$

The statistical autocorrelation function of the input signal vector  $\mathbf{u}(i)$  can also be expressed as:

$$\mathbf{R}_{wiener} = \mathbf{R}_{wiener-k} + \mathbf{M}_k \mathbf{M}_k^H \quad (3.52)$$

Where  $\mathbf{R}_{wiener-k}$  is the statistical autocorrelation function of the interference-plus-noise vectors, defined as:

$$\mathbf{R}_{wiener-k} = E[(\mathbf{M}_{-k} b_{-k}(i) + \mathbf{n}(i))(\mathbf{M}_{-k} b_{-k}(i) + \mathbf{n}(i))^H] \quad (3.53)$$

$$= \sum_{j \neq k} \mathbf{M}_j \mathbf{M}_j^H + 0.01 \mathbf{I}_{N_r} \quad (3.54)$$

Apply the Matrix Inversion Lemma, or called the Woodbury Matrix Identity, on Equation 3.52:

$$\mathbf{R}_{wiener}^{-1} = \mathbf{R}_{wiener-k}^{-1} - \frac{1}{1 + \mathbf{M}_k^H \mathbf{R}_{wiener-k}^{-1} \mathbf{M}_k} \mathbf{R}_{wiener-k}^{-1} \mathbf{M}_k \mathbf{M}_k^H \mathbf{R}_{wiener-k}^{-1} \quad (3.55)$$

For the convenience for the mathematical proof, denote  $\gamma = \mathbf{M}_k^H \mathbf{R}_{wiener-k}^{-1} \mathbf{M}_k$ , then:

$$\mathbf{R}_{wiener}^{-1} = \mathbf{R}_{wiener-k}^{-1} - \frac{1}{1 + \gamma} \mathbf{R}_{wiener-k}^{-1} \mathbf{M}_k \mathbf{M}_k^H \mathbf{R}_{wiener-k}^{-1} \quad (3.56)$$

So:



$$\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k = \mathbf{M}_k^H (\mathbf{R}_{wiener-k}^{-1} - \frac{1}{1+\gamma} \mathbf{R}_{wiener-k}^{-1} \mathbf{M}_k \mathbf{M}_k^H \mathbf{R}_{wiener-k}^{-1}) \mathbf{M}_k \quad (3.57)$$

$$= \mathbf{M}_k^H \mathbf{R}_{wiener-k}^{-1} \mathbf{M}_k - \frac{1}{1+\gamma} \mathbf{M}_k^H \mathbf{R}_{wiener-k}^{-1} \mathbf{M}_k \mathbf{M}_k^H \mathbf{R}_{wiener-k}^{-1} \mathbf{M}_k \quad (3.58)$$

$$= \mathbf{M}_k^H \mathbf{R}_{wiener-k}^{-1} \mathbf{M}_k - \frac{1}{1+\gamma} [\mathbf{M}_k^H \mathbf{R}_{wiener-k}^{-1} \mathbf{M}_k] [\mathbf{M}_k^H \mathbf{R}_{wiener-k}^{-1} \mathbf{M}_k] \quad (3.59)$$

$$= \mathbf{M}_k^H \mathbf{R}_{wiener-k}^{-1} \mathbf{M}_k - \frac{1}{1+\gamma} [\mathbf{M}_k^H \mathbf{R}_{wiener-k}^{-1} \mathbf{M}_k]^2 \quad (3.60)$$

$$= \gamma - \frac{1}{1+\gamma} \gamma^2 \quad (3.61)$$

$$= \gamma (1 - \frac{\gamma}{1+\gamma}) \quad (3.62)$$

$$= \frac{\gamma}{1+\gamma} \quad (3.63)$$

Conclusively, the SINR for the desired  $k$ th transmitted signal symbol in Wiener estimation is expressed as:

$$SINR_{wiener_k} = \frac{1}{1 - \frac{\gamma}{1+\gamma}} \frac{\gamma}{1+\gamma} \quad (3.64)$$

$$= \gamma \quad (3.65)$$

$$= \mathbf{M}_k^H \mathbf{R}_{wiener-k}^{-1} \mathbf{M}_k \quad (3.66)$$

### 3.2 Least Squares (LS) Estimation

In this research project, Wiener estimation algorithm is merely applied in computer-based numerical experiments to compute theoretical optimal bounds when the CSI is known *a priori*. Since in this research project, communications systems are designed using bi-directional training, LS estimation is the important algorithm used to update matrices of the precoder and the receive filter iteratively by transmitted training symbol sequences.

Wiener estimation is derived in the statistical sense, while LS estimation involves the time-averaging technique, which depends on the number of sample points acquired in the computation [44]. It can be imagined that the aim of LS estimation is making a curve model to make the sum of the squares of the difference between the curve model points and the sample measurement

points as small as possible, so that the curve model fits these sample measurement points [44]. LS estimation is a deterministic data fitting approach in regression analysis. In the computational term, LS algorithm is a batch-processing method, because it processes batches of input data points [44].

The mathematical derivation of LS estimation from the very beginning can be referred in [44] and is not going to be presented here. Only the extension of some expressions based on existing conclusions purified from [44] for MIMO communications systems is provided.

The system model adopted here is totally the same as the one used in Section 3.1. Nevertheless, in LS estimation, the time-averaging technique is introduced into use.

Recall Equation 3.3, where the autocorrelation function of the input signal vector and the cross-correlation function of the input signal vector and the desired signal symbol are time-averaging ones here. Denote  $L$  as the length of each transmitted training symbol sequence.

The time-averaging autocorrelation function  $\mathbf{R}_{ls}$  of the input signal vector  $\mathbf{u}(i)$  is:

$$\mathbf{R}_{ls} = \frac{1}{L} \sum_{i=1}^L \mathbf{u}(i) \mathbf{u}^H(i) \quad (3.67)$$

The time-averaging cross-correlation function  $\mathbf{p}_{ls_k}$  between the input signal vector  $\mathbf{u}(i)$  and the desired transmitted signal symbol  $b_k(i)$  is:

$$\mathbf{p}_{ls_k} = \frac{1}{L} \sum_{i=1}^L \mathbf{u}(i) b_k^*(i) \quad (3.68)$$

Therefore, the LS estimated receive filter  $\mathbf{g}_{ls_k}$  for the desired transmitted signal symbol  $b_k(i)$  is expressed as:

$$\mathbf{g}_{ls_k} = \mathbf{R}_{ls}^{-1} \mathbf{p}_{ls_k} \quad (3.69)$$

Thus the reconstructed  $k$ th transmitted signal symbol at time  $i$  is expressed as:

$$\hat{b}_k(i) = \mathbf{g}_{ls_k}^H \mathbf{u}(i) \quad (3.70)$$

The estimated error for the  $k$ th transmitted signal symbol under LS estimation is:

$$e_{ls_k}(i) = b_k(i) - \hat{b}_{ls_k}(i) \quad (3.71)$$

$$= b_k(i) - \mathbf{g}_{ls_k}^H \mathbf{u}(i) \quad (3.72)$$

The next step is to derive the expressions of MSE and SINR under LS estimation for the convenience of the later experimental analysis.

Firstly, derive the expression of MSE.

The expression of the MSE for the  $k$ th transmitted signal symbol is expressed as:

$$MSE_{ls_k} = E[|e_{ls_k}(i)e_{ls_k}^*(i)|] \quad (3.73)$$

$$= E[(b_k(i) - \mathbf{g}_{ls_k}^H \mathbf{u}(i))(b_k(i) - \mathbf{g}_{ls_k}^H \mathbf{u}(i))^*] \quad (3.74)$$

$$= E[(b_k(i) - \mathbf{g}_{ls_k}^H \mathbf{u}(i))(b_k^*(i) - \mathbf{g}_{ls_k}^T \mathbf{u}^*(i))] \quad (3.75)$$

$$= E[b_k(i)b_k^*(i)] - E[b_k(i)\mathbf{g}_{ls_k}^T \mathbf{u}^*(i)] - E[b_k^*(i)\mathbf{g}_{ls_k}^H \mathbf{u}(i)] + E[\mathbf{g}_{ls_k}^H \mathbf{u}(i)\mathbf{u}^H(i)\mathbf{g}_{ls_k}^T] \quad (3.76)$$

$$= E[b_k(i)b_k^*(i)] - \mathbf{g}_{ls_k}^T E[b_k(i)\mathbf{u}^*(i)] - \mathbf{g}_{ls_k}^H E[b_k^*(i)\mathbf{u}(i)] + \mathbf{g}_{ls_k}^H E[\mathbf{u}(i)\mathbf{u}^H(i)]\mathbf{g}_{ls_k}^T \quad (3.77)$$

$$= 1 - \mathbf{g}_{ls_k}^T \mathbf{p}_{wiener_k}^* - \mathbf{g}_{ls_k}^H \mathbf{p}_{wiener_k} + \mathbf{g}_{ls_k}^H \mathbf{R}_{wiener} \mathbf{g}_{ls_k} \quad (3.78)$$

Notice, in Equation 3.78, both  $\mathbf{p}_{wiener_k}$  and  $\mathbf{R}_{wiener}$  are statistical functions, rather than time-averaging ones.

Next, turn the attention to the expression of the SINR in terms of the  $k$ th transmitted signal symbol.

In this research project, for LS estimation, the SINR can be straightforwardly expressed by its literal significance, the ratio between the statistical mean of the desired signal power  $E[|\mathbf{g}_{ls_k}^H \mathbf{M}_k b_k(i)|^2]$  and the statistical mean of the sum of interference power and noise power  $E[\sum_{j \neq k} |\mathbf{g}_{ls_k}^H \mathbf{M}_j b_j(i)|^2] + E[|\mathbf{g}_{ls_k}^H \mathbf{n}(i)|^2]$ , without too many sophisticated calculations, viz:

$$SINR_{l_{s_k}} = \frac{E[|\mathbf{g}_{l_{s_k}}^H \mathbf{M}_k b_k(i)|^2]}{E[\sum_{j \neq k} |\mathbf{g}_{l_{s_k}}^H \mathbf{M}_j b_j(i)|^2] + E[|\mathbf{g}_{l_{s_k}}^H \mathbf{n}(i)|^2]} \quad (3.79)$$

$$= \frac{E[[\mathbf{g}_{l_{s_k}}^H \mathbf{M}_k b_k(i)][\mathbf{g}_{l_{s_k}}^H \mathbf{M}_k b_k(i)]^*]}{E[\sum_{j \neq k} [\mathbf{g}_{l_{s_k}}^H \mathbf{M}_j b_j(i)][\mathbf{g}_{l_{s_k}}^H \mathbf{M}_j b_j(i)]^*] + E[[\mathbf{g}_{l_{s_k}}^H \mathbf{n}(i)][\mathbf{g}_{l_{s_k}}^H \mathbf{n}(i)]^*]} \quad (3.80)$$

$$= \frac{E[[\mathbf{g}_{l_{s_k}}^H \mathbf{M}_k b_k(i)][b_k^*(i) \mathbf{M}_k^H \mathbf{g}_{l_{s_k}}]]}{E[\sum_{j \neq k} [\mathbf{g}_{l_{s_k}}^H \mathbf{M}_j b_j(i)][b_j^*(i) \mathbf{M}_j^H \mathbf{g}_{l_{s_k}}]] + E[[\mathbf{g}_{l_{s_k}}^H \mathbf{n}(i)][\mathbf{n}^H(i) \mathbf{g}_{l_{s_k}}]]} \quad (3.81)$$

$$= \frac{|\mathbf{g}_{l_{s_k}}^H \mathbf{M}_k E[b_k(i) b_k^*(i)] \mathbf{M}_k^H \mathbf{g}_{l_{s_k}}|}{\sum_{j \neq k} |\mathbf{g}_{l_{s_k}}^H \mathbf{M}_j E[b_j(i) b_j^*(i)] \mathbf{M}_j^H \mathbf{g}_{l_{s_k}}| + |\mathbf{g}_{l_{s_k}}^H E[\mathbf{n}(i) \mathbf{n}^H(i)] \mathbf{g}_{l_{s_k}}|} \quad (3.82)$$

$$= \frac{|\mathbf{g}_{l_{s_k}}^H \mathbf{M}_k \mathbf{M}_k^H \mathbf{g}_{l_{s_k}}|}{\sum_{j \neq k} |\mathbf{g}_{l_{s_k}}^H \mathbf{M}_j \mathbf{M}_j^H \mathbf{g}_{l_{s_k}}| + |\mathbf{g}_{l_{s_k}}^H 0.01 \mathbf{g}_{l_{s_k}}|} \quad (3.83)$$

$$= \frac{|\mathbf{g}_{l_{s_k}}^H \mathbf{M}_k|^2}{\sum_{j \neq k} |\mathbf{g}_{l_{s_k}}^H \mathbf{M}_j|^2 + 0.01 |\mathbf{g}_{l_{s_k}}^H \mathbf{g}_{l_{s_k}}|} \quad (3.84)$$

### 3.3 Reduced-Rank Filtering

In the engineering of MIMO communications systems, the precoder and the receive filter can be designed by adaptive bi-directional training, as introduced in Section 1.1. In the process of bi-directional training, training symbol sequences with a certain length are required as pilot signals to update the precoder and the receive filter in each iteration to form the effect of beam-forming. This engineering mechanism does work well when the scale of the MIMO communications system is limited to a certain range; however, when the system scale grows larger to cater more sophisticated needs, especially under mMIMO systems, the overhead of training symbol sequences grows linearly with the length of the filter [1]. This disruptive effect might severely reduce the working efficiency of the communications system. As a result, reduced-rank filtering technique is proposed as a solution to alleviate this technical problem.

In reduced-rank filtering, filtering and filter estimation occur within a lower-dimensional projection subspace, to reduce the required length of the training sequence, the number of filter parameters to estimate and the feedback requirement. In a large-scaled MIMO communications system, within a certain training length, reduced-rank filtering significantly outperforms the full-rank counterpart, in terms of the MSE or SINR, the two criteria used to evaluate the system performance. Nevertheless, this convenience is by no

means for free, as the training length gradually becomes long enough, the asymptotic MMSE achieved by reduced-rank filtering might be higher than the counterpart achieved by full-rank filtering. The smaller the rank, the higher the asymptotic MMSE achieved. When the rank of the reduced-rank projection subspace matrix is equal to that of the originally received signal space, the asymptotic MMSE achieved should be near with that of full-rank filtering.

There are several different methods to design the reduced-rank projection subspace, such as eigen-space methods (including Principal Components (PC) method, Generalized Side-lobe Canceller (GSC) method and cross-spectral method) and Krylov subspace methods (including MSWF method and rank-recursive (conjugate gradient) algorithm) [1]. In this research project, the Krylov subspace generated by the MSWF method is adopted to use as the reduced-rank projection subspace. Figure 3.3 visualizes the internal structure of a MSWF.

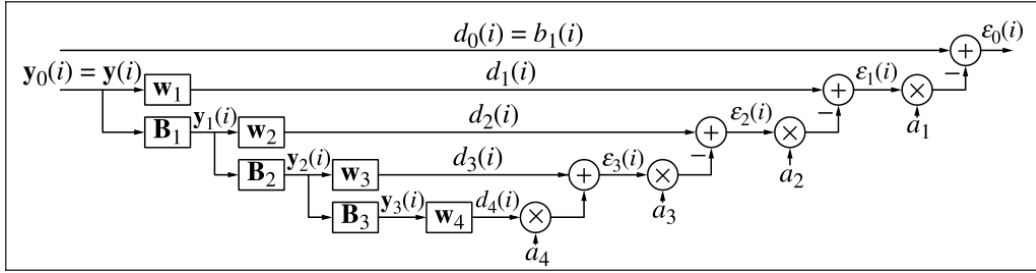


Figure 3.3: Multi-Stage Wiener Filter [1]

As the model of the MIMO communications system adopted in Section 3.1 suggests, it is assumed that there are one transmitter with  $N_t$  transmit antennas, one receiver with  $N_r$  receive antennas and  $N_s$  transmitted signal streams iterated between the transmitter terminal and the receiver terminal. When the rank of the Krylov subspace is set to be  $D$ ,  $1 \leq D < N_r$ , the  $N_r$ -by- $D$  Krylov subspace matrix  $S_k$  for the  $k$ th transmitted signal symbol is expressed to be:

$$\mathbf{S}_k = [\mathbf{p}_k \ \mathbf{R}\mathbf{p}_k \ \mathbf{R}^2\mathbf{p}_k \ \dots \ \mathbf{R}^{D-1}\mathbf{p}_k] \quad (3.85)$$

Where  $\mathbf{R}$  is the autocorrelation function of the input signal vector and  $\mathbf{p}_k$  is the cross-correlation function between the input signal vector and the desired signal symbol. Columns of this matrix span a  $D$ -dimensional Krylov subspace. To exploit the advantage brought by reduced-rank filtering over the full-rank counterpart, the received signal vector is always projected into

this subspace matrix to reduce the signal dimension.

In Wiener estimation, the so-called autocorrelation function  $\mathbf{R}$  and the cross-correlation function  $\mathbf{p}_k$  are the statistical ones, given by Equations 3.17 and 3.23, respectively; whereas in LS estimation, these two functions are the time-averaging ones, given by Equations 3.67 and 3.68, respectively.

Thus, in Wiener estimation, the  $N_r$ -by- $D$  Krylov subspace matrix for the  $k$ th transmitted signal symbol is expressed as:

$$\mathbf{S}_{wiener_k} = [\mathbf{p}_{wiener_k} \mathbf{R}_{wiener} \mathbf{p}_{wiener_k} \mathbf{R}_{wiener}^2 \mathbf{p}_{wiener_k} \cdots \mathbf{R}_{wiener}^{D-1} \mathbf{p}_{wiener_k}] \quad (3.86)$$

Similarly, in LS estimation, the Krylov subspace matrix for the  $k$ th transmitted signal symbol is presented as:

$$\mathbf{S}_{ls_k} = [\mathbf{p}_{ls_k} \mathbf{R}_{ls} \mathbf{p}_{ls_k} \mathbf{R}_{ls}^2 \mathbf{p}_{ls_k} \cdots \mathbf{R}_{ls}^{D-1} \mathbf{p}_{ls_k}] \quad (3.87)$$

In this section, procedures of mathematical derivations of receive filters, MSE expressions and SINR expressions under reduced-rank filtering are not going to be presented step by step, since they are all directly based on the derivations given in Sections 3.1 and 3.2. However, considering in reduced-rank filtering, the input signal vector is always projected into the Krylov subspace matrix, a simple and intuitive explanation is easy to understand: for the full-rank autocorrelation function  $\mathbf{R}$  of the input signal vector, it should be multiplied with a Krylov subspace matrix on the left and another one on the right to obtain the corresponding reduced-rank autocorrelation function, since this autocorrelation function contains a square of the input signal vectors; for the full-rank cross-correlation function  $\mathbf{p}_k$  between the input signal vector and the desired signal symbol, a Krylov subspace matrix should be multiplied on the left to get the corresponding reduced-rank cross-correlation function, because this cross-correlation involves an input signal vector. This explanation applies in both Wiener estimation and LS estimation.

Owning the Krylov subspace matrix, the reduced-rank received filter  $\mathbf{g}_{rr\ k}$  for the  $k$ th transmitted signal symbol is provided as:

$$\mathbf{g}_{rr\ k} = \mathbf{S}_k (\mathbf{S}_k^H \mathbf{R} \mathbf{S}_k)^{-1} \mathbf{S}_k^H \mathbf{p}_k \quad (3.88)$$

In Wiener estimation, the statistical autocorrelation function of the input signal vector is given by Equation 3.17 and the statistical cross-correlation

function between the input signal vector and the desired signal symbol is provided by Equation 3.23, thus the reduced-rank receive filter  $\mathbf{g}_{rr\ wiener_k}$  for the  $k$ th transmitted signal symbol is expressed as:

$$\mathbf{g}_{rr\ wiener_k} = \mathbf{S}_{wiener_k} (\mathbf{S}_{wiener_k}^H \mathbf{R}_{wiener} \mathbf{S}_{wiener_k})^{-1} \mathbf{S}_{wiener_k}^H \mathbf{p}_{wiener_k} \quad (3.89)$$

In LS estimation, the time-averaging autocorrelation function of the input signal vector is given by Equation 3.67 and the time-averaging cross-correlation function between the input signal vector and the desired signal symbol is provided by Equation 3.68, the reduced-rank receive filter  $\mathbf{g}_{rr\ ls_k}$  for the  $k$ th transmitted signal symbol is given as:

$$\mathbf{g}_{rr\ ls_k} = \mathbf{S}_{ls_k} (\mathbf{S}_{ls_k}^H \mathbf{R}_{ls} \mathbf{S}_{ls_k})^{-1} \mathbf{S}_{ls_k}^H \mathbf{p}_{ls_k} \quad (3.90)$$

The expressions of MSE and SINR under reduced-rank filtering are also presented.

Firstly, focus on the expression of MSE.

Under Wiener estimation, the expression of the MSE for the  $k$ th transmitted signal symbol is presented as:

$$MSE_{rr\ wiener_k} = 1 - \mathbf{p}_{wiener_k}^H \mathbf{S}_{wiener_k} (\mathbf{S}_{wiener_k}^H \mathbf{R}_{wiener} \mathbf{S}_{wiener_k})^{-1} \mathbf{S}_{wiener_k}^H \mathbf{p}_{wiener_k} \quad (3.91)$$

In LS estimation, the MSE of the  $k$ th transmitted signal symbol is presented as:

$$MSE_{rr\ ls_k} = 1 - \mathbf{g}_{rr\ ls_k}^T \mathbf{p}_{wiener_k}^* - \mathbf{g}_{rr\ ls_k}^H \mathbf{p}_{wiener_k} + \mathbf{g}_{rr\ ls_k}^H \mathbf{R}_{wiener} \mathbf{g}_{rr\ ls_k} \quad (3.92)$$

Notice, in Equation 3.92, both the  $\mathbf{p}_{wiener_k}$  and  $\mathbf{R}_{wiener}$  are statistical functions, not time-averaging ones.

Next, turn the attention to the expressions of SINR.

The expression of the SINR for the  $k$ th transmitted signal symbol in Wiener estimation under reduced-rank filtering is expressed as:

$$SINR_{rr\ wiener_k} = \mathbf{p}_{wiener_k}^H \mathbf{S}_{wiener_k} (\mathbf{S}_{wiener_k}^H \mathbf{R}_{wiener-k} \mathbf{S}_{wiener_k})^{-1} \mathbf{S}_{wiener_k}^H \mathbf{p}_{wiener_k} \quad (3.93)$$

Where  $\mathbf{R}_{wiener-k}$  is the statistical interference-plus-noise autocorrelation function, which is given by Equation 3.54.

In LS estimation, for the  $k$ th transmitted signal symbol, under reduced-rank filtering, the SINR is given as:

$$SINR_{rr\ l s_k} = \frac{|\mathbf{g}_{rr\ l s_k}^H \mathbf{M}_k|^2}{\sum_{j \neq k} |\mathbf{g}_{rr\ l s_k}^H \mathbf{M}_j|^2 + 0.01 |\mathbf{g}_{rr\ l s_k}^H \mathbf{g}_{rr\ l s_k}|} \quad (3.94)$$

The aforesaid expressions will be directly applied in the later experimental simulation analysis.

### 3.4 Bi-Directional Optimization and Training

In wireless communications systems, training is often used to update the precoder and the receive filter with the help of training symbol sequences. In uni-directional training, where training only exists in the forward direction, only the receive filter is updated by the received training symbol sequences. To further improve the design of a MIMO channel communications system, in this research project, bi-directional training, a reasonable extension of uni-directional training, is adopted to use.

In bi-directional training, training occurs in both the downlink direction and the uplink direction. In this research project, training in the forward direction is chosen to implement at first, which is from the transmitter terminal to the receiver terminal. In forward training, the transmitter sends a packet containing training symbol sequences via a randomly initialized normalized precoder to the side of the receiver, then the receiver updates its receive filter by these received training symbol sequences and normalizes the updated filter; in backward training, roles played by the precoder and the receive filter are swapped: the newly trained receive filter acts as the precoder for the receiver, which sends a packet containing training symbol sequences back to the transmitter terminal, then the transmitter updates its precoder also with power normalization. The aforementioned training procedure is iterated for the pre-set time or until the pre-set convergence requirement is met. After this bi-directional training, useful data information can be therewith transmitted.

Frames containing training symbol sequences and useful data information



in the process of bi-directional training can be visually illustrated by Figure 3.4.

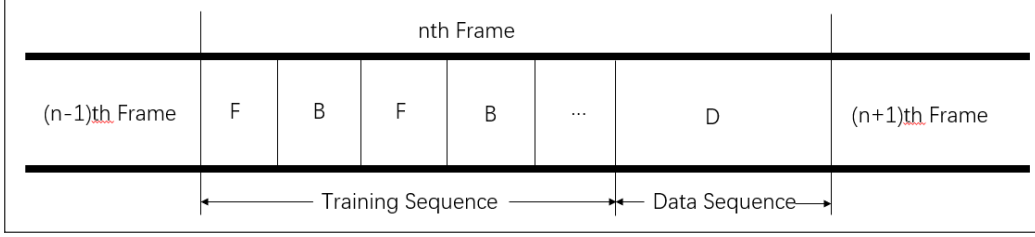


Figure 3.4: TDD data frames in bi-directional training

In Figure 3.4, the frame sitting in the middle is the  $n$ th frame concerned, the one on the leftmost is the  $(n - 1)$ th frame and the one on the rightmost is the  $(n + 1)$ th frame. The  $n$ th frame is started with the forward training symbol sequences  $F$  and the backward training symbol sequences  $B$  comes the next. In practice, training symbol sequences can also be started with the backward training sequences, depending on the need, such as in [28]. When bi-directional training is finished, whereafter useful data information can be started to transmit. The internal structure of each frame is the same. The whole process of bi-directional training is based on the TDD mechanism within a certain coherence time, thus the problem of channel variation is not considered.

An important problem of terminology is worth to be emphasized. In this research project report, there is a little difference between bi-directional optimization and bi-directional training. In the former one, it is assumed that both the transmitter and the receiver know the CSI *a priori*, thus Wiener estimation can be applied to achieve the statistical optimal performance and the filter estimation is not related with the length of each training symbol sequence; conversely, in the later one, the CSI is assumed to be unknown, both the precoder and the receive filter can only be updated via training symbol sequences using LS estimation, thus the filter estimation is firmly related with the length of each training symbol sequence. The longer the length of each training symbol sequence, the better the filter estimation and accordingly the better the system performance

Both bi-directional optimization and bi-directional training will be discussed in details. In both of them, the distributed mechanism is used, which means there is no such a centralized controller responsible for calculating the precoder matrix and the receive filter matrix, the two matrices can only be

computed in their respective terminals.

### 3.4.1 Bi-Directional Optimization

In bi-directional optimization, both of the transmitter and the receiver own the knowledge of the CSI *a priori*, accordingly, Wiener estimation can be directly applied to compute the optimal precoder and the optimal receive filter in each bi-directional optimization iteration. The procedure of bi-directional optimization can be given by mathematical expressions. The system model used is the same as the one presented in Section 3.1.

First of all, at the transmitter side, the initialized normalized precoder  $\mathbf{v}_k$  for the  $k$ th transmitted signal symbol, can be randomly generated, such as by the uniformly distributed random number generation scheme. This step is also the initialization of the whole bi-directional optimization algorithm.

Next, at the receiver side, owning the knowledge of the CSI, the receiver can straightforwardly compute the optimized receive filter for the  $k$ th transmitted signal symbol by Wiener estimation approach with power normalization as:

$$\mathbf{g}_{wiener_k} = \frac{(\mathbf{R}_{wiener})^{-1} \mathbf{p}_{wiener_k}}{|(\mathbf{R}_{wiener})^{-1} \mathbf{p}_{wiener_k}|} \quad (3.95)$$

Where the statistical autocorrelation function  $\mathbf{R}_{wiener}$  of the input signal vector is given by Equation 3.17, whereas the statistical cross-correlation function  $\mathbf{p}_k$  between the input signal vector and the desired signal symbol is provided by Equation 3.23.

If reduced-rank filtering is added into consideration, the power normalized reduced-rank receive filter for the  $k$ th transmitted signal symbol is:

$$\mathbf{g}_{rr\ wiener_k} = \frac{\mathbf{S}_{wiener_k} (\mathbf{S}_{wiener_k}^H \mathbf{R}_{wiener} \mathbf{S}_{wiener_k})^{-1} \mathbf{S}_{wiener_k}^H \mathbf{p}_{wiener_k}}{|\mathbf{S}_{wiener_k} (\mathbf{S}_{wiener_k}^H \mathbf{R}_{wiener} \mathbf{S}_{wiener_k})^{-1} \mathbf{S}_{wiener_k}^H \mathbf{p}_{wiener_k}|} \quad (3.96)$$

Where the statistical Krylov subspace matrix  $\mathbf{S}_{wiener_k}$  for the  $k$ th transmitted signal symbol is given by Equation 3.86.

Backward optimization comes the next. Although the algorithm applied is the same, due to the principle of channel reciprocity in the TDD transmission mechanism, in the backward direction, the MIMO channel matrix  $\bar{\mathbf{H}}$  is the transposed form of the forward MIMO channel matrix  $\mathbf{H}$ , viz:

$$\overleftarrow{\mathbf{H}} = \mathbf{H}^T \quad (3.97)$$

Also due to this principle, the complex conjugated receive filter  $\mathbf{g}_k^*$  acts as the precoder for the receiver in the backward direction.

In the transmitter side, at time  $i$ , the received signal vector  $\overleftarrow{\mathbf{u}}(i)$  is expressed as:

$$\overleftarrow{\mathbf{u}}(i) = \sqrt{P}\overleftarrow{\mathbf{H}}\mathbf{g}^*\overleftarrow{\mathbf{b}}(i) + \overleftarrow{\mathbf{n}}(i) \quad (3.98)$$

$$= \overleftarrow{\mathbf{M}}\overleftarrow{\mathbf{b}}(i) + \overleftarrow{\mathbf{n}}(i) \quad (3.99)$$

Where  $\overleftarrow{\mathbf{b}}(i)$  and  $\overleftarrow{\mathbf{n}}(i)$  are the transmitted signal vector and the additive noise at time  $i$  in the backward direction, respectively. The mixing matrix for the backward direction  $\overleftarrow{\mathbf{M}} = \sqrt{P}\overleftarrow{\mathbf{H}}\mathbf{g}^*$ .

Hence, the statistical autocorrelation function  $\overleftarrow{\mathbf{R}}_{wiener}$  of the input signal vector in the backward direction is:

$$\overleftarrow{\mathbf{R}}_{wiener} = E[\overleftarrow{\mathbf{u}}(i)\overleftarrow{\mathbf{u}}(i)^H] \quad (3.100)$$

$$= \overleftarrow{\mathbf{M}}\overleftarrow{\mathbf{M}}^H + 0.01I_{N_t} \quad (3.101)$$

Where  $I_{N_t}$  is a  $N_t$ -by- $N_t$  identity matrix.

The statistical cross-correlation function  $\overleftarrow{\mathbf{p}}_{wiener_k}$  for the  $k$ th transmitted signal symbol in the backward direction is:

$$\overleftarrow{\mathbf{p}}_{wiener_k} = E[\overleftarrow{\mathbf{u}}(i)\overleftarrow{b}_k(i)^*] \quad (3.102)$$

$$= \overleftarrow{\mathbf{M}}_k \quad (3.103)$$

Therefore, the power normalized optimized precoder for the  $k$ th transmitted signal symbol in the backward optimization is expressed as:

$$\overleftarrow{\mathbf{g}}_{wiener_k} = \frac{((\overleftarrow{\mathbf{R}}_{wiener})^{-1}\overleftarrow{\mathbf{p}}_{wiener_k})^*}{|((\overleftarrow{\mathbf{R}}_{wiener})^{-1}\overleftarrow{\mathbf{p}}_{wiener_k})^*|} \quad (3.104)$$

If considering reduced-rank filtering, in the backward direction, the statistical Krylov subspace  $\overleftarrow{\mathbf{S}}_{wiener_k}$  for the  $k$ th transmitted signal symbol is expressed as:

$$\overleftarrow{\mathbf{S}}_{wiener_k} = [\overleftarrow{\mathbf{p}}_{wiener_k} \overleftarrow{\mathbf{R}}_{wiener} \overleftarrow{\mathbf{p}}_{wiener_k} \overleftarrow{\mathbf{R}}_{wiener}^2 \overleftarrow{\mathbf{p}}_{wiener_k} \dots \overleftarrow{\mathbf{R}}_{wiener}^{D-1} \overleftarrow{\mathbf{p}}_{wiener_k}] \quad (3.105)$$

Therefore, in the backward direction, the power normalized precoder for the  $k$ th transmitted signal symbol is:

$$\overleftarrow{\mathbf{g}}_{rr\ wiener_k} = \frac{(\overleftarrow{\mathbf{S}}_{wiener_k} (\overleftarrow{\mathbf{S}}_{wiener_k}^H \overleftarrow{\mathbf{R}}_{wiener} \overleftarrow{\mathbf{S}}_{wiener_k})^{-1} \overleftarrow{\mathbf{S}}_{wiener_k}^H \overleftarrow{\mathbf{p}}_{wiener_k})^*}{|(\overleftarrow{\mathbf{S}}_{wiener_k} (\overleftarrow{\mathbf{S}}_{wiener_k}^H \overleftarrow{\mathbf{R}}_{wiener} \overleftarrow{\mathbf{S}}_{wiener_k})^{-1} \overleftarrow{\mathbf{S}}_{wiener_k}^H \overleftarrow{\mathbf{p}}_{wiener_k})^*|} \quad (3.106)$$

This is backward optimization, after which, forward optimization for the next round can be started.

In bi-directional optimization, these aforementioned steps are repeated for the pre-defined time or until the certain convergence requirement is met. Then the transmission of useful data information can be started with an optimized system performance.

### 3.4.2 Bi-Directional Training

What is different with bi-directional optimization is that, in bi-directional training, both of the transmitter and the receiver have no knowledge of the CSI. On account of the deficiency of the CSI, one approach is to estimate the CSI at first at the receiver terminal, then relay it back to the transmitter terminal. This approach is technically feasible and is adopted in several ways, such as in [46–49]. However, as the scale of the communications system becomes increasingly large, exchanging the channel information may be difficult to implement [28]. In this situation, bi-directional training is adopted to update the precoder and the receive filter by training symbol sequences. The procedure of bi-directional training will be given in mathematical expressions. The system model adopted is the same as the one presented in Section 3.1. The main steps in bi-directional training are basically the same with the counterparts in bi-directional optimization.

At first, looking at the transmitter side, the initialized normalized precoder  $\mathbf{v}_k$  for the  $k$ th transmitted signal symbol can be generated by the uniformly distributed random number generation scheme, as the initialization of the whole bi-directional training.

Then, in the receiver terminal, by received training symbol sequences, in which the length of each transmitted training symbol sequence is  $L$ , the power normalized trained receive filter for the  $kth$  transmitted signal symbol is:

$$\mathbf{g}_{ls_k} = \frac{(\mathbf{R}_{ls})^{-1} \mathbf{p}_{ls_k}}{|(\mathbf{R}_{ls})^{-1} \mathbf{p}_{ls_k}|} \quad (3.107)$$

Where the time-averaging autocorrelation function  $\mathbf{R}_{ls}$  between the input signal vector is provided by Equation 3.67, and the time-averaging cross-correlation function  $\mathbf{p}_{ls_k}$  between the input signal vector and the desired signal symbol is given by Equation 3.68.

What is more, when the reduced-rank filtering is put into consideration, the power normalized reduced-rank receive filter for the  $kth$  transmitted signal symbol is expressed by:

$$\mathbf{g}_{rr\ ls_k} = \frac{\mathbf{S}_{ls_k} (\mathbf{S}_{ls_k}^H \mathbf{R}_{ls} \mathbf{S}_{ls_k})^{-1} \mathbf{S}_{ls_k}^H \mathbf{p}_{ls_k}}{|\mathbf{S}_{ls_k} (\mathbf{S}_{ls_k}^H \mathbf{R}_{ls} \mathbf{S}_{ls_k})^{-1} \mathbf{S}_{ls_k}^H \mathbf{p}_{ls_k}|} \quad (3.108)$$

Where the time-averaging Krylov subspace matrix  $\mathbf{S}_{ls_k}$  for the  $kth$  transmitted signal symbol is provided by Equation 3.87.

Now, forward training is ended and backward training subsequently comes. Similarly as in bi-directional optimization, on account of the principle of channel reciprocity, now the backward MIMO channel matrix  $\overleftarrow{\mathbf{H}}$  is the transposed form of the forward MIMO channel matrix  $\mathbf{H}$ , and what acts as the precoder for the receiver is also the complex conjugated form of the receive filter  $\mathbf{g}_k^*$ . Thus, the time-averaging autocorrelation function  $\overleftarrow{\mathbf{R}}_{ls}$  between the input signal vector in the backward direction is given as:

$$\overleftarrow{\mathbf{R}}_{ls} = \frac{1}{L} \sum_{i=1}^L \overleftarrow{\mathbf{u}}(i) \overleftarrow{\mathbf{u}}(i)^H \quad (3.109)$$

The time-averaging cross-correlation function  $\overleftarrow{\mathbf{p}}_{ls_k}$ , for the  $kth$  transmitted signal symbol in the backward direction is expressed as:

$$\overleftarrow{\mathbf{p}}_{ls_k} = \frac{1}{L} \sum_{i=1}^L \overleftarrow{\mathbf{u}}(i) \overleftarrow{b}_k(i)^* \quad (3.110)$$

Therefore, the power normalized updated precoder for the  $kth$  transmitted signal symbol in backward training is given as:

$$\overleftarrow{\mathbf{g}}_{ls_k} = \frac{((\overleftarrow{\mathbf{R}}_{ls})^{-1} \overleftarrow{\mathbf{p}}_{ls_k})^*}{|((\overleftarrow{\mathbf{R}}_{ls})^{-1} \overleftarrow{\mathbf{p}}_{ls_k})^*|} \quad (3.111)$$

When reduced-rank filtering is put into consideration, in backward training, the time-averaging Krylov subspace  $\overleftarrow{\mathbf{S}}_{ls_k}$ , for the  $k$ th transmitted signal symbol is given as:

$$\overleftarrow{\mathbf{S}}_{ls_k} = [\overleftarrow{\mathbf{p}}_{ls_k} \ \overleftarrow{\mathbf{R}}_{ls} \overleftarrow{\mathbf{p}}_{ls_k} \ \overleftarrow{\mathbf{R}}_{ls}^2 \overleftarrow{\mathbf{p}}_{ls_k} \ \dots \ \overleftarrow{\mathbf{R}}_{ls}^{D-1} \overleftarrow{\mathbf{p}}_{ls_k}] \quad (3.112)$$

Thus, in backward training, the power normalized trained precoder for the  $k$ th transmitted signal symbol is:

$$\overleftarrow{\mathbf{g}}_{rr\ ls_k} = \frac{(\overleftarrow{\mathbf{S}}_{ls_k} (\overleftarrow{\mathbf{S}}_{ls_k}^H \overleftarrow{\mathbf{R}}_{ls} \overleftarrow{\mathbf{S}}_{ls_k})^{-1} \overleftarrow{\mathbf{S}}_{ls_k}^H \overleftarrow{\mathbf{p}}_{ls_k})^*}{|(\overleftarrow{\mathbf{S}}_{ls_k} (\overleftarrow{\mathbf{S}}_{ls_k}^H \overleftarrow{\mathbf{R}}_{ls} \overleftarrow{\mathbf{S}}_{ls_k})^{-1} \overleftarrow{\mathbf{S}}_{ls_k}^H \overleftarrow{\mathbf{p}}_{ls_k})^*|} \quad (3.113)$$

If it is needed, forward training for the next round can be started subsequently.

These aforesaid steps are repeated for the pre-defined time or until the pre-defined convergence criterion is met. After this bi-directional training, useful data information can be transmitted into the communications channel.

# Chapter 4

## Experiments and Results

Both theoretical background knowledge and involved algorithms have been introduced, which lay the cornerstone of the experimental simulation and results in this chapter. In this chapter, a series of step-by-step experiments will be performed and analyzed as the verification of aforementioned theories. The experimental simulation begins with uni-directional optimization and training in the scenario where there is only one transmitter-receiver pair, then proceeds to bi-directional optimization and training under the same scenario; in the end, bi-directional optimization and training under an interference network, where there are multiple transmitter-receiver pairs, are performed. Details of each experiment are demonstrated in each corresponding section. Program developed for the experimental simulation is attached in Appendices in the end of this research project report, where Appendix A contains main functions and Appendix B presents self-defined functions.

Before the presentation of experimental results, there are two key points worthy to be emphasized in advance.

Firstly, in this research project, training symbols are generated using the statistically randomly generated Binary Phase-Shift Keying (BPSK) scheme, the simplest form of Phase-Shift Keying (PSK), with the mean zero and variance one. Figure 4.1 visualizes the transmitted training symbol sequence. Useful data information can be generated using Quadrature Phase-Shift Keying (QPSK), a complex form of PSK, but the transmission of useful data information is beyond the scope of this research project. Noise symbols are generated by complex Gaussian distribution with the mean zero and variance 0.01.

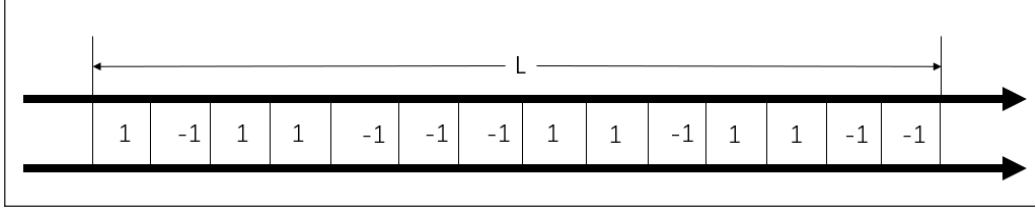


Figure 4.1: Transmitted training symbol sequence

In Figure 4.1, it can be seen that each statistically randomly generated training symbol is either 1 or -1, each with probability  $\frac{1}{2}$ . Here the length of this training symbol sequence  $L$  is 14, with half of them 1 and the rest of them -1; however, in practice, the length of each training symbol sequence can be set depending on the need.

Secondly, when there are more than one transmitted signal streams, the final output value of MSE or SINR is the averaged one over all transmitted signal streams. In the process of optimization or training, precoder vectors and receive filter vectors for all transmitted signal streams are optimized or updated simultaneously, which is named as group adaptation as discussed in [27].

## 4.1 Experimental Platform

This research project is purely based on computer software simulation and Matrix Laboratory (MATLAB) is used as the simulation platform. Developed by MathWorks, MATLAB is a high-level proprietary multi-paradigm programming language and a numeric computing platform. It is widely applied in various fields of research, including statistics, machine learning, control systems, signal processing, image processing, wireless communications, finance and economics, to perform matrix manipulations, function plotting, user interfacing and some other purposes.

## 4.2 Uni-Directional Optimization and Training

This section presents the simulation results for uni-directional optimization and training.



Same as the system model used in Section 3.1, it is assumed that there are one transmitter with  $N_t$  transmit antennas, one receiver with  $N_r$  receive antennas, transmit power for each transmit antenna  $P$  and  $N_s$  transmitted signal streams iterated between the transmitter side and the receiver side.

In the first experiment, what is simulated is MMSE vs background SNR and SINR vs background SNR, with the full-rank receiver under Wiener estimation. It is uni-directional optimization with the CSI known *a priori*, thus both the MMSE and SINR are optimal. Both of the number of transmit antennas  $N_t$  and the number of receive antennas  $N_r$  are 8, the number of transmitted signal stream  $N_s$  is 1, with the background SNR adjusted from 0 dB to 30 dB along the horizontal axis. Figure 4.2 is the plot for MMSE vs background SNR and Figure 4.3 is the plot for SINR vs background SNR.

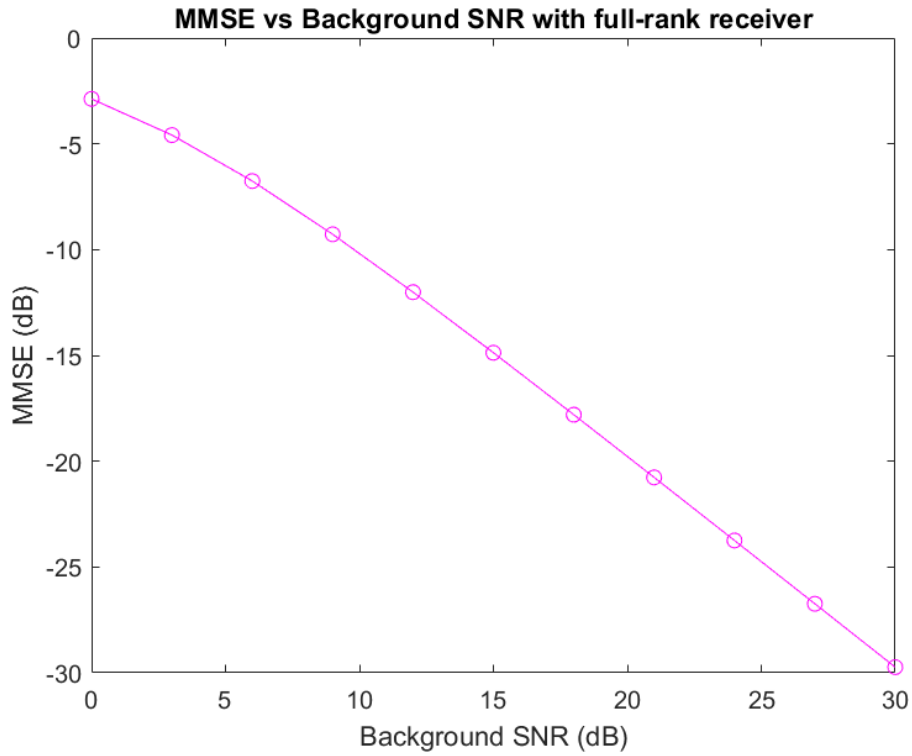


Figure 4.2: MMSE vs background SNR with the full-rank receiver, with the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8 and the number of transmitted signal stream  $N_s$  1

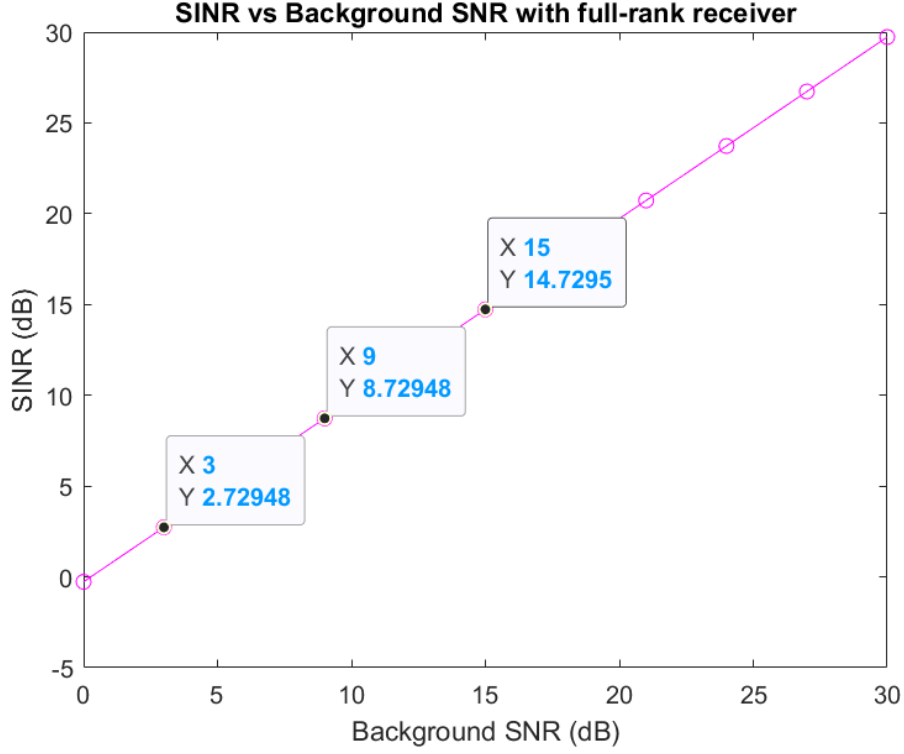


Figure 4.3: SINR vs background SNR with the full-rank receiver, with the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8 and the number of transmitted signal stream  $N_s$  1

In Figure 4.2, it can be observed that the MMSE monotonically decreases as the background SNR increases from 0 dB to 30 dB. Here the background SNR is expressed by  $\frac{P}{0.01}$ . Considering the statistical distribution of noise is kept unchanged with mean zero and variance 0.01, for the increase of the background SNR, the only change is the adjustment of the transmit power of each transmit antenna  $P$ .

In Figure 4.3, it can be seen that the SINR monotonically increases as the background SNR increases. By the three marked data points, the slope of the magenta SINR trend line can be calculated to be approximately 1, because in this system configuration, the number of transmitted signal stream is 1, without any interference streams, the SINR should theoretically be equal to the corresponding background SNR. However, in practice, due to the randomness of the normalized MIMO channel matrix and the normalized precoder matrix generated in this experiment, the SINR is always a little smaller than the corresponding background SNR.

From Figures 4.2 and 4.3, it can be concluded that, when the statistical distribution of the additive noise is kept unchanged, the larger the transmit power of each transmit antenna, the better the performance of the communications system.

If the setting of all parameters in this system configuration is kept unchanged, except adding the number of transmitted signal streams  $N_s$  from one to eight, the following two simulation results, Figures 4.4 and 4.5, are obtained:

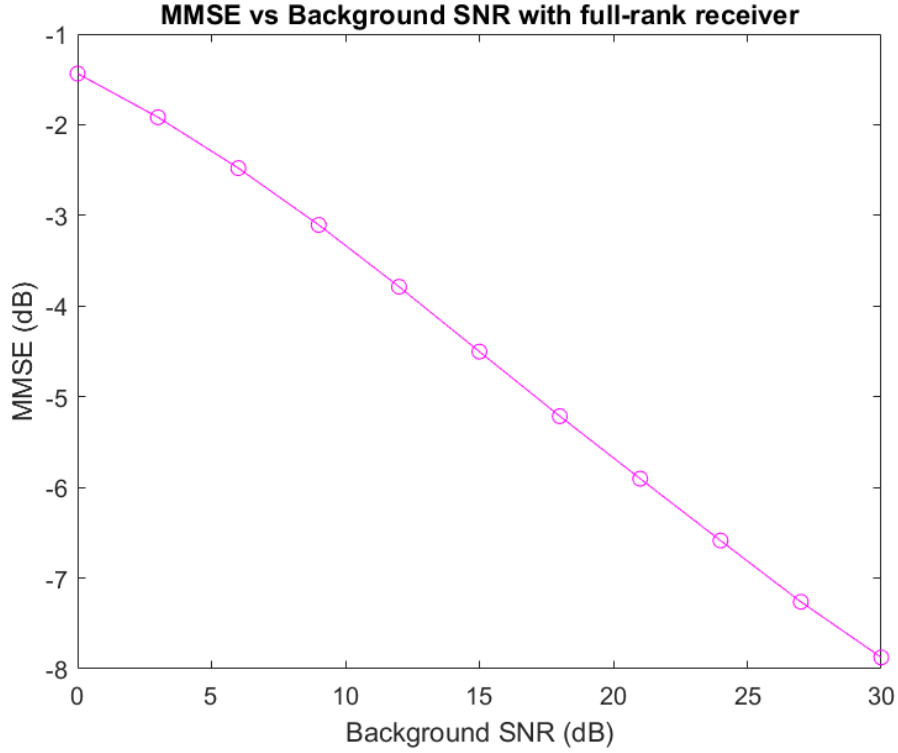


Figure 4.4: MMSE vs background SNR with the full-rank receiver, with the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8 and the number of transmitted signal streams  $N_s$  8

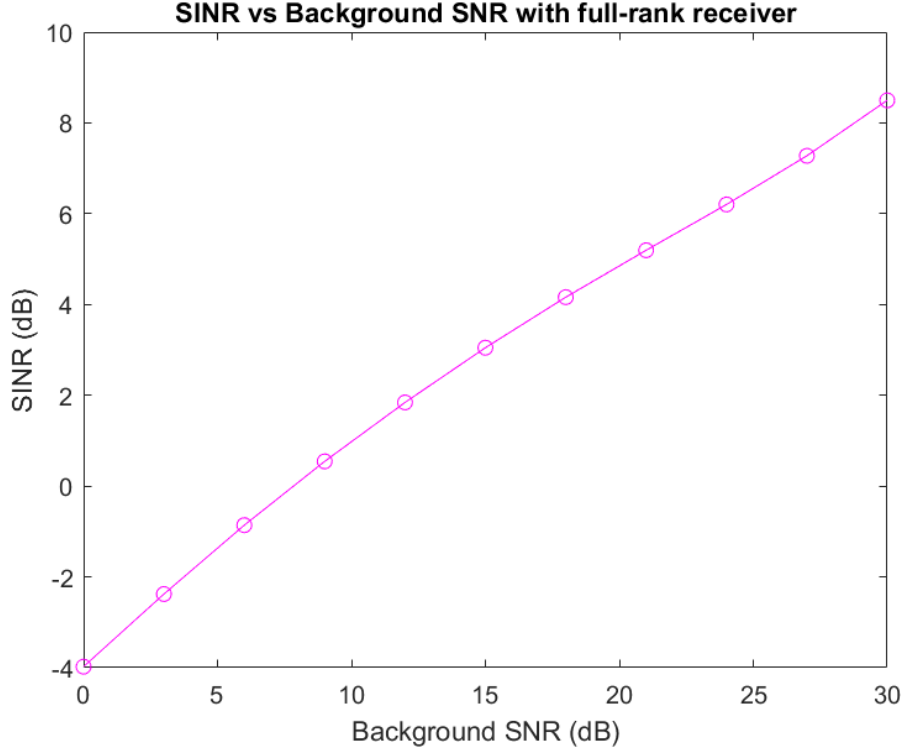


Figure 4.5: SINR vs background SNR with the full-rank receiver, with the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8 and the number of transmitted signal streams  $N_s$  8

From Figure 4.4, it can be easily observed that although the MMSE monotonically decreases with the increase of the background SNR, for each certain background SNR, the corresponding MMSE is much higher than the counterpart displayed in Figure 4.2, since here there are seven transmitted signal streams act as interference streams, which degrade the system performance.

By Figure 4.5, it can be seen that the slope of the magenta SINR trend line is much smaller than one and each SINR is much smaller than the corresponding background SNR. This is also due to the degradation caused by the seven interference streams. Actually, the MMSE has negative correlation with the SINR, increasing the former one is equivalent to decreasing the later one.

Now, the next experiment comes. Keep the number of transmitted signal streams  $N_s$  8, and plot the comparison between the full-rank receiver and two

reduced-rank receivers. Here the values of the two reduced-ranks are set to be two and four. Figures 4.6 and 4.7 show the comparison between the full-rank receiver with the two reduced-rank receivers, with respect to the change of the MMSE and SINR as the background SNR increases from 0 dB to 30 dB. In both figures, the magenta trend line refers to the full-rank receiver, while the cyan one and the red one refers to rank=2 and 4, respectively.

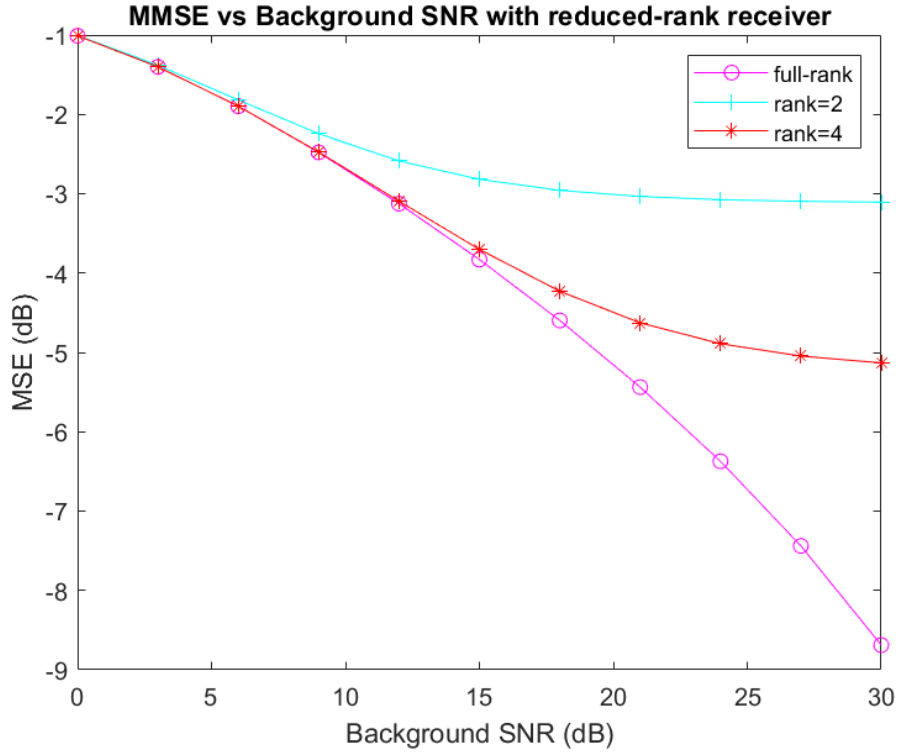


Figure 4.6: MMSE vs background SNR with reduced-rank receivers, with the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8, the number of transmitted signal streams  $N_s$  8 and ranks 2 and 4

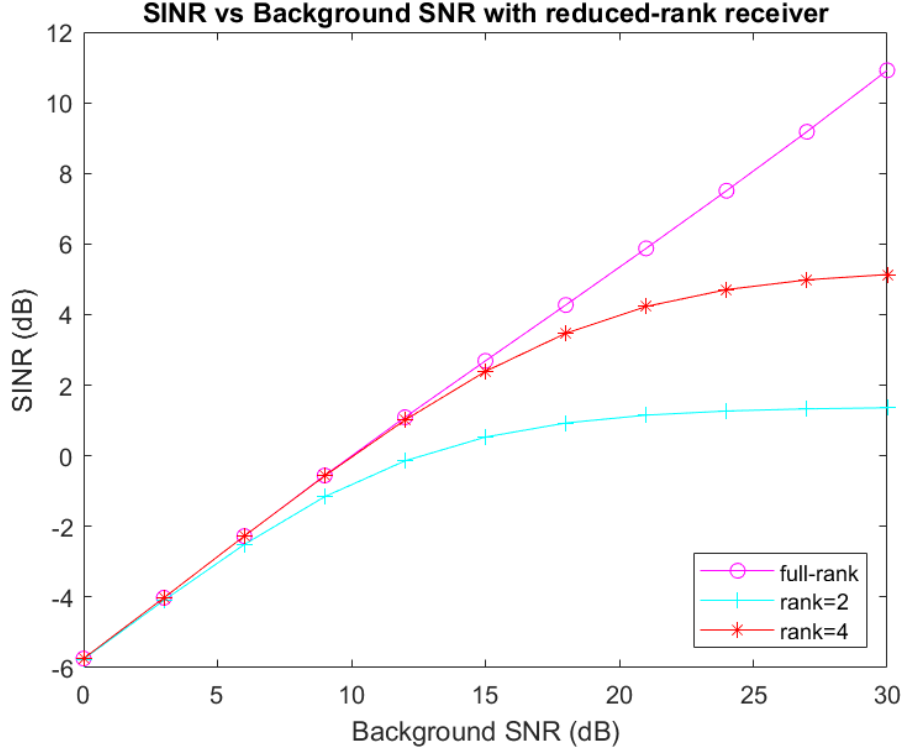


Figure 4.7: SINR vs background SNR with reduced-rank receivers, with the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8, the number of transmitted signal streams  $N_s$  8 and ranks 2 and 4

In Figure 4.6, it suggests that as the background SNR increases, the MMSE of the full-rank receiver becomes increasingly smaller than the counterparts of the two reduced-rank ones. It can be observed when the background SNR is larger than 5 dB, rank=2 trend line starts diverging with the other two ones; when the background SNR exceeds 15 dB, rank=4 trend line begins to diverge with the full-rank one.

In Figure 4.7, it can be seen that when the background SNR is large enough, the SINR of the full-rank receiver is significantly larger the counterparts of the two reduced-rank receivers.

From Figures 4.6 and 4.7, it can be concluded that when the background SNR is large enough, the MMSE and SINR of the full-rank receiver are smaller and larger than the counterparts of the reduced-rank ones, respectively. This might be because the reduced-rank filter not only reduces the received signal dimension, but also the DoFs available for suppressing inter-

ference and noise, accordingly, the ability of anti-interference-plus-noise is weakened.

Now, turn the attention from uni-directional optimization to uni-directional training. Same as the system model applied in Section 3.1, it is assumed that there are one transmitter with  $N_t$  transmit antennas, one receiver with  $N_r$  receive antennas, transmit power for each transmit antenna  $P$ ,  $N_s$  transmitted training symbol sequences iterated between the transmitter side and the receiver side and the length of each training symbol sequence  $L$ . Differs with uni-directional optimization, where only Wiener estimation exists, uni-directional training involves LS estimation.

It is worth mentioning that, in LS estimation, training symbol sequences are randomly generated, which involves a certain degree of randomness, hence the final training results in different trials are not exactly the same. To overcome this randomness, in this research project, for LS estimation, both the averaged training result averaged from several independent trials and one sample training result from a certain trial are presented.

In the next experiment, both the number of transmit antennas  $N_t$  and the number of receive antennas  $N_r$  are 8, the number of transmitted training symbol sequence  $N_s$  is 1, the length of each training symbol sequence is from 8 to 400 with increment 8 and the number of trials for plotting the averaged training result  $N_a$  is 20.

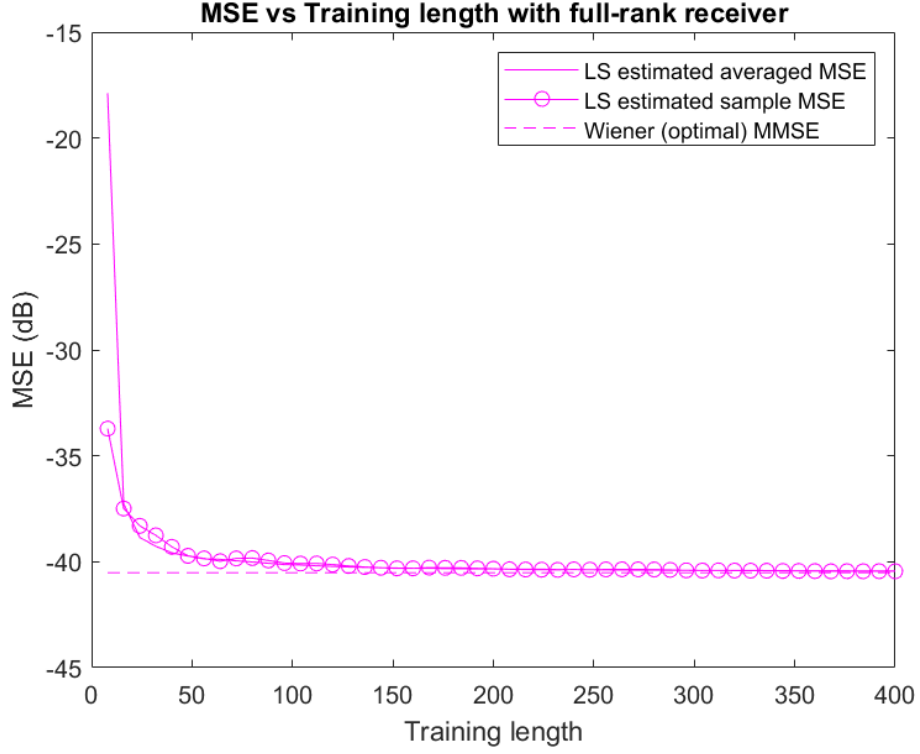


Figure 4.8: MSE vs training length with the full-rank receiver, with the transmit power of each transmit antenna  $P$  100, the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8, the number of transmitted training symbol sequence  $N_s$  1, the length of each training symbol sequence  $L$  from 8 to 400 with increment 8 and the number of trials for plotting the averaged training result  $N_a$  20



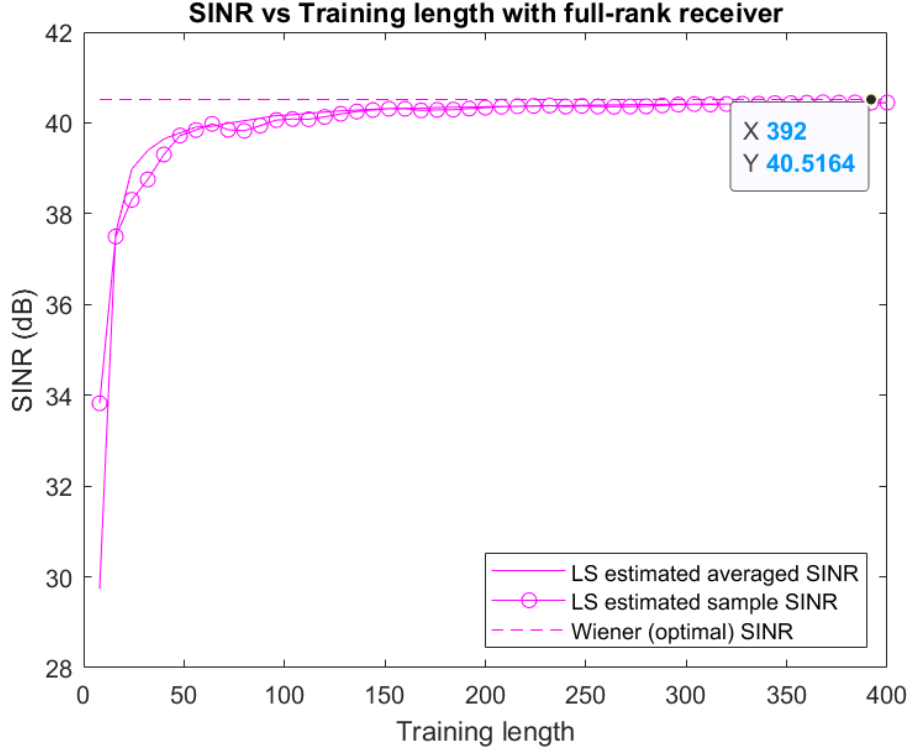


Figure 4.9: SINR vs training length with the full-rank receiver, with the transmit power of each transmit antenna  $P$  100, the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8, the number of transmitted training symbol sequence  $N_s$  1, the length of each training symbol sequence  $L$  from 8 to 400 with increment 8 and the number of trials for plotting the averaged training result  $N_a$  20

In Figure 4.8, it can be seen that as the training length increases, the LS estimated MSE gradually approaches the Wiener estimated MMSE. In this case, when the training length exceeds approximately 80, the LS estimated MSE asymptotically achieves its optimal value.

In Figure 4.9, similarly, it suggests that the LS estimated SINR gradually approaches the Wiener estimated optimal SINR as the training length increases. Here the value of background SNR is  $10\log_{10}\frac{100}{0.01} = 40\text{dB}$ , considering there is only one transmitted training symbol sequence, the asymptotically achieved optimal SINR should also be 40 dB. However, in this figure, the asymptotically achieved optimal SINR is around 40.5164 dB, a little larger than 40 dB. It might be due to the randomness coming from the normalized MIMO channel matrix and the normalized precoder matrix generated by the

random number generation mechanism.

Next, keep the setting of all parameters unchanged, but add the number of transmitted training symbol sequences  $N_s$  from 1 to 8. Figures 4.10 and 4.11 are obtained.

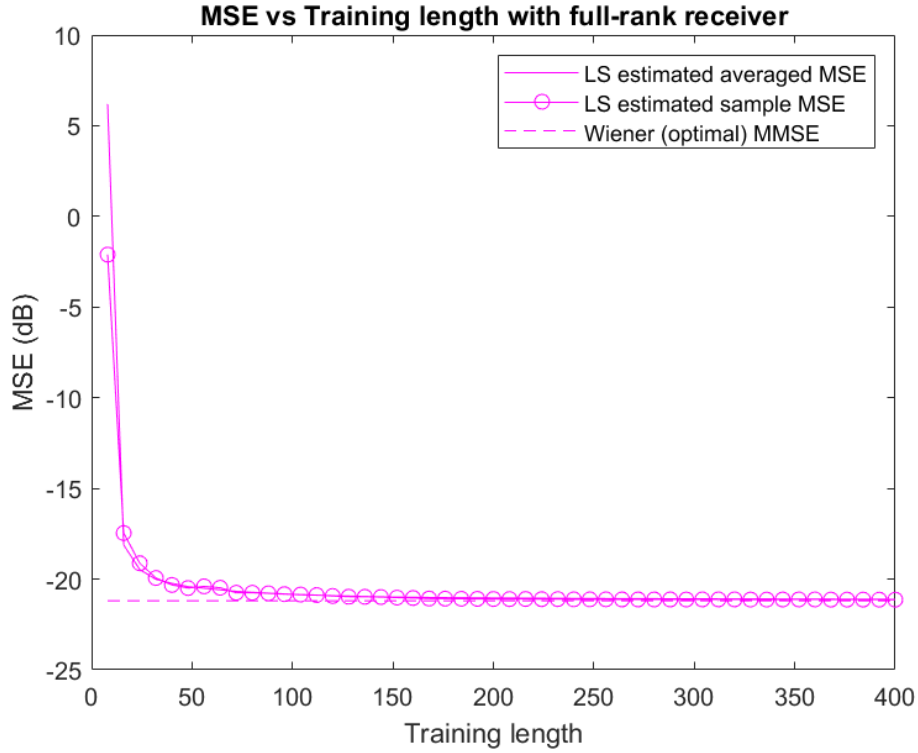


Figure 4.10: MSE vs training length with the full-rank receiver, with the transmit power of each transmit antenna  $P$  100, the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8, the number of transmitted training symbol sequences  $N_s$  8, the length of each training symbol sequence  $L$  from 8 to 400 with increment 8 and the number of trials for plotting the averaged training result  $N_a$  20

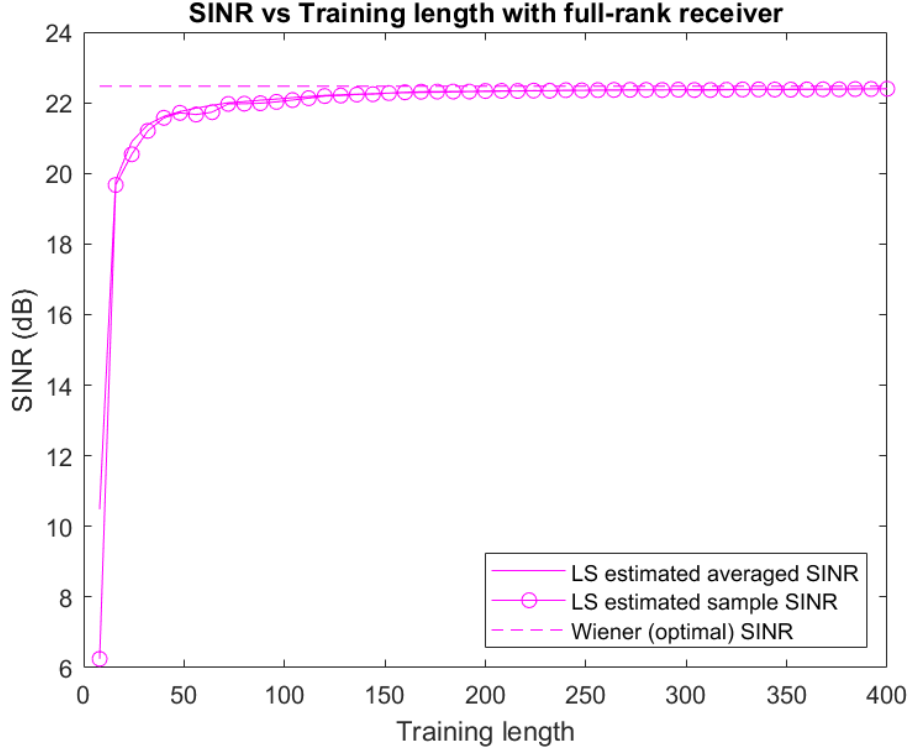


Figure 4.11: SINR vs training length with the full-rank receiver, with the transmit power of each transmit antenna  $P$  100, the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8, the number of transmitted training symbol sequences  $N_s$  8, the length of each training symbol sequence  $L$  from 8 to 400 with increment 8 and the number of trials for plotting the averaged training result  $N_a$  20

In Figure 4.10, on account of the interference from the other 7 transmitted training symbol sequences, the asymptotically achieved LS estimated MMSE is much smaller than the counterpart in Figure 4.8; similarly, in Figure 4.11, the asymptotically achieved LS estimated optimal SINR is significantly smaller than the counterpart in Figure 4.9 due to the same reason.

It is time to explore the advantage of reduced-rank filtering. Keep the setting of parameters of the last experiment unchanged, but add the simulation results of two reduced-rank receivers with ranks 2 and 4. Figures 4.12 and 4.13 are obtained.

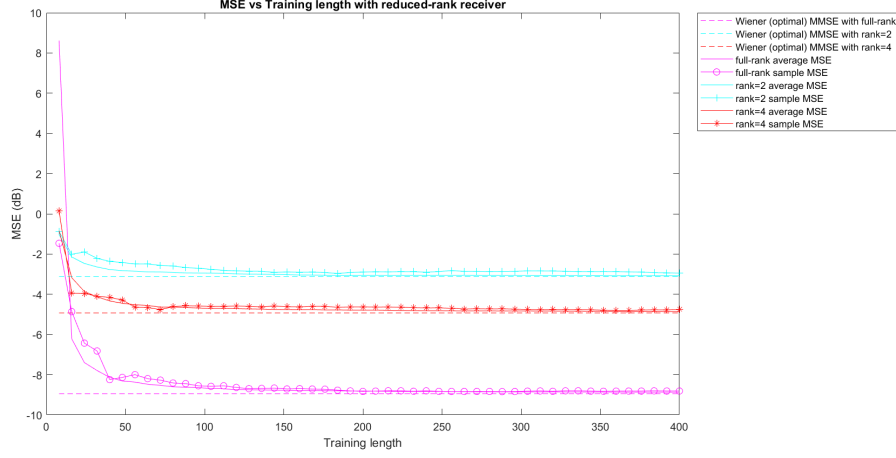


Figure 4.12: MSE vs training length with reduced-rank receivers, with the transmit power of each transmit antenna  $P$  100, the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8, the number of transmitted training symbol sequences  $N_s$  8, the length of each training symbol sequence  $L$  from 8 to 400 with increment 8, the number of trials for plotting the averaged training result  $N_a$  20 and ranks 2 and 4

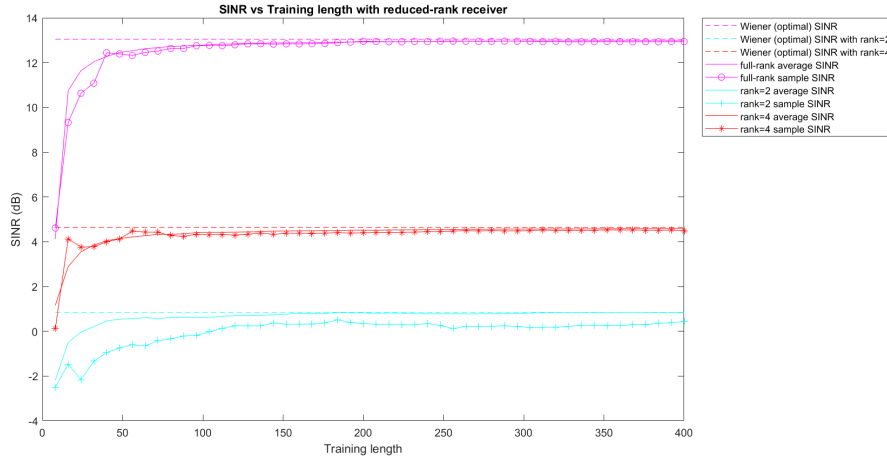


Figure 4.13: SINR vs training length with reduced-rank receiver, with the transmit power of each transmit antenna  $P$  100, the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8, the number of transmitted training symbol sequences  $N_s$  8, the length of each training symbol sequence  $L$  from 8 to 400 with increment 8, the number of trials for plotting the averaged training result  $N_a$  20 and ranks 2 and 4

In Figures 4.12 and 4.13, it can be seen that as the training length increases, under both full-rank filtering and reduced-rank filtering, the LS estimated performance trend line gradually approaches the corresponding Wiener optimal performance. When the training length exceed around 80, all of them achieve their optimal values. However, from these two figures, an obvious phenomenon can be observed that, since nearly the very beginning, the full-rank receiver always performs the best and rank=2 nearly always performs the worst. This might be due to the scale of this MIMO communications system is relatively small, it is hard for reduced-rank filtering to embody its technical advantage in this case. It is not the typical scenario where reduced-rank filtering is appropriate to use.

Nevertheless, if enlarging the scale of the transmit and receive antenna array, the simulation result will be different. For example, in the next experiment, there are 64 transmit antennas at the transmitter and 64 receive antennas at the receiver and keep the setting of other parameters unchanged. Figures 4.14 and 4.15 are obtained.

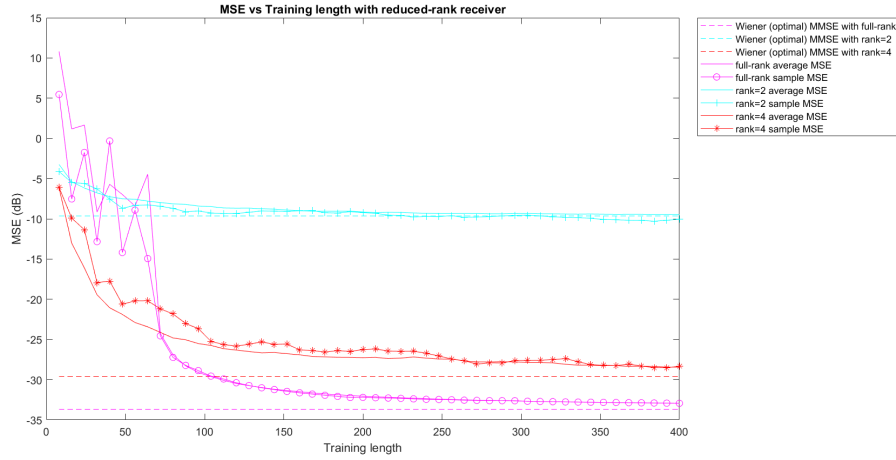


Figure 4.14: MSE vs training length with reduced-rank receivers, with the transmit power of each transmit antenna  $P$  100, the number of transmit antennas  $N_t$  64, the number of receive antennas  $N_r$  64, the number of transmitted training symbol sequences  $N_s$  8, the length of each training symbol sequence  $L$  from 8 to 400 with increment 8, the number of trials for plotting the averaged training result  $N_a$  20 and ranks 2 and 4

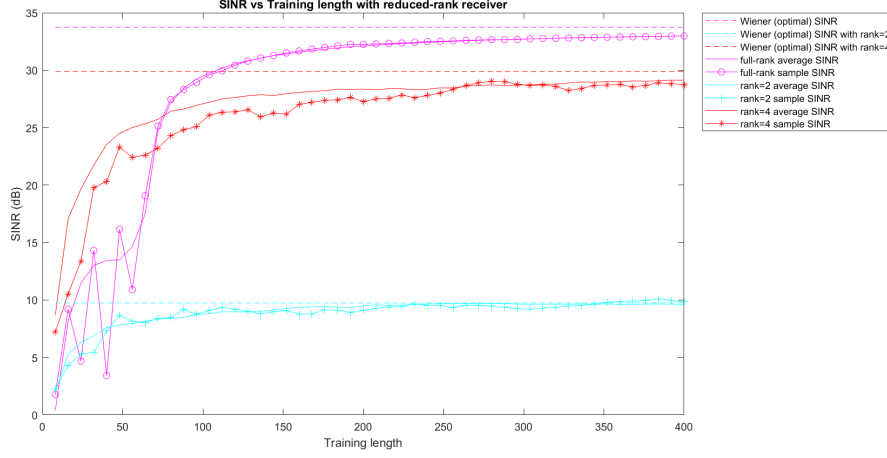


Figure 4.15: SINR vs training length with reduced-rank receivers, with the transmit power of each transmit antenna  $P$  100, the number of transmit antennas  $N_t$  64, the number of receive antennas  $N_r$  64, the number of transmitted training symbol sequences  $N_s$  8, the length of each training symbol sequence  $L$  from 8 to 400 with increment 8, the number of trials for plotting the averaged training result  $N_a$  20 and ranks 2 and 4

From Figures 4.14 and 4.15, from the very beginning to approximately the training length  $L$  equaling to 75, rank=4 outperforms the full-rank; after this point, the later one outperforms the former one. Rank=2 always performs the worst.

Next, increase the number of transmitted training symbol sequences  $N_s$  from 8 to 64, Figures 4.16 and 4.17 are obtained.

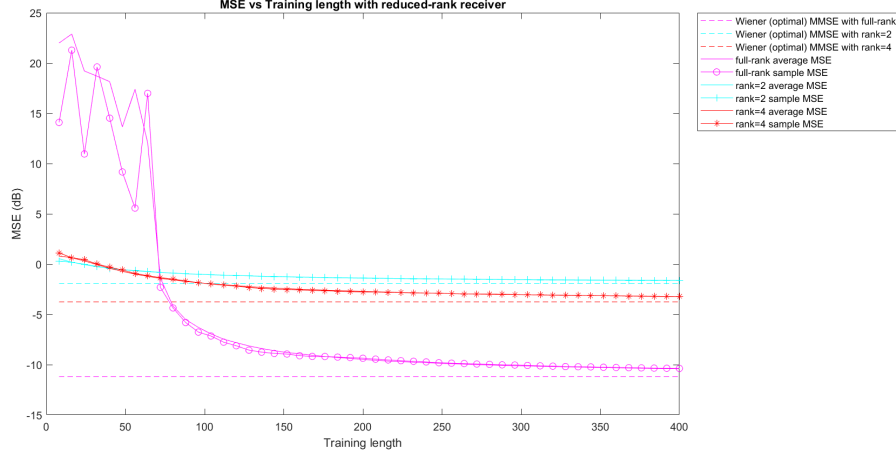


Figure 4.16: MSE vs training length with reduced-rank receivers, with the transmit power of each transmit antenna  $P$  100, the number of transmit antennas  $N_t$  64, the number of receive antennas  $N_r$  64, the number of transmitted training symbol sequences  $N_s$  64, the length of each transmitted training symbol sequence  $L$  from 8 to 400 with increment 8, the number of trials for plotting the averaged training result  $N_a$  20 and ranks 2 and 4

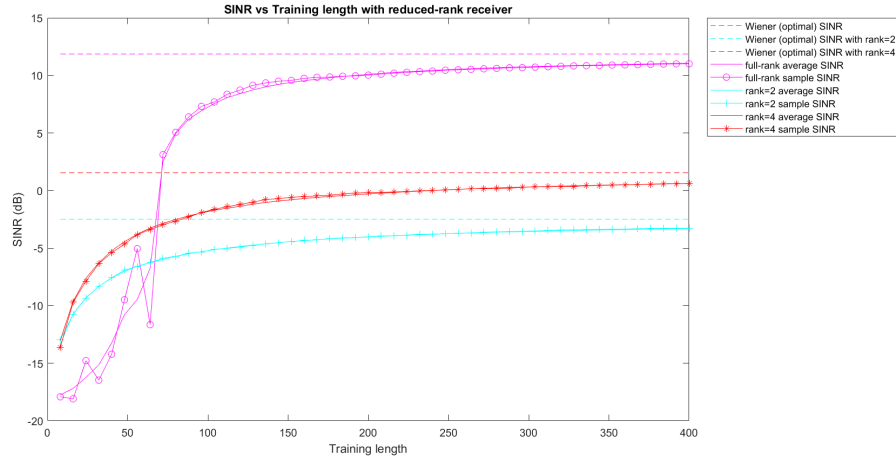


Figure 4.17: SINR vs training length with reduced-rank receivers, with the transmit power of each transmit antenna  $P$  100, the number of transmit antennas  $N_t$  64, the number of receive antennas  $N_r$  64, the number of transmitted training symbol sequences  $N_s$  64, the length of each training symbol sequence  $L$  from 8 to 400 with increment 8, the number of trials for plotting the averaged training result  $N_a$  20 and ranks 2 and 4

From Figures 4.16 and 4.17, it can be observed that in this case, when the training length  $L$  is less than around 75, the two reduced-rank receivers outperform the full-rank one; after that, the later one outperforms the former two ones.

Conclusively, these simulation results suggest that when the scale of the MIMO communications system is large enough, within a certain training length, reduced-rank filtering outperforms the full-rank one. However, because reduced-rank filtering reduces the DoFs available for suppressing interference and noise, accordingly weakens the ability of anti-interference-plus-noise, when the training length is long enough, its optimal performance is worse than the counterpart achieved by full-rank filtering.

### 4.3 Bi-Directional Optimization and Training

In this section, the simulation results of bi-directional optimization and training are presented.

Keep using the system model as presented in Section 3.1.

Firstly, make four experiments, MSE vs training length, SINR vs training length, MSE vs number of bi-directional optimization or training iterations and SINR vs number of bi-directional optimization or training iterations, with a full-rank transmitter-receiver pair. It is set that the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8 and the number of transmitted training symbol sequences  $N_s$  8.

In these four experiments, control variant method is used. In the experiments for MSE vs training length and SINR vs training length, the length of each transmitted training symbol sequence  $L$  is from 8 to 400 with increment 8, the number of bi-directional optimization or training iterations  $N_b$  is fixed to be 20, as shown in Figures 4.18 and 4.19; in the other two experiments, similarly, the length of each transmitted training symbol sequence  $L$  is fixed to be 40, the number of bi-directional optimization or training iterations  $N_b$  is from 1 to 20 with increment 1, as shown in Figures 4.20 and 4.21.



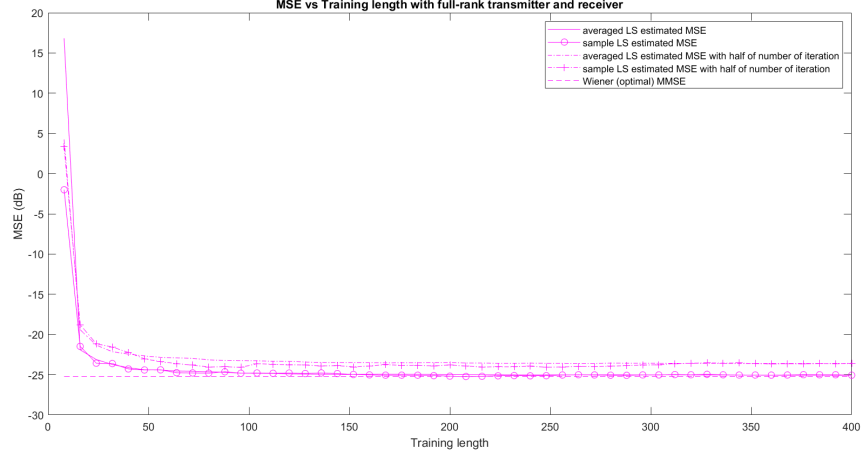


Figure 4.18: MSE vs training length with a full-rank transmitter-receiver pair, with the transmit power of each transmit or receive antenna  $P$  100, the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8, the number of transmitted training symbol sequences  $N_s$  8, the length of each transmitted training symbol sequence  $L$  from 8 to 400 with increment 8, the fixed number of bi-directional optimization or training iterations  $N_b$  20 and the number of trials for plotting the averaged training result  $N_a$  20

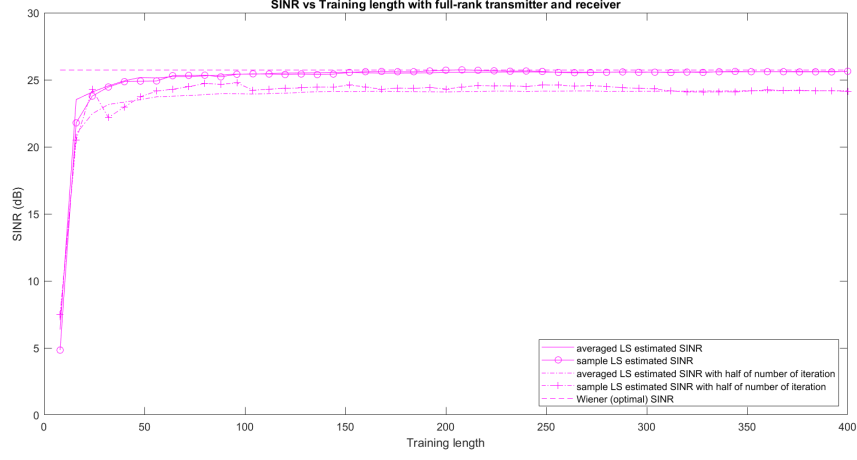


Figure 4.19: SINR vs training length with a full-rank transmitter-receiver pair, with the transmit power of each transmit or receive antenna  $P$  100, the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8, the number of transmitted training symbol sequences  $N_s$  8, the length of each transmitted training symbol sequence  $L$  from 8 to 400 with increment 8, the fixed number of bi-directional optimization or training iterations  $N_b$  20 and the number of trials for plotting the averaged training result  $N_a$  20

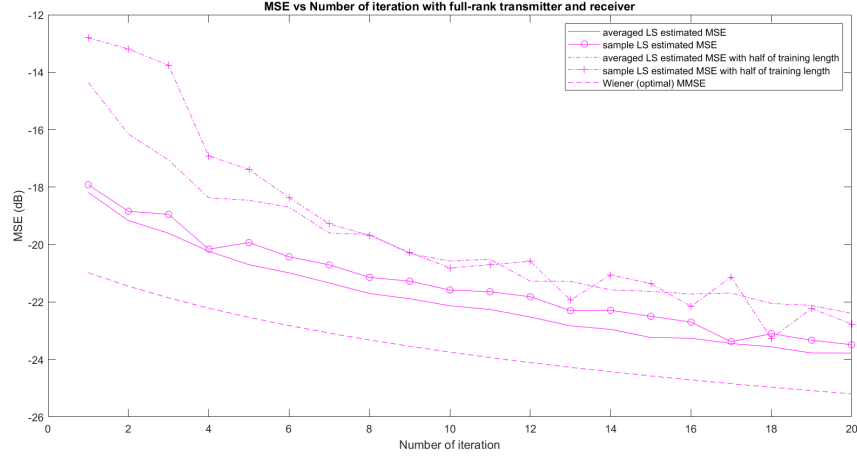


Figure 4.20: MSE vs number of bi-directional optimization or training iterations with a full-rank transmitter-receiver pair, with the transmit power of each transmit or receive antenna  $P$  100, the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8, the number of transmitted training symbol sequences  $N_s$  8, the number of bi-directional optimization or training iterations  $N_b$  from 1 to 20 with increment 1, the fixed length of each transmitted training symbol sequence  $L$  40 and the number of trials for plotting the averaged training result  $N_a$  20

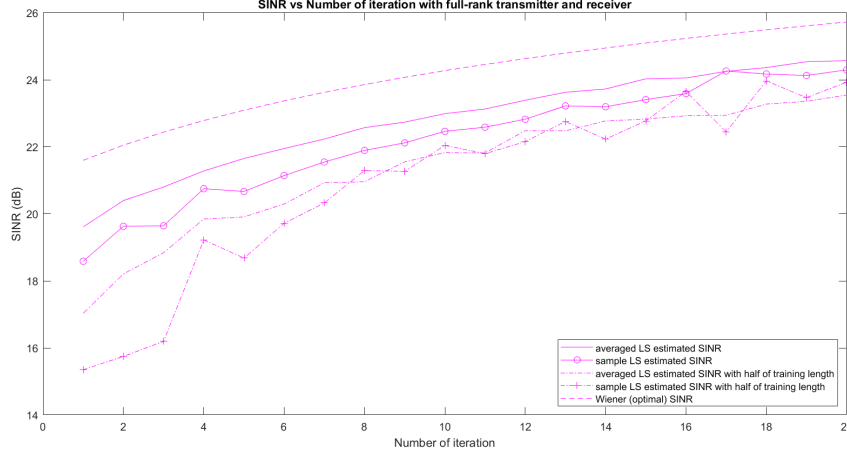


Figure 4.21: SINR vs number of bi-directional optimization or training iterations with a full-rank transmitter-receiver pair, with the transmit power of each transmit or receive antenna  $P$  100, the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8, the number of transmitted training symbol sequences  $N_s$  8, the number of bi-directional optimization or training iterations  $N_b$  from 1 to 20 with increment 1, the fixed length of each training symbol sequence  $L$  40 and the number of trials for plotting the averaged training result  $N_a$  20

In Figures 4.18 and 4.19, the solid trend line refers to the Wiener estimated performance, the dashed trend line refers to the LS estimated performance as explained by the figure caption and the dash-dot trend line acts as a comparison here, which refers to the performance with half of the number of bi-directional training iterations, viz 10 in this case. It can be observed that, when the number of bi-directional optimization or training iterations is fixed, as the training length increases, the LS estimated performance gradually approaches the corresponding Wiener estimated performance. In this case, when the training length exceeds around 80, the LS estimated performance approaches the Wiener estimated performance asymptotically. The LS estimated performance with half of the number of bi-directional training iterations is a little worse than the other LS estimated one.

From Figures 4.20 and 4.21, similarly, the dash-dot trend line is the LS estimated performance with half of the length of each transmitted training symbol sequence, acts as the comparison here. It can be observed that, when the length of each transmitted training symbol sequence is fixed, the system performance is significantly improved as the number of bi-directional

optimization or training iterations increases. Wiener estimated performance always outperforms the LS estimated counterpart in this case. The LS estimated performance with half of the length of each transmitted training symbol sequence is worse than the other LS estimated one.

In the next experiment, keep the setting of all parameters unchanged and add two reduced-rank transmitter-receiver pairs with ranks 2 and 4, respectively. Figures 4.22, 4.23, 4.24 and 4.25 present the simulation results.

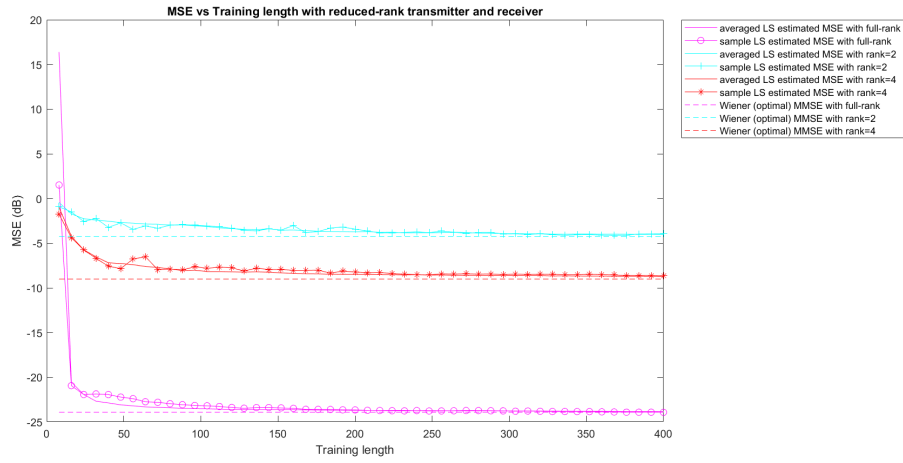


Figure 4.22: MSE vs training length with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna  $P$  100, the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8, the number of transmitted training symbol sequences  $N_s$  8, the length of each transmitted training symbol sequence  $L$  from 8 to 400 with increment 8, the fixed number of bi-directional optimization or training iterations  $N_b$  20, the number of trials for plotting the averaged training result  $N_a$  20 and ranks 2 and 4

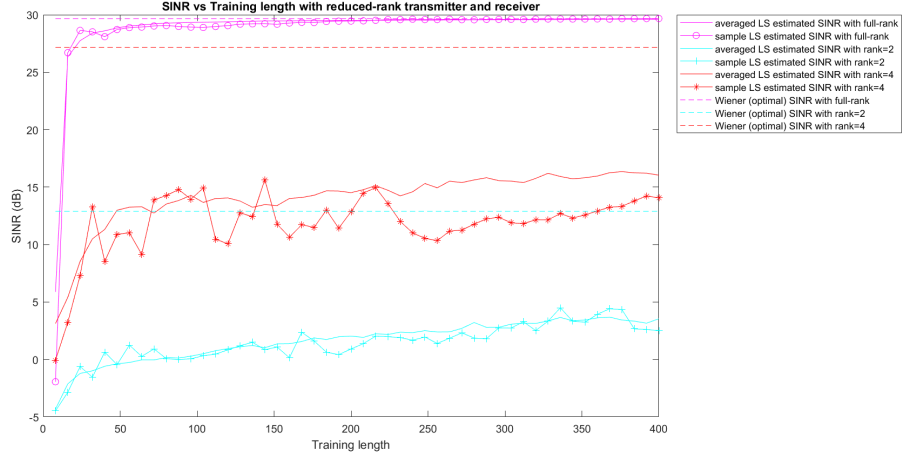


Figure 4.23: SINR vs training length with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna  $P$  100, the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8, the number of transmitted training symbol sequences  $N_s$  8, the length of each transmitted training symbol sequence  $L$  from 8 to 400 with increment 8, the fixed number of bi-directional optimization or training iterations  $N_b$  20, the number of trials for plotting the averaged training result  $N_a$  20 and ranks 2 and 4

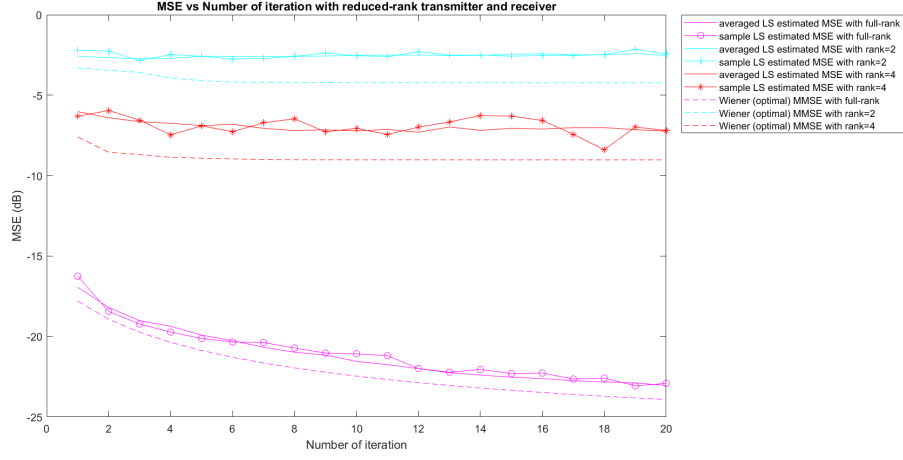


Figure 4.24: MSE vs number of bi-directional optimization or training iterations with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna  $P$  100, the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8, the number of transmitted training symbol sequences  $N_s$  8, the number of bi-directional optimization or training iterations  $N_b$  from 1 to 20 with increment 1, the fixed length of each transmitted training symbol sequence  $L$  40, the number of trials for plotting the averaged training result  $N_a$  20 and ranks 2 and 4

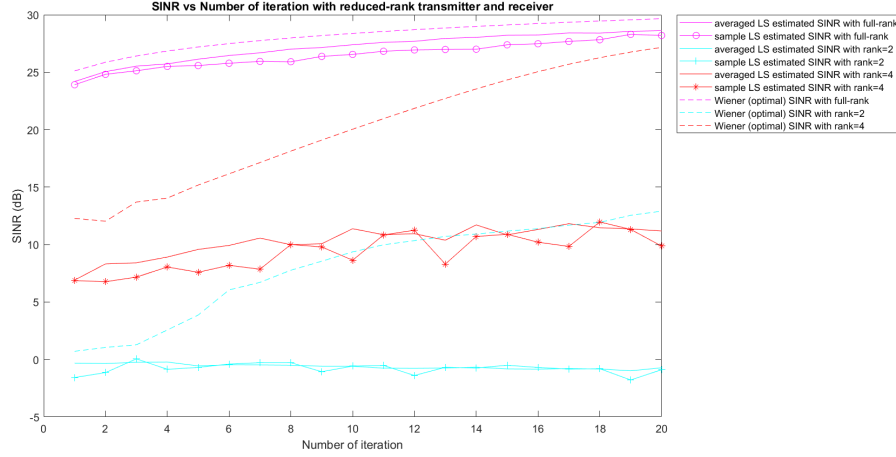


Figure 4.25: SINR vs number of bi-directional optimization or training iteration with two transmitter-receiver pairs, with the transmit power of each transmit or receive antenna  $P$  100, the number of transmit antennas  $N_t$  8, the number of receive antennas  $N_r$  8, the number of transmitted training symbol sequences  $N_s$  8, the number of bi-directional optimization or training iterations  $N_b$  from 1 to 20 with increment 1, the fixed length of each transmitted training symbol sequence  $L$  40, the number of trials for plotting the averaged training result  $N_a$  20 and ranks 2 and 4

From Figures 4.22 and 4.23, it can be observed that, when the number of bi-directional optimization or training iterations is fixed, as the training length increases, the LS estimated full-rank filtering performance gradually approaches the corresponding Wiener estimated counterpart. In these figures, it seems that the two LS estimated reduced-rank filtering performances do not approach their corresponding Wiener estimated counterparts, this might be because of the limitation of the range of training length in this experiment.

From the comparison between the full-rank filtering performance and the two reduced-rank filtering ones, since the scale of this MIMO communications system is relatively small, nearly from the very beginning, the full-rank one always performs the best, meanwhile the rank=4 one outperforms the rank=2 one.

From Figures 4.24 and 4.25, when the training length is fixed, the LS estimated performance improves as the number of bi-directional training iterations increases. Throughout the whole range of the horizontal axis, full-



rank filtering always performs the best and the rank=4 one outperforms the rank=2 one.

Then, take the same experiment in a MIMO communications system with a larger scale. Change the number of both transmit antennas  $N_t$  and receive antennas  $N_r$  and the number of transmitted training symbol sequences  $N_s$  from 8 to 64. Figures 4.26, 4.27, 4.28 and 4.29 present the experimental results.

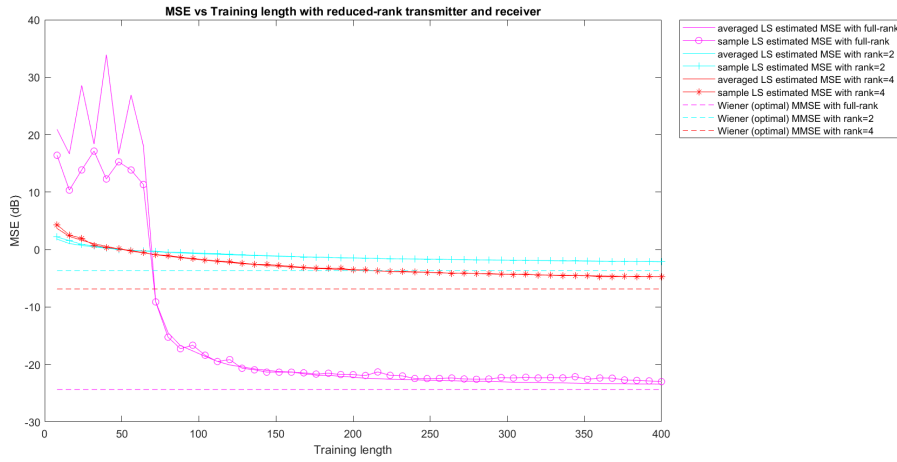


Figure 4.26: MSE vs training length with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna  $P$  100, the number of transmit antennas  $N_t$  64, the number of receive antennas  $N_r$  64, the number of transmitted training symbol sequences  $N_s$  64, the length of each transmitted training symbol sequence  $L$  from 8 to 400 with increment 8, the fixed number of bi-directional optimization or training iterations  $N_b$  20, the number of trials for plotting the averaged training result  $N_a$  20 and ranks 2 and 4

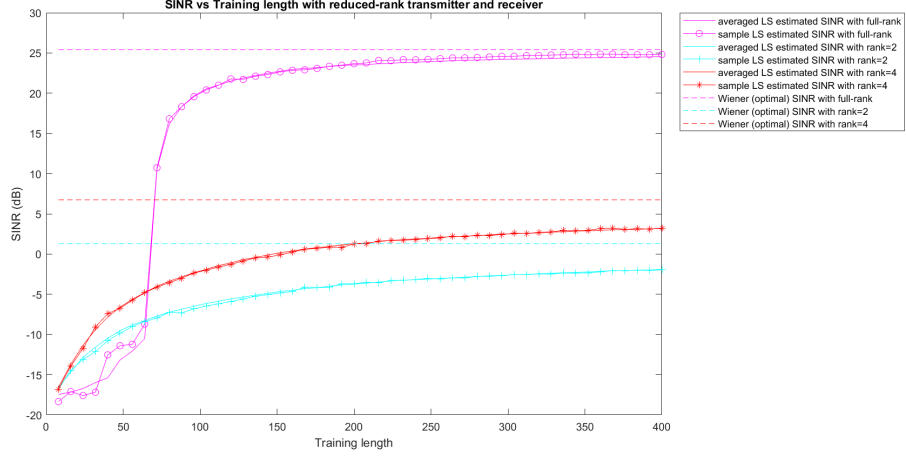


Figure 4.27: SINR vs training length with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna  $P$  100, the number of transmit antennas  $N_t$  64, the number of receive antennas  $N_r$  64, the number of transmitted training symbol sequences  $N_s$  64, the length of each transmitted training symbol sequence  $L$  from 8 to 400 with increment 8, the fixed number of bi-directional optimization or training iterations  $N_b$  20, the number of trials for plotting the averaged training result  $N_a$  20 and ranks 2 and 4

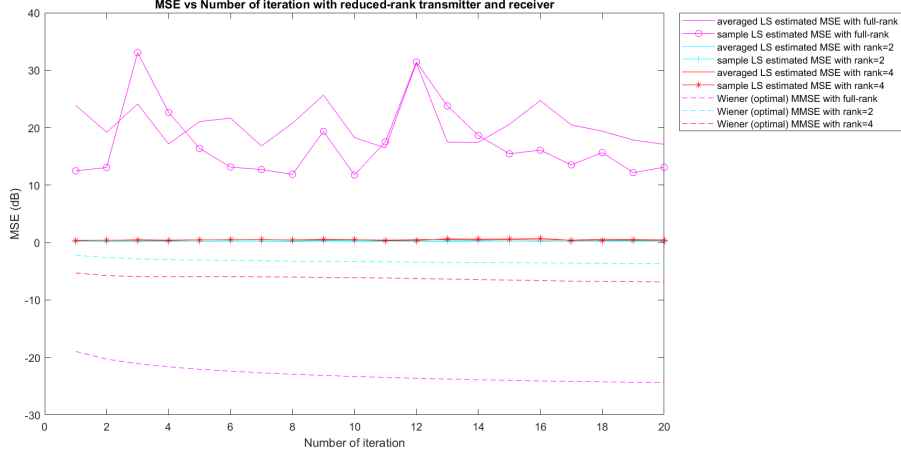


Figure 4.28: MSE vs number of bi-directional optimization or training iterations with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna  $P$  100, the number of transmit antennas  $N_t$  64, the number of receive antennas  $N_r$  64, the number of transmitted training symbol sequences  $N_s$  64, the number of bi-directional optimization or training iterations  $N_b$  from 1 to 20 with increment 1, the fixed length of each transmitted training symbol sequence  $L$  40, the number of trials for plotting the averaged training result  $N_a$  20 and ranks 2 and 4

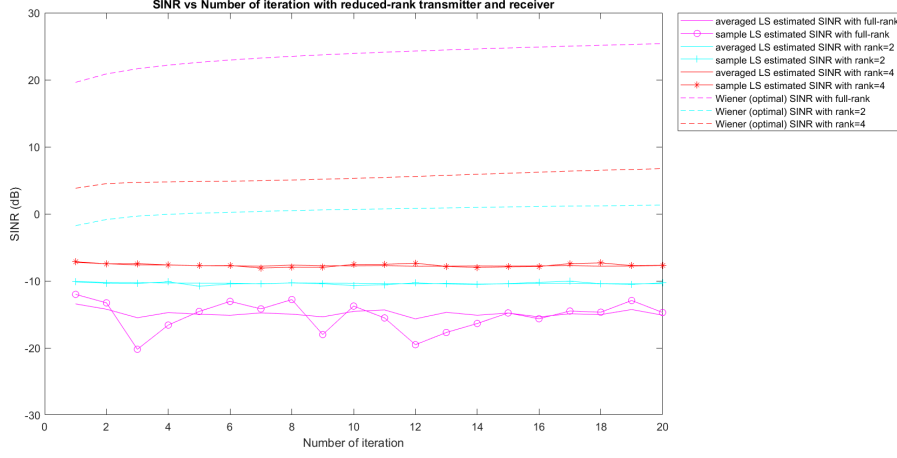


Figure 4.29: SINR vs number of bi-directional optimization or training iterations with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna  $P$  100, the number of transmit antennas  $N_t$  64, the number of receive antennas  $N_r$  64, the number of transmitted training symbol sequences  $N_s$  64, the number of bi-directional optimization or training iterations  $N_b$  from 1 to 20 with increment 1, the fixed length of each transmitted training symbol sequence  $L$  40, the number of trials for plotting the averaged training result  $N_a$  20 and ranks 2 and 4

From Figures 4.26 and 4.27, it can be found that when the number of bi-directional optimization or training iterations is fixed, from the very beginning to around training length equaling to 70, two reduced-rank transmitter-receiver pairs significantly outperform the full-rank one; after that, this relationship is reversed, especially when the training length is long enough, full-rank filtering significantly outperforms reduced-rank filtering. Each LS estimated performance approaches the corresponding Wiener estimated one asymptotically as the increase of training length. The larger the rank, the better the asymptotic optimal performance.

From Figures 4.28 and 4.29, it suggests that when the training length is fixed, the LS estimated performance improves as the increase of number of bi-directional optimization or training iterations, though in this case LS estimated performances do not improve obviously. It can be easily noticed that, across the whole range of the horizontal axis, two reduced-rank performances are significantly better than the performance of the full-rank one. Similarly, the larger the rank, the better the Wiener estimated performance.

Finally, add an appropriate modification on this system model to make an interference network, the structure of an interference network is schematically shown in Figure 4.30.

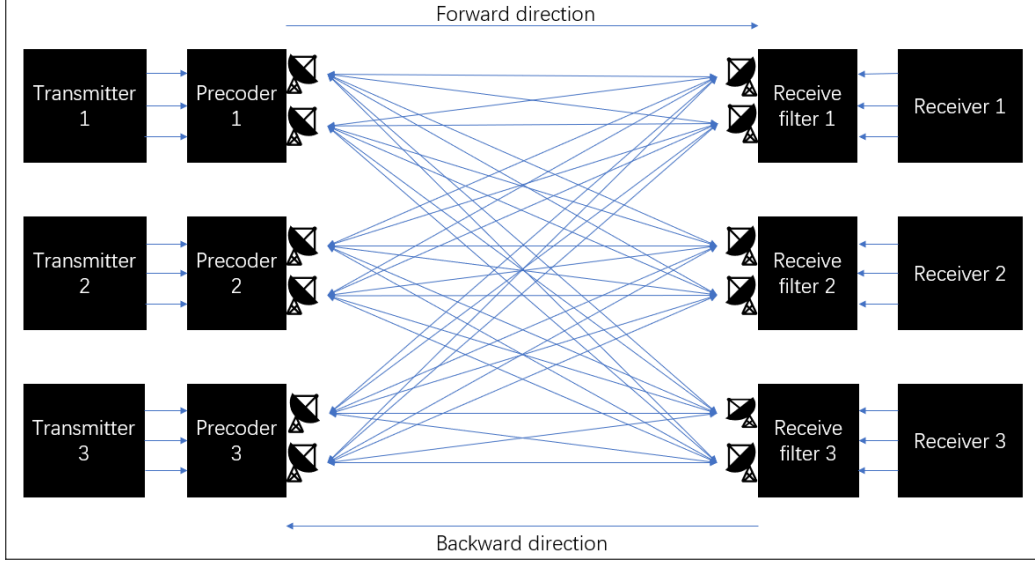


Figure 4.30: Model of interference networks

In Figure 4.30, it can be seen that an interference network consists of more than one transmitter-receiver pairs. In such a network, for each transmitted signal stream, interference comes from both its own transmitter and other transmitters within this network. Figure 4.30 only provides a sketch diagram of an interference network and draws three transmitter-receive pairs. In practice, it can be extended to more pairs.

In the setting of parameters for the interference network model used in this experiment, it is set that the number of transmitter-receiver pairs  $N_p$  is 3, the number of transmit antennas on each transmitter  $N_t$  is 64, the number of receive antennas on each receiver  $N_r$  is 64, the number of transmitted training symbol sequences  $N_s$  sent from each precoder is 64. Figure 4.31 plots the relationship between the total received sum rate and the increase of the training length, where the number of bi-directional optimization and training iterations  $N_b$  is fixed to be 10; Figure 4.32 plots the relationship between the total received sum rate and the increase of the number of bi-directional optimization or training iterations, where the fixed training length  $L$  is 20. The number of trials for plotting the averaged training result  $N_a$  is 10. In this experiment, the received sum rate is calculated by Equation

2.2, which is the Shannon-Hartley Theorem. To simplify the calculation, the value of bandwidth  $B$  is assumed to be 1.

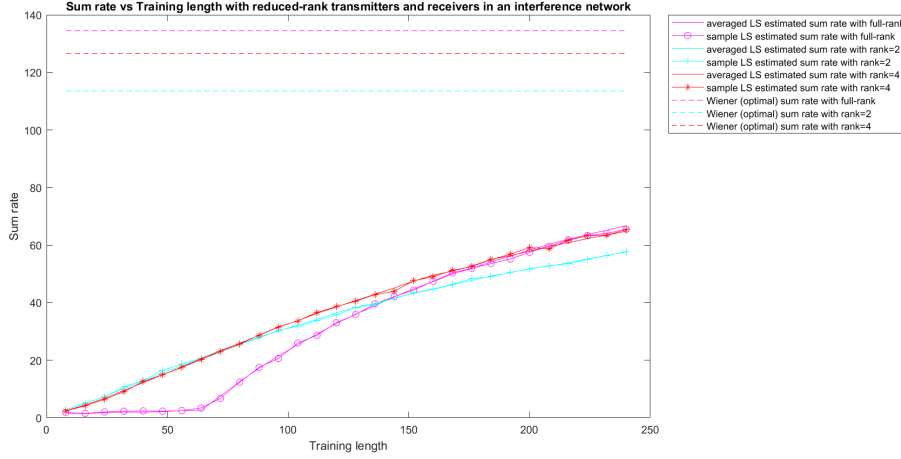


Figure 4.31: Sum rate vs training length with reduced-rank transmitters and receivers in an interference network, with the transmit power of each transmit or receive antenna  $P$  100, the number of transmitter-receiver pairs  $N_p$  3, the number of transmit antennas on each transmitter  $N_t$  64, the number of receive antennas on each receiver  $N_r$  64, the number of transmitted training symbol sequences for each transmitter-receiver pair  $N_s$  64, the length of each transmitted training symbol sequence  $L$  from 8 to 240 with increment 8, the fixed number of bi-directional optimization or training iterations  $N_b$  10, the number of trials for plotting the averaged training result  $N_a$  10 and ranks 2 and 4

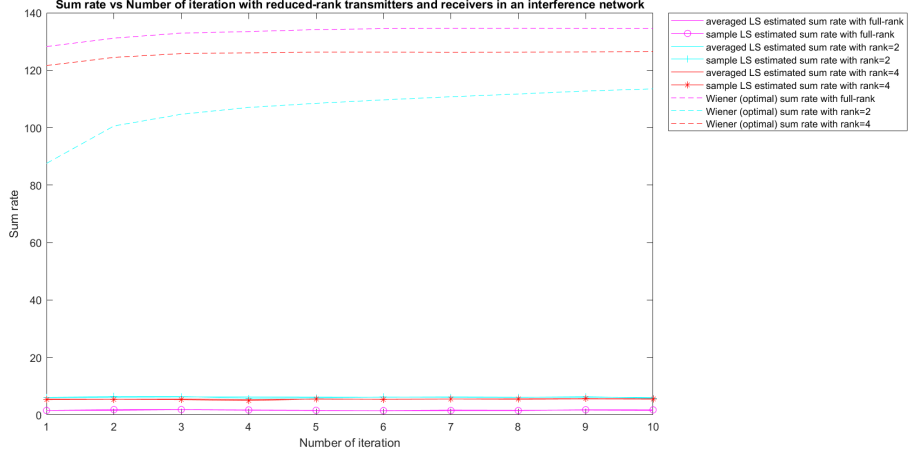


Figure 4.32: Sum rate vs number of bi-directional optimization or training iterations with reduced-rank transmitters and receivers in an interference network, with the transmit power of each transmit or receive antenna  $P$  100, the number of transmit antennas on each transmitter  $N_t$  64, the number of receive antennas on each receiver  $N_r$  64, the number of transmitted training symbol sequences for each transmitter-receiver pair  $N_s$  64, the number of bi-directional optimization or training iterations  $N_b$  from 1 to 20 with increment 1, the fixed length of each transmitted training symbol sequence  $L$  40, the number of trials for plotting the averaged training result  $N_a$  10 and ranks 2 and 4

From Figure 4.31, it can be observed that from the very beginning to training length equaling to 150, for LS estimation, the two reduced-rank networks significantly outperform the full-rank one; after that, the three networks nearly have the same performance. For Wiener estimation, it can be seen that the larger the rank, the better the optimal system performance. In this figure, the three LS estimated sum rate trend lines do not asymptotically approach their corresponding Wiener optimal values as the training length increases, this is because here the training length range is not long enough. When the training length is long enough, each LS estimated sum rate trend line must approach its corresponding Wiener optimal value asymptotically.

From Figure 4.32, it can be observed that the three Wiener estimated sum rates increase obviously as the number of bi-directional optimization iterations increases, the larger the rank, the better the Wiener estimated performance. However, it seems that the three LS estimated trend lines do not improve as the number of bi-directional training iterations increases, this

might be because here the scale of this communications system is significantly larger than before, this fixed training length is not helpful for significantly improving the system performance for each training iteration.



# Chapter 5

## Discussion

### 5.1 General Discussion

Until this stage, all background knowledge introduction, theoretical derivations and experimental simulation results have been presented. Chapter 4 presented a series of simulation results beginning with uni-directional optimization with full-rank filtering in a small-scaled MIMO communications system and ending with bi-directional training with reduced-rank filtering in a small-scaled interference network.

A series of experimental simulation results suggest that, when the scale of the MIMO communications system is relatively large and the training length resource is limited to a certain degree, reduced-rank filtering can outperform full-rank filtering with smaller signal subspace matrix ranks; however, since reduced-rank filtering reduces the DoFs for suppressing interference and noise, when the training length is long enough, the asymptotically optimal performance achieved by the reduced-rank filtering is worse than that achieved by the full-rank counterpart. Bi-directional training can improve the performance of a MIMO communications system by updating precoder and receive filter matrices using transmitted training symbol sequences back and forth between the transmitter side and the receiver side. In MIMO communications systems design, reduced-rank filtering can be combined with bi-directional training to achieve a better system performance.

Nevertheless, there might be a problem with the experimental simulation results about reduced-rank filtering. When running simulation programs involving reduced-rank filtering, MATLAB command window always prints the warning that matrix is close to singular or badly scaled, results may be

inaccurate, along with an extremely small matrix conditional number. The lower the matrix conditional number, the higher the degree of singularity of the matrix. This might be because in a MSWF-based reduced-rank projection matrix, columns are highly linearly correlated, which introduces the property of singularity in the matrix manipulation about reduced-rank filtering. Therefore, in simulation results, Wiener estimated and LS estimated reduced-rank performances may be somewhat inaccurate when the rank becomes relatively large, or the number of training samples are less than the size of the covariance matrix.

## 5.2 Future Works

Firstly, at the end of Section 4.3, bi-directional optimization and training with reduced-rank filtering in an interference network is simulated, however, this can also be extended to the scenario of cellular networks. In a cellular network, there are several divided land areas named cells, each cell is specifically served by one or more fixed base stations. Within each cell coverage area, the transmission and reception of information data, including voice, image and video, are coordinated by the fixed base stations.

Secondly, as discussed in Section 2.3, instead of purely software-based adaptive beamforming techniques, beamforming computation can also be performed on novel hardware platforms, such as FPGAs, to cater the increasingly complicated computational needs and accelerate signal processing. In future experiments, FPGAs-based adaptive beamforming processing can be performed, then compared with the purely software-based counterpart in terms of the processing quality and efficiency.

# Chapter 6

## Conclusion

In this research project report, four crucial algorithms are presented mathematically to lay the formulation of the experimental simulations; then, a series of experimental simulation results are presented to demonstrate aforementioned theories; in the end, simulation results are assessed, one problem is pointed out and two possible future improvement methods are discussed. It can be concluded that when the scale of a MIMO communications system is large and the training length resource is limited, reduced-rank filtering has the ability to achieve a better performance over its full-rank counterpart. By applying transmitted training symbol sequences, bi-directional training can update precoders and receive filters iteratively to optimize the system design. Reduced-rank filtering can be combined with bi-directional training to achieve a better system performance in MIMO communications systems design. This research project is by no means complete and there is still room for further studies in this topic.

# References

- [1] M.L. Honig. “*Overview of Multiuser Detection*”, pages 1–45. Wiley, 11 2008.
- [2] CTIA. “Number of mobile wireless cell sites in the United States from 2000 to 2019”. Statista.com. [https://www.statista.com/statistics/185854/monthly-number-of-cell-sites-in-the-united-states-since-june-1986/#:~:text=Mobile%20wireless%20cell%20sites%20in%20the%20United%20States%202000%2D2019&text=In%202019%2C%20there%20were%20395%2C562,antennas%20as%20per%20the%20source.\(accessed on Jan 31st. 2021\).](https://www.statista.com/statistics/185854/monthly-number-of-cell-sites-in-the-united-states-since-june-1986/#:~:text=Mobile%20wireless%20cell%20sites%20in%20the%20United%20States%202000%2D2019&text=In%202019%2C%20there%20were%20395%2C562,antennas%20as%20per%20the%20source.(accessed%20on%20Jan%2031st%202021).)
- [3] Statista.com. “Number of mobile internet users in the United States from 2015 to 2025”. Statista.com. [https://www.statista.com/statistics/275591/number-of-mobile-internet-user-in-usa/#:~:text=In%202020%2C%20274.7%20million%20people,population%20are%20mobile%20internet%20users.\(accessed on Jan 31st. 2021\).](https://www.statista.com/statistics/275591/number-of-mobile-internet-user-in-usa/#:~:text=In%202020%2C%20274.7%20million%20people,population%20are%20mobile%20internet%20users.(accessed%20on%20Jan%2031st%202021).)
- [4] Louis L. Scharf. “The SVD and reduced rank signal processing”. *Signal Process.*, 25(2):113–133, November 1991.
- [5] Don H. Johnson and Dan E. Dudgeon. “*Array Signal Processing: Concepts and Techniques*”. Simon & Schuster, Inc., USA, 1992.
- [6] P. A. Zulch, J. S. Goldstein, J. R. Guerci, and L. S. Reed. “Comparison of reduced-rank signal processing techniques”. In *Conference Record of Thirty-Second Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 421–425 vol.1, 1998.
- [7] L. Scharf and J. K. Thomas. “Wiener filters in canonical coordinates for transform coding, filtering, and quantizing”. *IEEE Transactions on Signal Processing*, 46(3):647–654, 1998.

- [8] D. A. Pados and S. N. Batalama. “Low-complexity blind detection of ds/cdma signals: auxiliary-vector receivers”. *IEEE Transactions on Communications*, 45(12):1586–1594, 1997.
- [9] M. L. Honig. “A comparison of subspace adaptive filtering techniques for ds-cdma interference suppression”. In *MILCOM 97 MILCOM 97 Proceedings*, volume 2, pages 836–840 vol.2, 1997.
- [10] H.V. Poor and G.W. Wornell. “*Wireless Communications: Signal Processing Perspectives*”. Prentice-Hall signal processing series. Prentice Hall PTR, 1998.
- [11] Xiaodong Wang and H. V. Poor. “Blind multiuser detection: a subspace approach”. *IEEE Transactions on Information Theory*, 44(2):677–690, 1998.
- [12] M. L. Honig and J. S. Goldstein. “Adaptive reduced-rank residual correlation algorithms for ds-cdma interference suppression”. In *Conference Record of Thirty-Second Asilomar Conference on Signals, Systems and Computers (Cat. No.98CH36284)*, volume 2, pages 1106–1110 vol.2, 1998.
- [13] D. A. Pados and S. N. Batalama. “Joint space-time auxiliary-vector filtering for DS/CDMA systems with antenna arrays”. *IEEE Transactions on Communications*, 47(9):1406–1415, 1999.
- [14] M. L. Honig and Weimin Xiao. “Performance of reduced-rank linear interference suppression”. *IEEE Transactions on Information Theory*, 47(5):1928–1946, 2001.
- [15] G. S. Rajappan and M. L. Honig. “Signature sequence adaptation for DS-CDMA with multipath”. *IEEE Journal on Selected Areas in Communications*, 20(2):384–395, 2002.
- [16] H. Zhou, M. L. Honig, J. Liu, and W. Xiao. “Bi-directional training for FDD systems”. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2019.
- [17] T. L. Marzetta. “How much training is required for Multiuser MIMO?”. In *2006 Fortieth Asilomar Conference on Signals, Systems and Computers*, pages 359–363, 2006.
- [18] X. Zhou, T. A. Lamehewa, P. Sadeghi, and S. Durrani. “Two-way training: optimal power allocation for pilot and data transmission”. *IEEE Transactions on Wireless Communications*, 9(2):564–569, 2010.

- [19] K. S. Gomadam, H. C. Papadopoulos, and C. . Sundberg. “Techniques for Multi-user MIMO with two-way training”. In *2008 IEEE International Conference on Communications*, pages 3360–3366, 2008.
- [20] R. Osawa, H. Murata, K. Yamamoto, and S. Yoshida. “Performance of two-way channel estimation technique for multi-user distributed antenna systems with spatial precoding”. In *2009 IEEE 70th Vehicular Technology Conference Fall*, pages 1–5, 2009.
- [21] C. Steger and A. Sabharwal. “Single-input two-way SIMO channel: diversity-multiplexing tradeoff with two-way training”. *IEEE Transactions on Wireless Communications*, 7(12):4877–4885, 2008.
- [22] L. P. Withers, R. M. Taylor, and D. M. Warne. “Echo-MIMO: A two-way channel training method for matched cooperative beamforming”. *IEEE Transactions on Signal Processing*, 56(9):4419–4432, 2008.
- [23] B. Rong, X. Qiu, M. Kadoch, S. Sun, and W. Li. “*5G Heterogeneous Networks: Self-organizing and Optimization*”. SpringerBriefs in Electrical and Computer Engineering. Springer International Publishing, 2016.
- [24] A. Katalinic, R. Nagy, and R. Zentner. “Benefits of MIMO systems in practice: Increased capacity, reliability and spectrum efficiency”. In *Proceedings ELMAR 2006*, pages 263–266, 2006.
- [25] Ehab Sahlli, Mahamod Ismail, Rosdiadee Nordin, and Nor Abdulah. “Beamforming techniques for massive MIMO systems in 5G: overview, classification, and trends for future research”. *Frontiers of Information Technology & Electronic Engineering*, 18:753–772, 06 2017.
- [26] E. G. Larsson, O. Edfors, F. Tufvesson, and T. L. Marzetta. “Massive MIMO for next generation wireless systems”. *IEEE Communications Magazine*, 52(2):186–195, 2014.
- [27] Yakun Sun and M. L. Honig. “Reduced-rank signature receiver adaptation”. *IEEE Transactions on Wireless Communications*, 5(10):2896–2902, 2006.
- [28] C. Shi, R. A. Berry, and M. L. Honig. “Bi-directional training for adaptive beamforming and power control in interference networks”. *IEEE Transactions on Signal Processing*, 62(3):607–618, 2014.
- [29] B. Zhuang, R. A. Berry, and M. L. Honig. “Interference alignment in MIMO cellular networks”. In *2011 IEEE International Conference on*

- Acoustics, Speech and Signal Processing (ICASSP)*, pages 3356–3359, 2011.
- [30] D. A. Schmidt, C. Shi, R. A. Berry, M. L. Honig, and W. Utschick. “Comparison of distributed beamforming algorithms for MIMO interference networks”. *IEEE Transactions on Signal Processing*, 61(13):3476–3489, 2013.
  - [31] CTIA. “U.S. imports of telecommunications equipment from 2002 to 2019”. Statista.com. <https://www.statista.com/statistics/221705/us-imports-of-telecommunications-equipment-from-world/>(accessed on Feb 2nd. 2021).
  - [32] H. Zhou, J. Liu, Q. Cheng, D. Maamari, W. Xiao, and A. C. K. Soong. “Bi-directional training with rank optimization and fairness control”. In *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pages 1–6, 2018.
  - [33] Balachander Ramamurthy. “*MIMO for Satellite Communication Systems*”. PhD thesis, University of South Australia, 2018.
  - [34] Yanchuan Huang, Paul Brennan, Dave Patrick, I Weller, Peters Roberts, and K Hughes. “FMCW based MIMO imaging radar for maritime navigation”. *Progress In Electromagnetics Research*, 115:327–342, 01 2011.
  - [35] B. D. Van Veen and K. M. Buckley. ”Beamforming: a versatile approach to spatial filtering”. *IEEE ASSP Magazine*, 5(2):4–24, 1988.
  - [36] Anjitha D and Shanmugha Sundaram G. A. “FPGA implementation of beamforming algorithm for terrestrial radar application”. In *2014 International Conference on Communication and Signal Processing*, pages 453–457, 2014.
  - [37] W. Shang, Z. Dou, W. Xue, and Y. Li. “Digital beamforming based on FPGA for phased array radar”. In *2017 Progress In Electromagnetics Research Symposium - Spring (PIERS)*, pages 437–440, 2017.
  - [38] Arun Singh. “A wireless networks flexible adoptive modulation and coding technique in advanced 4G LTE”. *International Journal of Information Technology*, 11:55–66, 02 2019.
  - [39] M. Viswanathan and V. Mathuranathan. “*Wireless Communication Systems in Matlab: (Black & White Edition)*”. Independently Published, 2018.

- [40] Q. Shi, M. Razaviyayn, Z. Luo, and C. He. “An iteratively weighted MMSE approach to distributed sum-utility maximization for a MIMO interfering broadcast channel. *IEEE Transactions on Signal Processing*, 59(9):4331–4340, 2011.
- [41] K. Shen and W. Yu. “Fractional programming for communication systems—Part I: Power control and beamforming. *IEEE Transactions on Signal Processing*, 66(10):2616–2630, 2018.
- [42] David MacKay. “*Information Theory, Inference, and Learning Algorithms*”, volume 50. Cambridge University Press, 01 2003.
- [43] D. J. Allred, Heejong Yoo, V. Krishnan, W. Huang, and D. V. Anderson. “LMS adaptive filters using distributed arithmetic for high throughput”. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 52(7):1327–1337, 2005.
- [44] S. Haykin and S.S. Haykin. “*Adaptive Filter Theory*”. Pearson, 2014.
- [45] A. Wang and J. O. Smith. “Some properties of tail-canceling IIR filters’. In *Proceedings of 1997 Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 4 pp.–, 1997.
- [46] M. Biguesh and A. B. Gershman. “Training-based MIMO channel estimation: a study of estimator tradeoffs and optimal training signals”. *IEEE Transactions on Signal Processing*, 54(3):884–893, 2006.
- [47] O. E. Ayach and R. W. Heath. “Interference alignment with analog channel state feedback. *IEEE Transactions on Wireless Communications*, 11(2):626–636, 2012.
- [48] Omar Ayach, Angel Lozano, and Robert Heath. “On the overhead of interference alignment: Training, feedback, and cooperation. *IEEE Transactions on Wireless Communications*, 11, 04 2012.
- [49] P. Komulainen, A. Tölli, and M. Juntti. “Effective CSI signaling and decentralized beam coordination in TDD multi-cell MIMO systems. *IEEE Transactions on Signal Processing*, 61(9):2204–2218, 2013.



# Appendices

# Appendix A

## Main functions

```
1 main_func_ls_bidirec_fr.m:
2
3 %% MIMO system bi-directional training with full-rank ...
   transmitter and receiver
4 %   Zikun Tan, MS
5
6 clear all;
7 close all;
8 clc;
9
10
11 %% System basic setting
12 %    $Y = \sqrt{P} \cdot H \cdot V \cdot b(i) + n(i)$ 
13
14 proj_Nt=8; ...
   % number of transmit antennas ...
15 proj_Nr=8; ...
   % number of receive antennas ...
16 proj_num_stream=8; ...
   % ...
   number of transmitted streams
17
18 proj_num_ite=20; ...
   % fixed number of iteration ...
19 proj_num_ite_ran=20; ...
   % ...
   range of number of iteration
20 proj_num_ite_base=1; ...
   % ...
```

```

    base of number of iteration
21
22 proj_train_leng=40; ...
                                     % ...
    fixed training length
23 proj_train_leng_ran=50; ...
                                     % ...
    length range of training sequence
24 proj_train_base=8; ...
                                     % ...
    base of training sequence
25
26 proj_ave_num=20; ...
                                     ...
    % number of trials for averaging
27
28 proj_pow=100; ...
                                     ...
    % fixed transmit power
29 proj_Gaussian_chan=Gaussian_chan_gen(proj_Nr,proj_Nt); ...
    % normalized Gaussian channel
30 proj_nor_precoder=precoder_gen(proj_Nt,proj_num_stream,1); ...
    % normalized initialized precoder
31
32
33 %% (MSE & SINR) vs Training length
34
35 %% Wiener (optimal) MMSE & SINR for fixed number of ...
    bi-directional optimization (for comparison)
36
37 [mmse_fixednumite,sinr_fixednumite,-]=wiener_mmse_sinr_fixednumite_bidirec_fr ...
    (proj_pow,proj_Gaussian_chan,proj_nor_precoder,proj_ave_num, ...
    proj_train_leng_ran,proj_train_base,proj_num_ite,0.01);
38
39
40 %% LS estimated MSE & SINR vs varied Training length & fixed ...
    Number of iteration
41
42 % with the fixed number of iteration
43 [mse_vs_trainleng_ave,mse_vs_trainleng_sam,sinr_vs_trainleng_ave, ...
    sinr_vs_trainleng_sam,train_leng]=ls_mse_sinr_vs_trainleng_bidirec_fr ...
    (proj_pow,proj_Gaussian_chan,proj_nor_precoder,proj_ave_num, ...
    proj_train_leng_ran,proj_train_base,proj_num_ite,0.01);
44 % with the half of the fixed number of iteration (for ...
    comparison)
45 [mse_vs_trainleng_ave_hf,mse_vs_trainleng_sam_hf,sinr_vs_trainleng_ave_hf, ...
    sinr_vs_trainleng_sam_hf,-]=ls_mse_sinr_vs_trainleng_bidirec_fr(proj_pow, ...
    proj_Gaussian_chan,proj_nor_precoder,proj_ave_num,proj_train_leng_ran, ...
    proj_train_base,proj_num_ite/2,0.01);

```

```

46
47 %% (MSE & SINR) vs Number of iteration
48
49 %% Wiener optimal (MMSE & SINR) vs (varied Number of ...
    iteration) (for comparison)
50
51 [mmse_variednumite,sinr_variednumite,~]=wiener_mmse_sinr_vs_numite_bidirec_fr ...
    (proj_pow,proj_Gaussian_chan,proj_nor_precoder,proj_num_ite_ran, ...
    proj_num_ite_base,0.01);
52
53
54 %% LS estimated (MSE & SINR) vs (fixed Training length & ...
    varied Number of iteration)
55
56 % with the fixed training length
57 [mse_vs_numite_ave,mse_vs_numite_sam,sinr_vs_numite_ave,sinr_vs_numite_sam, ...
    numite_leng]=ls_mse_sinr_vs_numite_bidirec_fr(proj_pow,proj_Gaussian_chan, ...
    proj_nor_precoder,proj_ave_num,proj_train_leng,proj_num_ite_ran, ...
    proj_num_ite_base,0.01);
58 % with half of the fixed training length (for comparison)
59 [mse_vs_numite_ave_hf,mse_vs_numite_sam_hf,sinr_vs_numite_ave_hf, ...
    sinr_vs_numite_sam_hf,~]=ls_mse_sinr_vs_numite_bidirec_fr(proj_pow, ...
    proj_Gaussian_chan,proj_nor_precoder,proj_ave_num,proj_train_leng/2, ...
    proj_num_ite_ran,proj_num_ite_base,0.01);
60
61
62 %% Plotting
63
64 % (MSE & SINR) vs Training length
65
66 % MSE vs Training length
67 figure(1);
68 plot(train_leng,mse_vs_trainleng_ave,'-m',train_leng,mse_vs_trainleng_sam, ...
    '-om',train_leng,mse_vs_trainleng_ave_hf,'-.m',train_leng, ...
    mse_vs_trainleng_sam_hf,'-.+m',train_leng,mmse_fixednumite,'--m');
69 title('MSE vs Training length with full-rank transmitter and ...
    receiver');
70 legend('averaged LS estimated MSE','sample LS estimated ...
    MSE','averaged LS estimated MSE with half of number of ...
    iteration','sample LS estimated MSE with half of number ...
    of iteration','Wiener (optimal) MMSE');
71 xlabel('Training length');
72 ylabel('MSE (dB)');
73
74 % SINR vs Training length
75 figure(2);
76 plot(train_leng,sinr_vs_trainleng_ave,'-m',train_leng,sinr_vs_trainleng_sam, ...
    '-om',train_leng,sinr_vs_trainleng_ave_hf,'-.m',train_leng, ...
    sinr_vs_trainleng_sam_hf,'-.+m',train_leng,sinr_fixednumite,'--m');

```

```

77 title('SINR vs Training length with full-rank transmitter ...
    and receiver');
78 legend('averaged LS estimated SINR','sample LS estimated ...
    SINR','averaged LS estimated SINR with half of number of ...
    iteration','sample LS estimated SINR with half of number ...
    of iteration','Wiener (optimal) SINR');
79 xlabel('Training length');
80 ylabel('SINR (dB)');
81
82 % (MSE & SINR) vs Number of iteration
83
84 % MSE vs Number of iteration
85 figure(3);
86 plot(numite_leng,mse_vs_numite_ave,'-m',numite_leng,mse_vs_numite_sam,'-om', ...
    numite_leng,mse_vs_numite_ave_hf,'-.m',numite_leng,mse_vs_numite_sam_hf, ...
    '-.+m',numite_leng,mmse_variednumite,'--m');
87 title('MSE vs Number of iteration with full-rank transmitter ...
    and receiver');
88 legend('averaged LS estimated MSE','sample LS estimated ...
    MSE','averaged LS estimated MSE with half of training ...
    length','sample LS estimated MSE with half of training ...
    length','Wiener (optimal) MMSE');
89 xlabel('Number of iteration');
90 ylabel('MSE (dB)');
91
92 % SINR vs Number of iteration
93 figure(4);
94 plot(numite_leng,sinr_vs_numite_ave,'-m',numite_leng,sinr_vs_numite_sam, ...
    '-om',numite_leng,sinr_vs_numite_ave_hf,'-.m',numite_leng, ...
    sinr_vs_numite_sam_hf,'-.+m',numite_leng,sinr_variednumite,'--m');
95 title('SINR vs Number of iteration with full-rank ...
    transmitter and receiver');
96 legend('averaged LS estimated SINR','sample LS estimated ...
    SINR','averaged LS estimated SINR with half of training ...
    length','sample LS estimated SINR with half of training ...
    length','Wiener (optimal) SINR');
97 xlabel('Number of iteration');
98 ylabel('SINR (dB)');
99
100
101
102
103 main_func_ls_bidirec_in_rr.m:
104
105 %% MIMO system bi-directional training with reduced-rank in ...
    an interference network
106 % Zikun Tan, MS
107
108

```

```

109 %% System basic setting
110 %   Y=sqrt(P)*H*V*b(i)+n(i)
111
112 proj_N=3; ...
                                     ...
        % number of transmitter-receiver pairs
113 proj_Nt=64; ...
                                     ...
        % number of transmit antennas
114 proj_Nr=64; ...
                                     ...
        % number of receive antennas
115 proj_num_stream=64; ...
                                     % ...
        number of transmitted streams
116
117 proj_num_ite=10; ...
                                     ...
        % fixed number of iteration
118 proj_num_ite_ran=10; ...
                                     % ...
        range of number of iteration
119 proj_num_ite_base=1; ...
                                     % ...
        base of number of iteration
120
121 proj_train_leng=20; ...
                                     % ...
        fixed training length
122 proj_train_leng_ran=30; ...
                                     % ...
        length range of training sequence
123 proj_train_base=8; ...
                                     % ...
        base of training sequence
124
125 proj_ave_num=20; ...
                                     ...
        % number of trials for averaging
126
127 red_rank=[2,4]; ...
                                     ...
        % reduced-rank
128
129 proj_pow=100; ...
                                     ...
        % transmit power
130 proj_Gaussian_chan_mat=in_Gaussian_chan_gen(proj_N,proj_Nr,proj_Nt); ...
        % Gaussian complex channel matrices

```

```

131 proj_nor_precoder_mat=in_precoder_gen(proj_N,proj_Nt,proj_num_stream,1); ...
    % normalized initialized precoders
132
133
134 %% Sum rate vs Training length
135
136 %% Wiener (optimal) Sum rate for fixed number of ...
    bi-directional optimization with full-rank & reduced-rank ...
    (for comparison)
137
138 % with full-rank
139 [sumrate_fixednumite_fr,-]=wiener_sumrate_fixednumite_bidirec_in_fr ...
    (proj_pow,proj_Gaussian_chan_mat,proj_nor_precoder_mat, ...
    proj_train_leng_ran,proj_train_base,proj_num_ite,0.01);
140 % with reduced-rank
141 [sumrate_fixednumite_rr,-]=wiener_sumrate_fixednumite_bidirec_in_rr ...
    (proj_pow,proj_Gaussian_chan_mat,proj_nor_precoder_mat,red_rank, ...
    proj_train_leng_ran,proj_train_base,proj_num_ite,0.01);
142
143
144 %% LS estimated Sum rate vs varied Training length & fixed ...
    Number of iteration with full-rank and reduced-rank
145
146 % with full-rank
147 [sumrate_vs_trainleng_ave_fr,sumrate_vs_trainleng_sam_fr,-]= ...
    ls_sumrate_vs_trainleng_bidirec_in_fr(proj_pow, ...
    proj_Gaussian_chan_mat,proj_nor_precoder_mat,proj_ave_num, ...
    proj_train_leng_ran,proj_train_base,proj_num_ite,0.01);
148 % with reduced-rank
149 [sumrate_vs_trainleng_ave_rr,sumrate_vs_trainleng_sam_rr, ...
    train_leng]=ls_sumrate_vs_trainleng_bidirec_in_rr(proj_pow, ...
    proj_Gaussian_chan_mat,proj_nor_precoder_mat,red_rank,proj_ave_num, ...
    proj_train_leng_ran,proj_train_base,proj_num_ite,0.01);
150
151
152 %% Sum rate vs Number of iteration
153
154 %% Wiener (optimal) Sum rate vs varied Number of iteration ...
    with full-rank & reduced-rank (for comparison)
155
156 % with full-rank
157 [sumrate_variednumite_fr,-]=wiener_sumrate_vs_numite_bidirec_in_fr ...
    (proj_pow,proj_Gaussian_chan_mat,proj_nor_precoder_mat, ...
    proj_num_ite_ran,proj_num_ite_base,0.01);
158 % with reduced-rank
159 [sumrate_variednumite_rr,-]=wiener_sumrate_vs_numite_bidirec_in_rr ...
    (proj_pow,proj_Gaussian_chan_mat,proj_nor_precoder_mat,red_rank, ...
    proj_num_ite_ran,proj_num_ite_base,0.01);
160

```

```

161
162 %% LS estimated Sum rate vs (fixed Training length & varied ...
    Number of iteration)
163
164 % with the fixed training length with full-rank
165 [sumrate_vs_numite_ave_fr,sumrate_vs_numite_sam_fr,-]= ...
    ls_sumrate_vs_numite_bidirec_in_fr(proj_pow,proj_Gaussian_chan_mat, ...
    proj_nor_precoder_mat,proj_ave_num,proj_train_leng,proj_num_ite_ran, ...
    proj_num_ite_base,0.01);
166 % with the fixed training length with reduced-rank
167 [sumrate_vs_numite_ave_rr,sumrate_vs_numite_sam_rr,numite_leng]= ...
    ls_sumrate_vs_numite_bidirec_in_rr(proj_pow,proj_Gaussian_chan_mat, ...
    proj_nor_precoder_mat,red_rank,proj_ave_num,proj_train_leng, ...
    proj_num_ite_ran,proj_num_ite_base,0.01);
168
169
170 %% Plotting
171
172 % Sum rate vs Training length
173 figure(1);
174 plot(train_leng,sumrate_vs_trainleng_ave_fr,'-m', ...
175      train_leng,sumrate_vs_trainleng_sam_fr,'-om', ...
176      train_leng,sumrate_vs_trainleng_ave_rr(1,:), '-c', ...
177      train_leng,sumrate_vs_trainleng_sam_rr(1,:), '-+c', ...
178      train_leng,sumrate_vs_trainleng_ave_rr(2,:), '-r', ...
179      train_leng,sumrate_vs_trainleng_sam_rr(2,:), '-*r', ...
180      train_leng,sumrate_fixednumite_fr,'--m', ...
181      train_leng,sumrate_fixednumite_rr(1,:), '--c', ...
182      train_leng,sumrate_fixednumite_rr(2,:), '--r');
183 title('Sum rate vs Training length with reduced-rank ...
    transmitters and receivers in an interference network');
184 legend('averaged LS estimated sum rate with full-rank', ...
185        'sample LS estimated sum rate with full-rank', ...
186        'averaged LS estimated sum rate with rank=2', ...
187        'sample LS estimated sum rate with rank=2', ...
188        'averaged LS estimated sum rate with rank=4', ...
189        'sample LS estimated sum rate with rank=4', ...
190        'Wiener (optimal) sum rate with full-rank', ...
191        'Wiener (optimal) sum rate with rank=2', ...
192        'Wiener (optimal) sum rate with rank=4');
193 xlabel('Training length');
194 ylabel('Sum rate');
195
196 % Sum rate vs Number of iteration
197 figure(2);
198 plot(numite_leng,sumrate_vs_numite_ave_fr,'-m', ...
199      numite_leng,sumrate_vs_numite_sam_fr,'-om', ...
200      numite_leng,sumrate_vs_numite_ave_rr(1,:), '-c', ...
201      numite_leng,sumrate_vs_numite_sam_rr(1,:), '-+c', ...

```



```

202     numite_leng,sumrate_vs_numite_ave_rr(2,:), '-r', ...
203     numite_leng,sumrate_vs_numite_sam_rr(2,:), '-*r', ...
204     numite_leng,sumrate_variednumite_fr, '--m', ...
205     numite_leng,sumrate_variednumite_rr(1,:), '--c', ...
206     numite_leng,sumrate_variednumite_rr(2,:), '--r');
207 title('Sum rate vs Number of iteration with reduced-rank ...
        transmitters and receivers in an interference network');
208 legend('averaged LS estimated sum rate with full-rank', ...
209         'sample LS estimated sum rate with full-rank', ...
210         'averaged LS estimated sum rate with rank=2', ...
211         'sample LS estimated sum rate with rank=2', ...
212         'averaged LS estimated sum rate with rank=4', ...
213         'sample LS estimated sum rate with rank=4', ...
214         'Wiener (optimal) sum rate with full-rank', ...
215         'Wiener (optimal) sum rate with rank=2', ...
216         'Wiener (optimal) sum rate with rank=4');
217 xlabel('Number of iteration');
218 ylabel('Sum rate');
219
220
221
222
223 main_func_ls_bidirec_rr.m:
224
225 %% MIMO system bi-directional training with reduced-rank
226 %   Zikun Tan, MS
227
228 clear all;
229 close all;
230 clc;
231
232
233 %% System basic setting
234 %    $Y = \sqrt{P} * H * V * b(i) + n(i)$ 
235
236 proj_Nt=64; ...
237
238                                     ...
239                                     % number of transmit antennas
240 proj_Nr=64; ...
241                                     ...
242                                     % number of receive antennas
243                                     % ...
244                                     number of transmitted streams
245
246 proj_num_stream=64; ...
247
248                                     ...
249                                     % fixed number of iteration
250 proj_num_ite=20; ...
251
252                                     ...

```

```

range of number of iteration                                % ...
242 proj_num_ite_base=1; ...                                % ...
base of number of iteration
243
244 proj_train_leng=40; ...                                % ...
fixed training length
245 proj_train_leng_ran=50; ...                            % ...
length range of training sequence
246 proj_train_base=8; ...                                % ...
base of training sequence
247
248 proj_ave_num=20; ...                                    ...
% number of trials for averaging
249
250 red_rank=[2,4]; ...                                    ...
% reduced-rank
251
252 proj_pow=100; ...                                      ...
% transmit power
253 proj_Gaussian_chan=Gaussian_chan_gen(proj_Nr,proj_Nt); ...
% Gaussian complex channel matrix
254 proj_nor_precoder=precoder_gen(proj_Nt,proj_num_stream,1); ...
% normalized initialized precoder
255
256
257 %% (MSE & SINR) vs Training length
258
259 %% Wiener (optimal) MMSE & SINR for fixed number of ...
bi-directional optimization with full-rank & reduced-rank ...
(for comparison)
260
261 % with full-rank
262 [mmse_fixednumite_fr,sinr_fixednumite_fr,-]= ...
wiener_mmse_sinr_fixednumite_bidirec_fr(proj_pow,proj_Gaussian_chan, ...
proj_nor_precoder,proj_train_leng_ran,proj_train_base,proj_num_ite, ...
0.01);
263 % with reduced-rank
264 [mmse_fixednumite_rr,sinr_fixednumite_rr,-]= ...
wiener_mmse_sinr_fixednumite_bidirec_rr(proj_pow,proj_Gaussian_chan, ...
proj_nor_precoder,red_rank,proj_train_leng_ran,proj_train_base, ...
proj_num_ite,0.01);

```

```

265
266
267 %% LS estimated MSE & SINR vs varied Training length & fixed ...
    Number of iteration with full-rank and reduced-rank
268
269 % with full-rank
270 [mse_vs_trainleng_ave_fr,mse_vs_trainleng_sam_fr, ...
    sinr_vs_trainleng_ave_fr,sinr_vs_trainleng_sam_fr,-]= ...
    ls_mse_sinr_vs_trainleng_bidirec_fr(proj_pow,proj_Gaussian_chan, ...
    proj_nor_precoder,proj_ave_num,proj_train_leng_ran,proj_train_base, ...
    proj_num_ite,0.01);
271 % with reduced-rank
272 [mse_vs_trainleng_ave_rr,mse_vs_trainleng_sam_rr, ...
    sinr_vs_trainleng_ave_rr,sinr_vs_trainleng_sam_rr,train_leng]= ...
    ls_mse_sinr_vs_trainleng_bidirec_rr(proj_pow,proj_Gaussian_chan, ...
    proj_nor_precoder,red_rank,proj_ave_num,proj_train_leng_ran, ...
    proj_train_base,proj_num_ite,0.01);
273
274
275 %% (MSE & SINR) vs Number of iteration
276
277 %% Wiener (optimal) MMSE & SINR vs varied Number of ...
    iteration with full-rank & reduced-rank (for comparison)
278
279 % with full-rank
280 [mmse_variednumite_fr,sinr_variednumite_fr,-]= ...
    wiener_mmse_sinr_vs_numite_bidirec_fr(proj_pow,proj_Gaussian_chan, ...
    proj_nor_precoder,proj_num_ite_ran,proj_num_ite_base,0.01);
281 % with reduced-rank
282 [mmse_variednumite_rr,sinr_variednumite_rr,-]= ...
    wiener_mmse_sinr_vs_numite_bidirec_rr(proj_pow,proj_Gaussian_chan, ...
    proj_nor_precoder,red_rank,proj_num_ite_ran,proj_num_ite_base,0.01);
283
284
285 %% LS estimated (MSE & SINR) vs (fixed Training length & ...
    varied Number of iteration)
286
287 % with the fixed training length with full-rank
288 [mse_vs_numite_ave_fr,mse_vs_numite_sam_fr,sinr_vs_numite_ave_fr, ...
    sinr_vs_numite_sam_fr,-]=ls_mse_sinr_vs_numite_bidirec_fr ...
    (proj_pow,proj_Gaussian_chan,proj_nor_precoder,proj_ave_num, ...
    proj_train_leng,proj_num_ite_ran,proj_num_ite_base,0.01);
289 % with the fixed training length with reduced-rank
290 [mse_vs_numite_ave_rr,mse_vs_numite_sam_rr,sinr_vs_numite_ave_rr, ...
    sinr_vs_numite_sam_rr,numite_leng]=ls_mse_sinr_vs_numite_bidirec_rr ...
    (proj_pow,proj_Gaussian_chan,proj_nor_precoder,red_rank, ...
    proj_ave_num,proj_train_leng,proj_num_ite_ran,proj_num_ite_base,0.01);
291
292

```

```

293 %% Plotting
294
295 % (MSE & SINR) vs Training length
296
297 % MSE vs Training length
298 figure(1);
299 plot(train_leng,mse_vs_trainleng_ave_fr,'-m', ...
300      train_leng,mse_vs_trainleng_sam_fr,'-om', ...
301      train_leng,mse_vs_trainleng_ave_rr(1,:), '-c', ...
302      train_leng,mse_vs_trainleng_sam_rr(1,:), '-+c', ...
303      train_leng,mse_vs_trainleng_ave_rr(2,:), '-r', ...
304      train_leng,mse_vs_trainleng_sam_rr(2,:), '-*r', ...
305      train_leng,mmse_fixednumite_fr,'--m', ...
306      train_leng,mmse_fixednumite_rr(1,:), '--c', ...
307      train_leng,mmse_fixednumite_rr(2,:), '--r');
308 title('MSE vs Training length with reduced-rank transmitter ...
        and receiver');
309 legend('averaged LS estimated MSE with full-rank', ...
310       'sample LS estimated MSE with full-rank', ...
311       'averaged LS estimated MSE with rank=2', ...
312       'sample LS estimated MSE with rank=2', ...
313       'averaged LS estimated MSE with rank=4', ...
314       'sample LS estimated MSE with rank=4', ...
315       'Wiener (optimal) MMSE with full-rank', ...
316       'Wiener (optimal) MMSE with rank=2', ...
317       'Wiener (optimal) MMSE with rank=4');
318 xlabel('Training length');
319 ylabel('MSE (dB)');
320
321 % SINR vs Training length
322 figure(2);
323 plot(train_leng,sinr_vs_trainleng_ave_fr,'-m', ...
324      train_leng,sinr_vs_trainleng_sam_fr,'-om', ...
325      train_leng,sinr_vs_trainleng_ave_rr(1,:), '-c', ...
326      train_leng,sinr_vs_trainleng_sam_rr(1,:), '-+c', ...
327      train_leng,sinr_vs_trainleng_ave_rr(2,:), '-r', ...
328      train_leng,sinr_vs_trainleng_sam_rr(2,:), '-*r', ...
329      train_leng,sinr_fixednumite_fr,'--m', ...
330      train_leng,sinr_fixednumite_rr(1,:), '--c', ...
331      train_leng,sinr_fixednumite_rr(2,:), '--r');
332 title('SINR vs Training length with reduced-rank transmitter ...
        and receiver');
333 legend('averaged LS estimated SINR with full-rank', ...
334       'sample LS estimated SINR with full-rank', ...
335       'averaged LS estimated SINR with rank=2', ...
336       'sample LS estimated SINR with rank=2', ...
337       'averaged LS estimated SINR with rank=4', ...
338       'sample LS estimated SINR with rank=4', ...
339       'Wiener (optimal) SINR with full-rank', ...

```

```

340         'Wiener (optimal) SINR with rank=2', ...
341         'Wiener (optimal) SINR with rank=4');
342 xlabel('Training length');
343 ylabel('SINR (dB)');
344
345 % (MSE & SINR) vs Number of iteration
346
347 % MSE vs Number of iteration
348 figure(3);
349 plot(numite_leng,mse_vs_numite_ave_fr,'-m', ...
350      numite_leng,mse_vs_numite_sam_fr,'-om', ...
351      numite_leng,mse_vs_numite_ave_rr(1,:), '-c', ...
352      numite_leng,mse_vs_numite_sam_rr(1,:), '-+c', ...
353      numite_leng,mse_vs_numite_ave_rr(2,:), '-r', ...
354      numite_leng,mse_vs_numite_sam_rr(2,:), '-*r', ...
355      numite_leng,mmse_variednumite_fr,'--m', ...
356      numite_leng,mmse_variednumite_rr(1,:), '--c', ...
357      numite_leng,mmse_variednumite_rr(2,:), '--r');
358 title('MSE vs Number of iteration with reduced-rank ...
        transmitter and receiver');
359 legend('averaged LS estimated MSE with full-rank', ...
360        'sample LS estimated MSE with full-rank', ...
361        'averaged LS estimated MSE with rank=2', ...
362        'sample LS estimated MSE with rank=2', ...
363        'averaged LS estimated MSE with rank=4', ...
364        'sample LS estimated MSE with rank=4', ...
365        'Wiener (optimal) MMSE with full-rank', ...
366        'Wiener (optimal) MMSE with rank=2', ...
367        'Wiener (optimal) MMSE with rank=4');
368 xlabel('Number of iteration');
369 ylabel('MSE (dB)');
370
371 % SINR vs Number of iteration
372 figure(4);
373 plot(numite_leng,sinr_vs_numite_ave_fr,'-m', ...
374      numite_leng,sinr_vs_numite_sam_fr,'-om', ...
375      numite_leng,sinr_vs_numite_ave_rr(1,:), '-c', ...
376      numite_leng,sinr_vs_numite_sam_rr(1,:), '-+c', ...
377      numite_leng,sinr_vs_numite_ave_rr(2,:), '-r', ...
378      numite_leng,sinr_vs_numite_sam_rr(2,:), '-*r', ...
379      numite_leng,sinr_variednumite_fr,'--m', ...
380      numite_leng,sinr_variednumite_rr(1,:), '--c', ...
381      numite_leng,sinr_variednumite_rr(2,:), '--r');
382 title('SINR vs Number of iteration with reduced-rank ...
        transmitter and receiver');
383 legend('averaged LS estimated SINR with full-rank', ...
384        'sample LS estimated SINR with full-rank', ...
385        'averaged LS estimated SINR with rank=2', ...
386        'sample LS estimated SINR with rank=2', ...

```

```

387         'averaged LS estimated SINR with rank=4', ...
388         'sample LS estimated SINR with rank=4', ...
389         'Wiener (optimal) SINR with full-rank', ...
390         'Wiener (optimal) SINR with rank=2', ...
391         'Wiener (optimal) SINR with rank=4');
392 xlabel('Number of iteration');
393 ylabel('SINR (dB)');
394
395
396
397
398 main_func_ls_fr.m:
399
400 %% MIMO system Least Squares (LS) receiver with full-rank
401 %   Zikun Tan, MS
402
403 clear all;
404 close all;
405 clc;
406
407
408 %% System basic setting
409 %   Y=sqrt(P)*H*V*b(i)+n(i)
410
411 proj_Nt=8; ...
412
413         % number of transmit antennas ...
414 proj_Nr=8; ...
415
416         % number of receive antennas ...
417 proj_num_stream=8; ...
418
419         % ...
420         number of transmitted streams
421         % ...
422 proj_train_leng_ran=50; ...
423
424         % ...
425         length steps of training sequence
426         % ...
427 proj_ave_num=20; ...
428
429         % ...
430         % number of trials
431         % ...
432 proj_train_base=8; ...
433
434         % ...
435         base of training sequence
436         % ...
437 proj_pow=100; ...
438
439         % ...
440         % transmit power
441         % ...
442         proj_Gaussian_chan=Gaussian_chan_gen(proj_Nr,proj_Nt); ...
443         % Gaussian complex channel matrix
444         proj_nor_precoder=precoder_gen(proj_Nt,proj_num_stream,1); ...
445         % normalized precoder

```

```

420 mix_mat=sqrt(proj_pow)*proj_Gaussian_chan*proj_nor_precoder; ...
    % mixing matrix
421
422
423 %% Wiener (optimal) estimated receive filter
424
425 [mmse_op,sinr_op]=wiener_mmse_sinr_fr(mix_mat,0.01);
426 mmse_op=mmse_op*ones(1,proj_train_leng_ran);
427 sinr_op=sinr_op*ones(1,proj_train_leng_ran);
428
429
430 %% LS estimated receive filter
431
432 [mse_ls_ave,mse_ls_sam,sinr_ls_ave,sinr_ls_sam,train_leng]= ...
    ls_mse_sinr_vs_trainleng_fr(mix_mat,proj_ave_num, ...
    proj_train_leng_ran,proj_train_base,0.01);
433
434
435 %% Plotting
436
437 % plot MSE vs Training length
438 figure(1);
439 plot(train_leng,mse_ls_ave,'-m',train_leng,mse_ls_sam,'-om', ...
    train_leng,mmse_op,'--m');
440 title('MSE vs Training length with full-rank receiver');
441 legend('LS estimated averaged MSE', ...
442        'LS estimated sample MSE', ...
443        'Wiener (optimal) MMSE');
444 xlabel('Training length');
445 ylabel('MSE (dB)');
446
447 % plot SINR vs Training length
448 figure(2);
449 plot(train_leng,sinr_ls_ave,'-m',train_leng,sinr_ls_sam,'-om', ...
    train_leng,sinr_op,'--m');
450 title('SINR vs Training length with full-rank receiver');
451 legend('LS estimated averaged SINR', ...
452        'LS estimated sample SINR', ...
453        'Wiener (optimal) SINR');
454 xlabel('Training length');
455 ylabel('SINR (dB)');
456
457
458
459
460 main_func_ls_rr_un.m:
461
462 %% MIMO system Least Squares (LS) receiver with ...
    reduced-rank, with time-averaging Krylov subspace

```

```

463 % Zikun Tan, MS
464
465 clear all;
466 close all;
467 clc;
468
469
470 %% System basic setting
471 % Y=sqrt(P)*H*V*b(i)+n(i)
472
473 proj_Nt=8; ...
                                     ...
                                     % number of transmit antennas
474 proj_Nr=8; ...
                                     ...
                                     % number of receive antennas
475 proj_num_stream=8; ...
                                     % ...
                                     number of transmitted streams
476
477 proj_train_leng_ran=50; ...
                                     % ...
                                     length range of training sequence
478 proj_train_base=8; ...
                                     % ...
                                     base of training sequence
479
480 proj_ave_num=20; ...
                                     ...
                                     % number of trials for averaging
481
482 proj_pow=100; ...
                                     ...
                                     % transmit power
483 proj_Gaussian_chan=Gaussian_chan_gen(proj_Nr,proj_Nt); ...
                                     % Gaussian complex channel
484 proj_nor_precoder=precoder_gen(proj_Nt,proj_num_stream,1); ...
                                     % normalized precoder
485 mix_mat=sqrt(proj_pow)*proj_Gaussian_chan*proj_nor_precoder; ...
                                     % initialized mixing matrix
486 red_rank=[2,4]; ...
                                     ...
                                     % reduced-rank
487
488
489 %% Wiener (optimal) estimated performance with full-rank
490
491 [mmse_wiener_fr,sinr_wiener_fr]=wiener_mmse_sinr_fr(mix_mat,0.01);
492 mmse_wiener_fr=mmse_wiener_fr*ones(1,proj_train_leng_ran);

```



```

493 sinr_wiener_fr=sinr_wiener_fr*ones(1,proj_train_leng_ran);
494
495
496 %% Wiener (optimal) estimated performance with reduced-rank
497
498 [mmse_wiener_rr,sinr_wiener_rr]=wiener_mmse_sinr_rr(mix_mat, ...
    red_rank,0.01);
499 mmse_wiener_rr=transpose(mmse_wiener_rr)*ones(1,proj_train_leng_ran);
500 sinr_wiener_rr=transpose(sinr_wiener_rr)*ones(1,proj_train_leng_ran);
501
502
503 %% LS estimated performance with full-rank (MSE & SINR) vs ...
    Training length
504
505 [mse_ls_fr_ave,mse_ls_fr_sam,sinr_ls_fr_ave,sinr_ls_fr_sam,-]= ...
    ls_mse_sinr_vs_trainleng_fr(mix_mat,proj_ave_num, ...
    proj_train_leng_ran,proj_train_base,0.01);
506
507
508 %% LS estimated performance with reduced-rank (MSE & SINR) ...
    vs Training length
509
510 [mse_ls_rr_ave,mse_ls_rr_sam,sinr_ls_rr_ave,sinr_ls_rr_sam, ...
    train_leng]=ls_mse_sinr_vs_trainleng_rr_un(mix_mat,red_rank, ...
    proj_ave_num,proj_train_leng_ran,proj_train_base,0.01);
511
512
513 %% Plotting
514
515 % plot the MSE vs Training length
516 figure(1);
517 plot(train_leng,mmse_wiener_fr,'--m', ...
    train_leng,mmse_wiener_rr(1,:), '--c', ...
    train_leng,mmse_wiener_rr(2,:), '--r', ...
    train_leng,mse_ls_fr_ave, '-m', ...
    train_leng,mse_ls_fr_sam, '-om', ...
    train_leng,mse_ls_rr_ave(1,:), '-c', ...
    train_leng,mse_ls_rr_sam(1,:), '-+c', ...
    train_leng,mse_ls_rr_ave(2,:), '-r', ...
    train_leng,mse_ls_rr_sam(2,:), '-*r');
526 title('MSE vs Training length with reduced-rank receiver');
527 legend('Wiener (optimal) MMSE with full-rank', ...
    'Wiener (optimal) MMSE with rank=2', ...
    'Wiener (optimal) MMSE with rank=4', ...
    'full-rank average MSE', ...
    'full-rank sample MSE', ...
    'rank=2 average MSE', ...
    'rank=2 sample MSE', ...
    'rank=4 average MSE', ...

```

```

535         'rank=4 sample MSE');
536 xlabel('Training length');
537 ylabel('MSE (dB)');
538
539 % plot the SINR vs Training length
540 figure(2);
541 plot(train_leng,sinr_wiener_fr,'--m', ...
542      train_leng,sinr_wiener_rr(1,:), '--c', ...
543      train_leng,sinr_wiener_rr(2,:), '--r', ...
544      train_leng,sinr_ls_fr_ave,'-m', ...
545      train_leng,sinr_ls_fr_sam,'-om', ...
546      train_leng,sinr_ls_rr_ave(1,:), '-c', ...
547      train_leng,sinr_ls_rr_sam(1,:), '-+c', ...
548      train_leng,sinr_ls_rr_ave(2,:), '-r', ...
549      train_leng,sinr_ls_rr_sam(2,:), '-*r');
550 title('SINR vs Training length with reduced-rank receiver');
551 legend('Wiener (optimal) SINR', ...
552        'Wiener (optimal) SINR with rank=2', ...
553        'Wiener (optimal) SINR with rank=4', ...
554        'full-rank average SINR', ...
555        'full-rank sample SINR', ...
556        'rank=2 average SINR', ...
557        'rank=2 sample SINR', ...
558        'rank=4 average SINR', ...
559        'rank=4 sample SINR');
560 xlabel('Training length');
561 ylabel('SINR (dB)');
562
563
564
565
566 main_func_wiener_fr.m:
567
568 %% MIMO system Wiener (optimal) receiver with full-rank ...
569     performance simulation
570 %   Zikun Tan, MS
571
572 clear all;
573 close all;
574 clc;
575
576 %% System basic setting
577 %    $Y = \sqrt{P} * H * V * b + n$ 
578
579 proj_Nt=8; ...
580
581     % number of transmit antennas
582 proj_Nr=8; ...

```

```

...
    % number of receive antennas
581 proj_num_stream=8; ...

    % ...
    number of transmitted streams
582 proj_xleng=11; ...

    % range of X-axis
    ...
583 proj_pow=zeros(1,proj_xleng); ...

    % transmit power
584 for m=1:proj_xleng
585     proj_pow(1,m)=0.01*10^((3*m-3)/10); ...
    % transmit power, ...
    making SNR from 0dB to 30dB
586 end
587 proj_Gaussian_chan=Gaussian_chan_gen(proj_Nr,proj_Nt); ...
    % Gaussian complex channel matrix
588 proj_nor_precoder=precoder_gen(proj_Nt,proj_num_stream,1); ...
    % normalized precoder

589
590
591 %% (MMSE & SINR) vs SNR
592
593 [mmse,sinr,snr]=wiener_mmse_sinr_vs_snr_fr(proj_pow, ...
    proj_Gaussian_chan,proj_nor_precoder,0.01);
594
595
596 %% Plotting (MMSE & SINR) vs SNR
597
598 % MMSE vs SNR
599 figure(1);
600 plot(snr,mmse,'-om');
601 title('MMSE vs Background SNR with full-rank receiver');
602 xlabel('Background SNR (dB)');
603 ylabel('MMSE (dB)');
604
605 % SINR vs SNR
606 figure(2);
607 plot(snr,sinr,'-om');
608 title('SINR vs Background SNR with full-rank receiver');
609 xlabel('Background SNR (dB)');
610 ylabel('SINR (dB)');
611
612
613
614
615 main_func_wiener_rr.m:
616
617 %% MIMO system Wiener (optimal) receiver with reduced-rank

```

```

618 % Zikun Tan, MS
619
620 clear all;
621 close all;
622 clc;
623
624
625 %% System basic setting
626 %  $Y = \sqrt{P} * H * V * b + n$ 
627
628 proj_Nt=8; ...
                                     ...
        % number of transmit antennas
629 proj_Nr=8; ...
                                     ...
        % number of receive antennas
630 proj_num_stream=8; ...
                                     % ...
        number of transmitted streams
631 proj_xleng=11; ...
                                     ...
        % range of X-axis
632 red_rank=[2,4]; ...
                                     ...
        % reduced rank
633 proj_pow=zeros(1,proj_xleng); ...
                                     % transmit power
634 for m=1:proj_xleng
635     proj_pow(1,m)=0.01*10^((3*m-3)/10); ...
                                     % transmit power, ...
        from 0dB to 30dB
636 end
637 proj_Gaussian_chan=Gaussian_chan_gen(proj_Nr,proj_Nt); ...
        % Gaussian complex channel matrix
638 proj_nor_precoder=precoder_gen(proj_Nt,proj_num_stream,1); ...
        % normalized precoder
639
640
641 %% Full-rank Wiener (optimal) (MMSE & SINR) vs SNR
642
643 [mmse_fr,sinr_fr,-]=wiener_mmse_sinr_vs_snr_fr(proj_pow, ...
        proj_Gaussian_chan,proj_nor_precoder,0.01);
644
645
646 %% Reduced-rank Wiener (optimal) (MMSE & SINR) vs SNR
647
648 [mmse_rr,sinr_rr,-]=wiener_mmse_sinr_vs_snr_rr(proj_pow, ...
        proj_Gaussian_chan,proj_nor_precoder,0.01,red_rank);
649

```

```

650
651 %% Plotting
652
653 snr=10*log10(proj_pow/0.01); ...
                                     % ...
        background SNR
654
655 % MMSE vs Background SNR
656 figure(1);
657 plot(snr,mmse_fr,'-om', ...
658      snr,mmse_rr(1,:), '-+c', ...
659      snr,mmse_rr(2,:), '-*r');
660 title('MMSE vs Background SNR with reduced-rank receiver');
661 legend('full-rank', ...
662        'rank=2', ...
663        'rank=4');
664 xlabel('Background SNR (dB)');
665 ylabel('MSE (dB)');
666
667 % SINR vs Background SNR
668 figure(2);
669 plot(snr,sinr_fr,'-om', ...
670      snr,sinr_rr(1,:), '-+c', ...
671      snr,sinr_rr(2,:), '-*r');
672 title('SINR vs Background SNR with reduced-rank receiver');
673 legend('full-rank', ...
674        'rank=2', ...
675        'rank=4');
676 xlabel('Background SNR (dB)');
677 ylabel('SINR (dB)');

```

# Appendix B

## Self-defined functions

```
1 bidirec_noiseseq_gen.m:
2
3 % Generate noise sequence symbols for all bi-directional ...
  training
4
5 function [noise_mat] = ...
  bidirec_noiseseq_gen(size_1,size_2,size_3)
6
7 noise_mat=zeros(size_1,size_2,size_3);
8 for m=1:size_1
9     noise_mat(m,:,:)=noise_mat_gen(size_2,size_3);
10 end
11
12 end
13
14
15
16
17 bidirec_trainseq_gen.m:
18
19 % Generate BPSK training sequences for all bi-directional ...
  training
20
21 function [trainseq_mat] = ...
  bidirec_trainseq_gen(size_1,size_2,size_3)
22
23 trainseq_mat=zeros(size_1,size_2,size_3);
24 for m=1:size_1
25     trainseq_mat(m,:,:)=bpsk_mat_gen(size_2,size_3);
26 end
27
28 end
```

```

29
30
31
32
33 bpsk_mat_gen.m:
34
35 % BPSK symbol sequence generator
36
37 function bpsk_mat = bpsk_mat_gen(num_stream,train_leng)
38
39 bpsk_mat=sign(rand(num_stream,train_leng)-0.5*ones( ...
    num_stream,train_leng));
40
41 end
42
43
44
45
46 Gaussian_chan_gen.m:
47
48 % Generate normalized Gaussian complex channel
49
50 function Gaussian_chan = Gaussian_chan_gen(Nr,Nt)
51
52 % normalized Gaussian distributed MIMO channel
53 Gaussian_chan=(1/sqrt(2*Nt))*(randn(Nr,Nt)+i*randn(Nr,Nt));
54
55 end
56
57
58
59
60 in_bidirec_noiseseq_gen.m:
61
62 % Generate noise sequence symbols for all bi-directional ...
    training in
63 % interference network
64
65 function [noise_mat] = ...
    in_bidirec_noiseseq_gen(size_1,size_2,size_3,size_4)
66
67 noise_mat=zeros(size_1,size_2,size_3,size_4);
68 for m=1:size_1
69     noise_mat(m,:,:,:)=bidirec_noiseseq_gen(size_2,size_3,size_4);
70 end
71
72 end
73
74

```

```

75
76
77 in_bidirec_trainseq_gen.m:
78
79 % Generate BPSK training sequences for all bi-directional ...
    training in
80 % interference network
81
82 function [trainseq_mat] = ...
    in_bidirec_trainseq_gen(size_1,size_2,size_3,size_4)
83
84 trainseq_mat=zeros(size_1,size_2,size_3,size_4);
85
86 for m=1:size_1
87     trainseq_mat(m,:,:,:)=bidirec_trainseq_gen(size_2,size_3,size_4);
88 end
89
90 end
91
92
93
94
95 in_Gaussian_chan_gen.m:
96
97 % Interference network Gaussian channels generator
98
99 function [Gaussian_chan_mat] = in_Gaussian_chan_gen(N,Nt,Nr)
100
101 Gaussian_chan_mat=zeros(N,N,Nr,Nt);
102
103 % assign for each channel
104 for m=1:N
105     for n=1:N
106         Gaussian_chan_mat(m,n,:,:)=Gaussian_chan_gen(Nr,Nt);
107     end
108 end
109
110 end
111
112
113
114
115 in_precoder_gen.m:
116
117 % Interference network precoders generator
118
119 function [precoder_mat] = in_precoder_gen(N,Nt,num_stream,pow)
120
121 precoder_mat=zeros(N,Nt,num_stream);

```



```

122
123 for m=1:N
124     precoder_mat(m, :, :) = precoder_gen(Nt, num_stream, pow);
125 end
126
127 end
128
129
130
131
132 ls_mse_sinr_vs_numite_bidirec_fr.m:
133
134 % LS estimated MSE & SINR vs Number of bi-directional ...
    training iteration with full-rank
135
136 function [mse_ave, mse_sam, sinr_ave, sinr_sam, num_ite_leng] = ...
    ls_mse_sinr_vs_numite_bidirec_fr(pow, Gaussian_chan, nor_precoder, ...
    ave_num, train_leng, num_ite_ran, num_ite_base, noise_covar)
137
138 % parameter extraction
139 [Nr, num_stream] = size(Gaussian_chan * nor_precoder);
140
141 % output container
142 mse = zeros(ave_num, num_ite_ran / num_ite_base);
143 sinr = zeros(ave_num, num_ite_ran / num_ite_base);
144
145 % for each trial in averaging
146 for m=1:ave_num
147     % training sequence
148     b = bidirec_trainseq_gen(2 * num_ite_ran + 1, num_stream, train_leng);
149     % noise sequence
150     noise = bidirec_noiseseq_gen(2 * num_ite_ran + 1, Nr, train_leng);
151
152     % for each number of bi-directional training iteration
153     for n=num_ite_base:num_ite_base:num_ite_ran
154         % initialized forward training
155         b_dot = reshape(b(1, :, :), [num_stream, train_leng]);
156         noise_dot = reshape(noise(1, :, :), [Nr, train_leng]);
157
158         % initialized mixing matrix
159         mix_mat = sqrt(pow) * Gaussian_chan * nor_precoder;
160         % sample correlation matrix
161         cor_mat_ls = (mix_mat * b_dot + noise_dot) ...
162             * ctranspose(mix_mat * b_dot + noise_dot) / train_leng;
163         % sample cross-correlation matrix
164         cro_cor_mat_ls = (mix_mat * b_dot + noise_dot) ...
165             * ctranspose(b_dot) / train_leng;
166         % LS estimated receive filter and power normalization
167         rece_fil_ls = cor_mat_ls \ cro_cor_mat_ls;

```

```

168         % power normalization
169         rece_fil_ls_nor=mat_pow_nor(rece_fil_ls);
170
171         % for bi-directional training with each number of ...
172         iteration
173         for n_d=1:n
174             % backward training
175             b_dot=reshape(b(2*n_d,:,:),[num_stream,train_leng]);
176             noise_dot=reshape(noise(2*n_d,:,:),[Nr,train_leng]);
177
178             % backward mixing matrix
179             mix_mat=transpose(sqrt(pow)*Gaussian_chan)* ...
180             conj(rece_fil_ls_nor);
181             % sample correlation matrix
182             cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
183             *ctranspose(mix_mat*b_dot+noise_dot)/ ...
184             train_leng;
185             % sample cross-correlation matrix
186             cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
187             *ctranspose(b_dot)/train_leng;
188             % LS estimated precoder
189             precoder_ls=conj(cor_mat_ls\cro_cor_mat_ls);
190             % power normalization
191             precoder_ls_nor=mat_pow_nor(precoder_ls);
192
193             % forward training
194             b_dot=reshape(b(2*n_d+1,:,:),[num_stream,train_leng]);
195             noise_dot=reshape(noise(2*n_d+1,:,:),[Nr,train_leng]);
196
197             % forward mixing matrix
198             mix_mat=sqrt(pow)*Gaussian_chan*precoder_ls_nor;
199             % sample correlation matrix
200             cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
201             *ctranspose(mix_mat*b_dot+noise_dot)/ ...
202             train_leng;
203             % sample cross-correlation matrix
204             cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
205             *ctranspose(b_dot)/train_leng;
206             % LS estimated receive filter
207             rece_fil_ls=cor_mat_ls\cro_cor_mat_ls;
208             % power normalization
209             rece_fil_ls_nor=mat_pow_nor(rece_fil_ls);
210         end
211
212         % statistical correlation matrix
213         cor_mat_op=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
214
215         % MSE & SINR computation
216         mse_temp=zeros(1,num_stream);

```

```

213         sinr_temp=zeros(1,num_stream);
214         for n_d=1:num_stream
215
216             % MSE computation
217             mse_temp_val=1-abs(transpose(rece_fil_ls(:,n_d))* ...
218                 conj(mix_mat(:,n_d))) ...
219                 -abs(ctranspose(rece_fil_ls(:,n_d))* ...
220                     mix_mat(:,n_d)) ...
221                 +abs(ctranspose(rece_fil_ls(:,n_d))* ...
222                     cor_mat_op*rece_fil_ls(:,n_d));
223             mse_temp(1,n_d)=mse_temp_val;
224
225             % signal power
226             signal_pow=abs(ctranspose(rece_fil_ls(:,n_d))* ...
227                 mix_mat(:,n_d))^2;
228
229             % interference power
230             interf_pow=0;
231             for n_dd=1:num_stream
232                 if n_dd==n_d
233                     % pass
234                 else
235                     interf_pow=interf_pow+abs(ctranspose ...
236                         (rece_fil_ls(:,n_d))*mix_mat(:,n_dd))^2;
237                 end
238             end
239
240             % noise power
241             noise_pow=abs(ctranspose(rece_fil_ls(:,n_d))* ...
242                 noise_covar*rece_fil_ls(:,n_d));
243
244             % SINR computation
245             sinr_temp_val=signal_pow/(interf_pow+noise_pow);
246             sinr_temp(1,n_d)=sinr_temp_val;
247         end
248
249         % average over all streams
250         mse(m,n/num_ite_base)=mean(mse_temp,2);
251         sinr(m,n/num_ite_base)=mean(sinr_temp,2);
252     end
253 end
254
255 % data averaging, sample data extraction and decibel conversion
256 mse_ave=10*log10(mean(mse));
257 mse_sam=10*log10(mse(1,:));
258 sinr_ave=10*log10(mean(sinr));
259 sinr_sam=10*log10(sinr(1,:));
260
261 % number of iteration axis

```

```

256 num_ite_leng=zeros(1,num_ite_ran/num_ite_base);
257 for m=1:num_ite_ran/num_ite_base
258     num_ite_leng(1,m)=m*num_ite_base;
259 end
260
261 end
262
263
264
265
266 ls_mse_sinr_vs_numite_bidirec_rr.m:
267
268 % LS estimated MSE & SINR vs Number of bi-directional ...
    training iteration with reduced-rank
269
270 function [mse_ave,mse_sam,sinr_ave,sinr_sam,num_ite_leng] = ...
    ls_mse_sinr_vs_numite_bidirec_rr(pow,Gaussian_chan,nor_precoder, ...
    red_rank,ave_num,train_leng,num_ite_ran,num_ite_base,noise_covar)
271
272 % parameter extraction
273 [Nr,num_stream]=size(Gaussian_chan*nor_precoder);
274
275 % output container
276 mse=zeros(size(red_rank,2),ave_num,num_ite_ran/num_ite_base);
277 sinr=zeros(size(red_rank,2),ave_num,num_ite_ran/num_ite_base);
278
279 % for every reduced-rank
280 for rr_ind=1:size(red_rank,2)
281     % assign current reduced-rank
282     rr=red_rank(rr_ind);
283
284     % for each trial in averaging
285     for m=1:ave_num
286         % training sequence
287         b=bidirec_trainseq_gen(2*num_ite_ran+1,num_stream,train_leng);
288         % noise sequence
289         noise=bidirec_noiseseq_gen(2*num_ite_ran+1,Nr,train_leng);
290
291         % for each number of bi-directional training iteration
292         for n=num_ite_base:num_ite_base:num_ite_ran
293
294             % initialized forward training
295
296             b_dot=reshape(b(1,:,:),[num_stream,train_leng]);
297             noise_dot=reshape(noise(1,:,:),[Nr,train_leng]);
298
299             % initialized mixing matrix:
300             mix_mat=sqrt(pow)*Gaussian_chan*nor_precoder;
301             % sample correlation matrix

```

```

302     cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
303         *ctranspose(mix_mat*b_dot+noise_dot)/ ...
           train_leng;
304     % sample cross-correlation matrix
305     cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
306         *ctranspose(b_dot)/train_leng;
307     % LS estimated receive filter and power ...
           normalization
308     [rece_fil_ls,~,~]=ls_rr_fil_mse_sinr_cal(mix_mat, ...
           cor_mat_ls,cro_cor_mat_ls,noise_covar,rr,false);
309     % power normalization
310     rece_fil_ls_nor=mat_pow_nor(rece_fil_ls);
311
312     % for bi-directional training with each number ...
           of iteration
313     for n_d=1:n
314
315         % backward training
316
317         b_dot=reshape(b(2*n_d,:,:),[num_stream,train_leng]);
318         noise_dot=reshape(noise(2*n_d,:,:),[Nr,train_leng]);
319
320         % backward mixing matrix
321         mix_mat=transpose(sqrt(pow)*Gaussian_chan)* ...
           conj(rece_fil_ls_nor);
322         % sample correlation matrix
323         cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
324             *ctranspose(mix_mat*b_dot+noise_dot)/ ...
           train_leng;
325         % sample cross-correlation matrix
326         cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
327             *ctranspose(b_dot)/train_leng;
328         % LS estimated precoder
329         [precoder_ls,~,~]=ls_rr_fil_mse_sinr_cal(mix_mat, ...
           cor_mat_ls,cro_cor_mat_ls,noise_covar,rr,false);
330         precoder_ls=conj(precoder_ls);
331         % power normalization
332         precoder_ls_nor=mat_pow_nor(precoder_ls);
333
334         % forward training
335
336         b_dot=reshape(b(2*n_d+1,:,:),[num_stream,train_leng]);
337         noise_dot=reshape(noise(2*n_d+1,:,:),[Nr,train_leng]);
338
339         % forward mixing matrix
340         mix_mat=sqrt(pow)*Gaussian_chan*precoder_ls_nor;
341         % sample correlation matrix
342         cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
343             *ctranspose(mix_mat*b_dot+noise_dot)/ ...

```

```

344         train_leng;
345         % sample cross-correlation matrix
346         cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
347             *ctranspose(b_dot)/train_leng;
348         % LS estimated receive filter
349         [rece_fil_ls,mse_val,sinr_val]= ...
350             ls_rr_fil_mse_sinr_cal(mix_mat,cro_cor_mat_ls, ...
351                 cro_cor_mat_ls,noise_covar,rr,true);
352         % power normalization
353         rece_fil_ls_nor=mat_pow_nor(rece_fil_ls);
354     end
355     % assign MSE and SINR values
356     mse(rr_ind,m,n/num_ite_base)=mse_val;
357     sinr(rr_ind,m,n/num_ite_base)=sinr_val;
358 end
359
360 % average data preallocation
361 mse_ave=zeros(size(red_rank,2),num_ite_ran/num_ite_base);
362 sinr_ave=zeros(size(red_rank,2),num_ite_ran/num_ite_base);
363
364 % averaging & decibel conversion
365 for m=1:size(red_rank,2)
366     mse_ave(m,:)=10*log10(mean(squeeze(mse(m,:,:))));
367     sinr_ave(m,:)=10*log10(mean(squeeze(sinr(m,:,:))));
368 end
369
370 % sample data preallocation
371 mse_sam=zeros(size(red_rank,2),num_ite_ran/num_ite_base);
372 sinr_sam=zeros(size(red_rank,2),num_ite_ran/num_ite_base);
373
374 % extraction & decibel conversion
375 for m=1:size(red_rank,2)
376     mse_sam(m,:)=10*log10(squeeze(mse(m,1,:)));
377     sinr_sam(m,:)=10*log10(squeeze(sinr(m,1,:)));
378 end
379
380 % number of iteration axis
381 num_ite_leng=zeros(1,num_ite_ran/num_ite_base);
382 for m=1:num_ite_ran/num_ite_base
383     num_ite_leng(1,m)=m*num_ite_base;
384 end
385
386 end
387
388
389

```

```

390
391 ls_mse_sinr_vs_trainleng_bidirec_fr.m
392
393 % LS estimated MSE & SINR vs Training length with full-rank
394
395 function [mse_ave,mse_sam,sinr_ave,sinr_sam,train_leng] = ...
    ls_mse_sinr_vs_trainleng_bidirec_fr(pow,Gaussian_chan, ...
    nor_precoder,ave_num,train_leng_ran,train_base,num_ite,noise_covar)
396
397 % parameter extraction
398 [Nr,num_stream]=size(Gaussian_chan*nor_precoder);
399 % output container
400 mse=zeros(ave_num,train_leng_ran);
401 sinr=zeros(ave_num,train_leng_ran);
402
403 % for each trial in averaging
404 for m=1:ave_num
405     % training sequence for each iteration
406     b=bidirec_trainseq_gen(2*num_ite+1,num_stream,train_base* ...
        train_leng_ran);
407     % noise sequence for each iteration
408     noise=bidirec_noiseseq_gen(2*num_ite+1,Nr,train_base* ...
        train_leng_ran);
409
410     % for each training length
411     for n=train_base:train_base:train_base*train_leng_ran
412         % initialized forward training
413         b_dot=reshape(b(1,:),[1:n]),[num_stream,n]);
414         noise_dot=reshape(noise(1,:),[1:n]),[Nr,n]);
415
416         % initialized mixing matrix
417         mix_mat=sqrt(pow)*Gaussian_chan*nor_precoder;
418
419         % sample correlation matrix
420         cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
            *ctranspose(mix_mat*b_dot+noise_dot)/n;
421         % sample cross-correlation matrix
422         cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
            *ctranspose(b_dot)/n;
423         % LS estimated receive filter and power normalization
424         rece_fil_ls=cor_mat_ls\cro_cor_mat_ls;
425         rece_fil_ls_nor=mat_pow_nor(rece_fil_ls);
426
427         % for each number of bi-directional training
428         for n_d=1:num_ite
429             % backward training
430             b_dot=reshape(b(2*n_d,:),[1:n]),[num_stream,n]);
431             noise_dot=reshape(noise(2*n_d,:),[1:n]),[Nr,n]);
432
433
434

```

```

435         % backward mixing matrix
436         mix_mat=transpose(sqrt(pow)*Gaussian_chan)* ...
            conj(rece_fil_ls_nor);
437         % sample correlation matrix
438         cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
            *ctranspose(mix_mat*b_dot+noise_dot)/n;
439         % sample cross-correlation matrix
440         cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
            *ctranspose(b_dot)/n;
441         % LS estimated precoder and power normalization
442         precoder_ls=conj(cor_mat_ls\cro_cor_mat_ls);
443         precoder_ls_nor=mat_pow_nor(precoder_ls);
444
445         % forward training
446         b_dot=reshape(b(2*n_d+1,:), [1:n]), [num_stream,n]);
447         noise_dot=reshape(noise(2*n_d+1,:), [1:n]), [Nr,n]);
448
449         % forward mixing matrix
450         mix_mat=sqrt(pow)*Gaussian_chan*precoder_ls_nor;
451         % sample correlation matrix
452         cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
            *ctranspose(mix_mat*b_dot+noise_dot)/n;
453         % sample cross-correlation matrix
454         cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
            *ctranspose(b_dot)/n;
455         % LS estimated combiner and power normalization
456         rece_fil_ls=cor_mat_ls\cro_cor_mat_ls;
457         rece_fil_ls_nor=mat_pow_nor(rece_fil_ls);
458
459     end
460
461     % statistical correlation matrix
462     cor_mat_op=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
463
464     % MSE & SINR computation
465     mse_temp=zeros(1,num_stream);
466     sinr_temp=zeros(1,num_stream);
467     for n_d=1:num_stream
468
469         % MSE computation
470         mse_temp_val=1-abs(transpose(rece_fil_ls(:,n_d))* ...
            conj(mix_mat(:,n_d))) ...
            -abs(ctranspose(rece_fil_ls(:,n_d))* ...
            mix_mat(:,n_d)) ...
            +abs(ctranspose(rece_fil_ls(:,n_d))* ...
            cor_mat_op*rece_fil_ls(:,n_d));
471         mse_temp(1,n_d)=mse_temp_val;
472
473         % signal power
474         signal_pow=abs(ctranspose(rece_fil_ls(:,n_d))* ...

```



```

mix_mat(:,n_d))^2;
480
481     % interference power
482     interf_pow=0;
483     for n_dd=1:num_stream
484         if n_dd==n_d
485             % pass
486         else
487             interf_pow=interf_pow+abs(ctranspose ...
                (rece_fil_ls(:,n_d))*mix_mat(:,n_dd))^2;
488         end
489     end
490
491     % noise power
492     noise_pow=abs(ctranspose(rece_fil_ls(:,n_d))* ...
        noise_covar*rece_fil_ls(:,n_d));
493
494     % SINR computation
495     sinr_temp_val=signal_pow/(interf_pow+noise_pow);
496     sinr_temp(1,n_d)=sinr_temp_val;
497 end
498
499     % average over all streams
500     mse(m,n/train_base)=mean(mse_temp,2);
501     sinr(m,n/train_base)=mean(sinr_temp,2);
502 end
503 end
504
505 % data averaging, sample data extraction and decibel conversion
506 mse_ave=10*log10(mean(mse));
507 mse_sam=10*log10(mse(1,:));
508 sinr_ave=10*log10(mean(sinr));
509 sinr_sam=10*log10(sinr(1,:));
510
511 % training length axis
512 train_leng=zeros(1,train_leng_ran);
513 for m=1:train_leng_ran
514     train_leng(1,m)=m*train_base;
515 end
516
517 end
518
519
520
521
522 ls_mse_sinr_vs_trainleng_bidirec_rr.m:
523
524 % LS estimated MSE & SINR vs Training length with reduced-rank
525

```

```

526 function [mse_ave,mse_sam,sinr_ave,sinr_sam,train_leng] = ...
    ls_mse_sinr_vs_trainleng_bidirec_rr(pow,Gaussian_chan,nor_precoder, ...
    red_rank,ave_num,train_leng_ran,train_base,num_ite,noise_covar)
527
528 % parameter extraction
529 [Nr,num_stream]=size(Gaussian_chan*nor_precoder);
530 % output container
531 mse=zeros(size(red_rank,2),ave_num,train_leng_ran);
532 sinr=zeros(size(red_rank,2),ave_num,train_leng_ran);
533
534 % for each reduced-rank
535 for rr_ind=1:size(red_rank,2)
536     % assign reduced-rank
537     rr=red_rank(rr_ind);
538
539     % for each trial in averaging
540     for m=1:ave_num
541
542         % training sequence
543         b=bidirec_trainseq_gen(2*num_ite+1,num_stream,train_base ...
            *train_leng_ran);
544         % noise sequence
545         noise=bidirec_noiseseq_gen(2*num_ite+1,Nr,train_base* ...
            train_leng_ran);
546
547         % for each training length
548         for n=train_base:train_base:train_base*train_leng_ran
549
550             % initialized forward training
551             b_dot=reshape(b(1,:),[1:n]),[num_stream,n]);
552             noise_dot=reshape(noise(1,:),[1:n]),[Nr,n]);
553
554             % initialized mixing matrix
555             mix_mat=sqrt(pow)*Gaussian_chan*nor_precoder;
556
557             % sample correlation matrix
558             cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                *ctranspose(mix_mat*b_dot+noise_dot)/n;
559             % sample cross-correlation matrix
560             cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                *ctranspose(b_dot)/n;
561             % reduced-rank LS estimated receive filter
562             [rece_fil,~,~]=ls_rr_fil_mse_sinr_cal(mix_mat, ...
                cor_mat_ls,cro_cor_mat_ls,noise_covar,rr,false);
563             % power normalization
564             rece_fil_nor=mat_pow_nor(rece_fil);
565
566             % for each number of bi-directional training
567             for n_d=1:num_ite

```

```

570
571     % backward training
572     b_dot=reshape(b(2*n_d,:), [1:n]), [num_stream,n]);
573     noise_dot=reshape(noise(2*n_d,:), [1:n]), [Nr,n]);
574
575     % backward mixing matrix
576     mix_mat=transpose(sqrt(pow)*Gaussian_chan)* ...
        conj(rece_fil_nor);
577     % sample correlation matrix
578     cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
        *ctranspose(mix_mat*b_dot+noise_dot)/n;
579
580     % sample cross-correlation matrix
581     cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
        *ctranspose(b_dot)/n;
582
583     % reduced-rank LS estimated precoder
584     [precoder,~,~]=ls_rr_fil_mse_sinr_cal(mix_mat, ...
        cor_mat_ls,cro_cor_mat_ls,noise_covar,rr,false);
585     precoder=conj(precoder);
586     % power normalization
587     precoder_nor=mat_pow_nor(precoder);
588
589     % forward training
590     b_dot=reshape(b(2*n_d+1,:), [1:n]), [num_stream,n]);
591     noise_dot=reshape(noise(2*n_d+1,:), [1:n]), [Nr,n]);
592
593     % forward mixing matrix
594     mix_mat=sqrt(pow)*Gaussian_chan*precoder_nor;
595     % sample correlation matrix
596     cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
        *ctranspose(mix_mat*b_dot+noise_dot)/n;
597
598     % sample cross-correlation matrix
599     cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
        *ctranspose(b_dot)/n;
600
601     % reduced-rank LS estimated receive filter
602     [rece_fil,mse_val,sinr_val]=ls_rr_fil_mse_sinr_cal ...
        (mix_mat,cor_mat_ls,cro_cor_mat_ls,noise_covar,rr, ...
        true);
603
604     % power normalization
605     rece_fil_nor=mat_pow_nor(rece_fil);
606
607     end
608
609     mse(rr_ind,m,n/train_base)=mse_val;
610     sinr(rr_ind,m,n/train_base)=sinr_val;
611
612     end
613
614     % average data preallocation
615     mse_ave=zeros(size(red_rank,2),train_leng_ran);

```

```

615 sinr_ave=zeros(size(red_rank,2),train_leng_ran);
616
617 % averaging & decibel conversion
618 for m=1:size(red_rank,2)
619     mse_ave(m,:)=10*log10(mean(squeeze(mse(m,:,:))));
620     sinr_ave(m,:)=10*log10(mean(squeeze(sinr(m,:,:))));
621 end
622
623 % sample data preallocation
624 mse_sam=zeros(size(red_rank,2),train_leng_ran);
625 sinr_sam=zeros(size(red_rank,2),train_leng_ran);
626
627 % extraction & decibel conversion
628 for m=1:size(red_rank,2)
629     mse_sam(m,:)=10*log10(squeeze(mse(m,1,:)));
630     sinr_sam(m,:)=10*log10(squeeze(sinr(m,1,:)));
631 end
632
633 % training length axis
634 train_leng=zeros(1,train_leng_ran);
635 for m=1:train_leng_ran
636     train_leng(1,m)=m*train_base;
637 end
638
639 end
640
641
642
643
644 ls_mse_sinr_vs_trainleng_fr.m:
645
646 % LS estimated MSE & SINR vs Training length with full-rank
647
648 function [mse_ave,mse_sam,sinr_ave,sinr_sam,train_leng] = ...
        ls_mse_sinr_vs_trainleng_fr(mix_mat,ave_num,train_leng_ran, ...
        train_base,noise_covar)
649
650 % parameter extraction
651 [Nr,num_stream]=size(mix_mat);
652
653 % statistical correlation matrix
654 cor_mat_op=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
655
656 % output container
657 mse=zeros(ave_num,train_leng_ran);
658 sinr=zeros(ave_num,train_leng_ran);
659
660 % for each trial
661 for m=1:ave_num

```

```

662 % training sequence
663 b=bpsk_mat_gen(num_stream,train_base*train_leng_ran);
664 % Gaussian additive noise
665 noise=noise_mat_gen(Nr,train_base*train_leng_ran);
666
667 % for each training length
668 for n=train_base:train_base:train_base*train_leng_ran
669
670     % sample correlation matrix
671     cor_mat_ls=(mix_mat*b(:, [1:n])+noise(:, [1:n]))* ...
        ctranspose(mix_mat*b(:, [1:n])+noise(:, [1:n]))/n;
672     % sample cross-correlation matrix
673     cro_cor_mat_ls=(mix_mat*b(:, [1:n])+noise(:, [1:n]))* ...
        ctranspose(b(:, [1:n]))/n;
674     % LS estimated receive filter
675     rece_fil_ls_fr=cor_mat_ls\cro_cor_mat_ls;
676
677     mse_temp=zeros(1,num_stream);
678     sinr_temp=zeros(1,num_stream);
679     for n_d=1:num_stream
680         % MSE computation
681         mse_temp_val=1-abs(transpose(rece_fil_ls_fr(:,n_d)) ...
            *conj(mix_mat(:,n_d))) ...
            -abs(ctranspose(rece_fil_ls_fr(:,n_d)) ...
            *mix_mat(:,n_d)) ...
            +abs(ctranspose(rece_fil_ls_fr(:,n_d)) ...
            *cor_mat_op*rece_fil_ls_fr(:,n_d));
682
683         mse_temp(1,n_d)=mse_temp_val;
684
685         % signal power
686         signal_pow=abs(ctranspose(rece_fil_ls_fr(:,n_d))* ...
            mix_mat(:,n_d))^2;
687
688         % interference power
689         interf_pow=0;
690         for n_dd=1:num_stream
691             if n_dd==n_d
692                 % pass
693             else
694                 interf_pow=interf_pow+abs(ctranspose ...
                    (rece_fil_ls_fr(:,n_d))*mix_mat(:,n_dd))^2;
695             end
696         end
697     end
698
699     % noise power
700     noise_pow=abs(ctranspose(rece_fil_ls_fr(:,n_d))* ...
        noise_covar*rece_fil_ls_fr(:,n_d));
701
702     % SINR computation

```

```

703         sinr_temp_val=signal_pow/(interf_pow+noise_pow);
704         sinr_temp(1,n_d)=sinr_temp_val;
705     end
706     % average over all streams
707     mse(m,n/train_base)=mean(mse_temp,2);
708     sinr(m,n/train_base)=mean(sinr_temp,2);
709 end
710 end
711
712 % averaging over trials & decibel conversion
713 mse_ave=10*log10(mean(mse));
714 sinr_ave=10*log10(mean(sinr));
715 % sample extracting & decibel conversion
716 mse_sam=10*log10(mse(1,:));
717 sinr_sam=10*log10(sinr(1,:));
718
719 % training length axis
720 train_leng=zeros(1,train_leng_ran);
721 for m=1:train_leng_ran
722     train_leng(1,m)=m*train_base;
723 end
724
725 end
726
727
728
729
730 ls_mse_sinr_vs_trainleng_rr_un.m:
731
732 % LS estimated MSE & SINR vs Training length with ...
733     reduced-rank and time-averaging Krylov subspace
734
735 function [mse_ave,mse_sam,sinr_ave,sinr_sam,train_leng] = ...
736     ls_mse_sinr_vs_trainleng_rr_un(mix_mat,red_rank,ave_num, ...
737     train_leng_ran,train_base,noise_covar)
738
739 % parameter extraction
740 [Nr,num_stream]=size(mix_mat);
741
742 % statistical correlation matrix
743 cor_mat_op=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
744
745 mse=zeros(size(red_rank,2),ave_num,train_leng_ran);
746 sinr=zeros(size(red_rank,2),ave_num,train_leng_ran);
747
748 % for each reduced-rank
749 for rr_ind=1:size(red_rank,2)
750     % assign reduced-rank
751     rr=red_rank(rr_ind);

```

```

749
750 % Krylov subspace preallocation
751 krylov=zeros(Nr,rr);
752
753 % for each trial
754 for m=1:ave_num
755
756     % training sequence
757     b=bpsk_mat_gen(num_stream,train_base*train_leng_ran);
758     % Gaussian additive noise
759     noise=noise_mat_gen(Nr,train_base*train_leng_ran);
760
761     % for each training length
762     for n=train_base:train_base:train_base*train_leng_ran
763
764         % sample correlation matrix
765         cor_mat_ls=(mix_mat*b(:,[1:n])+noise(:,[1:n]))* ...
            ctranspose(mix_mat*b(:,[1:n])+noise(:,[1:n]))/n;
766
767         % sample cross-correlation matrix
768         cro_cor_mat_ls=(mix_mat*b(:,[1:n])+noise(:,[1:n]))* ...
            ctranspose(b(:,[1:n]))/n;
769
770         mse_temp=zeros(1,num_stream);
771         sinr_temp=zeros(1,num_stream);
772
773         for n_d=1:num_stream
774             % Krylov subspace
775             for n_dot=1:rr
776                 % Krylov subspace
777                 krylov(:,n_dot)=cor_mat_ls^(n_dot-1)* ...
                    cro_cor_mat_ls(:,n_d);
778             end
779
780             rece_fil_ls=krylov*((ctranspose(krylov)*cor_mat_ls ...
                *krylov) ...
                \ctranspose(krylov))*cro_cor_mat_ls ...
                (:,n_d);
781
782             mse_temp_val=1-abs(transpose(rece_fil_ls)* ...
                conj(mix_mat(:,n_d))) ...
                -abs(ctranspose(rece_fil_ls)* ...
                    mix_mat(:,n_d)) ...
                +abs(ctranspose(rece_fil_ls)* ...
                    cor_mat_op*rece_fil_ls);
783
784             mse_temp(1,n_d)=mse_temp_val;
785
786             % signal power
787
788
789

```

```

790         signal_pow=abs(ctranspose(rece_fil_ls)* ...
            mix_mat(:,n_d))^2;
791
792         % interference power
793         interf_pow=0;
794         for n_dd=1:num_stream
795             if n_dd==n_d
796                 % pass
797             else
798                 interf_pow=interf_pow+abs ...
                    (ctranspose(rece_fil_ls)*mix_mat(:,n_dd))^2;
799             end
800         end
801
802         % noise power
803         noise_pow=abs(ctranspose(rece_fil_ls)* ...
            noise_covar*rece_fil_ls);
804
805         sinr_temp_val=signal_pow/(interf_pow+noise_pow);
806
807         sinr_temp(1,n_d)=sinr_temp_val;
808     end
809     % average over streams
810     mse_temp=mean(mse_temp,2);
811     sinr_temp=mean(sinr_temp,2);
812
813     mse(rr_ind,m,n/train_base)=mse_temp;
814     sinr(rr_ind,m,n/train_base)=sinr_temp;
815 end
816 end
817 end
818
819 % average data preallocation
820 mse_ave=zeros(size(red_rank,2),train_leng_ran);
821 sinr_ave=zeros(size(red_rank,2),train_leng_ran);
822
823 % averaging & decibel conversion
824 for m=1:size(red_rank,2)
825     mse_ave(m,:)=10*log10(mean(squeeze(mse(m,:,:))));
826     sinr_ave(m,:)=10*log10(mean(squeeze(sinr(m,:,:))));
827 end
828
829 % sample data preallocation
830 mse_sam=zeros(size(red_rank,2),train_leng_ran);
831 sinr_sam=zeros(size(red_rank,2),train_leng_ran);
832
833 % extraction & decibel conversion
834 for m=1:size(red_rank,2)
835     mse_sam(m,:)=10*log10(squeeze(mse(m,1,:)));

```



```

836     sinr_sam(m,:)=10*log10(squeeze(sinr(m,1,:)));
837 end
838
839 % training length axis
840 train_leng=zeros(1,train_leng_ran);
841 for m=1:train_leng_ran
842     train_leng(1,m)=m*train_base;
843 end
844
845 end
846
847
848
849
850 ls_rr_fil_mse_sinr_cal.m:
851
852 % LS estimated reduced-rank precoder & combiner, MSE and ...
    SINR calculator
853
854 function [fil,mse,sinr] = ...
    ls_rr_fil_mse_sinr_cal(mix_mat,cor_mat,cro_cor_mat,noise_covar,rr, ...
    flag)
855
856 % dimension extraction
857 [Nr,num_stream]=size(mix_mat);
858 % Krylov subspace preallocation
859 krylov=zeros(Nr,rr);
860 % filter preallocation
861 fil=zeros(Nr,num_stream);
862
863 % statistical correlation matrix
864 cor_mat_op=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
865
866 % for each transmitted stream, for computing filters ...
    (precoders or combiners)
867 for m=1:num_stream
868
869     % constructe Krylov subspace for the current transmitted ...
        stream
870     for n=1:rr
871         krylov(:,n)=cor_mat^(n-1)*cro_cor_mat(:,m);
872     end
873
874     % reduced-rank filter
875     fil(:,m)=krylov*((ctranspose(krylov)*cor_mat*krylov) ...
876         \ctranspose(krylov))*cro_cor_mat(:,m);
877 end
878
879 % need to compute MSE & SINR

```

```

880 if flag==true
881
882     mse_temp=zeros(1,num_stream);
883     sinr_temp=zeros(1,num_stream);
884
885     % for each transmitted stream, for computing MSE and SINR
886     for m=1:num_stream
887         % MSE
888         mse_temp_val=1-abs(transpose(fil(:,m))*conj(mix_mat(:,m))) ...
            ...
            -abs(ctranspose(fil(:,m))*mix_mat(:,m)) ...
            ...
            +abs(ctranspose(fil(:,m))*cor_mat_op ...
            *fil(:,m));
889         mse_temp(1,m)=mse_temp_val;
890
891         % signal power
892         signal_pow=abs(ctranspose(fil(:,m))*mix_mat(:,m))^2;
893
894         % interference power
895         interf_pow=0;
896         for m_dot=1:num_stream
897             if m_dot==m
898                 % pass
899             else
900                 interf_pow=interf_pow+abs(ctranspose(fil(:,m)) ...
                    *mix_mat(:,m_dot))^2;
901             end
902         end
903
904         % noise power
905         noise_pow=abs(ctranspose(fil(:,m))*noise_covar*fil(:,m));
906
907         % SINR
908         sinr_temp_val=signal_pow/(interf_pow+noise_pow);
909         sinr_temp(1,m)=sinr_temp_val;
910     end
911
912     % averaging over all transmitted streams
913     mse=mean(mse_temp,2);
914     sinr=mean(sinr_temp,2);
915
916 % no need to compute MSE & SINR
917 else
918     mse=0;
919     sinr=0;
920 end
921
922 end
923
924 end

```

```

925
926
927
928
929 ls_rr_fil_sumrate_in_cal.m:
930
931 % Interference network LS reduced-rank filter, sum rate ...
    calculator
932
933 function [fil1,fil2,fil3,sumrate] = ...
    ls_rr_fil_sumrate_in_cal(mix_mat11,mix_mat21,mix_mat31,cor_mat1, ...
    cro_cor_mat1, mix_mat22,mix_mat12,mix_mat32, ...
    cor_mat2,cro_cor_mat2, mix_mat33,mix_mat13,mix_mat23, ...
    cor_mat3,cro_cor_mat3,noise_covar, rr,flag)
934 % dimension extraction
935 [Nr,num_stream]=size(mix_mat11);
936 % Krylov subspace preallocation
937 krylov1=zeros(Nr,rr);
938 krylov2=zeros(Nr,rr);
939 krylov3=zeros(Nr,rr);
940 % filter preallocation
941 fil1=zeros(Nr,num_stream);
942 fil2=zeros(Nr,num_stream);
943 fil3=zeros(Nr,num_stream);
944
945 % for each transmitted stream, for computing filters
946 for m=1:num_stream
947
948     % constructe Krylov subspace for the current transmitted ...
        stream
949     for n=1:rr
950         krylov1(:,n)=cor_mat1^(n-1)*cro_cor_mat1(:,m);
951         krylov2(:,n)=cor_mat2^(n-1)*cro_cor_mat2(:,m);
952         krylov3(:,n)=cor_mat3^(n-1)*cro_cor_mat3(:,m);
953     end
954
955     % reduced-rank filters
956     fil1(:,m)=krylov1*((ctranspose(krylov1)*cor_mat1*krylov1) ...
        ...
957         \ctranspose(krylov1))*cro_cor_mat1(:,m);
958     fil2(:,m)=krylov2*((ctranspose(krylov2)*cor_mat2*krylov2) ...
        ...
959         \ctranspose(krylov2))*cro_cor_mat2(:,m);
960     fil3(:,m)=krylov3*((ctranspose(krylov3)*cor_mat3*krylov3) ...
        ...
961         \ctranspose(krylov3))*cro_cor_mat3(:,m);
962 end
963
964 % need to compute sum rate

```

```

965 if flag==true
966
967     sumrate1=zeros(1,num_stream);
968     sumrate2=zeros(1,num_stream);
969     sumrate3=zeros(1,num_stream);
970
971     % for each transmitted stream, for computing MSE and SINR
972     for m=1:num_stream
973
974         % signal power
975         signal_pow1=abs(ctranspose(fill(:,m))*mix_mat11(:,m))^2;
976         signal_pow2=abs(ctranspose(fil2(:,m))*mix_mat22(:,m))^2;
977         signal_pow3=abs(ctranspose(fil3(:,m))*mix_mat33(:,m))^2;
978
979         % interference power
980         interf_pow1=0;
981         interf_pow2=0;
982         interf_pow3=0;
983         % interference from its own transmitter
984         for m_dot=1:num_stream
985             if m_dot==m
986                 % pass
987             else
988                 interf_pow1=interf_pow1+abs(ctranspose(fill(:,m)) ...
989                     *mix_mat11(:,m_dot))^2;
990                 interf_pow2=interf_pow2+abs(ctranspose(fil2(:,m)) ...
991                     *mix_mat22(:,m_dot))^2;
992                 interf_pow3=interf_pow3+abs(ctranspose(fil3(:,m)) ...
993                     *mix_mat33(:,m_dot))^2;
994             end
995         end
996
997         % interference from other transmitters
998         for m_dot=1:num_stream
999             interf_pow1=interf_pow1+abs(ctranspose(fill(:,m)) ...
1000                 *mix_mat21(:,m_dot))^2 ...
1001                 +abs(ctranspose(fill(:,m)) ...
1002                     *mix_mat31(:,m_dot))^2;
1003             interf_pow2=interf_pow2+abs(ctranspose(fil2(:,m)) ...
1004                 *mix_mat12(:,m_dot))^2 ...
1005                 +abs(ctranspose(fil2(:,m)) ...
1006                     *mix_mat32(:,m_dot))^2;
1007             interf_pow3=interf_pow3+abs(ctranspose(fil3(:,m)) ...
1008                 *mix_mat13(:,m_dot))^2 ...
1009                 +abs(ctranspose(fil3(:,m)) ...
1010                     *mix_mat23(:,m_dot))^2;
1011         end
1012
1013         % noise power

```

```

1005         noise_pow1=abs(ctranspose(fil1(:,m))*noise_covar*fil1(:,m));
1006         noise_pow2=abs(ctranspose(fil2(:,m))*noise_covar*fil2(:,m));
1007         noise_pow3=abs(ctranspose(fil3(:,m))*noise_covar*fil3(:,m));
1008
1009         % sum rate
1010         sumrate1(1,m)=log2(1+signal_pow1/(interf_pow1+noise_pow1));
1011         sumrate2(1,m)=log2(1+signal_pow2/(interf_pow2+noise_pow2));
1012         sumrate3(1,m)=log2(1+signal_pow3/(interf_pow3+noise_pow3));
1013     end
1014
1015     % network sum rate
1016     sumrate=sum(sumrate1)+sum(sumrate2)+sum(sumrate3);
1017
1018     % no need to compute sum rate
1019 else
1020     sumrate=0;
1021 end
1022
1023 end
1024
1025
1026
1027
1028 ls_sumrate_vs_numite_bidirec_in_fr.m:
1029
1030 % LS estimated Sum rate vs Number of bi-directional training ...
1031 % iteration with full-rank
1032 % in interference network
1033 function [sumrate_ave,sumrate_sam,num_ite_leng] = ...
1034     ls_sumrate_vs_numite_bidirec_in_fr(pow,Gaussian_chan_mat, ...
1035     nor_precoder_mat,ave_num,train_leng,num_ite_ran,num_ite_base, ...
1036     noise_covar)
1037
1038 % parameter extraction
1039 N=size(Gaussian_chan_mat,1);
1040 Nr=size(squeeze(Gaussian_chan_mat(1,1,:,:)),1);
1041 num_stream=size(squeeze(nor_precoder_mat(1,:,:)),2);
1042 % temporary container
1043 sumrate1=zeros(1,num_stream);
1044 sumrate2=zeros(1,num_stream);
1045 sumrate3=zeros(1,num_stream);
1046 % output container
1047 sumrate=zeros(ave_num,num_ite_ran/num_ite_base);
1048
1049 % for each trial in averaging
1050 for m=1:ave_num
1051     % training sequence
1052     b=in_bidirec_trainseq_gen(N,2*num_ite_ran+1,num_stream, ...

```

```

train_leng);
1050 % noise sequence
1051 noise=in_bidirec.noiseseq_gen(N,2*num_ite_ran+1,Nr,train_leng);
1052
1053 % for each number of bi-directional training iteration
1054 for n=num_ite_base:num_ite_base:num_ite_ran
1055     % initialized forward training
1056     b1_dot=reshape(b(1,1,:,:),[num_stream,train_leng]);
1057     b2_dot=reshape(b(2,1,:,:),[num_stream,train_leng]);
1058     b3_dot=reshape(b(3,1,:,:),[num_stream,train_leng]);
1059     noise1_dot=reshape(noise(1,1,:,:),[Nr,train_leng]);
1060     noise2_dot=reshape(noise(2,1,:,:),[Nr,train_leng]);
1061     noise3_dot=reshape(noise(3,1,:,:),[Nr,train_leng]);
1062
1063     % receive filter updating, precoder --> receive filter
1064     % initialized mixing matrices
1065     mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:)) ...
        *squeeze(nor_precoder_mat(1,:,:));
1066     mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:)) ...
        *squeeze(nor_precoder_mat(2,:,:));
1067     mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:)) ...
        *squeeze(nor_precoder_mat(3,:,:));
1068     mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:)) ...
        *squeeze(nor_precoder_mat(1,:,:));
1069     mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
        *squeeze(nor_precoder_mat(2,:,:));
1070     mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
        *squeeze(nor_precoder_mat(3,:,:));
1071     mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
        *squeeze(nor_precoder_mat(1,:,:));
1072     mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
        *squeeze(nor_precoder_mat(2,:,:));
1073     mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
        *squeeze(nor_precoder_mat(3,:,:));
1074     % sample correlation matrices
1075     cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+mix_mat31* ...
        b3_dot+noise1_dot) ...
1076         *ctranspose(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
        mix_mat31*b3_dot+noise1_dot)/train_leng;
1077     cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+mix_mat32* ...
        b3_dot+noise2_dot) ...
1078         *ctranspose(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
        mix_mat32*b3_dot+noise2_dot)/train_leng;
1079     cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+mix_mat33* ...
        b3_dot+noise3_dot) ...
1080         *ctranspose(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
        mix_mat33*b3_dot+noise3_dot)/train_leng;
1081     % sample cross-correlation matrices
1082     cro_cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...

```

```

1083         mix_mat31*b3_dot+noise1_dot) ...
1084         *ctranspose(b1_dot)/train_leng;
1085 cro_cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
1086         mix_mat32*b3_dot+noise2_dot) ...
1087         *ctranspose(b2_dot)/train_leng;
1088 cro_cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
1089         mix_mat33*b3_dot+noise3_dot) ...
1090         *ctranspose(b3_dot)/train_leng;
1091 % LS estimated receive filters
1092 rece_fil1=cor_mat1\cro_cor_mat1;
1093 rece_fil2=cor_mat2\cro_cor_mat2;
1094 rece_fil3=cor_mat3\cro_cor_mat3;
1095 % power normalization
1096 rece_fil1_nor=mat_pow_nor(rece_fil1);
1097 rece_fil2_nor=mat_pow_nor(rece_fil2);
1098 rece_fil3_nor=mat_pow_nor(rece_fil3);
1099
1100 % for bi-directional training with each number of ...
1101 iteration
1102 for n_d=1:n
1103     % backward training
1104     b1_dot=reshape(b(1,2*n_d, :, :), [num_stream, train_leng]);
1105     b2_dot=reshape(b(2,2*n_d, :, :), [num_stream, train_leng]);
1106     b3_dot=reshape(b(3,2*n_d, :, :), [num_stream, train_leng]);
1107     noise1_dot=reshape(noise(1,2*n_d, :, :), [Nr, train_leng]);
1108     noise2_dot=reshape(noise(2,2*n_d, :, :), [Nr, train_leng]);
1109     noise3_dot=reshape(noise(3,2*n_d, :, :), [Nr, train_leng]);
1110
1111     % backward mixing matrices
1112     % precoder updating, precoder <-- receive filter
1113     mix_mat11=sqrt(pow)*transpose(squeeze ...
1114         (Gaussian_chan_mat(1,1, :, :)))*conj(rece_fil1_nor);
1115     mix_mat12=sqrt(pow)*transpose(squeeze ...
1116         (Gaussian_chan_mat(2,1, :, :)))*conj(rece_fil2_nor);
1117     mix_mat13=sqrt(pow)*transpose(squeeze ...
1118         (Gaussian_chan_mat(3,1, :, :)))*conj(rece_fil3_nor);
1119     mix_mat21=sqrt(pow)*transpose(squeeze ...
1120         (Gaussian_chan_mat(1,2, :, :)))*conj(rece_fil1_nor);
1121     mix_mat22=sqrt(pow)*transpose(squeeze ...
1122         (Gaussian_chan_mat(2,2, :, :)))*conj(rece_fil2_nor);
1123     mix_mat23=sqrt(pow)*transpose(squeeze ...
1124         (Gaussian_chan_mat(3,2, :, :)))*conj(rece_fil3_nor);
1125     mix_mat31=sqrt(pow)*transpose(squeeze ...
1126         (Gaussian_chan_mat(1,3, :, :)))*conj(rece_fil1_nor);
1127     mix_mat32=sqrt(pow)*transpose(squeeze ...
1128         (Gaussian_chan_mat(2,3, :, :)))*conj(rece_fil2_nor);
1129     mix_mat33=sqrt(pow)*transpose(squeeze ...
1130         (Gaussian_chan_mat(3,3, :, :)))*conj(rece_fil3_nor);
1131     % sample correlation matrices

```

```

1119     cor_mat1=(mix_mat11*b1_dot+mix_mat12*b2_dot+ ...
1120             mix_mat13*b3_dot+noise1_dot) ...
1121             *ctranspose(mix_mat11*b1_dot+ ...
1122             mix_mat12*b2_dot+mix_mat13*b3_dot+noise1_dot) ...
1123             /train_leng;
1124     cor_mat2=(mix_mat21*b1_dot+mix_mat22*b2_dot+ ...
1125             mix_mat23*b3_dot+noise2_dot) ...
1126             *ctranspose(mix_mat21*b1_dot+ ...
1127             mix_mat22*b2_dot+mix_mat23*b3_dot+noise2_dot) ...
1128             /train_leng;
1129     cor_mat3=(mix_mat31*b1_dot+mix_mat32*b2_dot+ ...
1130             mix_mat33*b3_dot+noise3_dot) ...
1131             *ctranspose(mix_mat31*b1_dot+mix_mat32* ...
1132             b2_dot+mix_mat33*b3_dot+noise3_dot) ...
1133             /train_leng;
1134     % sample cross-correlation matrices
1135     cro_cor_mat1=(mix_mat11*b1_dot+mix_mat12*b2_dot+ ...
1136             mix_mat13*b3_dot+noise1_dot) ...
1137             *ctranspose(b1_dot)/train_leng;
1138     cro_cor_mat2=(mix_mat21*b1_dot+mix_mat22*b2_dot+ ...
1139             mix_mat23*b3_dot+noise2_dot) ...
1140             *ctranspose(b2_dot)/train_leng;
1141     cro_cor_mat3=(mix_mat31*b1_dot+mix_mat32*b2_dot+ ...
1142             mix_mat33*b3_dot+noise3_dot) ...
1143             *ctranspose(b3_dot)/train_leng;
1144     % LS estimated precoders and power normalization
1145     precoder1=mat_pow_nor(conj(cor_mat1\cro_cor_mat1));
1146     precoder2=mat_pow_nor(conj(cor_mat2\cro_cor_mat2));
1147     precoder3=mat_pow_nor(conj(cor_mat3\cro_cor_mat3));
1148
1149     % forward training
1150     b1_dot=reshape(b(1,2*n_d+1,:),[num_stream,train_leng]);
1151     b2_dot=reshape(b(2,2*n_d+1,:),[num_stream,train_leng]);
1152     b3_dot=reshape(b(3,2*n_d+1,:),[num_stream,train_leng]);
1153     noise1_dot=reshape(noise(1,2*n_d+1,:),[Nr,train_leng]);
1154     noise2_dot=reshape(noise(2,2*n_d+1,:),[Nr,train_leng]);
1155     noise3_dot=reshape(noise(3,2*n_d+1,:),[Nr,train_leng]);
1156
1157     % forward mixing matrices
1158     % receive filter updating, precoder --> receive ...
1159     filter
1160     mix_mat11=sqrt(pow)*squeeze(Gaussian_chan.mat(1,1,:,:)) ...
1161             *precoder1;
1162     mix_mat21=sqrt(pow)*squeeze(Gaussian_chan.mat(2,1,:,:)) ...
1163             *precoder2;
1164     mix_mat31=sqrt(pow)*squeeze(Gaussian_chan.mat(3,1,:,:)) ...
1165             *precoder3;
1166     mix_mat12=sqrt(pow)*squeeze(Gaussian_chan.mat(1,2,:,:)) ...
1167             *precoder1;

```



```

1151     mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
        *precoder2;
1152     mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
        *precoder3;
1153     mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
        *precoder1;
1154     mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
        *precoder2;
1155     mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
        *precoder3;
1156     % sample correlation matrices
1157     cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+mix_mat31* ...
        b3_dot+noise1_dot) ...
1158         *ctranspose(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
        mix_mat31*b3_dot+noise1_dot)/train_leng;
1159     cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+mix_mat32* ...
        b3_dot+noise2_dot) ...
1160         *ctranspose(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
        mix_mat32*b3_dot+noise2_dot)/train_leng;
1161     cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+mix_mat33* ...
        b3_dot+noise3_dot) ...
1162         *ctranspose(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
        mix_mat33*b3_dot+noise3_dot)/train_leng;
1163     % sample cross-correlation matrices
1164     cro_cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
        mix_mat31*b3_dot+noise1_dot) ...
1165         *ctranspose(b1_dot)/train_leng;
1166     cro_cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
        mix_mat32*b3_dot+noise2_dot) ...
1167         *ctranspose(b2_dot)/train_leng;
1168     cro_cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
        mix_mat33*b3_dot+noise3_dot) ...
1169         *ctranspose(b3_dot)/train_leng;
1170     % LS estimated receive filters
1171     rece_fil1=cor_mat1\cro_cor_mat1;
1172     rece_fil2=cor_mat2\cro_cor_mat2;
1173     rece_fil3=cor_mat3\cro_cor_mat3;
1174     % power normalization
1175     rece_fil1_nor=mat_pow_nor(rece_fil1);
1176     rece_fil2_nor=mat_pow_nor(rece_fil2);
1177     rece_fil3_nor=mat_pow_nor(rece_fil3);
1178     end
1179
1180     % for each stream
1181     for n_d=1:num_stream
1182
1183         % signal power
1184         signal_pow1=abs(ctranspose(rece_fil1(:,n_d))* ...
        mix_mat11(:,n_d))^2;

```

```

1185     signal_pow2=abs(ctranspose(rece_fil2(:,n_d))* ...
        mix_mat22(:,n_d))^2;
1186     signal_pow3=abs(ctranspose(rece_fil3(:,n_d))* ...
        mix_mat33(:,n_d))^2;

1187
1188     % interference power
1189     interf_pow1=0;
1190     interf_pow2=0;
1191     interf_pow3=0;
1192     % interference from its own transmitter
1193     for n_dd=1:num_stream
1194         if n_dd==n_d
1195             % pass
1196         else
1197             interf_pow1=interf_pow1+abs(ctranspose ...
                (rece_fil1(:,n_d))*mix_mat11(:,n_dd))^2;
1198             interf_pow2=interf_pow2+abs(ctranspose ...
                (rece_fil2(:,n_d))*mix_mat22(:,n_dd))^2;
1199             interf_pow3=interf_pow3+abs(ctranspose ...
                (rece_fil3(:,n_d))*mix_mat33(:,n_dd))^2;
1200         end
1201     end
1202
1203     % interference from other transmitters
1204     for n_dd=1:num_stream
1205         interf_pow1=interf_pow1+abs(ctranspose ...
            (rece_fil1(:,n_d))*mix_mat21(:,n_dd))^2 ...
            +abs(ctranspose(rece_fil1 ...
                (:,n_d))*mix_mat31(:,n_dd))^2;
1206         interf_pow2=interf_pow2+abs(ctranspose ...
            (rece_fil2(:,n_d))*mix_mat12(:,n_dd))^2 ...
            +abs(ctranspose(rece_fil2 ...
                (:,n_d))*mix_mat32(:,n_dd))^2;
1207         interf_pow3=interf_pow3+abs(ctranspose ...
            (rece_fil3(:,n_d))*mix_mat13(:,n_dd))^2 ...
            +abs(ctranspose(rece_fil3 ...
                (:,n_d))*mix_mat23(:,n_dd))^2;
1208     end
1209
1210     % noise power
1211     noise_pow1=abs(ctranspose(rece_fil1(:,n_d))* ...
        noise_covar*rece_fil1(:,n_d));
1212     noise_pow2=abs(ctranspose(rece_fil2(:,n_d))* ...
        noise_covar*rece_fil2(:,n_d));
1213     noise_pow3=abs(ctranspose(rece_fil3(:,n_d))* ...
        noise_covar*rece_fil3(:,n_d));
1214
1215     % sum rate computation
1216     sumrate1(1,n_d)=log2(1+signal_pow1/(interf_pow1+ ...

```

```

        noise_pow1));
1217     sumrate2(1,n_d)=log2(1+signal_pow2/(interf_pow2+ ...
        noise_pow2));
1218     sumrate3(1,n_d)=log2(1+signal_pow3/(interf_pow3+ ...
        noise_pow3));
1219     end
1220
1221     % network sum rate
1222     sumrate(m,n/num_ite_base)=sum(sumrate1)+sum(sumrate2)+ ...
        sum(sumrate3);
1223     end
1224 end
1225
1226 % data averaging and extracting
1227 sumrate_ave=mean(sumrate);
1228 sumrate_sam=sumrate(1,:);
1229
1230 % number of iteration axis
1231 num_ite_leng=zeros(1,num_ite_ran/num_ite_base);
1232 for m=1:num_ite_ran/num_ite_base
1233     num_ite_leng(1,m)=m*num_ite_base;
1234 end
1235
1236 end
1237
1238
1239
1240
1241 ls_sumrate_vs_numite_bidirec_in_rr.m:
1242
1243 % LS estimated Sum rate vs Number of bi-directional training ...
    iteration with reduced-rank
1244 % in interference network
1245
1246 function [sumrate_ave,sumrate_sam,num_ite_leng] = ...
    ls_sumrate_vs_numite_bidirec_in_rr(pow,Gaussian_chan_mat, ...
    nor_precoder_mat,red_rank,ave_num,train_leng,num_ite_ran, ...
    num_ite_base,noise_covar)
1247
1248 % parameter extraction
1249 N=size(Gaussian_chan_mat,1);
1250 Nr=size(squeeze(Gaussian_chan_mat(1,1,:,:)),1);
1251 num_stream=size(squeeze(nor_precoder_mat(1,:,:)),2);
1252 % temporary container
1253 sumrate=zeros(size(red_rank,2),ave_num,num_ite_ran/num_ite_base);
1254 % output data
1255 sumrate_ave=zeros(size(red_rank,2),num_ite_ran/num_ite_base);
1256 sumrate_sam=zeros(size(red_rank,2),num_ite_ran/num_ite_base);
1257

```

```

1258 % for every reduced-rank
1259 for rr_ind=1:size(red_rank,2)
1260     % assign current reduced-rank
1261     rr=red_rank(rr_ind);
1262
1263     % for each trial in averaging
1264     for m=1:ave_num
1265         % training sequence
1266         b=in_bidirec_trainseq_gen(N,2*num_ite_ran+1,num_stream, ...
            train_leng);
1267         % noise sequence
1268         noise=in_bidirec_noiseseq_gen(N,2*num_ite_ran+1,Nr, ...
            train_leng);
1269
1270         % for each number of bi-directional training iteration
1271         for n=num_ite_base:num_ite_base:num_ite_ran
1272
1273             % initialized forward training
1274             % receive filter updating, precoder --> receive ...
            filter
1275             b1_dot=reshape(b(1,1,:,:),[num_stream,train_leng]);
1276             b2_dot=reshape(b(2,1,:,:),[num_stream,train_leng]);
1277             b3_dot=reshape(b(3,1,:,:),[num_stream,train_leng]);
1278             noise1_dot=reshape(noise(1,1,:,:),[Nr,train_leng]);
1279             noise2_dot=reshape(noise(2,1,:,:),[Nr,train_leng]);
1280             noise3_dot=reshape(noise(3,1,:,:),[Nr,train_leng]);
1281
1282             % initialized mixing matrices
1283             mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:)) ...
            *squeeze(nor_precoder_mat(1,:,:));
1284             mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:)) ...
            *squeeze(nor_precoder_mat(2,:,:));
1285             mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:)) ...
            *squeeze(nor_precoder_mat(3,:,:));
1286             mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:)) ...
            *squeeze(nor_precoder_mat(1,:,:));
1287             mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
            *squeeze(nor_precoder_mat(2,:,:));
1288             mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
            *squeeze(nor_precoder_mat(3,:,:));
1289             mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
            *squeeze(nor_precoder_mat(1,:,:));
1290             mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
            *squeeze(nor_precoder_mat(2,:,:));
1291             mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
            *squeeze(nor_precoder_mat(3,:,:));
1292             % sample correlation matrices
1293             cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
            mix_mat31*b3_dot+noise1_dot) ...

```

```

1294         *ctranspose(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
1295             mix_mat31*b3_dot+noise1_dot)/train_leng;
1296     cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+mix_mat32* ...
1297         b3_dot+noise2_dot) ...
1298         *ctranspose(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
1299             mix_mat32*b3_dot+noise2_dot)/train_leng;
1300     cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+mix_mat33* ...
1301         b3_dot+noise3_dot) ...
1302         *ctranspose(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
1303             mix_mat33*b3_dot+noise3_dot)/train_leng;
1304     % sample cross-correlation matrices
1305     cro_cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
1306         mix_mat31*b3_dot+noise1_dot) ...
1307         *ctranspose(b1_dot)/train_leng;
1308     cro_cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
1309         mix_mat32*b3_dot+noise2_dot) ...
1310         *ctranspose(b2_dot)/train_leng;
1311     cro_cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
1312         mix_mat33*b3_dot+noise3_dot) ...
1313         *ctranspose(b3_dot)/train_leng;
1314     % LS estimated receive filter and power ...
1315     normalization
1316     [rece_fil1,rece_fil2,rece_fil3,~]= ...
1317         ls_rr_filsumrate_in_cal ...
1318         (mix_mat11,mix_mat21,mix_mat31,cor_mat1, ...
1319             cro_cor_mat1, ...
1320             mix_mat22,mix_mat12,mix_mat32,cor_mat2, ...
1321             cro_cor_mat2, ...
1322             mix_mat33,mix_mat13,mix_mat23,cor_mat3, ...
1323             cro_cor_mat3,noise_covar,rr,false);
1324     rece_fil1=mat_pow_nor(rece_fil1);
1325     rece_fil2=mat_pow_nor(rece_fil2);
1326     rece_fil3=mat_pow_nor(rece_fil3);
1327
1328     % for bi-directional training with each number ...
1329     of iteration
1330     for n_d=1:n
1331
1332         % backward training
1333         % precoder updating: precoder <-- receive filter
1334         b1_dot=reshape(b(1,2*n_d,:), ...
1335             [num_stream,train_leng]);
1336         b2_dot=reshape(b(2,2*n_d,:), ...
1337             [num_stream,train_leng]);
1338         b3_dot=reshape(b(3,2*n_d,:), ...
1339             [num_stream,train_leng]);
1340         noise1_dot=reshape(noise(1,2*n_d,:), ...
1341             [Nr,train_leng]);
1342         noise2_dot=reshape(noise(2,2*n_d,:), ...

```

```

[Nr,train_leng]);
1325 noise3_dot=reshape(noise(3,2*n_d,,:), ...
[Nr,train_leng]);

1326
1327 % backward mixing matrices
1328 mix_mat11=sqrt(pow)*transpose(squeeze ...
(Gaussian_chan_mat(1,1,,:)))*conj(rece_fil1);
1329 mix_mat12=sqrt(pow)*transpose(squeeze ...
(Gaussian_chan_mat(2,1,,:)))*conj(rece_fil2);
1330 mix_mat13=sqrt(pow)*transpose(squeeze ...
(Gaussian_chan_mat(3,1,,:)))*conj(rece_fil3);
1331 mix_mat21=sqrt(pow)*transpose(squeeze ...
(Gaussian_chan_mat(1,2,,:)))*conj(rece_fil1);
1332 mix_mat22=sqrt(pow)*transpose(squeeze ...
(Gaussian_chan_mat(2,2,,:)))*conj(rece_fil2);
1333 mix_mat23=sqrt(pow)*transpose(squeeze ...
(Gaussian_chan_mat(3,2,,:)))*conj(rece_fil3);
1334 mix_mat31=sqrt(pow)*transpose(squeeze ...
(Gaussian_chan_mat(1,3,,:)))*conj(rece_fil1);
1335 mix_mat32=sqrt(pow)*transpose(squeeze ...
(Gaussian_chan_mat(2,3,,:)))*conj(rece_fil2);
1336 mix_mat33=sqrt(pow)*transpose(squeeze ...
(Gaussian_chan_mat(3,3,,:)))*conj(rece_fil3);
1337 % sample correlation matrices
1338 cor_mat1=(mix_mat11*b1_dot+mix_mat12*b2_dot+ ...
mix_mat13*b3_dot+noise1_dot) ...
1339 *ctranspose(mix_mat11*b1_dot+ ...
mix_mat12*b2_dot+mix_mat13*b3_dot+ ...
noise1_dot)/train_leng;
1340 cor_mat2=(mix_mat21*b1_dot+mix_mat22*b2_dot+ ...
mix_mat23*b3_dot+noise2_dot) ...
1341 *ctranspose(mix_mat21*b1_dot+ ...
mix_mat22*b2_dot+mix_mat23*b3_dot+ ...
noise2_dot)/train_leng;
1342 cor_mat3=(mix_mat31*b1_dot+mix_mat32*b2_dot ...
+mix_mat33*b3_dot+noise3_dot) ...
1343 *ctranspose(mix_mat31*b1_dot+ ...
mix_mat32*b2_dot+mix_mat33*b3_dot+ ...
noise3_dot)/train_leng;
1344 % sample cross-correlation matrices
1345 cro_cor_mat1=(mix_mat11*b1_dot+mix_mat12* ...
b2_dot+mix_mat13*b3_dot+noise1_dot) ...
1346 *ctranspose(b1_dot)/train_leng;
1347 cro_cor_mat2=(mix_mat21*b1_dot+mix_mat22* ...
b2_dot+mix_mat23*b3_dot+noise2_dot) ...
1348 *ctranspose(b2_dot)/train_leng;
1349 cro_cor_mat3=(mix_mat31*b1_dot+mix_mat32* ...
b2_dot+mix_mat33*b3_dot+noise3_dot) ...
1350 *ctranspose(b3_dot)/train_leng;

```

```

1351 % LS estimated precoders
1352 [precoder1,precoder2,precoder3,-]= ...
1353     ls_rr_fil_sumrate_in_cal ...
1354     (mix_mat11,mix_mat12,mix_mat13,cor_mat1, ...
1355     cro_cor_mat1, ...
1356     mix_mat22,mix_mat21,mix_mat23,cor_mat2, ...
1357     cro_cor_mat2, ...
1358     mix_mat33,mix_mat31,mix_mat32,cor_mat3, ...
1359     cro_cor_mat3,noise_covar,rr,false);
1360 precoder1=mat_pow_nor(conj(precoder1));
1361 precoder2=mat_pow_nor(conj(precoder2));
1362 precoder3=mat_pow_nor(conj(precoder3));
1363
1364 % forward training
1365 % receive filter: precoder --> receive filter
1366 b1_dot=reshape(b(1,2*n_d+1,:,:),[num_stream, ...
1367     train_leng]);
1368 b2_dot=reshape(b(2,2*n_d+1,:,:),[num_stream, ...
1369     train_leng]);
1370 b3_dot=reshape(b(3,2*n_d+1,:,:),[num_stream, ...
1371     train_leng]);
1372 noise1_dot=reshape(noise(1,2*n_d+1,:,:),[Nr, ...
1373     train_leng]);
1374 noise2_dot=reshape(noise(2,2*n_d+1,:,:),[Nr, ...
1375     train_leng]);
1376 noise3_dot=reshape(noise(3,2*n_d+1,:,:),[Nr, ...
1377     train_leng]);
1378
1379 % forward mixing matrices
1380 mix_mat11=sqrt(pow)*squeeze(Gaussian_chan.mat ...
1381     (1,1,:,:))*precoder1;
1382 mix_mat21=sqrt(pow)*squeeze(Gaussian_chan.mat ...
1383     (2,1,:,:))*precoder2;
1384 mix_mat31=sqrt(pow)*squeeze(Gaussian_chan.mat ...
1385     (3,1,:,:))*precoder3;
1386 mix_mat12=sqrt(pow)*squeeze(Gaussian_chan.mat ...
1387     (1,2,:,:))*precoder1;
1388 mix_mat22=sqrt(pow)*squeeze(Gaussian_chan.mat ...
1389     (2,2,:,:))*precoder2;
1390 mix_mat32=sqrt(pow)*squeeze(Gaussian_chan.mat ...
1391     (3,2,:,:))*precoder3;
1392 mix_mat13=sqrt(pow)*squeeze(Gaussian_chan.mat ...
1393     (1,3,:,:))*precoder1;
1394 mix_mat23=sqrt(pow)*squeeze(Gaussian_chan.mat ...
1395     (2,3,:,:))*precoder2;
1396 mix_mat33=sqrt(pow)*squeeze(Gaussian_chan.mat ...
1397     (3,3,:,:))*precoder3;
1398 % sample correlation matrices
1399 cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...

```

```

1381         mix_mat31*b3_dot+noise1_dot) ...
            *ctranspose(mix_mat11*b1_dot+mix_mat21* ...
                b2_dot+mix_mat31*b3_dot+noise1_dot)/ ...
                train_leng;
1382     cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
        mix_mat32*b3_dot+noise2_dot) ...
1383         *ctranspose(mix_mat12*b1_dot+ ...
            mix_mat22*b2_dot+mix_mat32*b3_dot+ ...
            noise2_dot)/train_leng;
1384     cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
        mix_mat33*b3_dot+noise3_dot) ...
1385         *ctranspose(mix_mat13*b1_dot+ ...
            mix_mat23*b2_dot+mix_mat33*b3_dot+ ...
            noise3_dot)/train_leng;
1386     % sample cross-correlation matrices
1387     cro_cor_mat1=(mix_mat11*b1_dot+mix_mat21* ...
        b2_dot+mix_mat31*b3_dot+noise1_dot) ...
1388         *ctranspose(b1_dot)/train_leng;
1389     cro_cor_mat2=(mix_mat12*b1_dot+mix_mat22* ...
        b2_dot+mix_mat32*b3_dot+noise2_dot) ...
1390         *ctranspose(b2_dot)/train_leng;
1391     cro_cor_mat3=(mix_mat13*b1_dot+mix_mat23* ...
        2_dot+mix_mat33*b3_dot+noise3_dot) ...
1392         *ctranspose(b3_dot)/train_leng;
1393     % LS estimated receive filters
1394     [rece_fil1,rece_fil2,rece_fil3,sumrate_val]= ...
        ls_rr_fil.sumrate_in_cal ...
1395         (mix_mat11,mix_mat21,mix_mat31, ...
            cor_mat1,cro_cor_mat1, ...
            mix_mat22,mix_mat12,mix_mat32, ...
            cor_mat2,cro_cor_mat2, ...
            mix_mat33,mix_mat13,mix_mat23, ...
            cor_mat3,cro_cor_mat3, ...
            noise_covar,rr,true);
1396     % power normalization
1397     rece_fil1=mat_pow_nor(rece_fil1);
1398     rece_fil2=mat_pow_nor(rece_fil2);
1399     rece_fil3=mat_pow_nor(rece_fil3);
1400     end
1401     sumrate(rr_ind,m,n/num_ite_base)=sumrate_val;
1402     end
1403     end
1404     end
1405
1406     % averaging and extracting
1407     for m=1:size(red_rank,2)
1408         sumrate_ave(m,:)=mean(squeeze(sumrate(m,:,:)));
1409         sumrate_sam(m,:)=squeeze(sumrate(m,1,:));
1410     end

```



```

1411
1412 % number of iteration axis
1413 num_ite_leng=zeros(1,num_ite_ran/num_ite_base);
1414 for m=1:num_ite_ran/num_ite_base
1415     num_ite_leng(1,m)=m*num_ite_base;
1416 end
1417
1418 end
1419
1420
1421
1422
1423 ls_sumrate_vs_trainleng_bidirec_in_fr.m:
1424
1425 % LS estimated Sum rate vs Training length with full-rank
1426 % in interference network
1427
1428 function [sumrate_ave,sumrate_sam,train_leng] = ...
    ls_sumrate_vs_trainleng_bidirec_in_fr(pow,Gaussian_chan_mat, ...
    nor_precoder_mat,ave_num,train_leng_ran,train_base,num_ite, ...
    noise_covar)
1429
1430 % parameter extraction
1431 N=size(Gaussian_chan_mat,1);
1432 Nr=size(squeeze(Gaussian_chan_mat(1,1,:,:)),1);
1433 num_stream=size(squeeze(nor_precoder_mat(1,:,:)),2);
1434
1435 % temporary container
1436 sumrate=zeros(ave_num,train_leng_ran);
1437
1438 % for each trial in averaging
1439 for m=1:ave_num
1440     % training sequence for each iteration
1441     b=in_bidirec_trainseq_gen(N,2*num_ite+1,num_stream, ...
        train_base*train_leng_ran);
1442     % noise sequence for each iteration
1443     noise=in_bidirec_noiseseq_gen(N,2*num_ite+1,Nr, ...
        train_base*train_leng_ran);
1444
1445     % for each training length
1446     for n=train_base:train_base:train_base*train_leng_ran
1447         % initialized forward training
1448         % receive filter updating, precoder --> receive filter
1449         b1_dot=reshape(b(1,1,:),[1:n]),[num_stream,n]);
1450         b2_dot=reshape(b(2,1,:),[1:n]),[num_stream,n]);
1451         b3_dot=reshape(b(3,1,:),[1:n]),[num_stream,n]);
1452         noise1_dot=reshape(noise(1,1,:),[1:n]),[Nr,n]);
1453         noise2_dot=reshape(noise(2,1,:),[1:n]),[Nr,n]);
1454         noise3_dot=reshape(noise(3,1,:),[1:n]),[Nr,n]);

```

```

1455
1456 % initialized mixing matrices
1457 mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:)) ...
    *squeeze(nor_precoder_mat(1,:,:));
1458 mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:)) ...
    *squeeze(nor_precoder_mat(2,:,:));
1459 mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:)) ...
    *squeeze(nor_precoder_mat(3,:,:));
1460 mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:)) ...
    *squeeze(nor_precoder_mat(1,:,:));
1461 mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
    *squeeze(nor_precoder_mat(2,:,:));
1462 mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
    *squeeze(nor_precoder_mat(3,:,:));
1463 mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
    *squeeze(nor_precoder_mat(1,:,:));
1464 mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
    *squeeze(nor_precoder_mat(2,:,:));
1465 mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
    *squeeze(nor_precoder_mat(3,:,:));
1466
1467 % sample correlation matrices
1468 cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
    mix_mat31*b3_dot+noise1_dot) ...
1469     *ctranspose(mix_mat11*b1_dot+ ...
    mix_mat21*b2_dot+mix_mat31*b3_dot+noise1_dot)/n;
1470 cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
    mix_mat32*b3_dot+noise2_dot) ...
1471     *ctranspose(mix_mat12*b1_dot+ ...
    mix_mat22*b2_dot+mix_mat32*b3_dot+noise2_dot)/n;
1472 cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
    mix_mat33*b3_dot+noise3_dot) ...
1473     *ctranspose(mix_mat13*b1_dot+ ...
    mix_mat23*b2_dot+mix_mat33*b3_dot+noise3_dot)/n;
1474 % sample cross-correlation matrices
1475 cro_cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
    mix_mat31*b3_dot+noise1_dot) ...
1476     *ctranspose(b1_dot)/n;
1477 cro_cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
    mix_mat32*b3_dot+noise2_dot) ...
1478     *ctranspose(b2_dot)/n;
1479 cro_cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
    mix_mat33*b3_dot+noise3_dot) ...
1480     *ctranspose(b3_dot)/n;
1481 % LS estimated receive filters
1482 rece_fil1=cor_mat1\cro_cor_mat1;
1483 rece_fil2=cor_mat2\cro_cor_mat2;
1484 rece_fil3=cor_mat3\cro_cor_mat3;
1485 % power normalization

```

```

1486     rece_fil1_nor=mat_pow_nor(rece_fil1);
1487     rece_fil2_nor=mat_pow_nor(rece_fil2);
1488     rece_fil3_nor=mat_pow_nor(rece_fil3);
1489
1490     % for each number of bi-directional training
1491     for n_d=1:num_ite
1492         % backward training
1493         % precoder updating, precoder <-- receive filter
1494         b1_dot=reshape(b(1,2*n_d,:), [1:n]), [num_stream,n]);
1495         b2_dot=reshape(b(2,2*n_d,:), [1:n]), [num_stream,n]);
1496         b3_dot=reshape(b(3,2*n_d,:), [1:n]), [num_stream,n]);
1497         noise1_dot=reshape(noise(1,2*n_d,:), [1:n]), [Nr,n]);
1498         noise2_dot=reshape(noise(2,2*n_d,:), [1:n]), [Nr,n]);
1499         noise3_dot=reshape(noise(3,2*n_d,:), [1:n]), [Nr,n]);
1500
1501         % backward mixing matrices
1502         mix_mat11=sqrt(pow)*transpose(squeeze ...
1503             (Gaussian_chan_mat(1,1, :, :))) * conj(rece_fil1_nor);
1504         mix_mat12=sqrt(pow)*transpose(squeeze ...
1505             (Gaussian_chan_mat(2,1, :, :))) * conj(rece_fil2_nor);
1506         mix_mat13=sqrt(pow)*transpose(squeeze ...
1507             (Gaussian_chan_mat(3,1, :, :))) * conj(rece_fil3_nor);
1508         mix_mat21=sqrt(pow)*transpose(squeeze ...
1509             (Gaussian_chan_mat(1,2, :, :))) * conj(rece_fil1_nor);
1510         mix_mat22=sqrt(pow)*transpose(squeeze ...
1511             (Gaussian_chan_mat(2,2, :, :))) * conj(rece_fil2_nor);
1512         mix_mat23=sqrt(pow)*transpose(squeeze ...
1513             (Gaussian_chan_mat(3,2, :, :))) * conj(rece_fil3_nor);
1514         mix_mat31=sqrt(pow)*transpose(squeeze ...
1515             (Gaussian_chan_mat(1,3, :, :))) * conj(rece_fil1_nor);
1516         mix_mat32=sqrt(pow)*transpose(squeeze ...
1517             (Gaussian_chan_mat(2,3, :, :))) * conj(rece_fil2_nor);
1518         mix_mat33=sqrt(pow)*transpose(squeeze ...
1519             (Gaussian_chan_mat(3,3, :, :))) * conj(rece_fil3_nor);
1520
1521         % sample correlation matrices
1522         cor_mat1=(mix_mat11*b1_dot+mix_mat12*b2_dot ...
1523             +mix_mat13*b3_dot+noise1_dot) ...
1524             *ctranspose(mix_mat11*b1_dot+ ...
1525                 mix_mat12*b2_dot+mix_mat13*b3_dot+ ...
1526                 noise1_dot)/n;
1527         cor_mat2=(mix_mat21*b1_dot+mix_mat22*b2_dot+ ...
1528             mix_mat23*b3_dot+noise2_dot) ...
1529             *ctranspose(mix_mat21*b1_dot+ ...
1530                 mix_mat22*b2_dot+mix_mat23*b3_dot+ ...
1531                 noise2_dot)/n;
1532         cor_mat3=(mix_mat31*b1_dot+mix_mat32*b2_dot+ ...
1533             mix_mat33*b3_dot+noise3_dot) ...
1534             *ctranspose(mix_mat31*b1_dot+ ...
1535                 mix_mat32*b2_dot+mix_mat33*b3_dot+ ...

```

```

                                noise3_dot)/n;
1518 % sample cross-correlation matrices
1519 cro_cor_mat1=(mix_mat11*b1_dot+mix_mat12*b2_dot ...
                                +mix_mat13*b3_dot+noise1_dot) ...
1520                                *ctranspose(b1_dot)/n;
1521 cro_cor_mat2=(mix_mat21*b1_dot+mix_mat22*b2_dot ...
                                +mix_mat23*b3_dot+noise2_dot) ...
1522                                *ctranspose(b2_dot)/n;
1523 cro_cor_mat3=(mix_mat31*b1_dot+mix_mat32*b2_dot ...
                                +mix_mat33*b3_dot+noise3_dot) ...
1524                                *ctranspose(b3_dot)/n;
1525 % LS estimated precoders and power normalization
1526 precoder1=mat_pow_nor(conj(cro_cor_mat1\cro_cor_mat1));
1527 precoder2=mat_pow_nor(conj(cro_cor_mat2\cro_cor_mat2));
1528 precoder3=mat_pow_nor(conj(cro_cor_mat3\cro_cor_mat3));
1529
1530 % forward training
1531 % receive filter updating, precoder --> receive ...
                                filter
1532 b1_dot=reshape(b(1,2*n_d+1,:), [1:n]), [num_stream,n]);
1533 b2_dot=reshape(b(2,2*n_d+1,:), [1:n]), [num_stream,n]);
1534 b3_dot=reshape(b(3,2*n_d+1,:), [1:n]), [num_stream,n]);
1535 noise1_dot=reshape(noise(1,2*n_d+1,:), [1:n]), [Nr,n]);
1536 noise2_dot=reshape(noise(2,2*n_d+1,:), [1:n]), [Nr,n]);
1537 noise3_dot=reshape(noise(3,2*n_d+1,:), [1:n]), [Nr,n]);
1538
1539 % forward mixing matrices
1540 mix_mat11=sqrt(pow)*squeeze(Gaussian_chan.mat ...
                                (1,1, :, :))*precoder1;
1541 mix_mat21=sqrt(pow)*squeeze(Gaussian_chan.mat ...
                                (2,1, :, :))*precoder2;
1542 mix_mat31=sqrt(pow)*squeeze(Gaussian_chan.mat ...
                                (3,1, :, :))*precoder3;
1543 mix_mat12=sqrt(pow)*squeeze(Gaussian_chan.mat ...
                                (1,2, :, :))*precoder1;
1544 mix_mat22=sqrt(pow)*squeeze(Gaussian_chan.mat ...
                                (2,2, :, :))*precoder2;
1545 mix_mat32=sqrt(pow)*squeeze(Gaussian_chan.mat ...
                                (3,2, :, :))*precoder3;
1546 mix_mat13=sqrt(pow)*squeeze(Gaussian_chan.mat ...
                                (1,3, :, :))*precoder1;
1547 mix_mat23=sqrt(pow)*squeeze(Gaussian_chan.mat ...
                                (2,3, :, :))*precoder2;
1548 mix_mat33=sqrt(pow)*squeeze(Gaussian_chan.mat ...
                                (3,3, :, :))*precoder3;
1549 % sample correlation matrices
1550 cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
                                mix_mat31*b3_dot+noise1_dot) ...
1551                                *ctranspose(mix_mat11*b1_dot+ ...

```

```

1552         mix_mat21*b2_dot+mix_mat31*b3_dot ...
            +noise1_dot)/n;
1553     cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
        mix_mat32*b3_dot+noise2_dot) ...
1554         *ctranspose(mix_mat12*b1_dot+ ...
            mix_mat22*b2_dot+mix_mat32*b3_dot+ ...
            noise2_dot)/n;
1555     cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
        mix_mat33*b3_dot+noise3_dot) ...
1556         *ctranspose(mix_mat13*b1_dot+ ...
            mix_mat23*b2_dot+mix_mat33*b3_dot+ ...
            noise3_dot)/n;
1557     % sample cross-correlation matrices
1558     cro_cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
        mix_mat31*b3_dot+noise1_dot) ...
1559         *ctranspose(b1_dot)/n;
1560     cro_cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
        mix_mat32*b3_dot+noise2_dot) ...
1561         *ctranspose(b2_dot)/n;
1562     cro_cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot ...
        +mix_mat33*b3_dot+noise3_dot) ...
1563         *ctranspose(b3_dot)/n;
1564     % LS estimated receive filters
1565     rece_fil1=cor_mat1\cro_cor_mat1;
1566     rece_fil2=cor_mat2\cro_cor_mat2;
1567     rece_fil3=cor_mat3\cro_cor_mat3;
1568     % power normalization
1569     rece_fil1_nor=mat_pow_nor(rece_fil1);
1570     rece_fil2_nor=mat_pow_nor(rece_fil2);
1571     rece_fil3_nor=mat_pow_nor(rece_fil3);
1572     end
1573     % data preallocation
1574     sumrate1=zeros(1,num.stream);
1575     sumrate2=zeros(1,num.stream);
1576     sumrate3=zeros(1,num.stream);
1577
1578     for n_d=1:num.stream
1579
1580         % signal power
1581         signal_pow1=abs(ctranspose(rece_fil1(:,n_d))* ...
            mix_mat11(:,n_d))^2;
1582         signal_pow2=abs(ctranspose(rece_fil2(:,n_d))* ...
            mix_mat22(:,n_d))^2;
1583         signal_pow3=abs(ctranspose(rece_fil3(:,n_d))* ...
            mix_mat33(:,n_d))^2;
1584
1585         % interference power
1586         interf_pow1=0;

```

```

1587         interf_pow2=0;
1588         interf_pow3=0;
1589         % interference from its transmitter
1590         for n_dd=1:num_stream
1591             if n_dd==n_d
1592                 % pass
1593             else
1594                 interf_pow1=interf_pow1+abs(ctranspose ...
1595                     (rece_fill(:,n_d))*mix_mat11(:,n_dd))^2;
1596                 interf_pow2=interf_pow2+abs(ctranspose ...
1597                     (rece_fil2(:,n_d))*mix_mat22(:,n_dd))^2;
1598                 interf_pow3=interf_pow3+abs(ctranspose ...
1599                     (rece_fil3(:,n_d))*mix_mat33(:,n_dd))^2;
1600             end
1601         end
1602
1603         % interference from other transmitters
1604         for n_dd=1:num_stream
1605             interf_pow1=interf_pow1+abs(ctranspose ...
1606                 (rece_fill(:,n_d))*mix_mat21(:,n_dd))^2 ...
1607                 +abs(ctranspose(rece_fill ...
1608                     (:,n_d))*mix_mat31(:,n_dd))^2;
1609             interf_pow2=interf_pow2+abs(ctranspose ...
1610                 (rece_fil2(:,n_d))*mix_mat12(:,n_dd))^2 ...
1611                 +abs(ctranspose(rece_fil2 ...
1612                     (:,n_d))*mix_mat32(:,n_dd))^2;
1613             interf_pow3=interf_pow3+abs(ctranspose ...
1614                 (rece_fil3(:,n_d))*mix_mat13(:,n_dd))^2 ...
1615                 +abs(ctranspose(rece_fil3 ...
1616                     (:,n_d))*mix_mat23(:,n_dd))^2;
1617         end
1618
1619         % noise power
1620         noise_pow1=abs(ctranspose(rece_fill(:,n_d))* ...
1621             noise_covar*rece_fill(:,n_d));
1622         noise_pow2=abs(ctranspose(rece_fil2(:,n_d))* ...
1623             noise_covar*rece_fil2(:,n_d));
1624         noise_pow3=abs(ctranspose(rece_fil3(:,n_d))* ...
1625             noise_covar*rece_fil3(:,n_d));
1626
1627         % sum rate computation
1628         sumrate1(1,n_d)=log2(1+signal_pow1/(interf_pow1+ ...
1629             noise_pow1));
1630         sumrate2(1,n_d)=log2(1+signal_pow2/(interf_pow2+ ...
1631             noise_pow2));
1632         sumrate3(1,n_d)=log2(1+signal_pow3/(interf_pow3+ ...
1633             noise_pow3));
1634     end
1635 end

```

```

1621         % network sum rate
1622         sumrate(m,n/train_base)=sum(sumrate1)+sum(sumrate2)+ ...
            sum(sumrate3);
1623     end
1624 end
1625
1626 % data averaging, sample data extraction
1627 sumrate_ave=mean(sumrate);
1628 sumrate_sam=sumrate(1,:);
1629
1630 % training length axis
1631 train_leng=zeros(1,train_leng_ran);
1632 for m=1:train_leng_ran
1633     train_leng(1,m)=m*train_base;
1634 end
1635
1636 end
1637
1638
1639
1640
1641 ls_sumrate_vs_trainleng_bidirec_in_rr.m:
1642
1643 % LS estimated Sum rate vs Training length with reduced-rank in
1644 % interference network
1645
1646 function [sumrate_ave,sumrate_sam,train_leng] = ...
    ls_sumrate_vs_trainleng_bidirec_in_rr(pow,Gaussian_chan_mat, ...
    nor_precoder_mat,red_rank,ave_num,train_leng_ran,train_base, ...
    num_ite,noise_covar)
1647
1648 % parameter extraction
1649 N=size(Gaussian_chan_mat,1);
1650 Nr=size(squeeze(Gaussian_chan_mat(1,1,:,:)),1);
1651 num_stream=size(squeeze(nor_precoder_mat(1,:,:)),2);
1652 % temporary container
1653 sumrate=zeros(size(red_rank,2),ave_num,train_leng_ran);
1654 % output data
1655 sumrate_ave=zeros(size(red_rank,2),train_leng_ran);
1656 sumrate_sam=zeros(size(red_rank,2),train_leng_ran);
1657
1658 % for each reduced-rank
1659 for rr_ind=1:size(red_rank,2)
1660     % assign reduced-rank
1661     rr=red_rank(rr_ind);
1662
1663     % for each trial in averaging
1664     for m=1:ave_num
1665

```

```

1666 % training sequence for each iteration
1667 b=in_bidirec_trainseq_gen(N,2*num_ite+1,num_stream, ...
    train_base*train_leng_ran);
1668 % noise sequence for each iteration
1669 noise=in_bidirec_noiseseq_gen(N,2*num_ite+1,Nr, ...
    train_base*train_leng_ran);
1670
1671 % for each training length
1672 for n=train_base:train_base:train_base*train_leng_ran
1673
1674 % initialized forward training
1675 % receive filter updating, precoder --> receive ...
    filter
1676 b1_dot=reshape(b(1,1,:),[1:n]),[num_stream,n]);
1677 b2_dot=reshape(b(2,1,:),[1:n]),[num_stream,n]);
1678 b3_dot=reshape(b(3,1,:),[1:n]),[num_stream,n]);
1679 noise1_dot=reshape(noise(1,1,:),[1:n]),[Nr,n]);
1680 noise2_dot=reshape(noise(2,1,:),[1:n]),[Nr,n]);
1681 noise3_dot=reshape(noise(3,1,:),[1:n]),[Nr,n]);
1682
1683 % initialized mixing matrices
1684 mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (1,1,,:))*squeeze(nor_precoder_mat(1,,:));
1685 mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (2,1,,:))*squeeze(nor_precoder_mat(2,,:));
1686 mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (3,1,,:))*squeeze(nor_precoder_mat(3,,:));
1687 mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (1,2,,:))*squeeze(nor_precoder_mat(1,,:));
1688 mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (2,2,,:))*squeeze(nor_precoder_mat(2,,:));
1689 mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (3,2,,:))*squeeze(nor_precoder_mat(3,,:));
1690 mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (1,3,,:))*squeeze(nor_precoder_mat(1,,:));
1691 mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (2,3,,:))*squeeze(nor_precoder_mat(2,,:));
1692 mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (3,3,,:))*squeeze(nor_precoder_mat(3,,:));
1693
1694 % sample correlation matrices
1695 cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
    mix_mat31*b3_dot+noise1_dot) ...
1696 *ctranspose(mix_mat11*b1_dot+mix_mat21* ...
    b2_dot+mix_mat31*b3_dot+noise1_dot)/n;
1697 cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
    mix_mat32*b3_dot+noise2_dot) ...
1698 *ctranspose(mix_mat12*b1_dot+mix_mat22* ...
    b2_dot+mix_mat32*b3_dot+noise2_dot)/n;

```



```

1699     cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
1700             mix_mat33*b3_dot+noise3_dot) ...
1701             *ctranspose(mix_mat13*b1_dot+mix_mat23*b2_dot ...
1702             +mix_mat33*b3_dot+noise3_dot)/n;
1703     % sample cross-correlation matrices
1704     cro_cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
1705             mix_mat31*b3_dot+noise1_dot) ...
1706             *ctranspose(b1_dot)/n;
1707     cro_cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
1708             mix_mat32*b3_dot+noise2_dot) ...
1709             *ctranspose(b2_dot)/n;
1710     cro_cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
1711             mix_mat33*b3_dot+noise3_dot) ...
1712             *ctranspose(b3_dot)/n;
1713     % reduced-rank LS estimated receive filters
1714     [rece_fil1,rece_fil2,rece_fil3,-]= ...
1715     ls_rr_fil_sumrate_in_cal ...
1716     (mix_mat11,mix_mat21,mix_mat31,cor_mat1, ...
1717     cro_cor_mat1, ...
1718     mix_mat22,mix_mat12,mix_mat32,cor_mat2, ...
1719     cro_cor_mat2, ...
1720     mix_mat33,mix_mat13,mix_mat23,cor_mat3, ...
1721     cro_cor_mat3,noise_covar,rr,false);
1722     % power normalization
1723     rece_fil1=mat_pow_nor(rece_fil1);
1724     rece_fil2=mat_pow_nor(rece_fil2);
1725     rece_fil3=mat_pow_nor(rece_fil3);
1726
1727     % for each number of bi-directional training
1728     for n_d=1:num_ite
1729
1730         % backward training
1731         % precoder updating: precoder <-- receive filter
1732         b1_dot=reshape(b(1,2*n_d,:),[1:n]),[num_stream,n]);
1733         b2_dot=reshape(b(2,2*n_d,:),[1:n]),[num_stream,n]);
1734         b3_dot=reshape(b(3,2*n_d,:),[1:n]),[num_stream,n]);
1735         noise1_dot=reshape(noise(1,2*n_d,:),[1:n]),[Nr,n]);
1736         noise2_dot=reshape(noise(2,2*n_d,:),[1:n]),[Nr,n]);
1737         noise3_dot=reshape(noise(3,2*n_d,:),[1:n]),[Nr,n]);
1738
1739         % backward mixing matrices
1740         mix_mat11=sqrt(pow)*transpose(squeeze ...
1741             (Gaussian_chan_mat(1,1,,:)))*conj(rece_fil1);
1742         mix_mat12=sqrt(pow)*transpose(squeeze ...
1743             (Gaussian_chan_mat(2,1,,:)))*conj(rece_fil2);
1744         mix_mat13=sqrt(pow)*transpose(squeeze ...
1745             (Gaussian_chan_mat(3,1,,:)))*conj(rece_fil3);
1746         mix_mat21=sqrt(pow)*transpose(squeeze ...
1747             (Gaussian_chan_mat(1,2,,:)))*conj(rece_fil1);

```

```

1735 mix_mat22=sqrt(pow)*transpose(squeeze ...
      (Gaussian_chan_mat(2,2, :, :))) *conj(rece_fil2);
1736 mix_mat23=sqrt(pow)*transpose(squeeze ...
      (Gaussian_chan_mat(3,2, :, :))) *conj(rece_fil3);
1737 mix_mat31=sqrt(pow)*transpose(squeeze ...
      (Gaussian_chan_mat(1,3, :, :))) *conj(rece_fil1);
1738 mix_mat32=sqrt(pow)*transpose(squeeze ...
      (Gaussian_chan_mat(2,3, :, :))) *conj(rece_fil2);
1739 mix_mat33=sqrt(pow)*transpose(squeeze ...
      (Gaussian_chan_mat(3,3, :, :))) *conj(rece_fil3);
1740 % sample correlation matrices
1741 cor_mat1=(mix_mat11*b1_dot+mix_mat12*b2_dot ...
      +mix_mat13*b3_dot+noise1_dot) ...
1742      *ctranspose(mix_mat11*b1_dot+ ...
      mix_mat12*b2_dot+mix_mat13*b3_dot+ ...
      noise1_dot)/n;
1743 cor_mat2=(mix_mat21*b1_dot+mix_mat22*b2_dot+ ...
      mix_mat23*b3_dot+noise2_dot) ...
1744      *ctranspose(mix_mat21*b1_dot+ ...
      mix_mat22*b2_dot+mix_mat23*b3_dot+ ...
      noise2_dot)/n;
1745 cor_mat3=(mix_mat31*b1_dot+mix_mat32*b2_dot+ ...
      mix_mat33*b3_dot+noise3_dot) ...
1746      *ctranspose(mix_mat31*b1_dot+ ...
      mix_mat32*b2_dot+mix_mat33*b3_dot+ ...
      noise3_dot)/n;
1747 % sample cross-correlation matrices
1748 cro_cor_mat1=(mix_mat11*b1_dot+mix_mat12* ...
      b2_dot+mix_mat13*b3_dot+noise1_dot) ...
1749      *ctranspose(b1_dot)/n;
1750 cro_cor_mat2=(mix_mat21*b1_dot+mix_mat22* ...
      b2_dot+mix_mat23*b3_dot+noise2_dot) ...
1751      *ctranspose(b2_dot)/n;
1752 cro_cor_mat3=(mix_mat31*b1_dot+mix_mat32* ...
      b2_dot+mix_mat33*b3_dot+noise3_dot) ...
1753      *ctranspose(b3_dot)/n;
1754 % reduced-rank LS estimated precoders
1755 [precoder1,precoder2,precoder3,-]= ...
      ls_rr_fil_sumrate_in_cal ...
1756      (mix_mat11,mix_mat12,mix_mat13,cor_mat1, ...
      cro_cor_mat1, ...
1757      mix_mat22,mix_mat21,mix_mat23,cor_mat2, ...
      cro_cor_mat2, ...
1758      mix_mat33,mix_mat31,mix_mat32,cor_mat3, ...
      cro_cor_mat3,noise_covar,rr,false);
1759 precoder1=mat_pow_nor(conj(precoder1));
1760 precoder2=mat_pow_nor(conj(precoder2));
1761 precoder3=mat_pow_nor(conj(precoder3));
1762

```

```

1763 % forward training
1764 % receive filter: precoder --> receive filter
1765 b1_dot=reshape(b(1,2*n_d+1,:),[1:n]),[num_stream,n]);
1766 b2_dot=reshape(b(2,2*n_d+1,:),[1:n]),[num_stream,n]);
1767 b3_dot=reshape(b(3,2*n_d+1,:),[1:n]),[num_stream,n]);
1768 noise1_dot=reshape(noise(1,2*n_d+1,:),[1:n]),[Nr,n]);
1769 noise2_dot=reshape(noise(2,2*n_d+1,:),[1:n]),[Nr,n]);
1770 noise3_dot=reshape(noise(3,2*n_d+1,:),[1:n]),[Nr,n]);
1771
1772 % forward mixing matrices
1773 mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (1,1, :, :))*precoder1;
1774 mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (2,1, :, :))*precoder2;
1775 mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (3,1, :, :))*precoder3;
1776 mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (1,2, :, :))*precoder1;
1777 mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (2,2, :, :))*precoder2;
1778 mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (3,2, :, :))*precoder3;
1779 mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (1,3, :, :))*precoder1;
1780 mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (2,3, :, :))*precoder2;
1781 mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (3,3, :, :))*precoder3;
1782 % sample correlation matrices
1783 cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
    mix_mat31*b3_dot+noise1_dot) ...
1784         *ctranspose(mix_mat11*b1_dot+mix_mat21* ...
    b2_dot+mix_mat31*b3_dot+noise1_dot)/n;
1785 cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
    mix_mat32*b3_dot+noise2_dot) ...
1786         *ctranspose(mix_mat12*b1_dot+mix_mat22* ...
    b2_dot+mix_mat32*b3_dot+noise2_dot)/n;
1787 cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
    mix_mat33*b3_dot+noise3_dot) ...
1788         *ctranspose(mix_mat13*b1_dot+mix_mat23* ...
    b2_dot+mix_mat33*b3_dot+noise3_dot)/n;
1789 % sample cross-correlation matrices
1790 cro_cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
    mix_mat31*b3_dot+noise1_dot) ...
1791         *ctranspose(b1_dot)/n;
1792 cro_cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
    mix_mat32*b3_dot+noise2_dot) ...
1793         *ctranspose(b2_dot)/n;
1794 cro_cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...

```

```

1795         mix_mat33*b3_dot+noise3_dot) ...
1796         *ctranspose(b3_dot)/n;
1797     % reduced-rank LS estimated receive filters
1798     [rece_fil1,rece_fil2,rece_fil3,sumrate_val]= ...
1799         ls_rr_fil_sumrate_in_cal ...
1800         (mix_mat11,mix_mat21,mix_mat31, ...
1801             cor_mat1,cro_cor_mat1, ...
1802             mix_mat22,mix_mat12,mix_mat32, ...
1803             cor_mat2,cro_cor_mat2, ...
1804             mix_mat33,mix_mat13,mix_mat23, ...
1805             cor_mat3,cro_cor_mat3,noise_covar ...
1806             ,rr,true);
1807     % power normalization
1808     rece_fil1=mat_pow_nor(rece_fil1);
1809     rece_fil2=mat_pow_nor(rece_fil2);
1810     rece_fil3=mat_pow_nor(rece_fil3);
1811 end
1812 sumrate(rr_ind,m,n/train_base)=sumrate_val;
1813 end
1814 end
1815 end
1816 % averaging and extracting
1817 for m=1:size(red_rank,2)
1818     sumrate_ave(m,:)=mean(squeeze(sumrate(m,:,:)));
1819     sumrate_sam(m,:)=squeeze(sumrate(m,1,:));
1820 end
1821 % training length axis
1822 train_leng=zeros(1,train_leng_ran);
1823 for m=1:train_leng_ran
1824     train_leng(1,m)=m*train_base;
1825 end
1826 end
1827
1828 mat_pow_nor.m:
1829
1830 % Power normalization for each beamformer for each matrix
1831
1832 function [mat] = mat_pow_nor(mat)
1833
1834 % extract the number of streams
1835 num_stream=size(mat,2);
1836
1837 % power normalization for each beamformer

```

```

1838 for m=1:num_stream
1839     mat(:,m)=mat(:,m)/norm(mat(:,m),'fro');
1840 end
1841
1842 end
1843
1844
1845
1846
1847 noise_mat_gen.m:
1848
1849 % Gaussian additive noise generator
1850
1851 function noise_mat = noise_mat_gen(Nr,train_leng)
1852
1853 noise_mat=(1/sqrt(2))*(randn(Nr,train_leng)/10+ ...
1854     i*randn(Nr,train_leng)/10);
1855 end
1856
1857
1858
1859
1860 precoder_gen.m:
1861
1862 % Normalized precoder matrix
1863
1864 function precoder = precoder_gen(Nt,num_stream,pow)
1865
1866 % randomly generated precoder
1867 precoder=rand(Nt,num_stream);
1868
1869 % power normalization for each beamformer
1870 for m=1:num_stream
1871     precoder(:,m)=sqrt(pow)*precoder(:,m)/norm(precoder(:,m),'fro');
1872 end
1873
1874 end
1875
1876
1877
1878
1879 wiener_fr_mmse_sinr_cal.m:
1880
1881 % Wiener estimated full-rank MMSE and SINR calculator
1882
1883 function [mmse,sinr] = ...
1884     wiener_fr_mmse_sinr_cal(mix_mat,cor_mat,combiner)

```

```

1884
1885 % extract parameter
1886 num_stream=size(mix_mat,2);
1887
1888 % temporary container
1889 mmse_temp=zeros(1,num_stream);
1890 sinr_temp=zeros(1,num_stream);
1891
1892 % calculate MMSE & SINR for each stream
1893 for m=1:num_stream
1894     % MMSE
1895     mmse_temp(1,m)=1-abs(ctranspose(combiner(:,m))* ...
        cor_mat*combiner(:,m));
1896     % SINR
1897     sinr_temp(1,m)=abs(ctranspose(mix_mat(:,m))*inv ...
        (cor_mat-mix_mat(:,m)*ctranspose(mix_mat(:,m)))*mix_mat ...
        (:,m));
1898 end
1899
1900 % average over all streams
1901 mmse=mean(mmse_temp,2);
1902 sinr=mean(sinr_temp,2);
1903
1904 end
1905
1906
1907
1908
1909 wiener_mmse_sinr_fixnumite_bidirec_fr.m:
1910
1911 % Wiener (optimal) MMSE & SINR for fixed number of ...
    bi-directional
1912 % optimization iteration with full-rank transmitter and receiver
1913
1914 function [mmse,sinr,train_leng] = ...
    wiener_mmse_sinr_fixednumite_bidirec_fr(pow,Gaussian_chan, ...
    nor_precoder,train_leng_ran,train_base,num_ite,noise_covar)
1915
1916 % parameter extraction
1917 [Nr,num_stream]=size(Gaussian_chan*nor_precoder);
1918 % output container
1919 mmse=zeros(1,train_leng_ran);
1920 sinr=zeros(1,train_leng_ran);
1921
1922 % initialized forward optimization
1923 % initialized mixing matrix
1924 mix_mat=sqrt(pow)*Gaussian_chan*nor_precoder;
1925 % statistical correlation matrix
1926 cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);

```

```

1927 % Wiener (optimal) receive filter
1928 rece_fil=cor_mat\mix_mat;
1929 % power normalization
1930 rece_fil_nor=mat_pow_nor(rece_fil);
1931
1932 % for each bi-directional optimization
1933 for m=1:num_lte
1934     % backward optimization
1935     mix_mat=transpose(sqrt(pow)*Gaussian_chan)* ...
        conj(rece_fil_nor);
1936     % statistical correlation matrix
1937     cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
1938     % Wiener (optimal) precoder
1939     precoder=conj(cor_mat\mix_mat);
1940     % power normalization
1941     precoder_nor=mat_pow_nor(precoder);
1942
1943     % forward optimization
1944     mix_mat=sqrt(pow)*Gaussian_chan*precoder_nor;
1945     % statistical correlation matrix
1946     cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
1947     % Wiener (optimal) receive filter
1948     rece_fil=cor_mat\mix_mat;
1949     % power normalization
1950     rece_fil_nor=mat_pow_nor(rece_fil);
1951 end
1952
1953 % data preallocation
1954 mmse_temp=zeros(1,num_stream);
1955 sinr_temp=zeros(1,num_stream);
1956
1957 % calculate MMSE & SINR
1958 % for each stream
1959 for n=1:num_stream
1960     % MMSE
1961     mmse_temp(1,n)=1-abs(ctranspose(rece_fil(:,n))*cor_mat* ...
        rece_fil(:,n));
1962     % SINR
1963     sinr_temp(1,n)=abs(ctranspose(mix_mat(:,n))*((cor_mat- ...
        mix_mat(:,n)*ctranspose(mix_mat(:,n))\mix_mat(:,n))));
1964 end
1965
1966 % averaging and decibel conversion
1967 mmse_val=10*log10(mean(mmse_temp,2));
1968 sinr_val=10*log10(mean(sinr_temp,2));
1969
1970 % training length axis
1971 train_leng=zeros(1,train_leng_ran);
1972 for m=1:train_leng_ran

```

```

1973     mmse(1,m)=mmse_val;
1974     sinr(1,m)=sinr_val;
1975     train_leng(1,m)=m*train_base;
1976 end
1977
1978 end
1979
1980
1981
1982
1983 wiener_mmse_sinr_fixednumite_bidirec_rr.m:
1984
1985 % Wiener (optimal) MMSE & SINR for fixed number of ...
    % bi-directional optimization iteration with reduced-rank
1986
1987 function [mmse,sinr,train_leng] = ...
    wiener_mmse_sinr_fixednumite_bidirec_rr(pow,Gaussian_chan, ...
    nor_precoder,red_rank,train_leng_ran,train_base,num_ite, ...
    noise_covar)
1988
1989 % parameter extraction
1990 [Nr]=size(Gaussian_chan,1);
1991 % temporary data preallocation
1992 mmse_temp=zeros(1,size(red_rank,2));
1993 sinr_temp=zeros(1,size(red_rank,2));
1994 % data preallocation
1995 mmse=zeros(size(red_rank,2),train_leng_ran);
1996 sinr=zeros(size(red_rank,2),train_leng_ran);
1997
1998 % for each reduced-rank
1999 for rr_ind=1:size(red_rank,2)
2000     % reduced-rank value
2001     rr=red_rank(rr_ind);
2002
2003     % initialized forward optimization
2004     % initialized mixing matrix
2005     mix_mat=sqrt(pow)*Gaussian_chan*nor_precoder;
2006     % statistical correlation matrix
2007     cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
2008     % reduced-rank Wiener (optimal) receive filter
2009     [rece_fil,~,~]=wiener_rr_fil_mmse_sinr_cal(cor_mat, ...
        mix_mat,rr);
2010     % power normalization
2011     rece_fil_nor=mat_pow_nor(rece_fil);
2012
2013     % for each bi-directional optimization
2014     for m=1:num_ite
2015         % backward optimization
2016         mix_mat=transpose(sqrt(pow)*Gaussian_chan)* ...

```



```

        conj(rece_fil_nor);
2017     % statistical correlation matrix
2018     cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
2019     % reduced-rank Wiener (optimal) precoder
2020     [precoder,~,~]=wiener_rr_fil_mmse_sinr_cal(cor_mat, ...
        mix_mat,rr);
2021     precoder=conj(precoder);
2022     % power normalization
2023     precoder_nor=mat_pow_nor(precoder);
2024
2025     % forward optimization
2026     mix_mat=sqrt(pow)*Gaussian_chan*precoder_nor;
2027     % statistical correlation matrix
2028     cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
2029     % reduced-rank Wiener (optimal) receive filter
2030     [rece_fil,mmse_val,sinr_val]=wiener_rr_fil_mmse_sinr_cal ...
        (cor_mat,mix_mat,rr);
2031     % power normalization
2032     rece_fil_nor=mat_pow_nor(rece_fil);
2033     end
2034
2035     % decibel conversion and assign data value to ...
        corresponding allocation
2036     mmse_temp(1,rr_ind)=10*log10(mmse_val);
2037     sinr_temp(1,rr_ind)=10*log10(sinr_val);
2038     end
2039
2040     % training length axis
2041     train_leng=zeros(1,train_leng_ran);
2042     for m=1:train_leng_ran
2043         mmse(1,m)=mmse_temp(1,1);
2044         mmse(2,m)=mmse_temp(1,2);
2045         sinr(1,m)=sinr_temp(1,1);
2046         sinr(2,m)=sinr_temp(1,2);
2047         train_leng(1,m)=m*train_base;
2048     end
2049
2050     end
2051
2052
2053
2054
2055     wiener_mmse_sinr_fr.m:
2056
2057     % Wiener (optimal) estimated MMSE & SINR in full-rank
2058
2059     function [mmse,sinr] = wiener_mmse_sinr_fr(mix_mat,noise_covar)
2060
2061     % parameter extraction

```

```

2062 [Nr,num_stream]=size(mix_mat);
2063
2064 % temporary container
2065 mmse=zeros(1,num_stream);
2066 sinr=zeros(1,num_stream);
2067
2068 % correlation matrix
2069 cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
2070 % full-rank receive filter
2071 rece_fil=cor_mat\mix_mat;
2072
2073 for m=1:num_stream
2074     mmse(1,m)=1-abs(ctranspose(rece_fil(:,m))*cor_mat* ...
        rece_fil(:,m));
2075     sinr(1,m)=abs(ctranspose(mix_mat(:,m))*((cor_mat- ...
        mix_mat(:,m)*ctranspose(mix_mat(:,m))\mix_mat(:,m))));
2076 end
2077
2078 % averaging & decibel conversion
2079 mmse=10*log10(mean(mmse,2));
2080 sinr=10*log10(mean(sinr,2));
2081
2082 end
2083
2084
2085
2086
2087 wiener_mmse_sinr_rr.m:
2088
2089 % Wiener (optimal) estimated MMSE & SINR in reduced-rank
2090
2091 function [mmse,sinr] = ...
        wiener_mmse_sinr_rr(mix_mat,red_rank,noise_covar)
2092
2093 % parameter extraction
2094 [Nr]=size(mix_mat,1);
2095
2096 % temporary container
2097 mmse=zeros(1,size(red_rank,2));
2098 sinr=zeros(1,size(red_rank,2));
2099
2100 % correlation matrix
2101 cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
2102
2103 for rr_ind=1:size(red_rank,2)
2104     % reduced-rank
2105     rr=red_rank(1,rr_ind);
2106     % compute MMSE & SINR for each reduced-rank
2107     [~,mmse(1,rr_ind),sinr(1,rr_ind)]= ...

```

```

        wiener_rr_fil_mmse_sinr_cal(cor_mat,mix_mat,rr);
2108 end
2109
2110 % decibel conversion
2111 mmse=10*log10(mmse);
2112 sinr=10*log10(sinr);
2113
2114 end
2115
2116
2117
2118
2119 wiener_mmse_sinr_vs_numite_bidirec_fr.m:
2120
2121 % Wiener (optimal) estimated (MMSE & SINR) vs Number of ...
    bi-directional optimization iteration with full-rank
2122
2123 function [mmse,sinr,num_ite_leng] = ...
    wiener_mmse_sinr_vs_numite_bidirec_fr(pow,Gaussian_chan, ...
    nor_precoder,num_ite_ran,num_ite_base,noise_covar)
2124
2125 % parameter extraction
2126 [Nr,num_stream]=size(Gaussian_chan*nor_precoder);
2127
2128 % output container
2129 mmse=zeros(1,num_ite_ran/num_ite_base);
2130 sinr=zeros(1,num_ite_ran/num_ite_base);
2131
2132 % temporary container
2133 mmse_temp=zeros(1,num_stream);
2134 sinr_temp=zeros(1,num_stream);
2135
2136 % for each number of bi-directional optimization iteration
2137 for m=num_ite_base:num_ite_base:num_ite_ran
2138     % initialized forward training
2139     % initialized mixing matrix
2140     mix_mat=sqrt(pow)*Gaussian_chan*nor_precoder;
2141     % statistical correlation matrix
2142     cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
2143     % Wiener estimated receive filter
2144     rece_fil=cor_mat\mix_mat;
2145     % power normalization
2146     rece_fil_nor=mat_pow_nor(rece_fil);
2147
2148     for m_d=1:m
2149         % backward optimization
2150         % mixing matrix
2151         mix_mat=transpose(sqrt(pow)*Gaussian_chan)* ...
            conj(rece_fil_nor);

```

```

2152         % statistical correlation matrix
2153         cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
2154         % Wiener estimated precoder
2155         precoder=conj(cor_mat\mix_mat);
2156         % power normalization
2157         precoder_nor=mat_pow_nor(precoder);
2158
2159         % forward optimization
2160         % mixing matrix
2161         mix_mat=sqrt(pow)*Gaussian_chan*precoder_nor;
2162         % statistical correlation matrix
2163         cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
2164         % Wiener estimated receive filter
2165         rece_fil=cor_mat\mix_mat;
2166         % power normalization
2167         rece_fil_nor=mat_pow_nor(rece_fil);
2168     end
2169
2170     % calculate MMSE & SINR
2171     % for each stream
2172     for n=1:num_stream
2173         % MMSE
2174         mmse_temp(1,n)=1-abs(ctranspose(rece_fil(:,n))* ...
2175             cor_mat*rece_fil(:,n));
2176         % SINR
2177         sinr_temp(1,n)=abs(ctranspose(mix_mat(:,n))*(( ...
2178             cor_mat-mix_mat(:,n)*ctranspose(mix_mat(:,n)))\ ...
2179             mix_mat(:,n)));
2180     end
2181
2182     % average over streams
2183     mmse(1,m/num_ite_base)=mean(mmse_temp,2);
2184     sinr(1,m/num_ite_base)=mean(sinr_temp,2);
2185 end
2186
2187 % decibel conversion
2188 mmse=10*log10(mmse);
2189 sinr=10*log10(sinr);
2190
2191 % number of iteration axis
2192 num_ite_leng=zeros(1,num_ite_ran/num_ite_base);
2193 for m=1:num_ite_ran/num_ite_base
2194     num_ite_leng(1,m)=m*num_ite_base;
2195 end
2196
2197 end

```

```

2198
2199 wiener_mmse_sinr_vs_numite_bidirec_rr.m:
2200
2201 % Wiener (optimal) estimated (MMSE & SINR) vs Number of ...
      bi-directional optimization iteration with reduced-rank
2202
2203 function [mmse,sinr,num_ite_leng] = ...
      wiener_mmse_sinr_vs_numite_bidirec_rr(pow,Gaussian_chan, ...
      nor_precoder,red_rank,num_ite_ran,num_ite_base,noise_covar)
2204
2205 % parameter extraction
2206 [Nr]=size(Gaussian_chan,1);
2207 % output container
2208 mmse=zeros(size(red_rank,2),num_ite_ran/num_ite_base);
2209 sinr=zeros(size(red_rank,2),num_ite_ran/num_ite_base);
2210
2211 % for each reduced-rank
2212 for rr_ind=1:size(red_rank,2)
2213     % assign current reduced-rank
2214     rr=red_rank(rr_ind);
2215
2216     % for each number of bi-directional training iteration
2217     for m=num_ite_base:num_ite_base:num_ite_ran
2218
2219         % initialized forward training
2220         % initialized mixing matrix
2221         mix_mat=sqrt(pow)*Gaussian_chan*nor_precoder;
2222         % statistical correlation matrix
2223         cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
2224         % Wiener estimated combiner
2225         [rece_fil,~,~]=wiener_rr_fil_mmse_sinr_cal(cor_mat, ...
            mix_mat,rr);
2226         % power normalization
2227         combiner_nor=mat_pow_nor(rece_fil);
2228
2229         for m_d=1:m
2230             % backward optimization
2231             % mixing matrix
2232             mix_mat=transpose(sqrt(pow)*Gaussian_chan)*conj( ...
                combiner_nor);
2233             % statistical correlation matrix
2234             cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar* ...
                eye(Nr);
2235             % Wiener estimated precoder
2236             [precoder,~,~]=wiener_rr_fil_mmse_sinr_cal(cor_mat, ...
                mix_mat,rr);
2237             precoder=conj(precoder);
2238             % power normalization
2239             precoder_nor=mat_pow_nor(precoder);

```

```

2240
2241         % forward optimization
2242         % mixing matrix
2243         mix_mat=sqrt(pow)*Gaussian_chan*precoder_nor;
2244         % statistical correlation matrix
2245         cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar* ...
            eye(Nr);
2246         % Wiener estimated combiner
2247         [rece_fil,mmse_val,sinr_val]= ...
            wiener_rr_fil_mmse_sinr_cal(cor_mat,mix_mat,rr);
2248         % power normalization
2249         combiner_nor=mat_pow_nor(rece_fil);
2250     end
2251     mmse(rr_ind,m/num_ite_base)=mmse_val;
2252     sinr(rr_ind,m/num_ite_base)=sinr_val;
2253 end
2254 end
2255
2256 % decibel conversion
2257 mmse=10*log10(mmse);
2258 sinr=10*log10(sinr);
2259
2260 % number of iteration axis
2261 num_ite_leng=zeros(1,num_ite_ran/num_ite_base);
2262 for m=1:num_ite_ran/num_ite_base
2263     num_ite_leng(1,m)=m*num_ite_base;
2264 end
2265
2266 end
2267
2268
2269
2270
2271 wiener_mmse_sinr_vs_snr_fr.m:
2272
2273 % Wiener (optimal) estimated MMSE & SINR vs SNR with full-rank
2274
2275 function [mmse,sinr,snr] = ...
            wiener_mmse_sinr_vs_snr_fr(pow,Gaussian_chan,nor_precoder, ...
            noise_covar)
2276
2277 % parameter extraction
2278 x_leng=size(pow,2);
2279 Nr=size(Gaussian_chan,1);
2280 num_stream=size(nor_precoder,2);
2281
2282 % output container
2283 mmse=zeros(1,x_leng);
2284 sinr=zeros(1,x_leng);

```

```

2285 snr=zeros(1,xleng);
2286
2287 % for each transmit power
2288 for m=1:xleng
2289     % temporary MMSE & SINR container
2290     mmse_temp=zeros(1,num_stream);
2291     sinr_temp=zeros(1,num_stream);
2292
2293     mix_mat=sqrt(pow(1,m))*Gaussian_chan*nor_precoder; ...
                % mixing matrix
2294     cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr); ...
                % correlation matrix
2295     rece_fil=cor_mat\mix_mat; ...
                % ...
                receive filter
2296
2297     % calculate MMSE & SINR
2298     % for each stream
2299     for n=1:num_stream
2300         % MMSE
2301         mmse_temp(1,n)=1-abs(ctranspose(rece_fil(:,n))* ...
                cor_mat*rece_fil(:,n));
2302         % SINR
2303         sinr_temp(1,n)=abs(ctranspose(mix_mat(:,n))*(( ...
                cor_mat-mix_mat(:,n)*ctranspose(mix_mat(:,n)))\ ...
                mix_mat(:,n)));
2304     end
2305
2306     % average over streams
2307     mmse(1,m)=mean(mmse_temp,2);
2308     sinr(1,m)=mean(sinr_temp,2);
2309     snr(1,m)=pow(1,m)/noise_covar;
2310 end
2311
2312 % decibal conversion
2313 mmse=10*log10(mmse);
2314 sinr=10*log10(sinr);
2315 snr=10*log10(snr);
2316
2317 end
2318
2319
2320
2321
2322 wiener_mmse_sinr_vs_snr_rr.m:
2323
2324 % Wiener (optimal) estimated MMSE & SINR vs SNR with ...
    reduced-rank
2325

```

```

2326 function [mmse,sinr,snr] = ...
        wiener_mmse_sinr_vs_snr_rr(pow,Gaussian_chan,nor_precoder, ...
        noise_covar,red_rank)
2327
2328 % parameter extraction
2329 xlen=size(pow,2);
2330 Nr=size(Gaussian_chan,1);
2331 num_stream=size(nor_precoder,2);
2332
2333 % output container
2334 mmse=zeros(size(red_rank,2),xlen);
2335 sinr=zeros(size(red_rank,2),xlen);
2336
2337 % for each reduced rank
2338 for rr_ind=1:size(red_rank,2)
2339     % reduced rank value
2340     rr=red_rank(rr_ind);
2341
2342     for m=1:xlen
2343         krylov=zeros(Nr,rr); ...
2344
2345         % Krylov subspace
2346         mix_mat=sqrt(pow(1,m))*Gaussian_chan*nor_precoder; ...
2347         % mixing matrix
2348         cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr); ...
2349         % correlation matrix
2350
2351         mmse_temp=zeros(1,num_stream);
2352         sinr_temp=zeros(1,num_stream);
2353         % for each stream
2354         for n=1:num_stream
2355             % Krylov subspace
2356             for n_dot=1:rr
2357                 krylov(:,n_dot)=cor_mat^(n_dot-1)*mix_mat(:,n);
2358             end
2359
2360             % MMSE
2361             tri_dia_cor_mat=ctranspose(krylov)*cor_mat*krylov; ...
2362             % tri-diagonal covariance matrix
2363             mmse_temp_val=1-abs(ctranspose(mix_mat(:,n))* ...
2364                 krylov*(tri_dia_cor_mat\ (ctranspose(krylov)))* ...
2365                 mix_mat(:,n));
2366             mmse_temp(1,n)=mmse_temp_val;
2367
2368             % SINR
2369             sinr_temp_val=abs(ctranspose(mix_mat(:,n))*krylov ...
2370                 ...
2371                 *((ctranspose(krylov)*(cor_mat- ...
2372                 mix_mat(:,n)*ctranspose(mix_mat ...

```



```

2364         (:,n))*krylov) ...
2365         \ctranspose(krylov))*mix_mat(:,n));
2366     end
2367
2368     % averaging over streams
2369     mmse(rr_ind,m)=mean(mmse_temp,2);
2370     sinr(rr_ind,m)=mean(sinr_temp,2);
2371 end
2372 end
2373
2374 % decibel conversion
2375 mmse=10*log10(mmse);
2376 sinr=10*log10(sinr);
2377 snr=19*log10(pow/noise_covar);
2378
2379 end
2380
2381
2382
2383
2384 wiener_rr_fil_mmse_sinr_cal.m:
2385
2386 % Wiener (optimal) reduced-rank filter, MMSE and SINR calculator
2387
2388 function [fil,mmse,sinr] = ...
2389     wiener_rr_fil_mmse_sinr_cal(corr_mat,mix_mat,rr)
2389
2390 % dimension extraction
2391 [Nr,num_stream]=size(mix_mat);
2392 % Krylov subspace preallocation
2393 krylov=zeros(Nr,rr);
2394 % filter preallocation
2395 fil=zeros(Nr,num_stream);
2396
2397 % temporary container
2398 mmse_temp=zeros(1,num_stream);
2399 sinr_temp=zeros(1,num_stream);
2400
2401 % for each transmitted stream
2402 for m=1:num_stream
2403
2404     % constructe Krylov subspace for the current transmitted ...
2405     stream
2406     for n=1:rr
2407         krylov(:,n)=corr_mat^(n-1)*mix_mat(:,m);
2408     end
2409
2409     % reduced-rank filter

```

```

2410     fil(:,m)=krylov*((ctranspose(krylov)*cor_mat*krylov) ...
2411         \ctranspose(krylov))*mix_mat(:,m);
2412
2413     % MSE
2414     tri_dia_cor_mat=ctranspose(krylov)*cor_mat*krylov; ...
                % tri-diagonal covariance matrix
2415     mmse_temp_val=1-abs(ctranspose(mix_mat(:,m))*krylov* ...
        (tri_dia_cor_mat\ctranspose(krylov))*mix_mat(:,m));
2416     mmse_temp(1,m)=mmse_temp_val;
2417
2418     % SINR
2419     sinr_temp_val=abs(ctranspose(mix_mat(:,m))*krylov ...
2420         *((ctranspose(krylov)*(cor_mat-mix_mat ...
        (:,m)*ctranspose(mix_mat(:,m)))*krylov) ...
        ...
2421         \ctranspose(krylov))*mix_mat(:,m));
2422
2423     sinr_temp(1,m)=sinr_temp_val;
2424 end
2425
2426 % averaging over all transmitted streams
2427 mmse=mean(mmse_temp,2);
2428 sinr=mean(sinr_temp,2);
2429
2430 end
2431
2432
2433
2434
2435 wiener_rr_fil.sumrate_in.cal.m:
2436
2437 % Interference network Wiener (optimal) reduced-rank filter, ...
    sum rate calculator
2438
2439 function [fil1,fil2,fil3,sumrate] = ...
    wiener_rr_fil.sumrate_in.cal(cor_mat1,mix_mat11, ...
    cor_mat2,mix_mat22, cor_mat3,mix_mat33,rr)
2440
2441 % dimension extraction
2442 [Nr,num_stream]=size(mix_mat11);
2443 % Krylov subspaces preallocation
2444 krylov1=zeros(Nr,rr);
2445 krylov2=zeros(Nr,rr);
2446 krylov3=zeros(Nr,rr);
2447 % filters preallocation
2448 fil1=zeros(Nr,num_stream);
2449 fil2=zeros(Nr,num_stream);
2450 fil3=zeros(Nr,num_stream);
2451

```

```

2452 % temporary container
2453 sumrate1=zeros(1,num_stream);
2454 sumrate2=zeros(1,num_stream);
2455 sumrate3=zeros(1,num_stream);
2456
2457 % for each transmitted stream
2458 for m=1:num_stream
2459
2460     % constructe Krylov subspaces for the current ...
        transmitted stream
2461     for n=1:rr
2462         krylov1(:,n)=cor_mat1^(n-1)*mix_mat11(:,m);
2463         krylov2(:,n)=cor_mat2^(n-1)*mix_mat22(:,m);
2464         krylov3(:,n)=cor_mat3^(n-1)*mix_mat33(:,m);
2465     end
2466
2467     % reduced-rank filters
2468     fil1(:,m)=krylov1*((ctranspose(krylov1)*cor_mat1*krylov1) ...
        ...
        \ctranspose(krylov1))*mix_mat11(:,m);
2469     fil2(:,m)=krylov2*((ctranspose(krylov2)*cor_mat2*krylov2) ...
        ...
        \ctranspose(krylov2))*mix_mat22(:,m);
2471     fil3(:,m)=krylov3*((ctranspose(krylov3)*cor_mat3*krylov3) ...
        ...
        \ctranspose(krylov3))*mix_mat33(:,m);
2473
2474
2475     % sum rate
2476     sumrate1(1,m)=log2(1+abs(ctranspose(mix_mat11(:,m))*krylov1 ...
        ...
        *((ctranspose(krylov1)*(cor_mat1-mix_mat11(:,m) ...
        *ctranspose(mix_mat11(:,m)))*krylov1) ...
        \ctranspose(krylov1))*mix_mat11(:,m)));
2478     sumrate2(1,m)=log2(1+abs(ctranspose(mix_mat22(:,m))*krylov2 ...
        ...
        *((ctranspose(krylov2)*(cor_mat2-mix_mat22(:,m) ...
        *ctranspose(mix_mat22(:,m)))*krylov2) ...
        \ctranspose(krylov2))*mix_mat22(:,m)));
2481     sumrate3(1,m)=log2(1+abs(ctranspose(mix_mat33(:,m))*krylov3 ...
        ...
        *((ctranspose(krylov3)*(cor_mat3-mix_mat33(:,m) ...
        *ctranspose(mix_mat33(:,m)))*krylov3) ...
        \ctranspose(krylov3))*mix_mat33(:,m)));
2484
2485 end
2486
2487 % network sum rate
2488 sumrate=sum(sumrate1)+sum(sumrate2)+sum(sumrate3);
2489
2490 end

```

```

2491
2492
2493
2494
2495 wiener_sumrate_fixednumite_bidirec_in_fr.m:
2496
2497 % Wiener (optimal) Sum rate for fixed number of ...
    bi-directional optimization iteration with full-rank
2498 % in interference network
2499
2500 function [sumrate,train_leng] = ...
    wiener_sumrate_fixednumite_bidirec_in_fr(pow,Gaussian_chan_mat, ...
    nor_precoder_mat,train_leng_ran,train_base,num_ite,noise_covar)
2501
2502 % parameter extraction
2503 Nr=size(squeeze(Gaussian_chan_mat(1,1,:,:)),1);
2504 num_stream=size(squeeze(nor_precoder_mat(1,:,:)),2);
2505 % output data
2506 sumrate=zeros(1,train_leng_ran);
2507
2508 % initialized forward optimization
2509 % receive filter updating, precoder --> receive filter
2510 % initialized mixing matrices
2511 mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:))* ...
    squeeze(nor_precoder_mat(1,:,:));
2512 mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:))* ...
    squeeze(nor_precoder_mat(2,:,:));
2513 mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:))* ...
    squeeze(nor_precoder_mat(3,:,:));
2514 mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:))* ...
    squeeze(nor_precoder_mat(1,:,:));
2515 mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:))* ...
    squeeze(nor_precoder_mat(2,:,:));
2516 mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:))* ...
    squeeze(nor_precoder_mat(3,:,:));
2517 mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:))* ...
    squeeze(nor_precoder_mat(1,:,:));
2518 mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:))* ...
    squeeze(nor_precoder_mat(2,:,:));
2519 mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:))* ...
    squeeze(nor_precoder_mat(3,:,:));
2520 % statistical correlation matrices
2521 cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
2522     +mix_mat21*ctranspose(mix_mat21) ...
2523     +mix_mat31*ctranspose(mix_mat31) ...
2524     +noise_covar*eye(Nr);
2525 cor_mat2=mix_mat12*ctranspose(mix_mat12) ...
2526     +mix_mat22*ctranspose(mix_mat22) ...
2527     +mix_mat32*ctranspose(mix_mat32) ...

```

```

2528         +noise_covar*eye(Nr);
2529 cor_mat3=mix_mat13*ctranspose(mix_mat13) ...
2530         +mix_mat23*ctranspose(mix_mat23) ...
2531         +mix_mat33*ctranspose(mix_mat33) ...
2532         +noise_covar*eye(Nr);
2533 % Wiener (optimal) receive filters with power normalization
2534 rece_fil1=mat_pow_nor(cor_mat1\mix_mat11);
2535 rece_fil2=mat_pow_nor(cor_mat2\mix_mat22);
2536 rece_fil3=mat_pow_nor(cor_mat3\mix_mat33);
2537
2538 % for each bi-directional optimization
2539 for m=1:num_ite
2540     % backward optimization
2541     % precoder updating, precoder <-- receive filter
2542     % mixing matrices
2543     mix_mat11=sqrt(pow)*transpose(squeeze(Gaussian_chan.mat ...
2544         (1,1, :, :))) *conj(rece_fil1);
2545     mix_mat12=sqrt(pow)*transpose(squeeze(Gaussian_chan.mat ...
2546         (2,1, :, :))) *conj(rece_fil2);
2547     mix_mat13=sqrt(pow)*transpose(squeeze(Gaussian_chan.mat ...
2548         (3,1, :, :))) *conj(rece_fil3);
2549     mix_mat21=sqrt(pow)*transpose(squeeze(Gaussian_chan.mat ...
2550         (1,2, :, :))) *conj(rece_fil1);
2551     mix_mat22=sqrt(pow)*transpose(squeeze(Gaussian_chan.mat ...
2552         (2,2, :, :))) *conj(rece_fil2);
2553     mix_mat23=sqrt(pow)*transpose(squeeze(Gaussian_chan.mat ...
2554         (3,2, :, :))) *conj(rece_fil3);
2555     mix_mat31=sqrt(pow)*transpose(squeeze(Gaussian_chan.mat ...
2556         (1,3, :, :))) *conj(rece_fil1);
2557     mix_mat32=sqrt(pow)*transpose(squeeze(Gaussian_chan.mat ...
2558         (2,3, :, :))) *conj(rece_fil2);
2559     mix_mat33=sqrt(pow)*transpose(squeeze(Gaussian_chan.mat ...
2560         (3,3, :, :))) *conj(rece_fil3);
2561 % statistical correlation matrices
2562 cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
2563         +mix_mat12*ctranspose(mix_mat12) ...
2564         +mix_mat13*ctranspose(mix_mat13) ...
2565         +noise_covar*eye(Nr);
2566 cor_mat2=mix_mat21*ctranspose(mix_mat21) ...
2567         +mix_mat22*ctranspose(mix_mat22) ...
2568         +mix_mat23*ctranspose(mix_mat23) ...
2569         +noise_covar*eye(Nr);
2570 cor_mat3=mix_mat31*ctranspose(mix_mat31) ...
2571         +mix_mat32*ctranspose(mix_mat32) ...
2572         +mix_mat33*ctranspose(mix_mat33) ...
2573         +noise_covar*eye(Nr);
2574 % Wiener (optimal) precoders with power normalization
2575 precoder1=mat_pow_nor(conj(cor_mat1\mix_mat11));
2576 precoder2=mat_pow_nor(conj(cor_mat2\mix_mat22));

```

```

2568     precoder3=mat_pow_nor(conj(cor_mat3\mix_mat33));
2569
2570     % forward optimization
2571     % for receiver, precoder --> receive filter
2572     % mixing matrices
2573     mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:)) ...
        *precoder1;
2574     mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:)) ...
        *precoder2;
2575     mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:)) ...
        *precoder3;
2576     mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:)) ...
        *precoder1;
2577     mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
        *precoder2;
2578     mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
        *precoder3;
2579     mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
        *precoder1;
2580     mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
        *precoder2;
2581     mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
        *precoder3;
2582     % statistical correlation matrices
2583     cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
        +mix_mat21*ctranspose(mix_mat21) ...
2584         +mix_mat31*ctranspose(mix_mat31) ...
2585         +noise_covar*eye(Nr);
2586     cor_mat2=mix_mat12*ctranspose(mix_mat12) ...
        +mix_mat22*ctranspose(mix_mat22) ...
2587         +mix_mat32*ctranspose(mix_mat32) ...
2588         +noise_covar*eye(Nr);
2589     cor_mat3=mix_mat13*ctranspose(mix_mat13) ...
        +mix_mat23*ctranspose(mix_mat23) ...
2590         +mix_mat33*ctranspose(mix_mat33) ...
2591         +noise_covar*eye(Nr);
2592     % Wiener (optimal) receive filters with power normalization
2593     rece_fil1=mat_pow_nor(cor_mat1\mix_mat11);
2594     rece_fil2=mat_pow_nor(cor_mat2\mix_mat22);
2595     rece_fil3=mat_pow_nor(cor_mat3\mix_mat33);
2596 end
2597
2600
2601 % data preallocation
2602 sumrate1=zeros(1,num_stream);
2603 sumrate2=zeros(1,num_stream);
2604 sumrate3=zeros(1,num_stream);
2605
2606 % for each stream
2607 for m=1:num_stream

```

```

2608     % calculate sum rate
2609     sumrate1(1,m)=log2(1+abs(ctranspose(mix_mat11(:,m))* ...
        ((cor_mat1-mix_mat11(:,m)*ctranspose(mix_mat11(:,m))) ...
        \mix_mat11(:,m))));
2610     sumrate2(1,m)=log2(1+abs(ctranspose(mix_mat22(:,m))* ...
        ((cor_mat2-mix_mat22(:,m)*ctranspose(mix_mat22(:,m))) ...
        \mix_mat22(:,m))));
2611     sumrate3(1,m)=log2(1+abs(ctranspose(mix_mat33(:,m))* ...
        ((cor_mat3-mix_mat33(:,m)*ctranspose(mix_mat33(:,m))) ...
        \mix_mat33(:,m))));
2612 end
2613
2614 % network sum rate
2615 sumrate_val=sum(sumrate1)+sum(sumrate2)+sum(sumrate3);
2616
2617 % training length axis
2618 train_leng=zeros(1,train_leng_ran);
2619 for m=1:train_leng_ran
2620     sumrate(1,m)=sumrate_val;
2621     train_leng(1,m)=m*train_base;
2622 end
2623
2624 end
2625
2626
2627
2628
2629 wiener_sumrate_fixednumite_bidirec_in_rr.m:
2630
2631 % Wiener (optimal) Sum rate for fixed number of ...
    bi-directional optimization iteration with reduced-rank
2632 % in interference network
2633
2634 function [sumrate,train_leng] = ...
    wiener_sumrate_fixednumite_bidirec_in_rr(pow,Gaussian_chan_mat, ...
    nor_precoder_mat,red_rank,train_leng_ran,train_base,num_ite, ...
    noise_covar)
2635
2636 % parameter extraction
2637 Nr=size(squeeze(Gaussian_chan_mat(1,1,:,:)),1);
2638 % temporary container
2639 sumrate_temp=zeros(1,size(red_rank,2));
2640
2641 % for each reduced-rank
2642 for rr_ind=1:size(red_rank,2)
2643     % reduced-rank value
2644     rr=red_rank(rr_ind);
2645
2646     % initialized forward optimization

```

```

2647 % receive filter updating, precoder --> receive filter
2648 % initialized mixing matrices
2649 mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:)) ...
      *squeeze(nor_precoder_mat(1,:,:));
2650 mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:)) ...
      *squeeze(nor_precoder_mat(2,:,:));
2651 mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:)) ...
      *squeeze(nor_precoder_mat(3,:,:));
2652 mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:)) ...
      *squeeze(nor_precoder_mat(1,:,:));
2653 mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
      *squeeze(nor_precoder_mat(2,:,:));
2654 mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
      *squeeze(nor_precoder_mat(3,:,:));
2655 mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
      *squeeze(nor_precoder_mat(1,:,:));
2656 mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
      *squeeze(nor_precoder_mat(2,:,:));
2657 mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
      *squeeze(nor_precoder_mat(3,:,:));
2658 % statistical correlation matrices
2659 cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
      +mix_mat21*ctranspose(mix_mat21) ...
      +mix_mat31*ctranspose(mix_mat31) ...
      +noise_covar*eye(Nr);
2660
2661 cor_mat2=mix_mat12*ctranspose(mix_mat12) ...
      +mix_mat22*ctranspose(mix_mat22) ...
      +mix_mat32*ctranspose(mix_mat32) ...
      +noise_covar*eye(Nr);
2662
2663 cor_mat3=mix_mat13*ctranspose(mix_mat13) ...
      +mix_mat23*ctranspose(mix_mat23) ...
      +mix_mat33*ctranspose(mix_mat33) ...
      +noise_covar*eye(Nr);
2664
2665 % reduced-rank Wiener (optimal) receive filters
2666 [rece_fil1,rece_fil2,rece_fil3,-]= ...
      wiener_rr_fil.sumrate_in_cal(cor_mat1,mix_mat11, ...
      cor_mat2,mix_mat22, cor_mat3,mix_mat33,rr);
2667
2668 rece_fil1=mat_pow_nor(rece_fil1);
2669 rece_fil2=mat_pow_nor(rece_fil2);
2670 rece_fil3=mat_pow_nor(rece_fil3);
2671
2672 % for each bi-directional optimization
2673 for m=1:num_ite
2674     % backward optimization
2675     % precoder updating, precoder <-- receive filter
2676     % mixing matrices
2677     mix_mat11=sqrt(pow)*transpose(squeeze ...
      (Gaussian_chan_mat(1,1,:,:))*conj(rece_fil1));
2678     mix_mat12=sqrt(pow)*transpose(squeeze ...

```



```

2684         (Gaussian_chan_mat(2,1,,:)))*conj(rece_fil2);
2685     mix_mat13=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(3,1,,:)))*conj(rece_fil3);
2686     mix_mat21=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(1,2,,:)))*conj(rece_fil1);
2687     mix_mat22=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(2,2,,:)))*conj(rece_fil2);
2688     mix_mat23=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(3,2,,:)))*conj(rece_fil3);
2689     mix_mat31=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(1,3,,:)))*conj(rece_fil1);
2690     mix_mat32=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(2,3,,:)))*conj(rece_fil2);
2691     mix_mat33=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(3,3,,:)))*conj(rece_fil3);
2692 % statistical correlation matrices
2693 cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
2694         +mix_mat12*ctranspose(mix_mat12) ...
2695         +mix_mat13*ctranspose(mix_mat13) ...
2696         +noise_covar*eye(Nr);
2697 cor_mat2=mix_mat21*ctranspose(mix_mat21) ...
2698         +mix_mat22*ctranspose(mix_mat22) ...
2699         +mix_mat23*ctranspose(mix_mat23) ...
2700         +noise_covar*eye(Nr);
2701 cor_mat3=mix_mat31*ctranspose(mix_mat31) ...
2702         +mix_mat32*ctranspose(mix_mat32) ...
2703         +mix_mat33*ctranspose(mix_mat33) ...
2704         +noise_covar*eye(Nr);
2705 % reduced-rank Wiener (optimal) precoders
2706 [precoder1,precoder2,precoder3,-]= ...
        wiener_rr_fil_sumrate_in_cal(cor_mat1,mix_mat11, ...
        cor_mat2,mix_mat22, cor_mat3,mix_mat33,rr);
2707 precoder1=mat_pow_nor(conj(precoder1));
2708 precoder2=mat_pow_nor(conj(precoder2));
2709 precoder3=mat_pow_nor(conj(precoder3));
2710 % forward optimization
2711 % for receiver, precoder --> receive filter
2712 % mixing matrices
2713 mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (1,1,,:))*precoder1;
2714 mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (2,1,,:))*precoder2;
2715 mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (3,1,,:))*precoder3;
2716 mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (1,2,,:))*precoder1;
2717 mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (2,2,,:))*precoder2;

```

```

2718     mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (3,2,:,:) * precoder3;
2719     mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (1,3,:,:) * precoder1;
2720     mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (2,3,:,:) * precoder2;
2721     mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (3,3,:,:) * precoder3;
2722     % statistical correlation matrices
2723     cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
        +mix_mat21*ctranspose(mix_mat21) ...
2724     +mix_mat31*ctranspose(mix_mat31) ...
2725     +noise_covar*eye(Nr);
2726     cor_mat2=mix_mat12*ctranspose(mix_mat12) ...
        +mix_mat22*ctranspose(mix_mat22) ...
2727     +mix_mat32*ctranspose(mix_mat32) ...
2728     +noise_covar*eye(Nr);
2729     cor_mat3=mix_mat13*ctranspose(mix_mat13) ...
        +mix_mat23*ctranspose(mix_mat23) ...
2730     +mix_mat33*ctranspose(mix_mat33) ...
2731     +noise_covar*eye(Nr);
2732     % reduced-rank Wiener (optimal) receive filters
2733     [rece_fil1,rece_fil2,rece_fil3,sumrate_val]= ...
        wiener_rr_fil.sumrate_in_cal(cor_mat1,mix_mat11, ...
        cor_mat2,mix_mat22, cor_mat3,mix_mat33,rr);
2734     rece_fil1=mat_pow_nor(rece_fil1);
2735     rece_fil2=mat_pow_nor(rece_fil2);
2736     rece_fil3=mat_pow_nor(rece_fil3);
2737     end
2738
2739     sumrate_temp(1,rr_ind)=sumrate_val;
2740     end
2741
2742     % training length axis
2743     train_leng=zeros(1,train_leng_ran);
2744     sumrate=zeros(size(red_rank,2),train_leng_ran);
2745     for m=1:train_leng_ran
2746         sumrate(1,m)=sumrate_temp(1,1);
2747         sumrate(2,m)=sumrate_temp(1,2);
2748         train_leng(1,m)=m*train_base;
2749     end
2750
2751     end
2752
2753     end
2754
2755     wiener_sumrate_vs_numite_bidirec_in_fr.m:
2756
2757
2758
2759
2760

```

```

2761 % Wiener (optimal) estimated Sum rate vs Number of ...
      bi-directional optimization iteration
2762 % with full-rank in interference network
2763
2764 function [sumrate,num_ite_leng] = ...
      wiener_sumrate_vs_numite_bidirec_in_fr(pow,Gaussian_chan_mat, ...
      nor_precoder_mat,num_ite_ran,num_ite_base,noise_covar)
2765
2766 % parameter extraction
2767 Nr=size(squeeze(Gaussian_chan_mat(1,1,:,:)),1);
2768 num_stream=size(squeeze(nor_precoder_mat(1,:,:)),2);
2769 % temporary containers
2770 sumrate1=zeros(1,num_stream);
2771 sumrate2=zeros(1,num_stream);
2772 sumrate3=zeros(1,num_stream);
2773 % output data
2774 sumrate=zeros(1,num_ite_ran/num_ite_base);
2775
2776 % for each number of bi-directional optimization iteration
2777 for m=num_ite_base:num_ite_base:num_ite_ran
2778     % initialized forward training
2779     % receive filter updating, precoder --> receive filter
2780     % initialized mixing matrices
2781     mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:)) ...
        *squeeze(nor_precoder_mat(1,:,:));
2782     mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:)) ...
        *squeeze(nor_precoder_mat(2,:,:));
2783     mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:)) ...
        *squeeze(nor_precoder_mat(3,:,:));
2784     mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:)) ...
        *squeeze(nor_precoder_mat(1,:,:));
2785     mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
        *squeeze(nor_precoder_mat(2,:,:));
2786     mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
        *squeeze(nor_precoder_mat(3,:,:));
2787     mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
        *squeeze(nor_precoder_mat(1,:,:));
2788     mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
        *squeeze(nor_precoder_mat(2,:,:));
2789     mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
        *squeeze(nor_precoder_mat(3,:,:));
2790     % statistical correlation matrices
2791     cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
        +mix_mat21*ctranspose(mix_mat21) ...
2792     +mix_mat31*ctranspose(mix_mat31) ...
2793     +noise_covar*eye(Nr);
2794     cor_mat2=mix_mat12*ctranspose(mix_mat12) ...
2795     +mix_mat22*ctranspose(mix_mat22) ...
2796     +mix_mat32*ctranspose(mix_mat32) ...
2797

```

```

2798         +noise_covar*eye(Nr);
2799     cor_mat3=mix_mat13*ctranspose(mix_mat13) ...
2800         +mix_mat23*ctranspose(mix_mat23) ...
2801         +mix_mat33*ctranspose(mix_mat33) ...
2802         +noise_covar*eye(Nr);
2803     % Wiener estimated receive filters with power normalization
2804     rece_fil1=mat_pow_nor(cor_mat1\mix_mat11);
2805     rece_fil2=mat_pow_nor(cor_mat2\mix_mat22);
2806     rece_fil3=mat_pow_nor(cor_mat3\mix_mat33);
2807
2808     for m_d=1:m
2809         % backward optimization
2810         % precoder updating, precoder <-- receive filter
2811         % mixing matrices
2812         mix_mat11=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
2813             (1,1,:,:)))*conj(rece_fil1);
2814         mix_mat12=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
2815             (2,1,:,:)))*conj(rece_fil2);
2816         mix_mat13=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
2817             (3,1,:,:)))*conj(rece_fil3);
2818         mix_mat21=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
2819             (1,2,:,:)))*conj(rece_fil1);
2820         mix_mat22=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
2821             (2,2,:,:)))*conj(rece_fil2);
2822         mix_mat23=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
2823             (3,2,:,:)))*conj(rece_fil3);
2824         mix_mat31=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
2825             (1,3,:,:)))*conj(rece_fil1);
2826         mix_mat32=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
2827             (2,3,:,:)))*conj(rece_fil2);
2828         mix_mat33=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
2829             (3,3,:,:)))*conj(rece_fil3);
2830         % statistical correlation matrices
2831         cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
2832             +mix_mat12*ctranspose(mix_mat12) ...
2833             +mix_mat13*ctranspose(mix_mat13) ...
2834             +noise_covar*eye(Nr);
2835         cor_mat2=mix_mat21*ctranspose(mix_mat21) ...
2836             +mix_mat22*ctranspose(mix_mat22) ...
2837             +mix_mat23*ctranspose(mix_mat23) ...
2838             +noise_covar*eye(Nr);
2839         cor_mat3=mix_mat31*ctranspose(mix_mat31) ...
2840             +mix_mat32*ctranspose(mix_mat32) ...
2841             +mix_mat33*ctranspose(mix_mat33) ...
2842             +noise_covar*eye(Nr);
2843         % Wiener estimated precoders
2844         precoder1=mat_pow_nor(conj(cor_mat1\mix_mat11));
2845         precoder2=mat_pow_nor(conj(cor_mat2\mix_mat22));
2846         precoder3=mat_pow_nor(conj(cor_mat3\mix_mat33));

```

```

2838
2839 % forward optimization
2840 % for receiver, precoder --> receive filter
2841 % mixing matrices
2842 mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:)) ...
      *precoder1;
2843 mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:)) ...
      *precoder2;
2844 mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:)) ...
      *precoder3;
2845 mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:)) ...
      *precoder1;
2846 mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
      *precoder2;
2847 mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
      *precoder3;
2848 mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
      *precoder1;
2849 mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
      *precoder2;
2850 mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
      *precoder3;
2851 % statistical correlation matrices
2852 cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
      +mix_mat21*ctranspose(mix_mat21) ...
2853 +mix_mat31*ctranspose(mix_mat31) ...
2854 +noise_covar*eye(Nr);
2855
2856 cor_mat2=mix_mat12*ctranspose(mix_mat12) ...
      +mix_mat22*ctranspose(mix_mat22) ...
2857 +mix_mat32*ctranspose(mix_mat32) ...
2858 +noise_covar*eye(Nr);
2859
2860 cor_mat3=mix_mat13*ctranspose(mix_mat13) ...
      +mix_mat23*ctranspose(mix_mat23) ...
2861 +mix_mat33*ctranspose(mix_mat33) ...
2862 +noise_covar*eye(Nr);
2863
2864 % Wiener estimated receive filters with power ...
      normalization
2865 rece_fil1=mat_pow_nor(cor_mat1\mix_mat11);
2866 rece_fil2=mat_pow_nor(cor_mat2\mix_mat22);
2867 rece_fil3=mat_pow_nor(cor_mat3\mix_mat33);
2868 end
2869
2870 % for each stream
2871 for n=1:num_stream
2872 % calculate sum rate
2873 sumrate1(1,n)=log2(1+abs(ctranspose(mix_mat11(:,n)) ...
      *((cor_mat1-mix_mat11(:,n)*ctranspose(mix_mat11(:,n))) ...
      \mix_mat11(:,n))));
2874 sumrate2(1,n)=log2(1+abs(ctranspose(mix_mat22(:,n)) ...

```

```

                * ((cor_mat2-mix_mat22(:,n)*ctranspose(mix_mat22(:,n))) ...
                \mix_mat22(:,n))));
2875    sumrate3(1,n)=log2(1+abs(ctranspose(mix_mat33(:,n))* ...
                ((cor_mat3-mix_mat33(:,n)*ctranspose(mix_mat33(:,n))) ...
                \mix_mat33(:,n))));
2876    end
2877
2878    % network sum rate
2879    sumrate(1,m/num_ite_base)=sum(sumrate1)+sum(sumrate2)+ ...
        sum(sumrate3);
2880 end
2881
2882 % number of iteration axis
2883 num_ite_leng=zeros(1,num_ite_ran/num_ite_base);
2884 for m=1:num_ite_ran/num_ite_base
2885     num_ite_leng(1,m)=m*num_ite_base;
2886 end
2887
2888 end
2889
2890
2891
2892
2893 wiener_sumrate_vs_numite_bidirec_in_rr.m:
2894
2895 % Wiener (optimal) estimated Sum rate vs Number of ...
    bi-directional optimization iteration
2896 % with reduced-rank in interference network
2897
2898 function [sumrate,num_ite_leng] = ...
    wiener_sumrate_vs_numite_bidirec_in_rr(pow,Gaussian_chan_mat, ...
    nor_precoder_mat,red_rank,num_ite_ran,num_ite_base,noise_covar)
2899
2900 % parameter extraction
2901 Nr=size(squeeze(Gaussian_chan_mat(1,1,:,:)),1);
2902 % output container
2903 sumrate=zeros(size(red_rank,2),num_ite_ran/num_ite_base);
2904
2905 % for each reduced-rank
2906 for rr_ind=1:size(red_rank,2)
2907     % assign current reduced-rank
2908     rr=red_rank(rr_ind);
2909
2910     % for each number of bi-directional training iteration
2911     for m=num_ite_base:num_ite_base:num_ite_ran
2912
2913         % initialized forward training
2914         % receive filter updating, precoder --> receive filter
2915         % initialized mixing matrices

```

```

2916 mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:)) ...
      *squeeze(nor_precoder_mat(1,:,:));
2917 mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:)) ...
      *squeeze(nor_precoder_mat(2,:,:));
2918 mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:)) ...
      *squeeze(nor_precoder_mat(3,:,:));
2919 mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:)) ...
      *squeeze(nor_precoder_mat(1,:,:));
2920 mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
      *squeeze(nor_precoder_mat(2,:,:));
2921 mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
      *squeeze(nor_precoder_mat(3,:,:));
2922 mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
      *squeeze(nor_precoder_mat(1,:,:));
2923 mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
      *squeeze(nor_precoder_mat(2,:,:));
2924 mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
      *squeeze(nor_precoder_mat(3,:,:));
2925 % statistical correlation matrices
2926 cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
      +mix_mat21*ctranspose(mix_mat21) ...
2927 +mix_mat31*ctranspose(mix_mat31) ...
2928 +noise_covar*eye(Nr);
2929 cor_mat2=mix_mat12*ctranspose(mix_mat12) ...
      +mix_mat22*ctranspose(mix_mat22) ...
2930 +mix_mat32*ctranspose(mix_mat32) ...
2931 +noise_covar*eye(Nr);
2932 cor_mat3=mix_mat13*ctranspose(mix_mat13) ...
      +mix_mat23*ctranspose(mix_mat23) ...
2933 +mix_mat33*ctranspose(mix_mat33) ...
2934 +noise_covar*eye(Nr);
2935 % reduced-rank Wiener (optimal) receive filters
2936 [rece_fil1,rece_fil2,rece_fil3,-]= ...
2937 wiener_rr_fil.sumrate.in.cal(cor_mat1,mix_mat11, ...
      cor_mat2,mix_mat22, cor_mat3,mix_mat33,rr);
2938 rece_fil1=mat_pow_nor(rece_fil1);
2939 rece_fil2=mat_pow_nor(rece_fil2);
2940 rece_fil3=mat_pow_nor(rece_fil3);
2941
2942 for m_d=1:m
2943
2944     % backward optimization
2945     % precoder updating, precoder <-- receive filter
2946     % mixing matrices
2947 mix_mat11=sqrt(pow)*transpose(squeeze ...
      (Gaussian_chan_mat(1,1,:,:))*conj(rece_fil1);
2948 mix_mat12=sqrt(pow)*transpose(squeeze ...
      (Gaussian_chan_mat(2,1,:,:))*conj(rece_fil2);
2949 mix_mat13=sqrt(pow)*transpose(squeeze ...

```

```

2952         (Gaussian_chan_mat(3,1,,:)))*conj(rece_fil3);
mix_mat21=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(1,2,,:)))*conj(rece_fil1);
2953 mix_mat22=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(2,2,,:)))*conj(rece_fil2);
2954 mix_mat23=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(3,2,,:)))*conj(rece_fil3);
2955 mix_mat31=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(1,3,,:)))*conj(rece_fil1);
2956 mix_mat32=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(2,3,,:)))*conj(rece_fil2);
2957 mix_mat33=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(3,3,,:)))*conj(rece_fil3);
2958 % statistical correlation matrices
2959 cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
2960         +mix_mat12*ctranspose(mix_mat12) ...
2961         +mix_mat13*ctranspose(mix_mat13) ...
2962         +noise_covar*eye(Nr);
2963 cor_mat2=mix_mat21*ctranspose(mix_mat21) ...
2964         +mix_mat22*ctranspose(mix_mat22) ...
2965         +mix_mat23*ctranspose(mix_mat23) ...
2966         +noise_covar*eye(Nr);
2967 cor_mat3=mix_mat31*ctranspose(mix_mat31) ...
2968         +mix_mat32*ctranspose(mix_mat32) ...
2969         +mix_mat33*ctranspose(mix_mat33) ...
2970         +noise_covar*eye(Nr);
2971 % reduced-rank Wiener (optimal) precoders
2972 [precoder1,precoder2,precoder3,-]= ...
        wiener_rr_fil_sumrate_in_cal(cor_mat1,mix_mat11, ...
        cor_mat2,mix_mat22, cor_mat3,mix_mat33,rr);
2973 precoder1=mat_pow_nor(conj(precoder1));
2974 precoder2=mat_pow_nor(conj(precoder2));
2975 precoder3=mat_pow_nor(conj(precoder3));
2976
2977 % forward optimization
2978 % for receiver, precoder --> receive filter
2979 % mixing matrices
2980 mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (1,1,,:))*precoder1;
2981 mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (2,1,,:))*precoder2;
2982 mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (3,1,,:))*precoder3;
2983 mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (1,2,,:))*precoder1;
2984 mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (2,2,,:))*precoder2;
2985 mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (3,2,,:))*precoder3;

```



```

2986         mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat ...
           (1,3,:,:))*precoder1;
2987         mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat ...
           (2,3,:,:))*precoder2;
2988         mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat ...
           (3,3,:,:))*precoder3;
2989         % statistical correlation matrices
2990         cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
           +mix_mat21*ctranspose(mix_mat21) ...
2991         +mix_mat31*ctranspose(mix_mat31) ...
           +noise_covar*eye(Nr);
2992         cor_mat2=mix_mat12*ctranspose(mix_mat12) ...
           +mix_mat22*ctranspose(mix_mat22) ...
2993         +mix_mat32*ctranspose(mix_mat32) ...
           +noise_covar*eye(Nr);
2994         cor_mat3=mix_mat13*ctranspose(mix_mat13) ...
           +mix_mat23*ctranspose(mix_mat23) ...
2995         +mix_mat33*ctranspose(mix_mat33) ...
           +noise_covar*eye(Nr);
2996         % reduced-rank Wiener (optimal) receive filters
2997         [rece_fil1,rece_fil2,rece_fil3,sumrate_val]= ...
           wiener_rr_fil_sumrate_in_cal(cor_mat1,mix_mat11, ...
           ...
3004
3005
3006         rece_fil1=mat_pow_nor(rece_fil1);
3007         rece_fil2=mat_pow_nor(rece_fil2);
3008         rece_fil3=mat_pow_nor(rece_fil3);
3009     end
3010     sumrate(rr_ind,m/num_ite_base)=sumrate_val;
3011 end
3012 end
3013
3014 % number of iteration axis
3015 num_ite_leng=zeros(1,num_ite_ran/num_ite_base);
3016 for m=1:num_ite_ran/num_ite_base
3017     num_ite_leng(1,m)=m*num_ite_base;
3018 end
3019
3020 end

```