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

List of Figures

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.

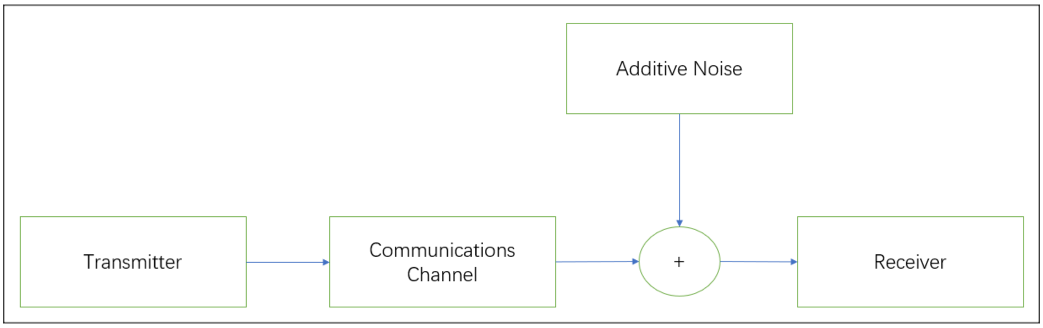

Figure 2.1: Basic model of communications systems

In Figure 2.1, the message is initially sent from the transmitter, therewith 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.

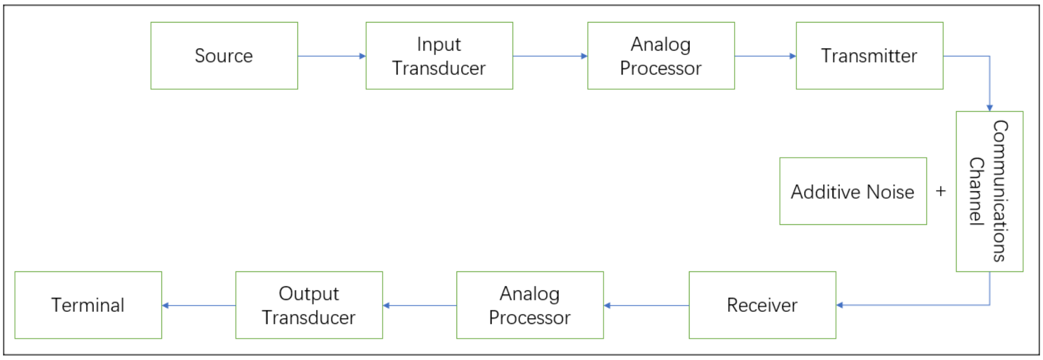

Figure 2.2: Model of electronic communications systems

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.

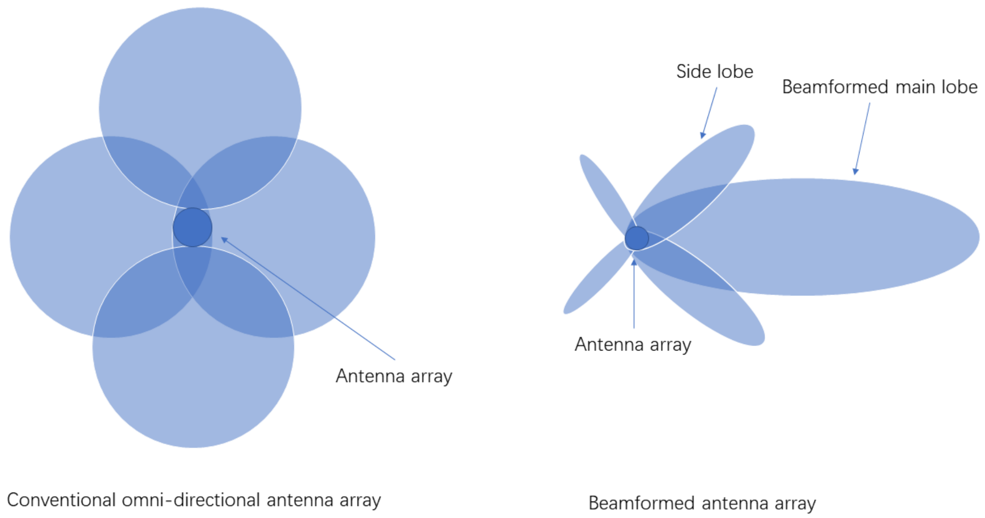

Figure 2.3: Conventional omni-directional antenna array vs Beamformed antenna array

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.

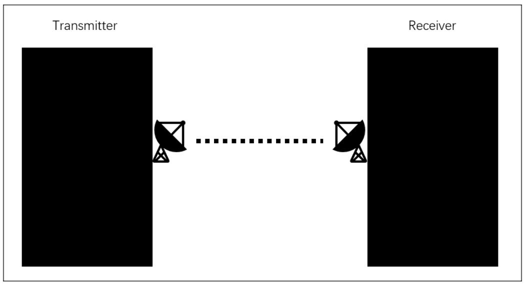

Figure 2.4: Single-input single-output systems

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 throughout is severely limited by the scale of the antenna array.

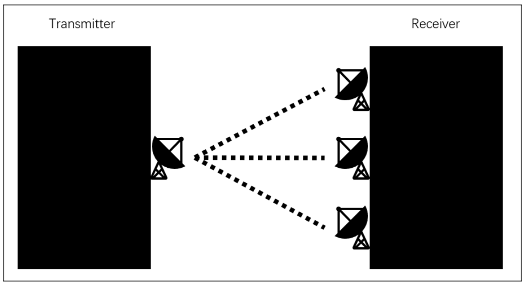

Figure 2.5: Single-input multi-output systems

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.

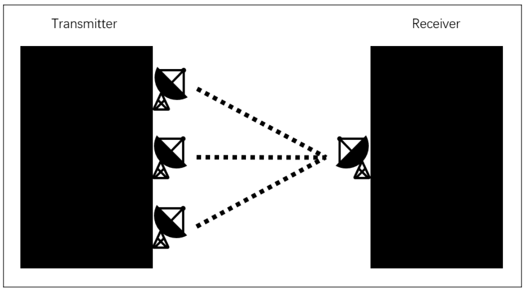

Figure 2.6: Multi-input single-output systems

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.

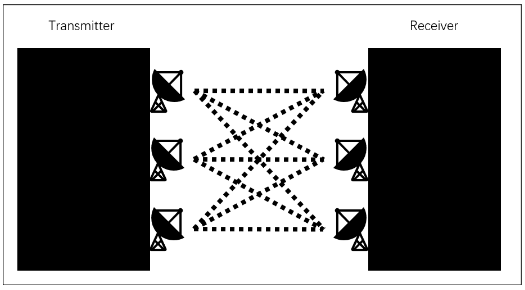

Figure 2.7: Multi-input multi-output systems

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:

$$\mathbf{y} = \mathbf{H}\mathbf{x}+\mathbf{n} \tag{2.1}$$

Where received signal $\mathbf{y} = \begin{bmatrix} y_1 & y_2 & y_3 & \dots & \dots & y_{Nr} \end{bmatrix}^T$, transmitted signal $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & \dots & x_{Nt} \end{bmatrix}^T$, 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. $\mathbf{n} = \begin{bmatrix} n_1 & n_2 & n_3 & \dots & \dots & n_{Nr} \end{bmatrix}^T$ 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.

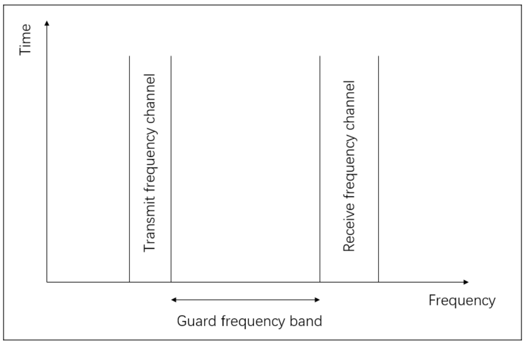

Figure 2.8: Frequency division duplex

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.

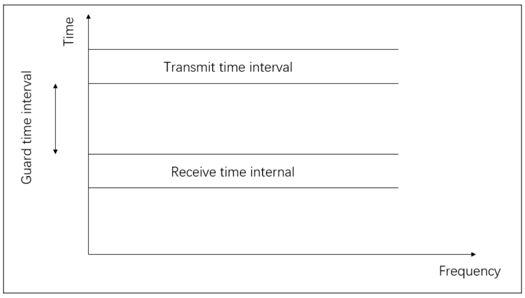

Figure 2.9: Time division duplex

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.

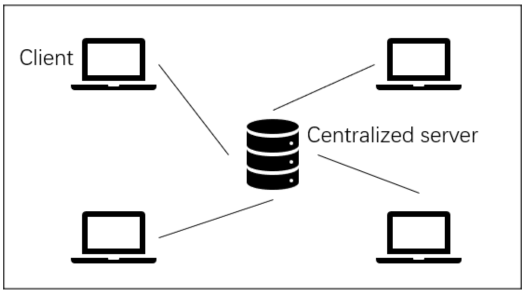

Figure 2.10: Centralized configuration

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.

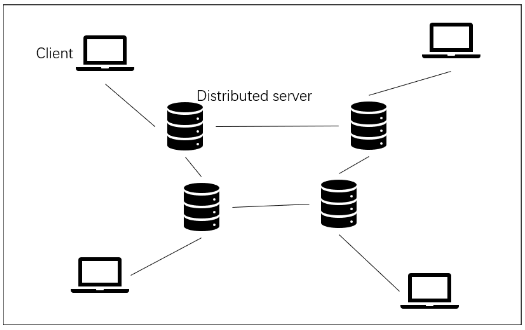

Figure 2.11: Distributed configuration

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 transmitted 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 ϵ, 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 ϵ [42].

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

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

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 $\frac{\mathbf{S}}{\mathbf{I+N}}$ 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.

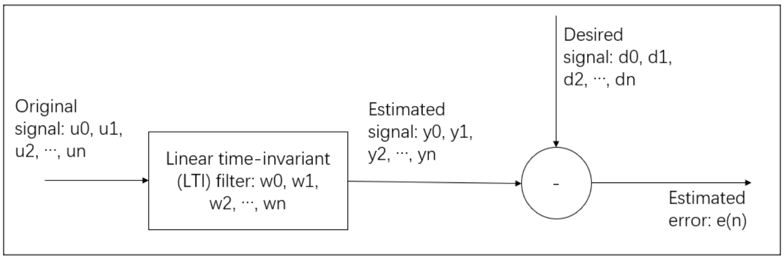

Figure 3.1: Digital estimator

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.

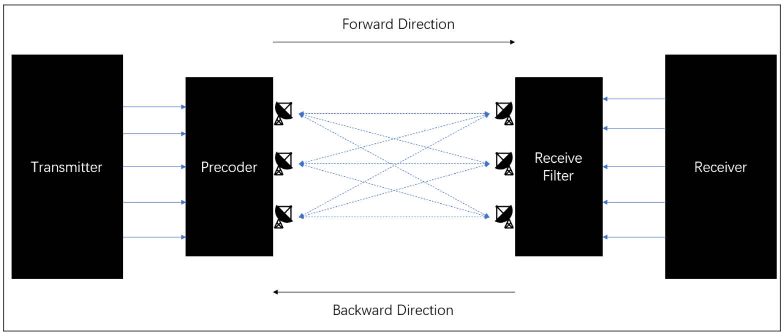

Figure 3.2: MIMO communications systems with precoding and receive filtering

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:

$$\begin{aligned} \mathbf{u}(i) &= \sqrt{P}\mathbf{H}\mathbf{v}\mathbf{b}(i) + \mathbf{n}(i) & (3.1) \\ &= \mathbf{M}\mathbf{b}(i) + \mathbf{n}(i) & (3.2) \end{aligned}$$

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 kth transmitted signal symbol $b_k(i)$, which is the kth element of the vector $\mathbf{b}(i)$, is expressed as:

$$\mathbf{g}_k = \mathbf{R}^{-1}\mathbf{p}_k \tag{3.3}$$

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 kth transmitted signal symbol is given as:

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

The estimated error for the kth transmitted signal symbol is:

$$\begin{aligned} e_k(i) &= b_k(i) - \hat{b_k(i)} && (3.5) \\ &= b_k(i) - \mathbf{g}_k^H \mathbf{u}(i) && (3.6) \end{aligned}$$

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:

$$\begin{aligned} E[\mathbf{b}(i)\mathbf{b}^H(i)] &= \mathbf{I}_{N_s} && (3.7) \\ E[\mathbf{b}(i)\mathbf{n}^H(i)] &= \mathbf{O}_{N_s,N_r} && (3.8) \\ E[\mathbf{n}(i)\mathbf{b}^H(i)] &= \mathbf{O}_{N_r,N_s} && (3.9) \\ E[\mathbf{n}(i)\mathbf{n}^H(i)] &= 0.01\mathbf{I}_{N_r} && (3.10) \end{aligned}$$

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:

$$
\begin{aligned}
\mathbf{R}_{wiener} &= E[\mathbf{u}(i)\mathbf{u}^H(i)] && (3.11)\\
&= E[[\sqrt{P}\mathbf{H}\mathbf{v}\mathbf{b}(i)+\mathbf{n}(i)][\sqrt{P}\mathbf{H}\mathbf{v}\mathbf{b}(i)+\mathbf{n}(i)]^H] && (3.12)\\
&= E[[\mathbf{M}\mathbf{b}(i)+\mathbf{n}(i)][\mathbf{M}\mathbf{b}(i)+\mathbf{n}(i)]^H] && (3.13)\\
&= E[[\mathbf{M}\mathbf{b}(i)+\mathbf{n}(i)][\mathbf{b}^H(i)\mathbf{M}^H+\mathbf{n}^H(i)]] && (3.14)\\
&= E[\mathbf{M}\mathbf{b}(i)\mathbf{b}^H(i)\mathbf{M}^H]+E[\mathbf{M}\mathbf{b}(i)\mathbf{n}^H(i)]+E[\mathbf{n}(i)\mathbf{b}^H(i)\mathbf{M}^H]+E[\mathbf{n}(i)\mathbf{n}^H(i)] && (3.15)\\
&= \mathbf{M}E[\mathbf{b}(i)\mathbf{b}^H(i)]\mathbf{M}^H+\mathbf{M}E[\mathbf{b}(i)\mathbf{n}^H(i)]+E[\mathbf{n}(i)\mathbf{b}^H(i)]\mathbf{M}^H+E[\mathbf{n}(i)\mathbf{n}^H(i)] && (3.16)\\
&= \mathbf{M}\mathbf{M}^H+0.01\mathbf{I}_{N_r} && (3.17)
\end{aligned}
$$

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

$$
\begin{aligned}
\mathbf{p}_{wiener_k} &= E[\mathbf{u}(i)b_k^*(i)] && (3.18)\\
&= E[[\sqrt{P}\mathbf{H}\mathbf{v}\mathbf{b}(i)+\mathbf{n}(i)]b_k^*(i)] && (3.19)\\
&= E[[\mathbf{M}\mathbf{b}(i)+\mathbf{n}(i)]b_k^*(i)] && (3.20)\\
&= E[\mathbf{M}\mathbf{b}(i)b_k^*(i)]+E[\mathbf{n}(i)b_k^*(i)] && (3.21)\\
&= \mathbf{M}E[\mathbf{b}(i)b_k^*(i)]+E[\mathbf{n}(i)b_k^*(i)] && (3.22)
\end{aligned}
$$

Where $E[\mathbf{b}(i)b_k^*(i)]$ is a N_s-by-1 vertical vector, in which only the kth 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:

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

Where the N_r-by-1 vector $\mathbf{M}_k$ is the kth 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 kth 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 kth transmitted signal symbol $b_k(i)$ is expressed as:

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

Thus the reconstructed kth transmitted signal symbol at time i under Wiener estimation is:

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

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

$$\begin{aligned} e_{wiener_k}(i) &= b_k(i) - \hat{b_{wiener_k}}(i) && (3.26)\\ &= b_k(i) - \mathbf{g}_{wiener_k}^H \mathbf{u}(i) && (3.27) \end{aligned}$$

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.

$$\begin{aligned} MMSE_{wiener_k} &= E[e_{wiener_k}(i) e_{wiener_k}^*(i)] && (3.28)\\ &= E[(b_k(i) - \mathbf{g}_{wiener_k}^H \mathbf{u}(i))(b_k(i) - \mathbf{g}_{wiener_k}^H \mathbf{u}(i))^*] && (3.29)\\ &= E[(b_k(i) - \mathbf{g}_{wiener_k}^H \mathbf{u}(i))(b_k^*(i) - \mathbf{u}^H(i)\mathbf{g}_{wiener_k})] && (3.30)\\ &= E[b_k(i)b_k^*(i)] - E[b_k(i)\mathbf{u}^H(i)\mathbf{g}_{wiener_k}] - E[\mathbf{g}_{wiener_k}^H \mathbf{u}(i) b_k^*(i)] && (3.31)\\ +E[\mathbf{g}_{wiener_k}^H \mathbf{u}(i)\mathbf{u}^H(i)\mathbf{g}_{wiener_k}] & && (3.32)\\ &= E[b_k(i)b_k^*(i)] - E[b_k(i)\mathbf{u}^H(i)]\mathbf{g}_{wiener_k} - \mathbf{g}_{wiener_k}^H E[\mathbf{u}(i) b_k^*(i)] && (3.33)\\ +\mathbf{g}_{wiener_k}^H E[\mathbf{u}(i)\mathbf{u}^H(i)]\mathbf{g}_{wiener_k} & && (3.34)\\ &= 1 - \mathbf{p}_{wiener_k}^H \mathbf{g}_{wiener_k} - \mathbf{g}_{wiener_k}^H \mathbf{p}_{wiener_k} && (3.35)\\ +\mathbf{g}_{wiener_k}^H \mathbf{R}_{wiener} \mathbf{g}_{wiener_k} & && (3.36)\\ &= 1 - \mathbf{p}_{wiener_k}^H \mathbf{g}_{wiener_k} - \mathbf{g}_{wiener_k}^H \mathbf{p}_{wiener_k} + \mathbf{p}_{wiener_k}^H \mathbf{g}_{wiener_k} && (3.37)\\ &= 1 - \mathbf{g}_{wiener_k}^H \mathbf{p}_{wiener_k} && (3.38)\\ &= 1 - \mathbf{g}_{wiener_k}^H \mathbf{R}_{wiener} \mathbf{g}_{wiener_k} && (3.39)\\ &= 1 - \mathbf{p}_{wiener_k}^H \mathbf{g}_{wiener_k} && (3.40)\\ &= 1 - \mathbf{p}_{wiener_k}^H \mathbf{R}_{wiener}^{-1} \mathbf{p}_{wiener_k} && (3.41) \end{aligned}$$

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:

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

Where the term $\mathbf{M}_k b_k(i)$ is the component for the desired kth 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 jth column of the mixing matrix $\mathbf{M}$ while $b_j(i)$ is the jth 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 $E[[\mathbf{g}^H_{wiener_k}\mathbf{M}_k b_k(i)][\mathbf{g}^H_{wiener_k}\mathbf{M}_k b_k(i)]^*]$ and the statistical mean of the total received signal power is expressed to be $E[[\mathbf{g}^H_{wiener_k}\mathbf{u}(i)][\mathbf{g}^H_{wiener_k}\mathbf{u}(i)]^*]$, so the statistical mean of the sum of interference power and noise power can be directly presented by a subtraction, as $E[[\mathbf{g}^H_{wiener_k}\mathbf{u}(i)][\mathbf{g}^H_{wiener_k}\mathbf{u}(i)]^*] - E[[\mathbf{g}^H_{wiener_k}\mathbf{M}_k b_k(i)][\mathbf{g}^H_{wiener_k}\mathbf{M}_k b_k(i)]^*]$.

Define:

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

Correspondingly, the SINR can be expressed as:

$$\begin{aligned}
SINR_{wiener_k} &= \frac{E[[\mathbf{g}^H_{wiener_k}\mathbf{M}_k b_k(i)][\mathbf{g}^H_{wiener_k}\mathbf{M}_k b_k(i)]^*]}{E[[\mathbf{g}^H_{wiener_k}\mathbf{u}(i)][\mathbf{g}^H_{wiener_k}\mathbf{u}(i)]^*] - E[[\mathbf{g}^H_{wiener_k}\mathbf{M}_k b_k(i)][\mathbf{g}^H_{wiener_k}\mathbf{M}_k b_k(i)]^*]} && (3.44)\\
&= \frac{E[[\mathbf{g}^H_{wiener_k}\mathbf{M}_k b_k(i)][b^*_k(i)\mathbf{M}^H_k\mathbf{g}_{wiener_k}]]}{E[[\mathbf{g}^H_{wiener_k}\mathbf{u}(i)][\mathbf{u}^H(i)\mathbf{g}_{wiener_k}]] - E[[\mathbf{g}^H_{wiener_k}\mathbf{M}_k b_k(i)][b^*_k(i)\mathbf{M}^H_k\mathbf{g}_{wiener_k}]]} && (3.45)\\
&= \frac{\mathbf{g}^H_{wiener_k}\mathbf{M}_k E[b_k(i)b^*_k(i)]\mathbf{M}^H_k\mathbf{g}_{wiener_k}}{\mathbf{g}^H_{wiener_k}E[\mathbf{u}(i)\mathbf{u}^H(i)]\mathbf{g}_{wiener_k} - \mathbf{g}^H_{wiener_k}\mathbf{M}_k E[b_k(i)b^*_k(i)]\mathbf{M}^H_k\mathbf{g}_{wiener_k}} && (3.46)\\
&= \frac{\mathbf{g}^H_{wiener_k}\mathbf{M}_k\mathbf{M}^H_k\mathbf{g}_{wiener_k}}{\mathbf{g}^H_{wiener_k}\mathbf{R}_{wiener}\mathbf{g}_{wiener_k} - \mathbf{g}^H_{wiener_k}\mathbf{M}_k\mathbf{M}^H_k\mathbf{g}_{wiener_k}} && (3.47)
\end{aligned}$$

Recall Equation 3.24, get:

$$
\begin{aligned}
SINR_{wiener_k} &= \frac{\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k \mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k}{\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{R}_{wiener} \mathbf{R}_{wiener}^{-1} \mathbf{M}_k - \mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k \mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k} && (3.48)\\
&= \frac{[\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k][\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k]}{\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k - [\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k][\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k]} && (3.49)\\
&= \frac{|\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k|^2}{\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k - |\mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k|^2} && (3.50)\\
&= \frac{1}{1 - \mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k} \mathbf{M}_k^H \mathbf{R}_{wiener}^{-1} \mathbf{M}_k && (3.51)
\end{aligned}
$$

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

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

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

$$
\begin{aligned}
\mathbf{R}_{wiener_{-k}} &= E[(\mathbf{M}_{-k} b_{-k}(i) + \mathbf{n}(i))(\mathbf{M}_{-k} b_{-k}(i) + \mathbf{n}(i))^H] && (3.53)\\
&= \sum_{j \neq k} \mathbf{M}_j \mathbf{M}_j^H + 0.01 \mathbf{I}_{N_r} && (3.54)
\end{aligned}
$$

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

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

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

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

So:

$$
\begin{aligned}
\mathbf{M}_k^H\mathbf{R}_{wiener}^{-1}\mathbf{M}_k &= \mathbf{M}_k^H(\mathbf{R}_{wiener_{-k}}^{-1} - \frac{1}{1+\gamma}\mathbf{R}_{wiener_{-k}}^{-1}\mathbf{M}_k\mathbf{M}_k^H\mathbf{R}_{wiener_{-k}}^{-1})\mathbf{M}_k & (3.57)\\
&= \mathbf{M}_k^H\mathbf{R}_{wiener_{-k}}^{-1}\mathbf{M}_k - \frac{1}{1+\gamma}\mathbf{M}_k^H\mathbf{R}_{wiener_{-k}}^{-1}\mathbf{M}_k\mathbf{M}_k^H\mathbf{R}_{wiener_{-k}}^{-1}\mathbf{M}_k & (3.58)\\
&= \mathbf{M}_k^H\mathbf{R}_{wiener_{-k}}^{-1}\mathbf{M}_k - \frac{1}{1+\gamma}[\mathbf{M}_k^H\mathbf{R}_{wiener_{-k}}^{-1}\mathbf{M}_k][\mathbf{M}_k^H\mathbf{R}_{wiener_{-k}}^{-1}\mathbf{M}_k] & (3.59)\\
&= \mathbf{M}_k^H\mathbf{R}_{wiener_{-k}}^{-1}\mathbf{M}_k - \frac{1}{1+\gamma}[\mathbf{M}_k^H\mathbf{R}_{wiener_{-k}}^{-1}\mathbf{M}_k]^2 & (3.60)\\
&= \gamma - \frac{1}{1+\gamma}\gamma^2 & (3.61)\\
&= \gamma(1 - \frac{\gamma}{1+\gamma}) & (3.62)\\
&= \frac{\gamma}{1+\gamma} & (3.63)
\end{aligned}
$$

Conclusively, the SINR for the desired *kth* transmitted signal symbol in Wiener estimation is expressed as:

$$
\begin{aligned}
SINR_{wiener_k} &= \frac{1}{1-\frac{\gamma}{1+\gamma}}\frac{\gamma}{1+\gamma} & (3.64)\\
&= \gamma & (3.65)\\
&= \mathbf{M}_k^H\mathbf{R}_{wiener_{-k}}^{-1}\mathbf{M}_k & (3.66)
\end{aligned}
$$

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:

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

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:

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

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

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

Thus the reconstructed kth transmitted signal symbol at time i is expressed as:

$$\hat{b_k(i)} = \mathbf{g}_{ls_k}^{H}\mathbf{u(i)} \tag{3.70}$$

The estimated error for the kth transmitted signal symbol under LS estimation is:

$$
\begin{aligned}
e_{ls_k}(i) &= b_k(i) - \hat{b_{ls_k}}(i) && (3.71)\\
&= b_k(i) - \mathbf{g}_{ls_k}^H \mathbf{u}(i) && (3.72)
\end{aligned}
$$

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 kth transmitted signal symbol is expressed as:

$$
\begin{aligned}
MSE_{ls_k} &= E[|e_{ls_k}(i)e_{ls_k}^*(i)|] && (3.73)\\
&= E[[b_k(i) - \mathbf{g}_{ls_k}^H \mathbf{u}(i)][b_k(i) - \mathbf{g}_{ls_k}^H \mathbf{u}(i)]^*] && (3.74)\\
&= E[[b_k(i) - \mathbf{g}_{ls_k}^H \mathbf{u}(i)][b_k^*(i) - \mathbf{g}_{ls_k}^T \mathbf{u}^*(i)]] && (3.75)\\
&= E[b_k(i)b_k^*(i)] - E[b_k(i)\mathbf{g}_{ls_k}^T \mathbf{u}^*(i)] - E[b_k^*(i)\mathbf{g}_{ls_k}^H \mathbf{u}(i)] + E[\mathbf{g}_{ls_k}^H \mathbf{u}(i)\mathbf{u}^H(i)\mathbf{g}_{ls_k}] && (3.76)\\
&= E[b_k(i)b_k^*(i)] - \mathbf{g}_{ls_k}^T E[b_k(i)\mathbf{u}^*(i)] - \mathbf{g}_{ls_k}^H E[b_k^*(i)\mathbf{u}(i)] + \mathbf{g}_{ls_k}^H E[\mathbf{u}(i)\mathbf{u}^H(i)]\mathbf{g}_{ls_k} && (3.77)\\
&= 1 - \mathbf{g}_{ls_k}^T \mathbf{p}_{wiener_k}^* - \mathbf{g}_{ls_k}^H \mathbf{p}_{wiener_k} + \mathbf{g}_{ls_k}^H \mathbf{R}_{wiener}\mathbf{g}_{ls_k} && (3.78)
\end{aligned}
$$

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 kth 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:

$$
\begin{aligned}
SINR_{ls_k} &= \frac{E[|\mathbf{g}_{ls_k}^H \mathbf{M}_k b_k(i)|^2]}{E[\sum_{j\neq k} |\mathbf{g}_{ls_k}^H \mathbf{M}_j b_j(i)|^2] + E[|\mathbf{g}_{ls_k}^H \mathbf{n}(i)|^2]} &(3.79)\\
&= \frac{E[[\mathbf{g}_{ls_k}^H \mathbf{M}_k b_k(i)][\mathbf{g}_{ls_k}^H \mathbf{M}_k b_k(i)]^*]}{E[\sum_{j\neq k}[\mathbf{g}_{ls_k}^H \mathbf{M}_j b_j(i)][\mathbf{g}_{ls_k}^H \mathbf{M}_j b_j(i)]^*] + E[[\mathbf{g}_{ls_k}^H \mathbf{n}(i)][\mathbf{g}_{ls_k}^H \mathbf{n}(i)]^*]} &(3.80)\\
&= \frac{E[[\mathbf{g}_{ls_k}^H \mathbf{M}_k b_k(i)][b_k^*(i)\mathbf{M}_k^H \mathbf{g}_{ls_k}]]}{E[\sum_{j\neq k}[\mathbf{g}_{ls_k}^H \mathbf{M}_j b_j(i)][b_j^*(i)\mathbf{M}_j^H \mathbf{g}_{ls_k}]] + E[[\mathbf{g}_{ls_k}^H \mathbf{n}(i)][\mathbf{n}^H(i)\mathbf{g}_{ls_k}]]} &(3.81)\\
&= \frac{|\mathbf{g}_{ls_k}^H \mathbf{M}_k E[b_k(i)b_k^*(i)]\mathbf{M}_k^H \mathbf{g}_{ls_k}|}{\sum_{j\neq k} |\mathbf{g}_{ls_k}^H \mathbf{M}_j E[b_j(i)b_j^*(i)]\mathbf{M}_j^H \mathbf{g}_{ls_k}| + |\mathbf{g}_{ls_k}^H E[\mathbf{n}(i)\mathbf{n}^H(i)]\mathbf{g}_{ls_k}|} &(3.82)\\
&= \frac{|\mathbf{g}_{ls_k}^H \mathbf{M}_k \mathbf{M}_k^H \mathbf{g}_{ls_k}|}{\sum_{j\neq k} |\mathbf{g}_{ls_k}^H \mathbf{M}_j \mathbf{M}_j^H \mathbf{g}_{ls_k}| + |\mathbf{g}_{ls_k}^H 0.01 \mathbf{g}_{ls_k}|} &(3.83)\\
&= \frac{|\mathbf{g}_{ls_k}^H \mathbf{M}_k|^2}{\sum_{j\neq k} |\mathbf{g}_{ls_k}^H \mathbf{M}_j|^2 + 0.01|\mathbf{g}_{ls_k}^H \mathbf{g}_{ls_k}|} &(3.84)
\end{aligned}
$$

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 beamforming. 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.

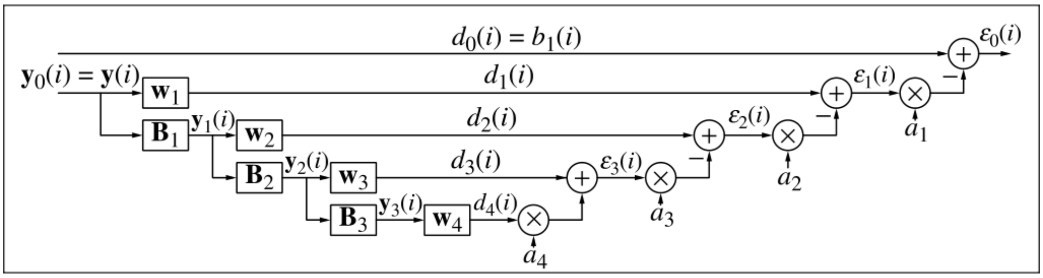

Figure 3.3: Multi-Stage Wiener Filter [1]

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 kth transmitted signal symbol is expressed to be:

$$\mathbf{S}_k = [\mathbf{p}_k \; \mathbf{R}\mathbf{p}_k \; \mathbf{R}^2\mathbf{p}_k \ldots \; \mathbf{R}^{D-1}\mathbf{p}_k] \tag{3.85}$$

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 kth transmitted signal symbol is expressed as:

$$\mathbf{S}_{wiener_k} = [\mathbf{p}_{wiener_k} \ \mathbf{R}_{wiener}\mathbf{p}_{wiener_k} \ \mathbf{R}^2_{wiener}\mathbf{p}_{wiener_k} \ldots \ \mathbf{R}^{D-1}_{wiener}\mathbf{p}_{wiener_k}] \tag{3.86}$$

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

$$\mathbf{S}_{ls_k} = [\mathbf{p}_{ls_k} \ \mathbf{R}_{ls}\mathbf{p}_{ls_k} \ \mathbf{R}^2_{ls}\mathbf{p}_{ls_k} \ldots \ \mathbf{R}^{D-1}_{ls}\mathbf{p}_{ls_k}] \tag{3.87}$$

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 kth transmitted signal symbol is provided as:

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

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 kth transmitted signal symbol is expressed as:

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

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 kth transmitted signal symbol is given as:

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

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 kth transmitted signal symbol is presented as:

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

In LS estimation, the MSE of the kth transmitted signal symbol is presented as:

$$MSE_{rr\ ls_k} = 1 - \mathbf{g}^T_{rr\ ls_k}\mathbf{p}^*_{wiener_k} - \mathbf{g}^H_{rr\ ls_k}\mathbf{p}_{wiener_k} + \mathbf{g}^H_{rr\ ls_k}\mathbf{R}_{wiener}\mathbf{g}_{rr\ ls_k} \qquad (3.92)$$

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 kth transmitted signal symbol in Wiener estimation under reduced-rank filtering is expressed as:

$$SINR_{rr\ wiener_k} = \mathbf{p}^H_{wiener_k}\mathbf{S}_{wiener_k}(\mathbf{S}^H_{wiener_k}\mathbf{R}_{wiener_{-k}}\mathbf{S}_{wiener_k})^{-1}\mathbf{S}^H_{wiener_k}\mathbf{p}_{wiener_k} \qquad (3.93)$$

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 kth transmitted signal symbol, under reduced-rank filtering, the SINR is given as:

$$SINR_{rr\ ls_k} = \frac{|\mathbf{g}^H_{rr\ ls_k}\mathbf{M}_k|^2}{\sum_{j\neq k}|\mathbf{g}^H_{rr\ ls_k}\mathbf{M}_j|^2 + 0.01|\mathbf{g}^H_{rr\ ls_k}\mathbf{g}_{rr\ ls_k}|} \tag{3.94}$$

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.

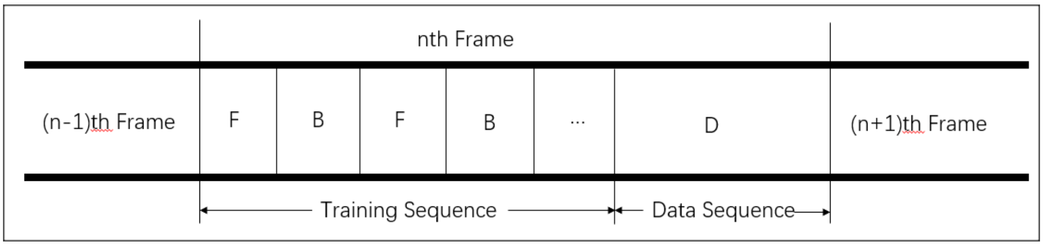

Figure 3.4: TDD data frames in bi-directional training

In Figure 3.4, the frame sitting in the middle is the nth 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 nth 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 kth 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 kth transmitted signal symbol by Wiener estimation approach with power normalization as:

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

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 kth transmitted signal symbol is:

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

Where the statistical Krylov subspace matrix $\mathbf{S}_{wiener_k}$ for the kth 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 $\overleftarrow{\mathbf{H}}$ 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:

$$\begin{aligned} \overleftarrow{\mathbf{u}(i)} &= \sqrt{P}\overleftarrow{\mathbf{H}}\mathbf{g}^*\overleftarrow{\mathbf{b}(i)} + \overleftarrow{\mathbf{n}(i)} & (3.98) \\ &= \overleftarrow{\mathbf{M}}\overleftarrow{\mathbf{b}(i)} + \overleftarrow{\mathbf{n}(i)} & (3.99) \end{aligned}$$

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:

$$\begin{aligned} \overleftarrow{\mathbf{R}_{wiener}} &= E[\overleftarrow{\mathbf{u}(i)}\overleftarrow{\mathbf{u}(i)}^H] & (3.100) \\ &= \overleftarrow{\mathbf{M}}\overleftarrow{\mathbf{M}}^H + 0.01 I_{N_t} & (3.101) \end{aligned}$$

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 kth transmitted signal symbol in the backward direction is:

$$\begin{aligned} \overleftarrow{\mathbf{p}_{wiener_k}} &= E[\overleftarrow{\mathbf{u}(i)}\overleftarrow{b_k(i)}^*] & (3.102) \\ &= \overleftarrow{\mathbf{M}_k} & (3.103) \end{aligned}$$

Therefore, the power normalized optimized precoder for the kth 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 kth transmitted signal symbol is expressed as:

$$\overleftarrow{\mathbf{S}_{wiener_k}} = [\overleftarrow{\mathbf{p}_{wiener_k}}\ \overleftarrow{\mathbf{R}_{wiener}}\overleftarrow{\mathbf{p}_{wiener_k}}\ \overleftarrow{\mathbf{R}_{wiener}}^2\overleftarrow{\mathbf{p}_{wiener_k}} \ldots \overleftarrow{\mathbf{R}_{wiener}}^{D-1}\overleftarrow{\mathbf{p}_{wiener_k}}] \tag{3.105}$$

Therefore, in the backward direction, the power normalized precoder for the kth transmitted signal symbol is:

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

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 kth 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:

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

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:

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

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:

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

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

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

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 kth transmitted signal symbol is given as:

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

Thus, in backward training, the power normalized trained precoder for the kth 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.

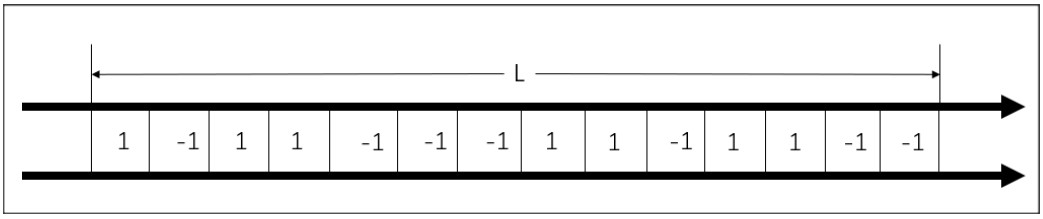

Figure 4.1: Transmitted training symbol sequence

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.

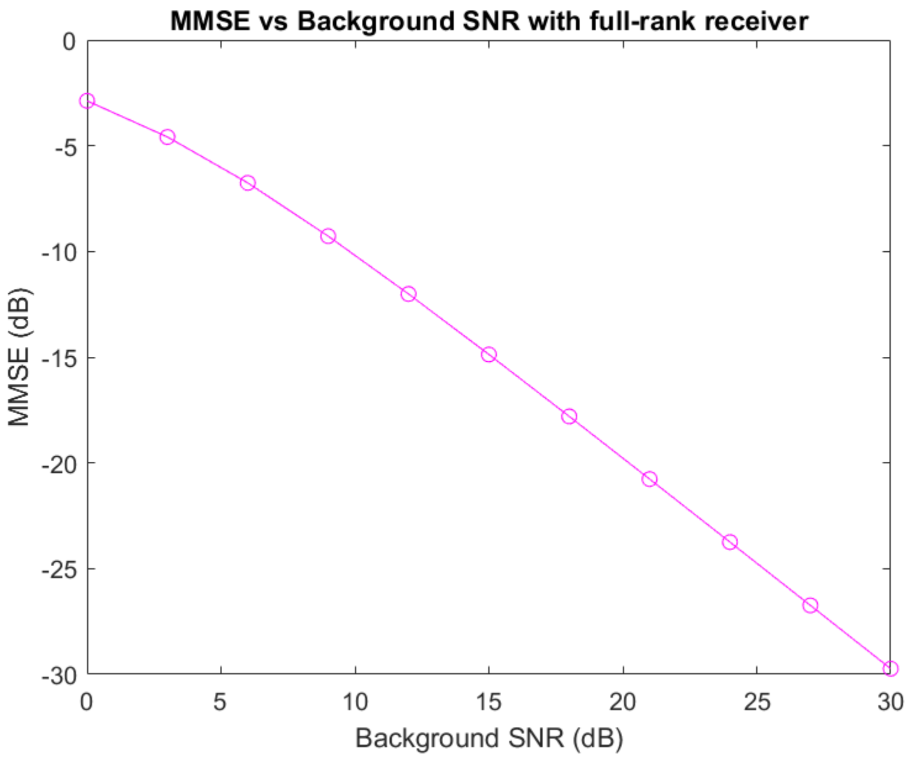

Figure 4.2: MMSE vs background SNR with the full-rank receiver, with the number of transmit antennas N_t 8, the number of receive antennas N_r 8 and the number of transmitted signal stream N_s 1

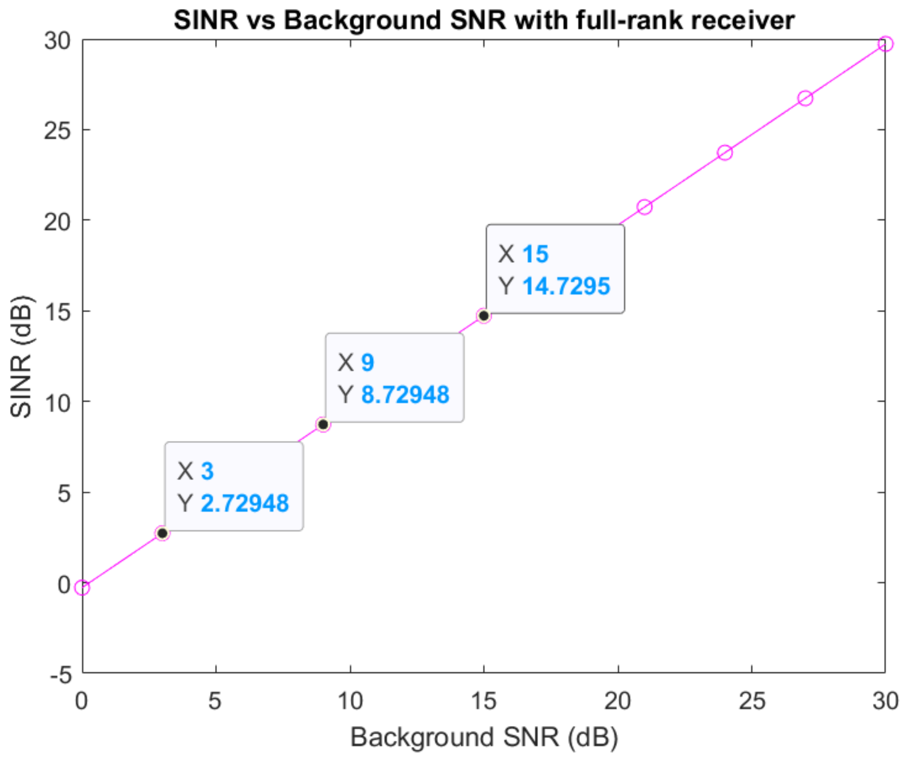

Figure 4.3: SINR vs background SNR with the full-rank receiver, with the number of transmit antennas N_t 8, the number of receive antennas N_r 8 and the number of transmitted signal stream N_s 1

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:

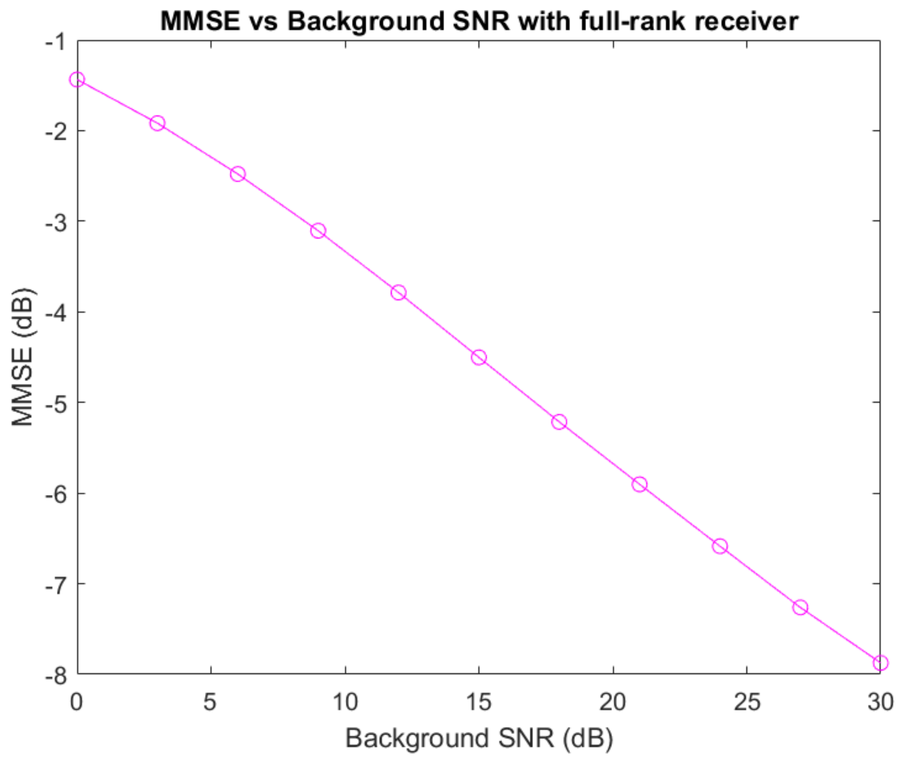

Figure 4.4: MMSE vs background SNR with the full-rank receiver, with the number of transmit antennas N_t 8, the number of receive antennas N_r 8 and the number of transmitted signal streams N_s 8

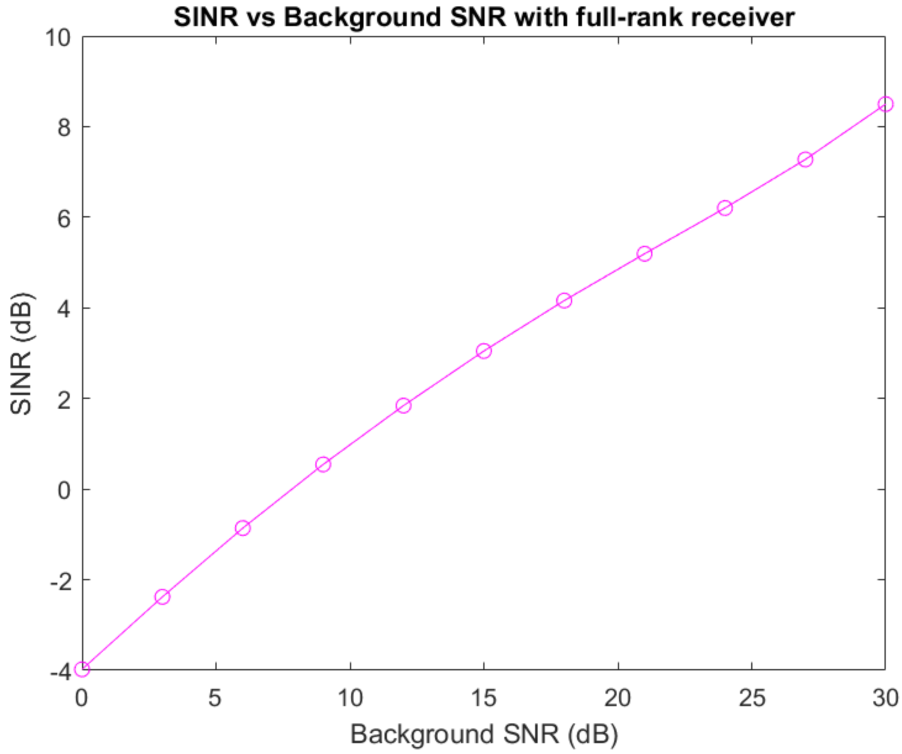

Figure 4.5: SINR vs background SNR with the full-rank receiver, with the number of transmit antennas N_t 8, the number of receive antennas N_r 8 and the number of transmitted signal streams N_s 8

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.

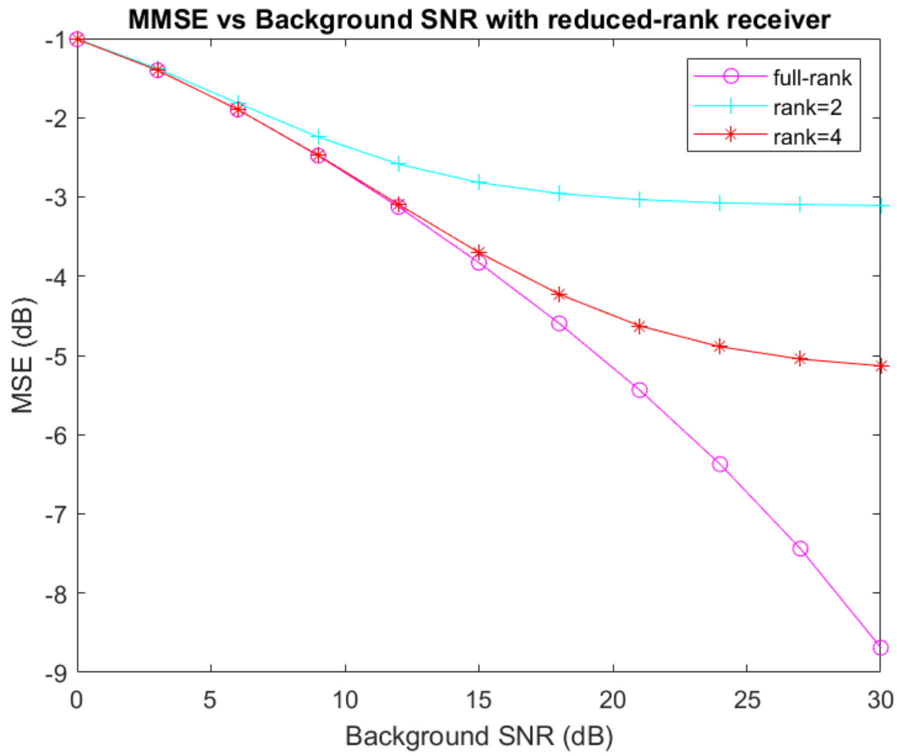

Figure 4.6: MMSE vs background SNR with reduced-rank receivers, with the number of transmit antennas N_t 8, the number of receive antennas N_r 8, the number of transmitted signal streams N_s 8 and ranks 2 and 4

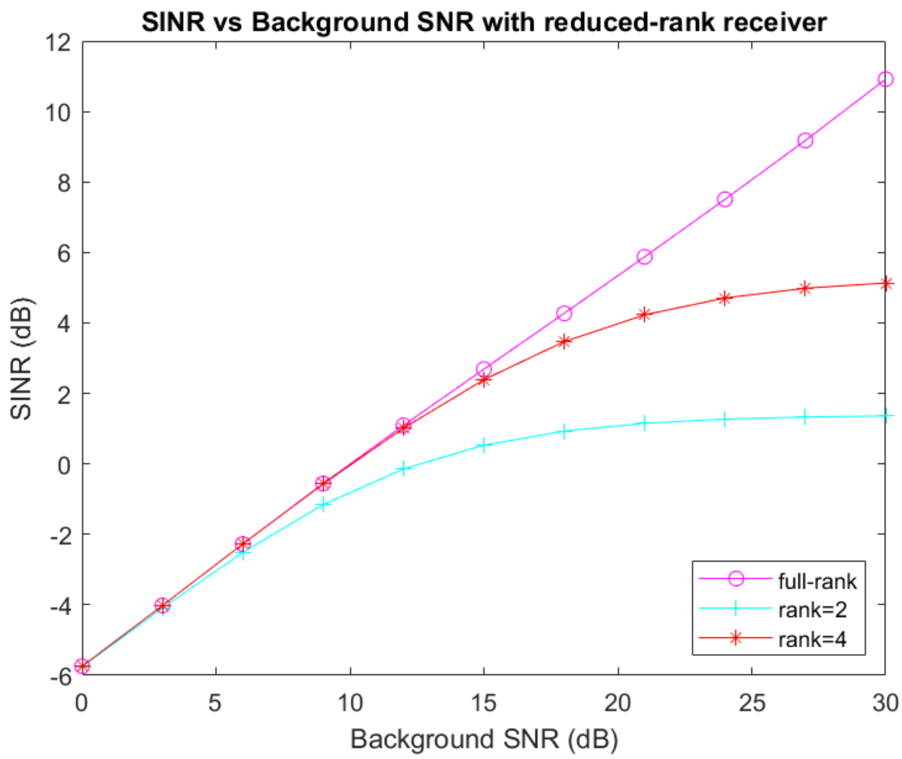

Figure 4.7: SINR vs background SNR with reduced-rank receivers, with the number of transmit antennas N_t 8, the number of receive antennas N_r 8, the number of transmitted signal streams N_s 8 and ranks 2 and 4

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 interference 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.

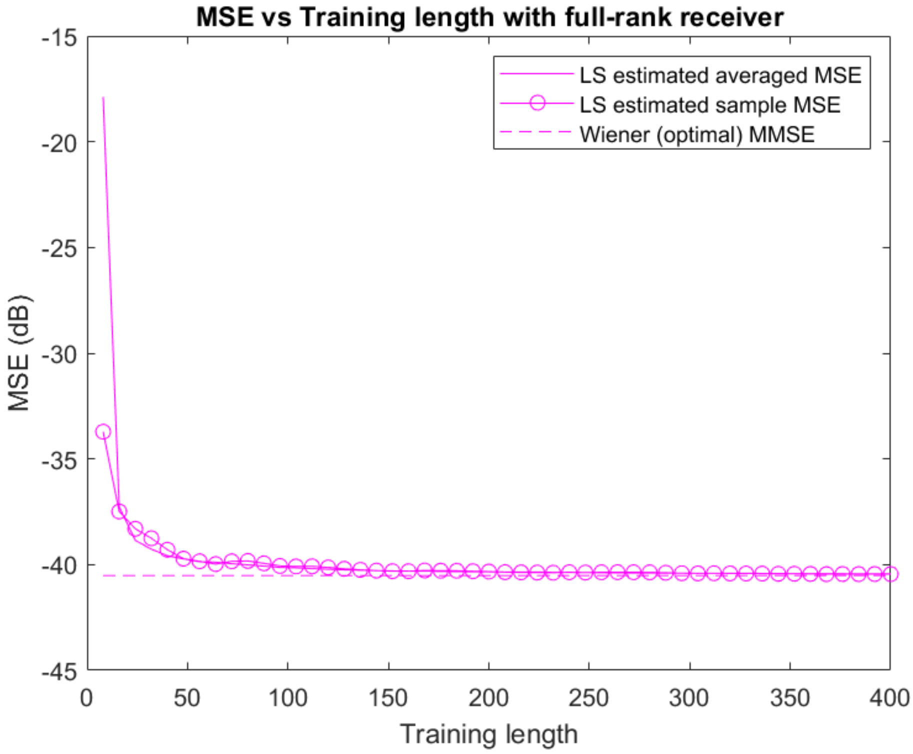

Figure 4.8: MSE vs training length with the full-rank receiver, with the transmit power of each transmit antenna P 100, the number of transmit antennas N_t 8, the number of receive antennas N_r 8, the number of transmitted training symbol sequence N_s 1, the length of each training symbol sequence L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

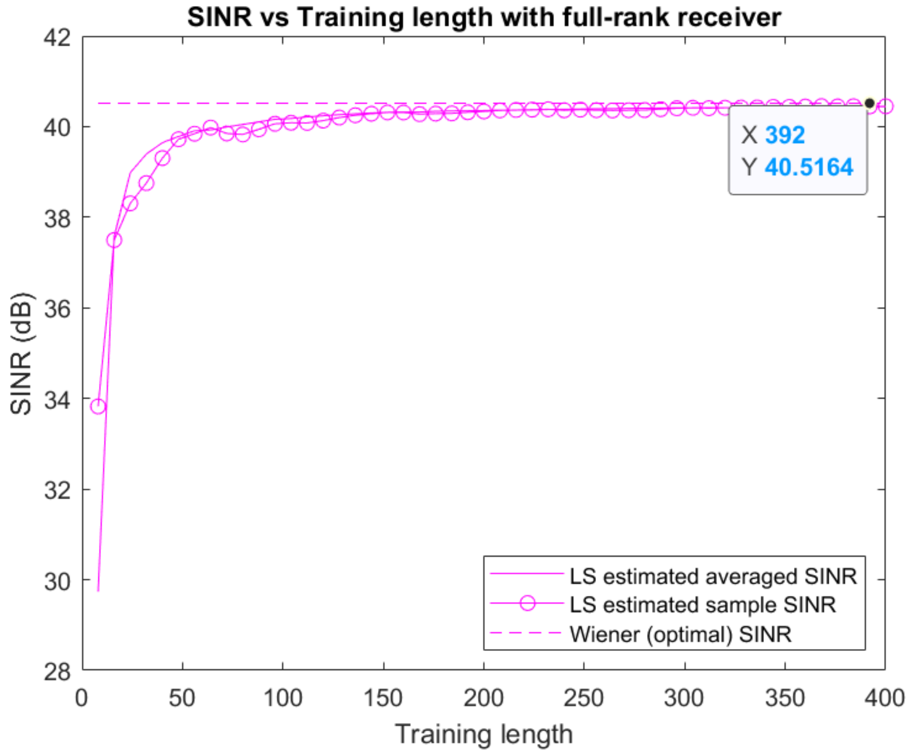

Figure 4.9: SINR vs training length with the full-rank receiver, with the transmit power of each transmit antenna P 100, the number of transmit antennas N_t 8, the number of receive antennas N_r 8, the number of transmitted training symbol sequence N_s 1, the length of each training symbol sequence L 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 $10log10\frac{100}{0.01} = 40dB$, 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.

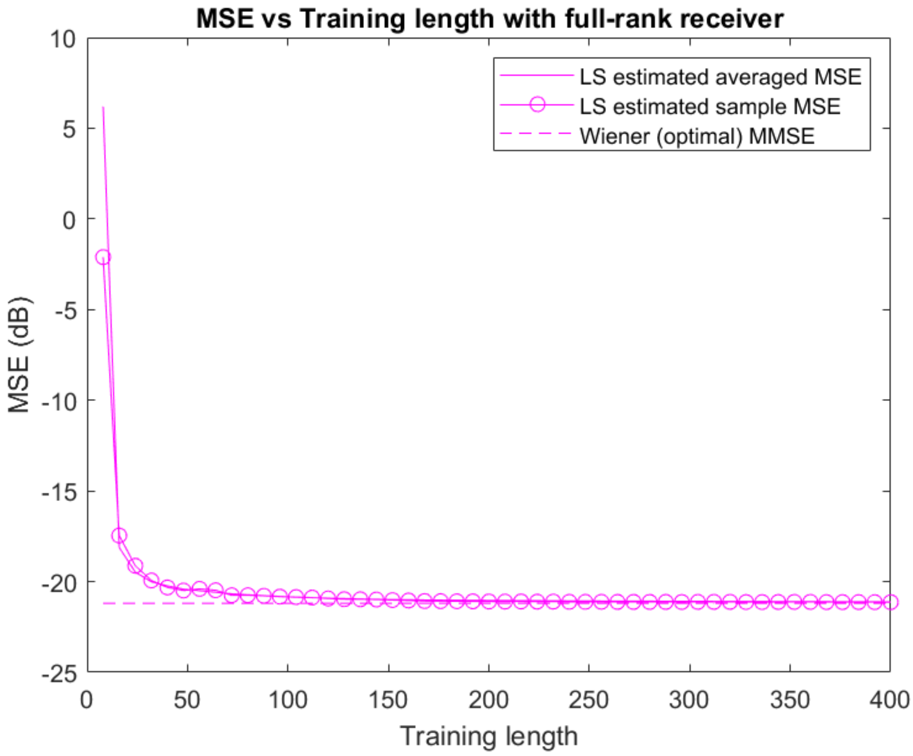

Figure 4.10: MSE vs training length with the full-rank receiver, with the transmit power of each transmit antenna P 100, the number of transmit antennas N_t 8, the number of receive antennas N_r 8, the number of transmitted training symbol sequences N_s 8, the length of each training symbol sequence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

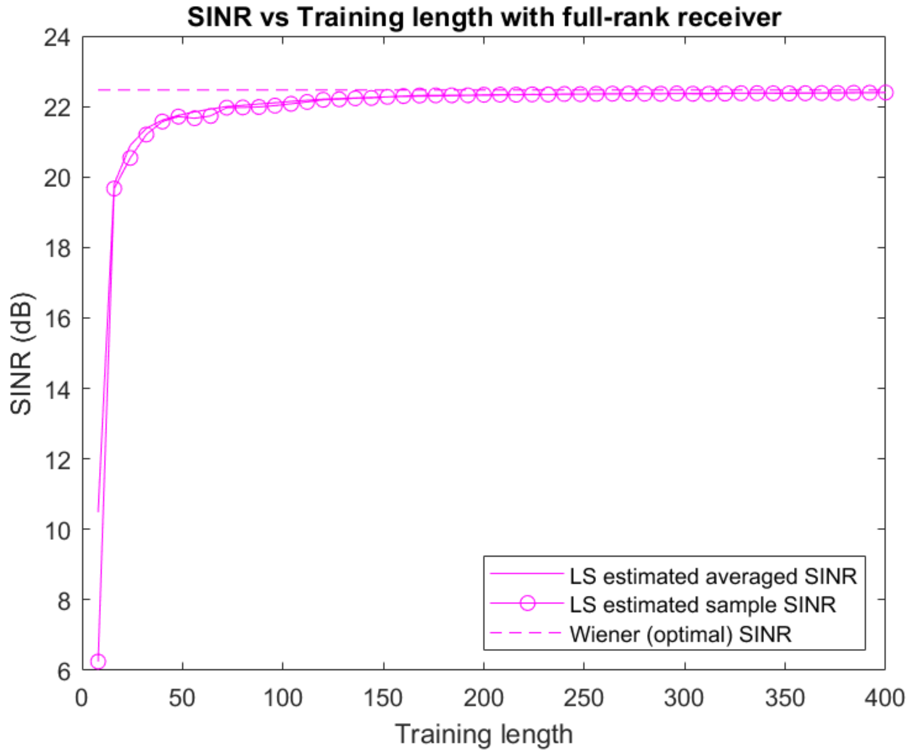

Figure 4.11: SINR vs training length with the full-rank receiver, with the transmit power of each transmit antenna P 100, the number of transmit antennas N_t 8, the number of receive antennas N_r 8, the number of transmitted training symbol sequences N_s 8, the length of each training symbol sequence 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.

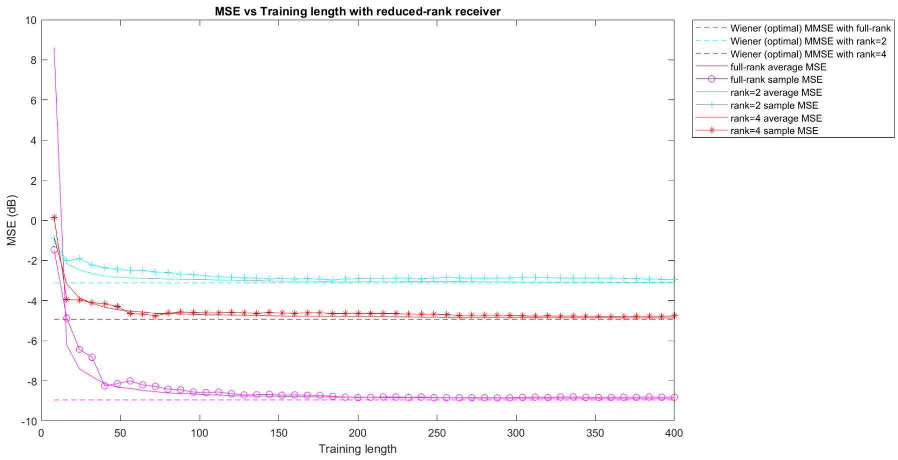

Figure 4.12: MSE vs training length with reduced-rank receivers, with the transmit power of each transmit antenna P 100, the number of transmit antennas N_t 8, the number of receive antennas N_r 8, the number of transmitted training symbol sequences N_s 8, the length of each training symbol sequence 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

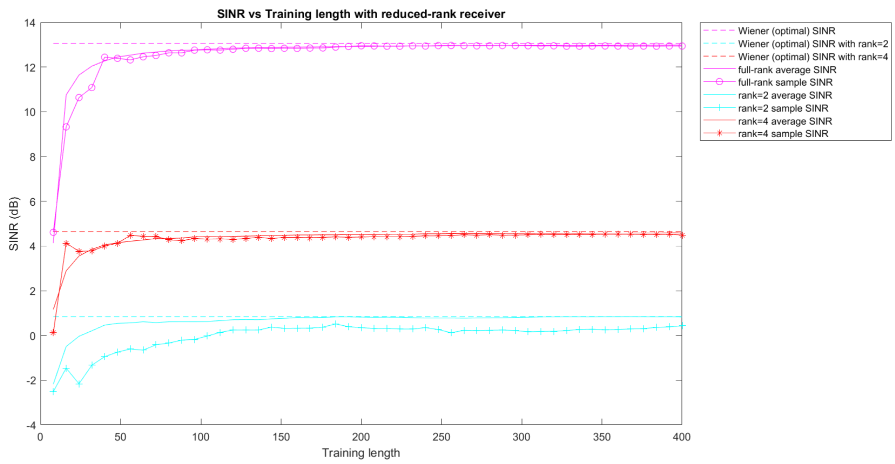

Figure 4.13: SINR vs training length with reduced-rank receiver, with the transmit power of each transmit antenna P 100, the number of transmit antennas N_t 8, the number of receive antennas N_r 8, the number of transmitted training symbol sequences N_s 8, the length of each training symbol sequence 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.

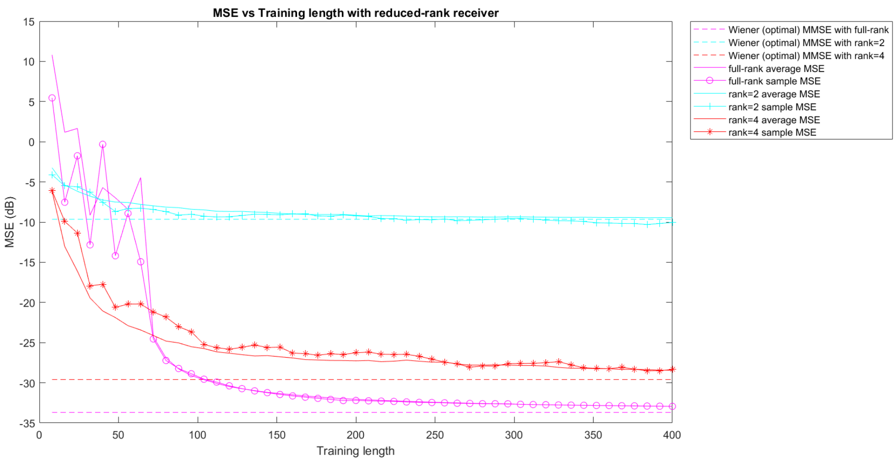

Figure 4.14: MSE vs training length with reduced-rank receivers, with the transmit power of each transmit antenna P 100, the number of transmit antennas N_t 64, the number of receive antennas N_r 64, the number of transmitted training symbol sequences N_s 8, the length of each training symbol sequence 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

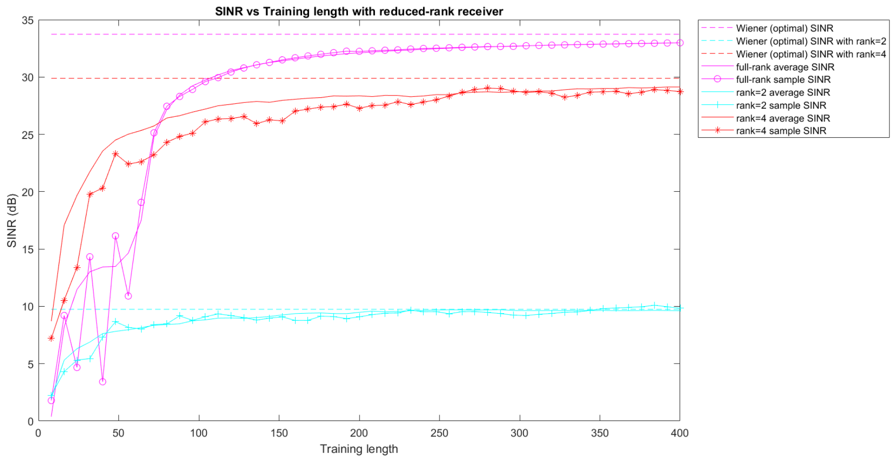

Figure 4.15: SINR vs training length with reduced-rank receivers, with the transmit power of each transmit antenna P 100, the number of transmit antennas N_t 64, the number of receive antennas N_r 64, the number of transmitted training symbol sequences N_s 8, the length of each training symbol sequence 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.

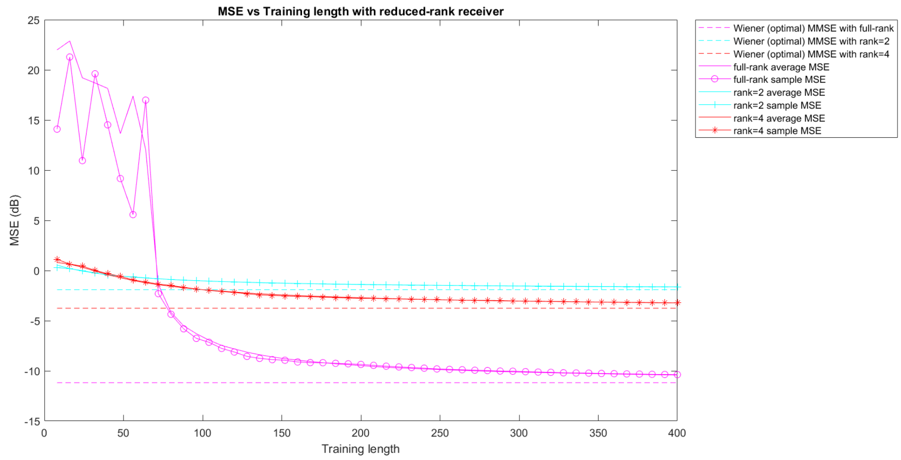

Figure 4.16: MSE vs training length with reduced-rank receivers, with the transmit power of each transmit antenna P 100, the number of transmit antennas N_t 64, the number of receive antennas N_r 64, the number of transmitted training symbol sequences N_s 64, the length of each transmitted training symbol 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

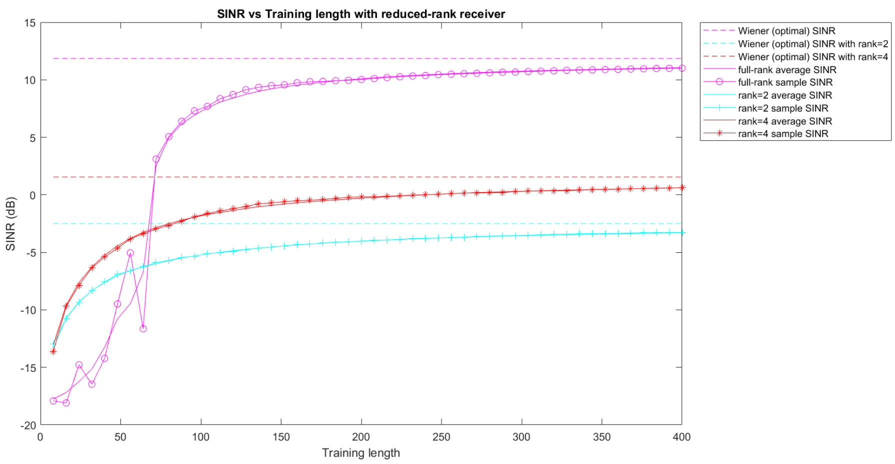

Figure 4.17: SINR vs training length with reduced-rank receivers, with the transmit power of each transmit antenna P 100, the number of transmit antennas N_t 64, the number of receive antennas N_r 64, the number of transmitted training symbol sequences N_s 64, the length of each training symbol sequence 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.

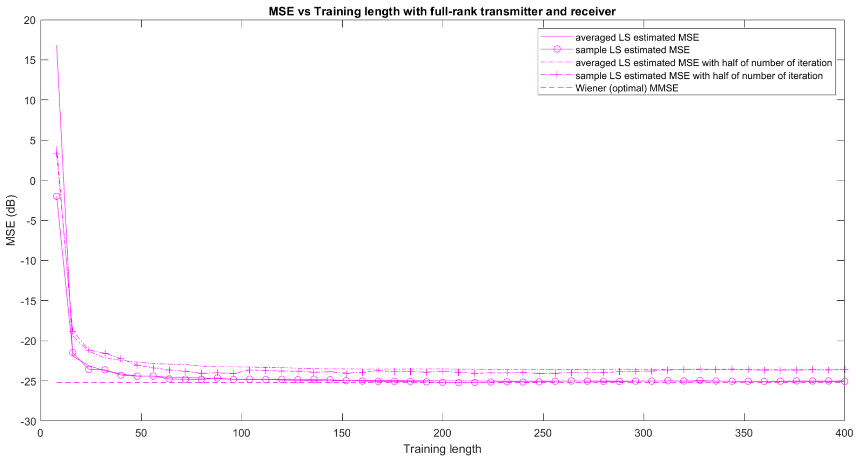

Figure 4.18: MSE vs training length with a full-rank transmitter-receiver pair, with the transmit power of each transmit or receive antenna P 100, the number of transmit antennas N_t 8, the number of receive antennas N_r 8, the number of transmitted training symbol sequences N_s 8, the length of each 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

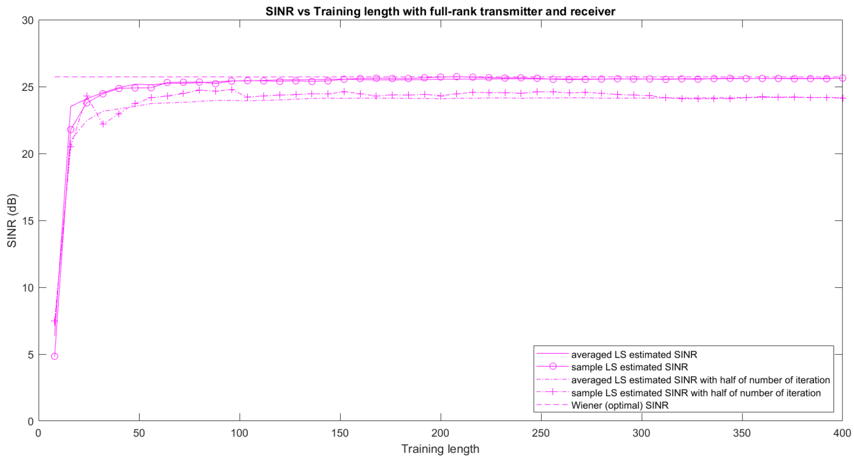

Figure 4.19: SINR vs training length with a full-rank transmitter-receiver pair, with the transmit power of each transmit or receive antenna P 100, the number of transmit antennas N_t 8, the number of receive antennas N_r 8, the number of transmitted training symbol sequences N_s 8, the length of each 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

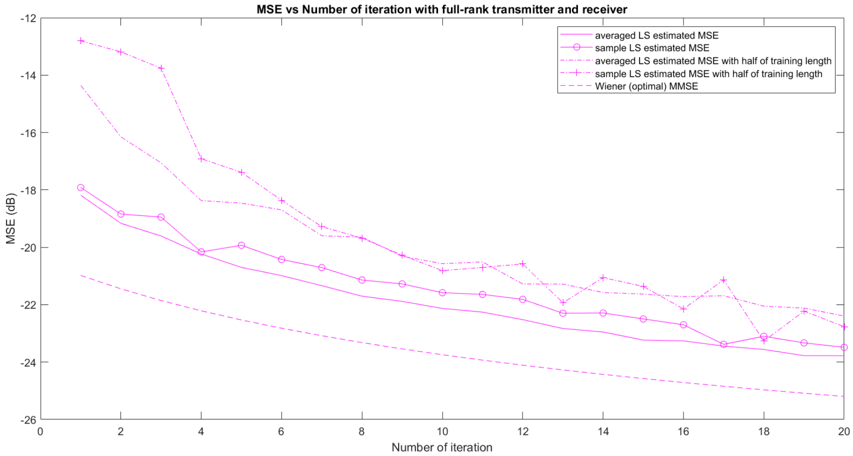

Figure 4.20: MSE vs number of bi-directional optimization or training iterations with a full-rank transmitter-receiver pair, with the transmit power of each transmit or receive antenna P 100, the number of transmit antennas N_t 8, the number of receive antennas N_r 8, the number of transmitted training 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

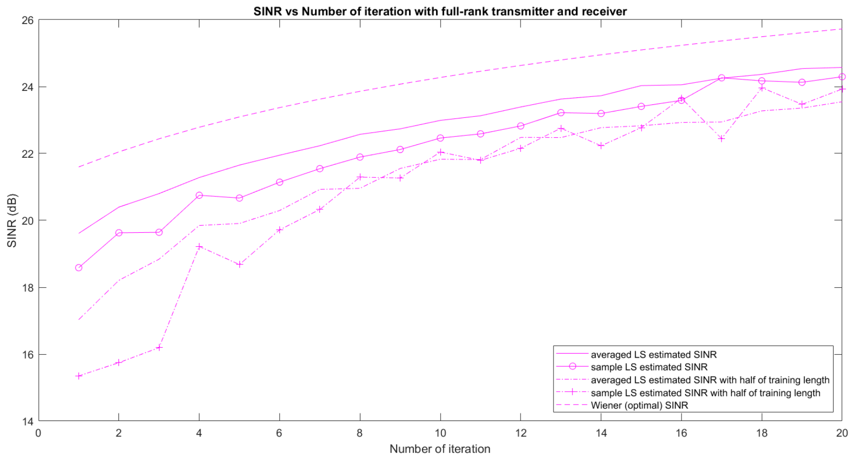

Figure 4.21: SINR vs number of bi-directional optimization or training iterations with a full-rank transmitter-receiver pair, with the transmit power of each transmit or receive antenna P 100, the number of transmit antennas N_t 8, the number of receive antennas N_r 8, the number of transmitted training 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.

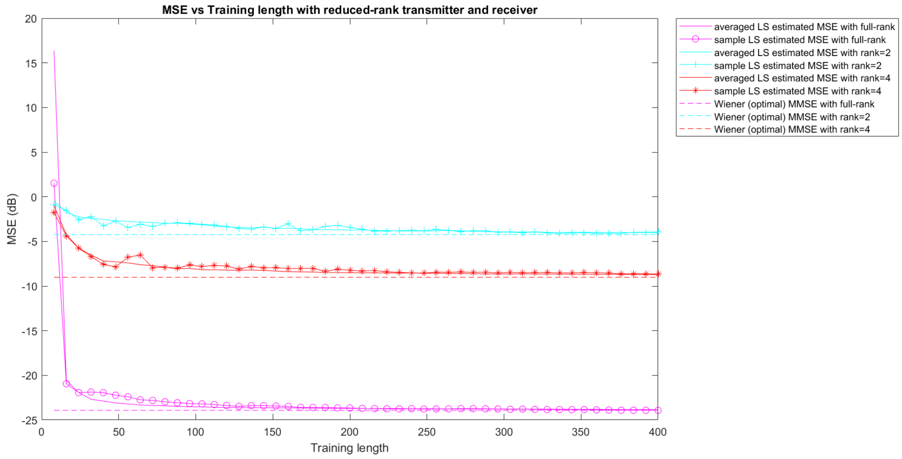

Figure 4.22: MSE vs training length with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna P 100, the number of transmit antennas N_t 8, the number of receive antennas N_r 8, the number of transmitted training symbol sequences N_s 8, the length of 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

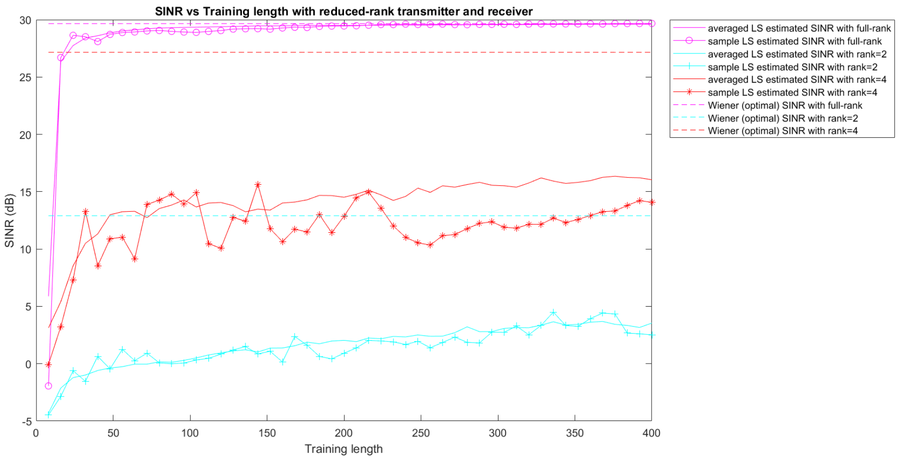

Figure 4.23: SINR vs training length with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna P 100, the number of transmit antennas N_t 8, the number of receive antennas N_r 8, the number of transmitted training symbol sequences N_s 8, the length 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

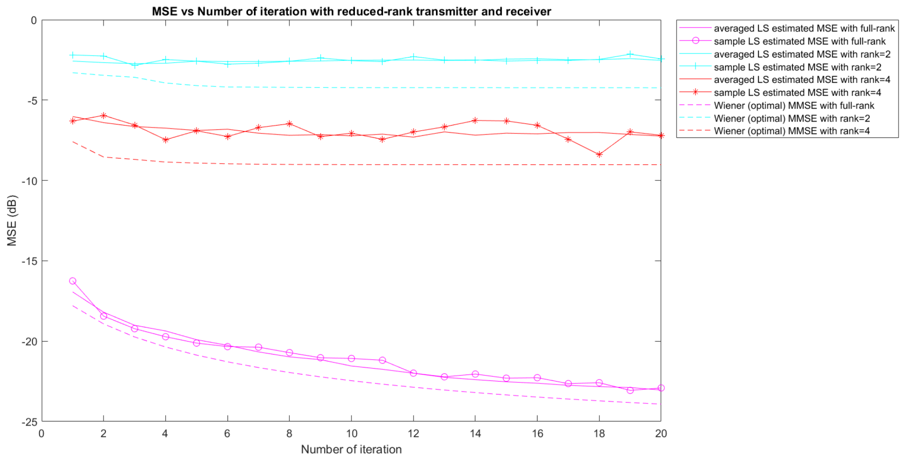

Figure 4.24: MSE vs number of bi-directional optimization or training iterations with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna P 100, the number of transmit antennas N_t 8, the number of receive antennas N_r 8, the number of transmitted 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

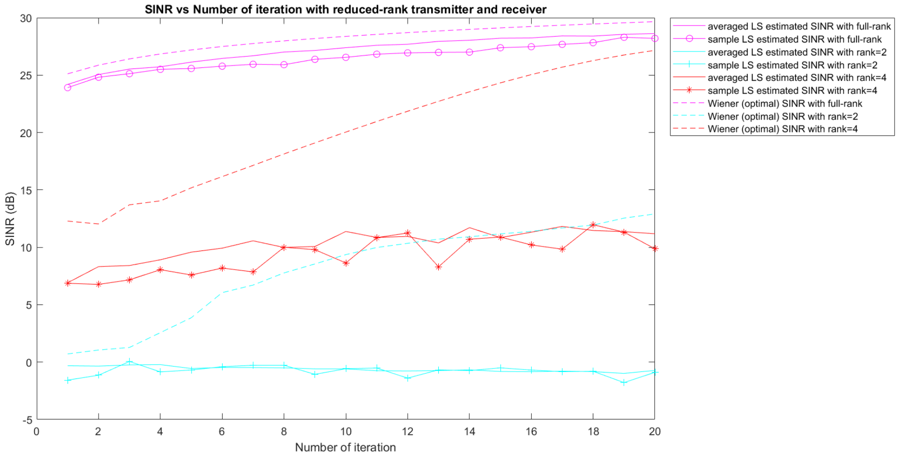

Figure 4.25: SINR vs number of bi-directional optimization or training iteration with two transmitter-receiver pairs, with the transmit power of each transmit or receive antenna P 100, the number of transmit antennas N_t 8, the number of receive antennas N_r 8, the number of transmitted training symbol 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.

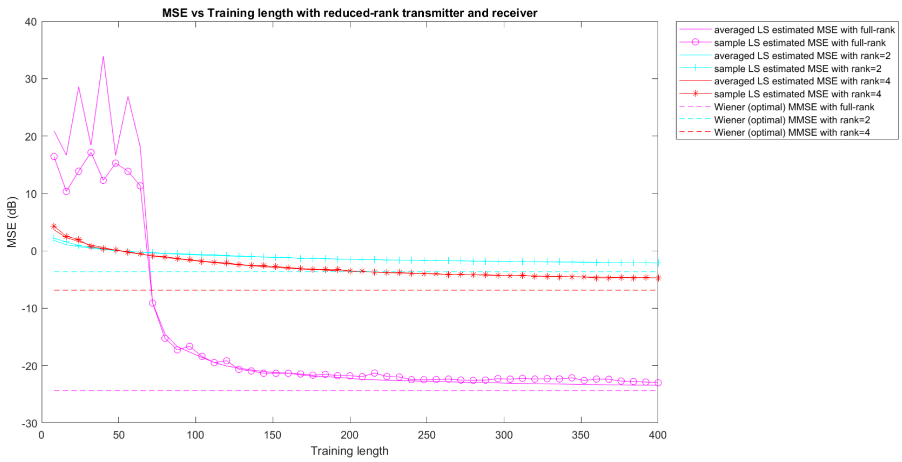

Figure 4.26: MSE vs training length with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna P 100, the number of transmit antennas N_t 64, the number of receive antennas N_r 64, the number of transmitted training symbol sequences N_s 64, the length 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

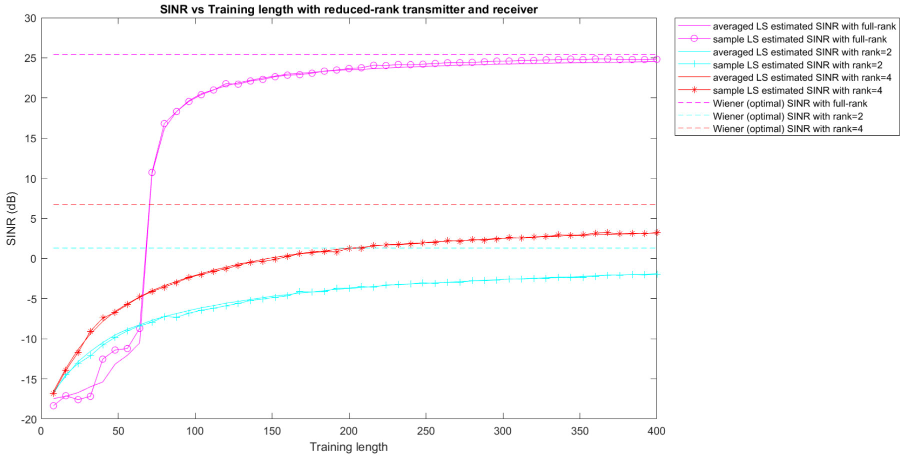

Figure 4.27: SINR vs training length with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna P 100, the number of transmit antennas N_t 64, the number of receive antennas N_r 64, the number of transmitted training symbol sequences N_s 64, the length 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

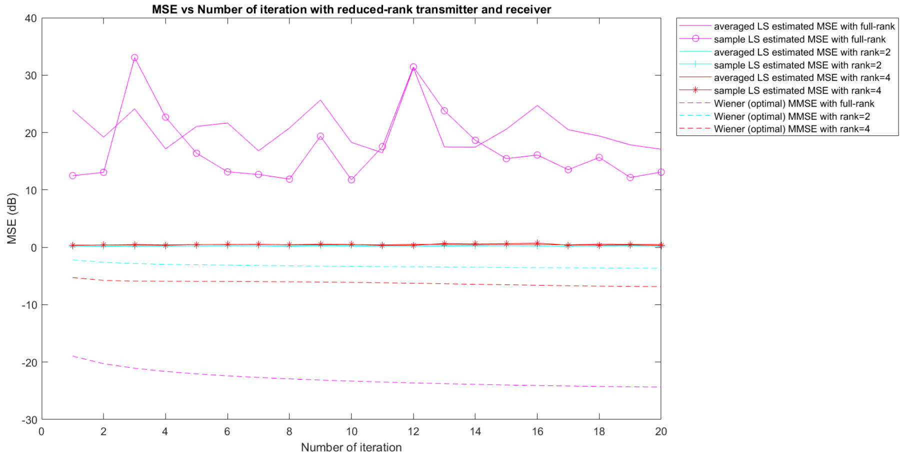

Figure 4.28: MSE vs number of bi-directional optimization or training iterations with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna P 100, the number of transmit antennas N_t 64, the number of receive antennas N_r 64, the number of transmitted 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

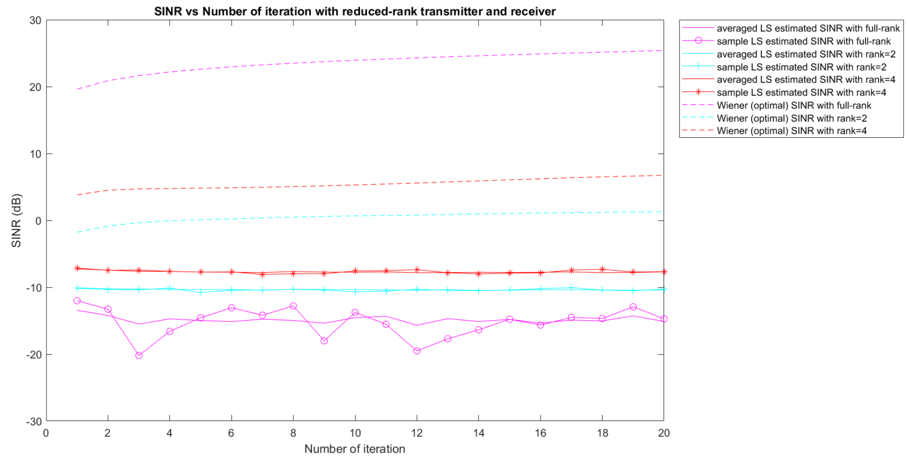

Figure 4.29: SINR vs number of bi-directional optimization or training iterations with two reduced-rank transmitter-receiver pairs, with the transmit power of each transmit or receive antenna P 100, the number of transmit antennas N_t 64, the number of receive antennas N_r 64, the number of transmitted 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.

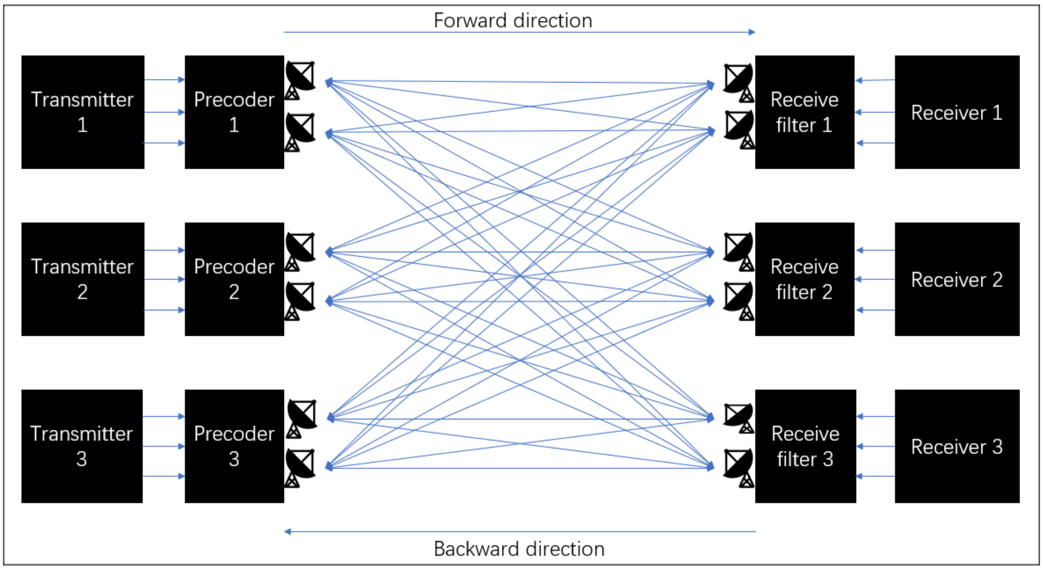

Figure 4.30: Model of interference networks

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, in 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.

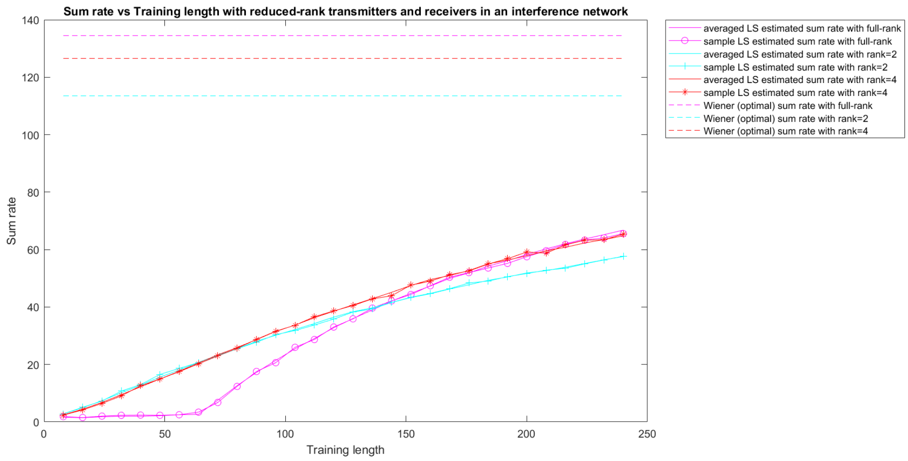

Figure 4.31: Sum rate vs training length with reduced-rank transmitters and receivers in an interference network, with the transmit power of each transmit or receive antenna P 100, the number of transmitter-receiver pairs N_p 3, the number of transmit antennas on each transmitter N_t 64, the number of 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

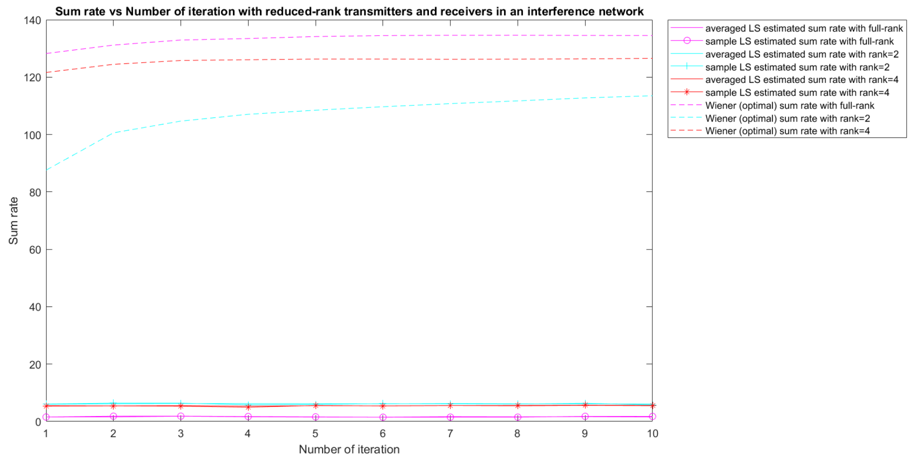

Figure 4.32: Sum rate vs number of bi-directional optimization or training iterations with reduced-rank transmitters and receivers in an interference network, with the transmit power of each transmit or receive antenna P 100, the number of transmit antennas on each transmitter N_t 64, the number of receive 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).

[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).

[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. Lamahewa, 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. Warme. “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. *"InformationTheory, 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

```
main_func_ls_bidirec_fr.m:

%% MIMO system bi-directional training with full-rank ...
    transmitter and receiver
%  Zikun Tan, MS

clear all;
close all;
clc;


%% System basic setting
%  Y=sqrt(P)*H*V*b(i)+n(i)

proj_Nt=8; ...
                                                                        ...
    % number of transmit antennas
proj_Nr=8; ...
                                                                        ...
    % number of receive antennas
proj_num_stream=8; ...
                                                                  % ...
    number of transmitted streams

proj_num_ite=20; ...
                                                                   ...
    % fixed number of iteration
proj_num_ite_ran=20; ...
                                                                % ...
    range of number of iteration
proj_num_ite_base=1; ...
                                                                % ...
```

```
    base of number of iteration

proj_train_leng=40; ...
                                                                      % ...
    fixed training length
proj_train_leng_ran=50; ...
                                                                  % ...
    length range of training sequence
proj_train_base=8; ...
                                                                       % ...
    base of training sequence

proj_ave_num=20; ...
                                                                         ...
    % number of trials for averaging

proj_pow=100; ...
                                                                            ...
    % fixed transmit power
proj_Gaussian_chan=Gaussian_chan_gen(proj_Nr,proj_Nt); ...
                        % normalized Gaussian channel
proj_nor_precoder=precoder_gen(proj_Nt,proj_num_stream,1); ...
                    % normalized initialized precoder


%% (MSE & SINR) vs Training length

%% Wiener (optimal) MMSE & SINR for fixed number of ...
    bi-directional optimization (for comparison)

[mmse_fixednumite,sinr_fixednumite,¬]=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);


%% LS estimated MSE & SINR vs varied Training length & fixed ...
    Number of iteration

% with the fixed number of iteration
[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);
% with the half of the fixed number of iteration (for ...
    comparison)
[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);
```

```
%% (MSE & SINR) vs Number of iteration

%% Wiener optimal (MMSE & SINR) vs (varied Number of ...
    iteration) (for comparison)

[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);


%% LS estimated (MSE & SINR) vs (fixed Training length & ...
    varied Number of iteration)

% with the fixed training length
[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);
% with half of the fixed training length (for comparison)
[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);


%% Plotting

% (MSE & SINR) vs Training length

% MSE vs Training length
figure(1);
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');
title('MSE vs Training length with full-rank transmitter and ...
    receiver');
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');
xlabel('Training length');
ylabel('MSE (dB)');

% SINR vs Training length
figure(2);
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');
```

```
title('SINR vs Training length with full-rank transmitter ...
    and receiver');
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');
xlabel('Training length');
ylabel('SINR (dB)');

% (MSE & SINR) vs Number of iteration

% MSE vs Number of iteration
figure(3);
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');
title('MSE vs Number of iteration with full-rank transmitter ...
    and receiver');
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');
xlabel('Number of iteration');
ylabel('MSE (dB)');

% SINR vs Number of iteration
figure(4);
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');
title('SINR vs Number of iteration with full-rank ...
    transmitter and receiver');
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');
xlabel('Number of iteration');
ylabel('SINR (dB)');




main_func_ls_bidirec_in_rr.m:

%% MIMO system bi-directional training with reduced-rank in ...
    an interference network
%  Zikun Tan, MS


```

```
%% System basic setting
%  Y=sqrt(P)*H*V*b(i)+n(i)

proj_N=3; ...
                                                                              ...
   % number of transmitter-receiver pairs
proj_Nt=64; ...
                                                                            ...
   % number of transmit antennas
proj_Nr=64; ...
                                                                            ...
   % number of receive antennas
proj_num_stream=64; ...
                                                                    % ...
   number of transmitted streams

proj_num_ite=10; ...
                                                                        ...
   % fixed number of iteration
proj_num_ite_ran=10; ...
                                                                   % ...
   range of number of iteration
proj_num_ite_base=1; ...
                                                                   % ...
   base of number of iteration

proj_train_leng=20; ...
                                                                    % ...
   fixed training length
proj_train_leng_ran=30; ...
                                                                % ...
   length range of training sequence
proj_train_base=8; ...
                                                                     % ...
   base of training sequence

proj_ave_num=20; ...
                                                                        ...
   % number of trials for averaging

red_rank=[2,4]; ...
                                                                         ...
   % reduced-rank

proj_pow=100; ...
                                                                           ...
   % transmit power
proj_Gaussian_chan_mat=in_Gaussian_chan_gen(proj_N,proj_Nr,proj_Nt); ...
         % Gaussian complex channel matrices
```

```
proj_nor_precoder_mat=in_precoder_gen(proj_N,proj_Nt,proj_num_stream,1); ...
      % normalized initialized precoders


%% Sum rate vs Training length

%% Wiener (optimal) Sum rate for fixed number of ...
    bi-directional optimization with full-rank & reduced-rank ...
    (for comparison)

% with full-rank
[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);
% with reduced-rank
[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);


%% LS estimated Sum rate vs varied Training length & fixed ...
    Number of iteration with full-rank and reduced-rank

% with full-rank
[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);
% with reduced-rank
[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);


%% Sum rate vs Number of iteration

%% Wiener (optimal) Sum rate vs varied Number of iteration ...
    with full-rank & reduced-rank (for comparison)

% with full-rank
[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);
% with reduced-rank
[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);

```

```
%% LS estimated Sum rate vs (fixed Training length & varied ...
    Number of iteration)

% with the fixed training length with full-rank
[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);
% with the fixed training length with reduced-rank
[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);


%% Plotting

% Sum rate vs Training length
figure(1);
plot(train_leng,sumrate_vs_trainleng_ave_fr,'-m', ...
     train_leng,sumrate_vs_trainleng_sam_fr,'-om', ...
     train_leng,sumrate_vs_trainleng_ave_rr(1,:),'-c', ...
     train_leng,sumrate_vs_trainleng_sam_rr(1,:),'-+c', ...
     train_leng,sumrate_vs_trainleng_ave_rr(2,:),'-r', ...
     train_leng,sumrate_vs_trainleng_sam_rr(2,:),'-*r', ...
     train_leng,sumrate_fixednumite_fr,'--m', ...
     train_leng,sumrate_fixednumite_rr(1,:),'--c', ...
     train_leng,sumrate_fixednumite_rr(2,:),'--r');
title('Sum rate vs Training length with reduced-rank ...
    transmitters  and receivers in an interference network');
legend('averaged LS estimated sum rate with full-rank', ...
       'sample LS estimated sum rate with full-rank', ...
       'averaged LS estimated sum rate with rank=2', ...
       'sample LS estimated sum rate with rank=2', ...
       'averaged LS estimated sum rate with rank=4', ...
       'sample LS estimated sum rate with rank=4', ...
       'Wiener (optimal) sum rate with full-rank', ...
       'Wiener (optimal) sum rate with rank=2', ...
       'Wiener (optimal) sum rate with rank=4');
xlabel('Training length');
ylabel('Sum rate');

% Sum rate vs Number of iteration
figure(2);
plot(numite_leng,sumrate_vs_numite_ave_fr,'-m', ...
     numite_leng,sumrate_vs_numite_sam_fr,'-om', ...
     numite_leng,sumrate_vs_numite_ave_rr(1,:),'-c', ...
     numite_leng,sumrate_vs_numite_sam_rr(1,:),'-+c', ...
```

```
    numite_leng,sumrate_vs_numite_ave_rr(2,:),'-r', ...
    numite_leng,sumrate_vs_numite_sam_rr(2,:),'-*r', ...
    numite_leng,sumrate_variednumite_fr,'--m', ...
    numite_leng,sumrate_variednumite_rr(1,:),'--c', ...
    numite_leng,sumrate_variednumite_rr(2,:),'--r');
title('Sum rate vs Number of iteration with reduced-rank  ...
   transmitters and receivers in an interference network');
legend('averaged LS estimated sum rate with full-rank', ...
       'sample LS estimated sum rate with full-rank', ...
       'averaged LS estimated sum rate with rank=2', ...
       'sample LS estimated sum rate with rank=2', ...
       'averaged LS estimated sum rate with rank=4', ...
       'sample LS estimated sum rate with rank=4', ...
       'Wiener (optimal) sum rate with full-rank', ...
       'Wiener (optimal) sum rate with rank=2', ...
       'Wiener (optimal) sum rate with rank=4');
xlabel('Number of iteration');
ylabel('Sum rate');




main_func_ls_bidirec_rr.m:

%% MIMO system bi-directional training with reduced-rank
%  Zikun Tan, MS

clear all;
close all;
clc;


%% System basic setting
%  Y=sqrt(P)*H*V*b(i)+n(i)

proj_Nt=64; ...
                                                                      ...
   % number of transmit antennas
proj_Nr=64; ...
                                                                      ...
   % number of receive antennas
proj_num_stream=64; ...
                                                                 % ...
   number of transmitted streams

proj_num_ite=20; ...
                                                                  ...
   % fixed number of iteration
proj_num_ite_ran=20; ...
```

```
                                                                      % ...
   range of number of iteration
proj_num_ite_base=1; ...
                                                                      % ...
   base of number of iteration

proj_train_leng=40; ...
                                                                       % ...
   fixed training length
proj_train_leng_ran=50; ...
                                                                  % ...
   length range of training sequence
proj_train_base=8; ...
                                                                        % ...
   base of training sequence

proj_ave_num=20; ...
                                                                          ...
   % number of trials for averaging

red_rank=[2,4]; ...
                                                                           ...
   % reduced-rank

proj_pow=100; ...
                                                                             ...
   % transmit power
proj_Gaussian_chan=Gaussian_chan_gen(proj_Nr,proj_Nt); ...
                    % Gaussian complex channel matrix
proj_nor_precoder=precoder_gen(proj_Nt,proj_num_stream,1); ...
                    % normalized initialized precoder


%% (MSE & SINR) vs Training length

%% Wiener (optimal) MMSE & SINR for fixed number of ...
   bi-directional optimization with full-rank & reduced-rank ...
   (for comparison)

% with full-rank
[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);
% with reduced-rank
[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);
```

```
%% LS estimated MSE & SINR vs varied Training length & fixed ...
    Number of iteration with full-rank and reduced-rank

% with full-rank
[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);
% with reduced-rank
[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);


%% (MSE & SINR) vs Number of iteration

%% Wiener (optimal) MMSE & SINR vs varied Number of ...
    iteration with full-rank & reduced-rank (for comparison)

% with full-rank
[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);
% with reduced-rank
[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);


%% LS estimated (MSE & SINR) vs (fixed Training length & ...
    varied Number of iteration)

% with the fixed training length with full-rank
[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);
% with the fixed training length with reduced-rank
[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);
```

```
%% Plotting

% (MSE & SINR) vs Training length

% MSE vs Training length
figure(1);
plot(train_leng,mse_vs_trainleng_ave_fr,'-m', ...
     train_leng,mse_vs_trainleng_sam_fr,'-om', ...
     train_leng,mse_vs_trainleng_ave_rr(1,:),'-c', ...
     train_leng,mse_vs_trainleng_sam_rr(1,:),'-+c', ...
     train_leng,mse_vs_trainleng_ave_rr(2,:),'-r', ...
     train_leng,mse_vs_trainleng_sam_rr(2,:),'-*r', ...
     train_leng,mmse_fixednumite_fr,'--m', ...
     train_leng,mmse_fixednumite_rr(1,:),'--c', ...
     train_leng,mmse_fixednumite_rr(2,:),'--r');
title('MSE vs Training length with reduced-rank transmitter ...
    and receiver');
legend('averaged LS estimated MSE with full-rank', ...
       'sample LS estimated MSE with full-rank', ...
       'averaged LS estimated MSE with rank=2', ...
       'sample LS estimated MSE with rank=2', ...
       'averaged LS estimated MSE with rank=4', ...
       'sample LS estimated MSE with rank=4', ...
       'Wiener (optimal) MMSE with full-rank', ...
       'Wiener (optimal) MMSE with rank=2', ...
       'Wiener (optimal) MMSE with rank=4');
xlabel('Training length');
ylabel('MSE (dB)');

% SINR vs Training length
figure(2);
plot(train_leng,sinr_vs_trainleng_ave_fr,'-m', ...
     train_leng,sinr_vs_trainleng_sam_fr,'-om', ...
     train_leng,sinr_vs_trainleng_ave_rr(1,:),'-c', ...
     train_leng,sinr_vs_trainleng_sam_rr(1,:),'-+c', ...
     train_leng,sinr_vs_trainleng_ave_rr(2,:),'-r', ...
     train_leng,sinr_vs_trainleng_sam_rr(2,:),'-*r', ...
     train_leng,sinr_fixednumite_fr,'--m', ...
     train_leng,sinr_fixednumite_rr(1,:),'--c', ...
     train_leng,sinr_fixednumite_rr(2,:),'--r');
title('SINR vs Training length with reduced-rank transmitter ...
    and receiver');
legend('averaged LS estimated SINR with full-rank', ...
       'sample LS estimated SINR with full-rank', ...
       'averaged LS estimated SINR with rank=2', ...
       'sample LS estimated SINR with rank=2', ...
       'averaged LS estimated SINR with rank=4', ...
       'sample LS estimated SINR with rank=4', ...
       'Wiener (optimal) SINR with full-rank', ...
```

```
       'Wiener (optimal) SINR with rank=2', ...
       'Wiener (optimal) SINR with rank=4');
xlabel('Training length');
ylabel('SINR (dB)');

% (MSE & SINR) vs Number of iteration

% MSE vs Number of iteration
figure(3);
plot(numite_leng,mse_vs_numite_ave_fr,'-m', ...
     numite_leng,mse_vs_numite_sam_fr,'-om', ...
     numite_leng,mse_vs_numite_ave_rr(1,:),'-c', ...
     numite_leng,mse_vs_numite_sam_rr(1,:),'-+c', ...
     numite_leng,mse_vs_numite_ave_rr(2,:),'-r', ...
     numite_leng,mse_vs_numite_sam_rr(2,:),'-*r', ...
     numite_leng,mmse_variednumite_fr,'--m', ...
     numite_leng,mmse_variednumite_rr(1,:),'--c', ...
     numite_leng,mmse_variednumite_rr(2,:),'--r');
title('MSE vs Number of iteration with reduced-rank ...
   transmitter and receiver');
legend('averaged LS estimated MSE with full-rank', ...
       'sample LS estimated MSE with full-rank', ...
       'averaged LS estimated MSE with rank=2', ...
       'sample LS estimated MSE with rank=2', ...
       'averaged LS estimated MSE with rank=4', ...
       'sample LS estimated MSE with rank=4', ...
       'Wiener (optimal) MMSE with full-rank', ...
       'Wiener (optimal) MMSE with rank=2', ...
       'Wiener (optimal) MMSE with rank=4');
xlabel('Number of iteration');
ylabel('MSE (dB)');

% SINR vs Number of iteration
figure(4);
plot(numite_leng,sinr_vs_numite_ave_fr,'-m', ...
     numite_leng,sinr_vs_numite_sam_fr,'-om', ...
     numite_leng,sinr_vs_numite_ave_rr(1,:),'-c', ...
     numite_leng,sinr_vs_numite_sam_rr(1,:),'-+c', ...
     numite_leng,sinr_vs_numite_ave_rr(2,:),'-r', ...
     numite_leng,sinr_vs_numite_sam_rr(2,:),'-*r', ...
     numite_leng,sinr_variednumite_fr,'--m', ...
     numite_leng,sinr_variednumite_rr(1,:),'--c', ...
     numite_leng,sinr_variednumite_rr(2,:),'--r');
title('SINR vs Number of iteration with reduced-rank ...
   transmitter and receiver');
legend('averaged LS estimated SINR with full-rank', ...
       'sample LS estimated SINR with full-rank', ...
       'averaged LS estimated SINR with rank=2', ...
       'sample LS estimated SINR with rank=2', ...
```

```
        'averaged LS estimated SINR with rank=4', ...
        'sample LS estimated SINR with rank=4', ...
        'Wiener (optimal) SINR with full-rank', ...
        'Wiener (optimal) SINR with rank=2', ...
        'Wiener (optimal) SINR with rank=4');
xlabel('Number of iteration');
ylabel('SINR (dB)');




main_func_ls_fr.m:

%% MIMO system Least Squares (LS) receiver with full-rank
%  Zikun Tan, MS

clear all;
close all;
clc;


%% System basic setting
%  Y=sqrt(P)*H*V*b(i)+n(i)

proj_Nt=8; ...
                                                                    ...
   % number of transmit antennas
proj_Nr=8; ...
                                                                    ...
   % number of receive antennas
proj_num_stream=8; ...
                                                              % ...
   number of transmitted streams
proj_train_leng_ran=50; ...
                                                          % ...
   length steps of training sequence
proj_ave_num=20; ...
                                                               ...
   % number of trials
proj_train_base=8; ...
                                                              % ...
   base of training sequence
proj_pow=100; ...
                                                                  ...
   % transmit power
proj_Gaussian_chan=Gaussian_chan_gen(proj_Nr,proj_Nt); ...
                       % Gaussian complex channel matrix
proj_nor_precoder=precoder_gen(proj_Nt,proj_num_stream,1); ...
                    % normalized precoder
```

```
mix_mat=sqrt(proj_pow)*proj_Gaussian_chan*proj_nor_precoder; ...
                    % mixing matrix


%% Wiener (optimal) estimated receive filter

[mmse_op,sinr_op]=wiener_mmse_sinr_fr(mix_mat,0.01);
mmse_op=mmse_op*ones(1,proj_train_leng_ran);
sinr_op=sinr_op*ones(1,proj_train_leng_ran);


%% LS estimated receive filter

[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);


%% Plotting

% plot MSE vs Training length
figure(1);
plot(train_leng,mse_ls_ave,'-m',train_leng,mse_ls_sam,'-om', ...
    train_leng,mmse_op,'--m');
title('MSE vs Training length with full-rank receiver');
legend('LS estimated averaged MSE', ...
       'LS estimated sample MSE', ...
       'Wiener (optimal) MMSE');
xlabel('Training length');
ylabel('MSE (dB)');

% plot SINR vs Training length
figure(2);
plot(train_leng,sinr_ls_ave,'-m',train_leng,sinr_ls_sam,'-om', ...
    train_leng,sinr_op,'--m');
title('SINR vs Training length with full-rank receiver');
legend('LS estimated averaged SINR', ...
       'LS estimated sample SINR', ...
       'Wiener (optimal) SINR');
xlabel('Training length');
ylabel('SINR (dB)');




main_func_ls_rr_un.m:

%% MIMO system Least Squares (LS) receiver with ...
    reduced-rank, with time-averaging Krylov subspace
```

```
%  Zikun Tan, MS

clear all;
close all;
clc;


%% System basic setting
%  Y=sqrt(P)*H*V*b(i)+n(i)

proj_Nt=8; ...
                                                                              ...
    % number of transmit antennas
proj_Nr=8; ...
                                                                              ...
    % number of receive antennas
proj_num_stream=8; ...
                                                                      % ...
    number of transmitted streams

proj_train_leng_ran=50; ...
                                                                 % ...
    length range of training sequence
proj_train_base=8; ...
                                                                      % ...
    base of training sequence

proj_ave_num=20; ...
                                                                        ...
    % number of trials for averaging

proj_pow=100; ...
                                                                           ...
    % transmit power
proj_Gaussian_chan=Gaussian_chan_gen(proj_Nr,proj_Nt); ...
                        % Gaussian complex channel
proj_nor_precoder=precoder_gen(proj_Nt,proj_num_stream,1); ...
                    % normalized precoder
mix_mat=sqrt(proj_pow)*proj_Gaussian_chan*proj_nor_precoder; ...
                  % initialized mixing matrix
red_rank=[2,4]; ...
                                                                         ...
    % reduced-rank


%% Wiener (optimal) estimated performance with full-rank

[mmse_wiener_fr,sinr_wiener_fr]=wiener_mmse_sinr_fr(mix_mat,0.01);
mmse_wiener_fr=mmse_wiener_fr*ones(1,proj_train_leng_ran);
```

```
sinr_wiener_fr=sinr_wiener_fr*ones(1,proj_train_leng_ran);


%% Wiener (optimal) estimated performance with reduced-rank

[mmse_wiener_rr,sinr_wiener_rr]=wiener_mmse_sinr_rr(mix_mat, ...
    red_rank,0.01);
mmse_wiener_rr=transpose(mmse_wiener_rr)*ones(1,proj_train_leng_ran);
sinr_wiener_rr=transpose(sinr_wiener_rr)*ones(1,proj_train_leng_ran);


%% LS estimated performance with full-rank (MSE & SINR) vs ...
    Training length

[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);


%% LS estimated performance with reduced-rank (MSE & SINR) ...
    vs Training length

[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);


%% Plotting

% plot the MSE vs Training length
figure(1);
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');
title('MSE vs Training length with reduced-rank receiver');
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', ...
```

```
        'rank=4 sample MSE');
xlabel('Training length');
ylabel('MSE (dB)');

% plot the SINR vs Training length
figure(2);
plot(train_leng,sinr_wiener_fr,'--m', ...
     train_leng,sinr_wiener_rr(1,:),'--c', ...
     train_leng,sinr_wiener_rr(2,:),'--r', ...
     train_leng,sinr_ls_fr_ave,'-m', ...
     train_leng,sinr_ls_fr_sam,'-om', ...
     train_leng,sinr_ls_rr_ave(1,:),'-c', ...
     train_leng,sinr_ls_rr_sam(1,:),'-+c', ...
     train_leng,sinr_ls_rr_ave(2,:),'-r', ...
     train_leng,sinr_ls_rr_sam(2,:),'-*r');
title('SINR vs Training length with reduced-rank receiver');
legend('Wiener (optimal) SINR', ...
        'Wiener (optimal) SINR with rank=2', ...
        'Wiener (optimal) SINR with rank=4', ...
        'full-rank average SINR', ...
        'full-rank sample SINR', ...
        'rank=2 average SINR', ...
        'rank=2 sample SINR', ...
        'rank=4 average SINR', ...
        'rank=4 sample SINR');
xlabel('Training length');
ylabel('SINR (dB)');




main_func_wiener_fr.m:

%% MIMO system Wiener (optimal) receiver with full-rank ...
    performance simulation
%  Zikun Tan, MS

clear all;
close all;
clc;


%% System basic setting
%  Y=sqrt(P)*H*V*b+n

proj_Nt=8; ...
                                                                          ...
    % number of transmit antennas
proj_Nr=8; ...
```

```
                                                                                                    ...
    % number of receive antennas
proj_num_stream=8; ...
                                                                                        % ...
    number of transmitted streams
proj_xleng=11; ...
                                                                                              ...
    % range of X-axis
proj_pow=zeros(1,proj_xleng); ...
                                                         % transmit power
for m=1:proj_xleng
    proj_pow(1,m)=0.01*10^((3*m-3)/10); ...
                                                   % transmit power, ...
        making SNR from 0dB to 30dB
end
proj_Gaussian_chan=Gaussian_chan_gen(proj_Nr,proj_Nt); ...
                        % Gaussian complex channel matrix
proj_nor_precoder=precoder_gen(proj_Nt,proj_num_stream,1); ...
                    % normalized precoder


%% (MMSE & SINR) vs SNR

[mmse,sinr,snr]=wiener_mmse_sinr_vs_snr_fr(proj_pow, ...
    proj_Gaussian_chan,proj_nor_precoder,0.01);


%% Plotting (MMSE & SINR) vs SNR

% MMSE vs SNR
figure(1);
plot(snr,mmse,'-om');
title('MMSE vs Background SNR with full-rank receiver');
xlabel('Background SNR (dB)');
ylabel('MMSE (dB)');

% SINR vs SNR
figure(2);
plot(snr,sinr,'-om');
title('SINR vs Background SNR with full-rank receiver');
xlabel('Background SNR (dB)');
ylabel('SINR (dB)');




main_func_wiener_rr.m:

%% MIMO system Wiener (optimal) receiver with reduced-rank
```

```
%  Zikun Tan, MS

clear all;
close all;
clc;


%% System basic setting
%  Y=sqrt(P)*H*V*b+n

proj_Nt=8; ...
                                                                    ...
   % number of transmit antennas
proj_Nr=8; ...
                                                                    ...
   % number of receive antennas
proj_num_stream=8; ...
                                                            % ...
   number of transmitted streams
proj_xleng=11; ...
                                                                ...
   % range of X-axis
red_rank=[2,4]; ...
                                                               ...
   % reduced rank
proj_pow=zeros(1,proj_xleng); ...
                                                   % transmit power
for m=1:proj_xleng
    proj_pow(1,m)=0.01*10^((3*m-3)/10); ...
                                              % transmit power, ...
       from 0dB to 30dB
end
proj_Gaussian_chan=Gaussian_chan_gen(proj_Nr,proj_Nt); ...
                        % Gaussian complex channel matrix
proj_nor_precoder=precoder_gen(proj_Nt,proj_num_stream,1); ...
                    % normalized precoder


%% Full-rank Wiener (optimal) (MMSE & SINR) vs SNR

[mmse_fr,sinr_fr,¬]=wiener_mmse_sinr_vs_snr_fr(proj_pow, ...
   proj_Gaussian_chan,proj_nor_precoder,0.01);


%% Reduced-rank Wiener (optimal) (MMSE & SINR) vs SNR

[mmse_rr,sinr_rr,¬]=wiener_mmse_sinr_vs_snr_rr(proj_pow, ...
   proj_Gaussian_chan,proj_nor_precoder,0.01,red_rank);

```

```

%% Plotting

snr=10*log10(proj_pow/0.01); ...
                                                        % ...
    background SNR

% MMSE vs Background SNR
figure(1);
plot(snr,mmse_fr,'-om', ...
     snr,mmse_rr(1,:),'-+c', ...
     snr,mmse_rr(2,:),'-*r');
title('MMSE vs Background SNR with reduced-rank receiver');
legend('full-rank', ...
       'rank=2', ...
       'rank=4');
xlabel('Background SNR (dB)');
ylabel('MSE (dB)');

% SINR vs Background SNR
figure(2);
plot(snr,sinr_fr,'-om', ...
     snr,sinr_rr(1,:),'-+c', ...
     snr,sinr_rr(2,:),'-*r');
title('SINR vs Background SNR with reduced-rank receiver');
legend('full-rank', ...
       'rank=2', ...
       'rank=4');
xlabel('Background SNR (dB)');
ylabel('SINR (dB)');
```

Appendix B

Self-defined functions

```
bidirec_noiseseq_gen.m:

% Generate noise sequence symbols for all bi-directional ...
    training

function [noise_mat] = ...
    bidirec_noiseseq_gen(size_1,size_2,size_3)

noise_mat=zeros(size_1,size_2,size_3);
for m=1:size_1
     noise_mat(m,:,:)=noise_mat_gen(size_2,size_3);
end

end




bidirec_trainseq_gen.m:

% Generate BPSK training sequences for all bi-directional ...
    training

function [trainseq_mat] = ...
    bidirec_trainseq_gen(size_1,size_2,size_3)

trainseq_mat=zeros(size_1,size_2,size_3);
for m=1:size_1
     trainseq_mat(m,:,:)=bpsk_mat_gen(size_2,size_3);
end

end
```

```
bpsk_mat_gen.m:

% BPSK symbol sequence generator

function bpsk_mat = bpsk_mat_gen(num_stream,train_leng)

bpsk_mat=sign(rand(num_stream,train_leng)-0.5*ones( ...
    num_stream,train_leng));

end




Gaussian_chan_gen.m:

% Generate normalized Gaussian complex channel

function Gaussian_chan = Gaussian_chan_gen(Nr,Nt)

% normalized Gaussian distributed MIMO channel
Gaussian_chan=(1/sqrt(2*Nt))*(randn(Nr,Nt)+i*randn(Nr,Nt));

end




in_bidirec_noiseseq_gen.m:

% Generate noise sequence symbols for all bi-directional ...
    training in
% interference network

function [noise_mat] = ...
    in_bidirec_noiseseq_gen(size_1,size_2,size_3,size_4)

noise_mat=zeros(size_1,size_2,size_3,size_4);
for m=1:size_1
    noise_mat(m,:,:,:)=bidirec_noiseseq_gen(size_2,size_3,size_4);
end

end
```

```
in_bidirec_trainseq_gen.m:

% Generate BPSK training sequences for all bi-directional ...
    training in
% interference network

function [trainseq_mat] = ...
    in_bidirec_trainseq_gen(size_1,size_2,size_3,size_4)

trainseq_mat=zeros(size_1,size_2,size_3,size_4);

for m=1:size_1
    trainseq_mat(m,:,:,:)=bidirec_trainseq_gen(size_2,size_3,size_4);
end

end




in_Gaussian_chan_gen.m:

% Interference network Gaussian channels generator

function [Gaussian_chan_mat] = in_Gaussian_chan_gen(N,Nt,Nr)

Gaussian_chan_mat=zeros(N,N,Nr,Nt);

% assign for each channel
for m=1:N
    for n=1:N
        Gaussian_chan_mat(m,n,:,:)=Gaussian_chan_gen(Nr,Nt);
    end
end

end




in_precoder_gen.m:

% Interference network precoders generator

function [precoder_mat] = in_precoder_gen(N,Nt,num_stream,pow)

precoder_mat=zeros(N,Nt,num_stream);
```

```

for m=1:N
    precoder_mat(m,:,:)=precoder_gen(Nt,num_stream,pow);
end

end




ls_mse_sinr_vs_numite_bidirec_fr.m:

% LS estimated MSE & SINR vs Number of bi-directional ...
    training iteration with full-rank

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)

% parameter extraction
[Nr,num_stream]=size(Gaussian_chan*nor_precoder);

% output container
mse=zeros(ave_num,num_ite_ran/num_ite_base);
sinr=zeros(ave_num,num_ite_ran/num_ite_base);

% for each trial in averaging
for m=1:ave_num
    % training sequence
    b=bidirec_trainseq_gen(2*num_ite_ran+1,num_stream,train_leng);
    % noise sequence
    noise=bidirec_noiseseq_gen(2*num_ite_ran+1,Nr,train_leng);

    % for each number of bi-directional training iteration
    for n=num_ite_base:num_ite_base:num_ite_ran
        % initialized forward training
        b_dot=reshape(b(1,:,:),[num_stream,train_leng]);
        noise_dot=reshape(noise(1,:,:),[Nr,train_leng]);

        % initialized mixing matrix
        mix_mat=sqrt(pow)*Gaussian_chan*nor_precoder;
        % sample correlation matrix
        cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                    *ctranspose(mix_mat*b_dot+noise_dot)/train_leng;
        % sample cross-correlation matrix
        cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                        *ctranspose(b_dot)/train_leng;
        % LS estimated receive filter and power normalization
        rece_fil_ls=cor_mat_ls\cro_cor_mat_ls;
```

```
        % power normalization
        rece_fil_ls_nor=mat_pow_nor(rece_fil_ls);

        % for bi-directional training with each number of ...
            iteration
        for n_d=1:n
            % backward training
            b_dot=reshape(b(2*n_d,:,:),[num_stream,train_leng]);
            noise_dot=reshape(noise(2*n_d,:,:),[Nr,train_leng]);

            % backward mixing matrix
            mix_mat=transpose(sqrt(pow)*Gaussian_chan)* ...
                conj(rece_fil_ls_nor);
            % sample correlation matrix
            cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                        *ctranspose(mix_mat*b_dot+noise_dot)/ ...
                            train_leng;
            % sample cross-correlation matrix
            cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                             *ctranspose(b_dot)/train_leng;
            % LS estimated precoder
            precoder_ls=conj(cor_mat_ls\cro_cor_mat_ls);
            % power normalization
            precoder_ls_nor=mat_pow_nor(precoder_ls);

            % forward training
            b_dot=reshape(b(2*n_d+1,:,:),[num_stream,train_leng]);
            noise_dot=reshape(noise(2*n_d+1,:,:),[Nr,train_leng]);

            % forward mixing matrix
            mix_mat=sqrt(pow)*Gaussian_chan*precoder_ls_nor;
            % sample correlation matrix
            cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                        *ctranspose(mix_mat*b_dot+noise_dot)/ ...
                            train_leng;
            % sample cross-correlation matrix
            cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                             *ctranspose(b_dot)/train_leng;
            % LS estimated receive filter
            rece_fil_ls=cor_mat_ls\cro_cor_mat_ls;
            % power normalization
            rece_fil_ls_nor=mat_pow_nor(rece_fil_ls);
        end

        % statistical correlation matrix
        cor_mat_op=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);

        % MSE & SINR computation
        mse_temp=zeros(1,num_stream);
```

```
        sinr_temp=zeros(1,num_stream);
        for n_d=1:num_stream

            % MSE computation
            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));
            mse_temp(1,n_d)=mse_temp_val;

            % signal power
            signal_pow=abs(ctranspose(rece_fil_ls(:,n_d))* ...
               mix_mat(:,n_d))^2;

            % interference power
            interf_pow=0;
            for n_dd=1:num_stream
                if n_dd==n_d
                    % pass
                else
                    interf_pow=interf_pow+abs(ctranspose ...
                       (rece_fil_ls(:,n_d))*mix_mat(:,n_dd))^2;
                end
            end

            % noise power
            noise_pow=abs(ctranspose(rece_fil_ls(:,n_d))* ...
               noise_covar*rece_fil_ls(:,n_d));

            % SINR computation
            sinr_temp_val=signal_pow/(interf_pow+noise_pow);
            sinr_temp(1,n_d)=sinr_temp_val;
        end

        % average over all streams
        mse(m,n/num_ite_base)=mean(mse_temp,2);
        sinr(m,n/num_ite_base)=mean(sinr_temp,2);
    end
end

% data averaging, sample data extraction and decibel conversion
mse_ave=10*log10(mean(mse));
mse_sam=10*log10(mse(1,:));
sinr_ave=10*log10(mean(sinr));
sinr_sam=10*log10(sinr(1,:));

% number of iteration axis
```

```
num_ite_leng=zeros(1,num_ite_ran/num_ite_base);
for m=1:num_ite_ran/num_ite_base
    num_ite_leng(1,m)=m*num_ite_base;
end

end




ls_mse_sinr_vs_numite_bidirec_rr.m:

% LS estimated MSE & SINR vs Number of bi-directional ...
   training iteration with reduced-rank

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)

% parameter extraction
[Nr,num_stream]=size(Gaussian_chan*nor_precoder);

% output container
mse=zeros(size(red_rank,2),ave_num,num_ite_ran/num_ite_base);
sinr=zeros(size(red_rank,2),ave_num,num_ite_ran/num_ite_base);

% for every reduced-rank
for rr_ind=1:size(red_rank,2)
    % assign current reduced-rank
    rr=red_rank(rr_ind);

    % for each trial in averaging
    for m=1:ave_num
        % training sequence
        b=bidirec_trainseq_gen(2*num_ite_ran+1,num_stream,train_leng);
        % noise sequence
        noise=bidirec_noiseseq_gen(2*num_ite_ran+1,Nr,train_leng);

        % for each number of bi-directional training iteration
        for n=num_ite_base:num_ite_base:num_ite_ran

            % initialized forward training

            b_dot=reshape(b(1,:,:),[num_stream,train_leng]);
            noise_dot=reshape(noise(1,:,:),[Nr,train_leng]);

            % initialized mixing matrix:
            mix_mat=sqrt(pow)*Gaussian_chan*nor_precoder;
            % sample correlation matrix
```

```
            cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                        *ctranspose(mix_mat*b_dot+noise_dot)/ ...
                            train_leng;
            % sample cross-correlation matrix
            cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                            *ctranspose(b_dot)/train_leng;
            % LS estimated receive filter and power ...
                normalization
            [rece_fil_ls,¬,¬]=ls_rr_fil_mse_sinr_cal(mix_mat, ...
                cor_mat_ls,cro_cor_mat_ls,noise_covar,rr,false);
            % power normalization
            rece_fil_ls_nor=mat_pow_nor(rece_fil_ls);

            % for bi-directional training with each number ...
                of iteration
            for n_d=1:n

                % backward training

                b_dot=reshape(b(2*n_d,:,:),[num_stream,train_leng]);
                noise_dot=reshape(noise(2*n_d,:,:),[Nr,train_leng]);

                % backward mixing matrix
                mix_mat=transpose(sqrt(pow)*Gaussian_chan)* ...
                    conj(rece_fil_ls_nor);
                % sample correlation matrix
                cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                            *ctranspose(mix_mat*b_dot+noise_dot)/ ...
                                train_leng;
                % sample cross-correlation matrix
                cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                                *ctranspose(b_dot)/train_leng;
                % LS estimated precoder
                [precoder_ls,¬,¬]=ls_rr_fil_mse_sinr_cal(mix_mat, ...
                    cor_mat_ls,cro_cor_mat_ls,noise_covar,rr,false);
                precoder_ls=conj(precoder_ls);
                % power normalization
                precoder_ls_nor=mat_pow_nor(precoder_ls);

                % forward training

                b_dot=reshape(b(2*n_d+1,:,:),[num_stream,train_leng]);
                noise_dot=reshape(noise(2*n_d+1,:,:),[Nr,train_leng]);

                % forward mixing matrix
                mix_mat=sqrt(pow)*Gaussian_chan*precoder_ls_nor;
                % sample correlation matrix
                cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                            *ctranspose(mix_mat*b_dot+noise_dot)/ ...
```

```
                                    train_leng;
                        % sample cross-correlation matrix
                        cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                                       *ctranspose(b_dot)/train_leng;
                        % LS estimated receive filter
                        [rece_fil_ls,mse_val,sinr_val]= ...
                            ls_rr_fil_mse_sinr_cal(mix_mat,cor_mat_ls, ...
                            cro_cor_mat_ls,noise_covar,rr,true);
                        % power normalization
                        rece_fil_ls_nor=mat_pow_nor(rece_fil_ls);
                    end

                    % assign MSE and SINR values
                    mse(rr_ind,m,n/num_ite_base)=mse_val;
                    sinr(rr_ind,m,n/num_ite_base)=sinr_val;
                end
            end
        end

        % average data preallocation
        mse_ave=zeros(size(red_rank,2),num_ite_ran/num_ite_base);
        sinr_ave=zeros(size(red_rank,2),num_ite_ran/num_ite_base);

        % averaging & decibel conversion
        for m=1:size(red_rank,2)
            mse_ave(m,:)=10*log10(mean(squeeze(mse(m,:,:))));
            sinr_ave(m,:)=10*log10(mean(squeeze(sinr(m,:,:))));
        end

        % sample data preallocation
        mse_sam=zeros(size(red_rank,2),num_ite_ran/num_ite_base);
        sinr_sam=zeros(size(red_rank,2),num_ite_ran/num_ite_base);

        % extraction & decibel conversion
        for m=1:size(red_rank,2)
            mse_sam(m,:)=10*log10(squeeze(mse(m,1,:)));
            sinr_sam(m,:)=10*log10(squeeze(sinr(m,1,:)));
        end

        % number of iteration axis
        num_ite_leng=zeros(1,num_ite_ran/num_ite_base);
        for m=1:num_ite_ran/num_ite_base
            num_ite_leng(1,m)=m*num_ite_base;
        end

        end
```

```
ls_mse_sinr_vs_trainleng_bidirec_fr.m

% LS estimated MSE & SINR vs Training length with full-rank

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)

% parameter extraction
[Nr,num_stream]=size(Gaussian_chan*nor_precoder);
% output container
mse=zeros(ave_num,train_leng_ran);
sinr=zeros(ave_num,train_leng_ran);

% for each trial in averaging
for m=1:ave_num
    % training sequence for each iteration
    b=bidirec_trainseq_gen(2*num_ite+1,num_stream,train_base* ...
        train_leng_ran);
    % noise sequence for each iteration
    noise=bidirec_noiseseq_gen(2*num_ite+1,Nr,train_base* ...
        train_leng_ran);

    % for each training length
    for n=train_base:train_base:train_base*train_leng_ran
        % initialized forward training
        b_dot=reshape(b(1,:,[1:n]),[num_stream,n]);
        noise_dot=reshape(noise(1,:,[1:n]),[Nr,n]);

        % initialized mixing matrix
        mix_mat=sqrt(pow)*Gaussian_chan*nor_precoder;

        % sample correlation matrix
        cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                   *ctranspose(mix_mat*b_dot+noise_dot)/n;
        % sample cross-correlation matrix
        cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                       *ctranspose(b_dot)/n;
        % LS estimated receive filter and power normalization
        rece_fil_ls=cor_mat_ls\cro_cor_mat_ls;
        rece_fil_ls_nor=mat_pow_nor(rece_fil_ls);

        % for each number of bi-directional training
        for n_d=1:num_ite
            % backward training
            b_dot=reshape(b(2*n_d,:,[1:n]),[num_stream,n]);
            noise_dot=reshape(noise(2*n_d,:,[1:n]),[Nr,n]);

```

```
        % backward mixing matrix
        mix_mat=transpose(sqrt(pow)*Gaussian_chan)* ...
            conj(rece_fil_ls_nor);
        % sample correlation matrix
        cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                    *ctranspose(mix_mat*b_dot+noise_dot)/n;
        % sample cross-correlation matrix
        cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                        *ctranspose(b_dot)/n;
        % LS estimated precoder and power normalization
        precoder_ls=conj(cor_mat_ls\cro_cor_mat_ls);
        precoder_ls_nor=mat_pow_nor(precoder_ls);

        % forward training
        b_dot=reshape(b(2*n_d+1,:,[1:n]),[num_stream,n]);
        noise_dot=reshape(noise(2*n_d+1,:,[1:n]),[Nr,n]);

        % forward mixing matrix
        mix_mat=sqrt(pow)*Gaussian_chan*precoder_ls_nor;
        % sample correlation matrix
        cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                    *ctranspose(mix_mat*b_dot+noise_dot)/n;
        % sample cross-correlation matrix
        cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                        *ctranspose(b_dot)/n;
        % LS estimated combiner and power normalization
        rece_fil_ls=cor_mat_ls\cro_cor_mat_ls;
        rece_fil_ls_nor=mat_pow_nor(rece_fil_ls);
    end

    % statistical correlation matrix
    cor_mat_op=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);

    % MSE & SINR computation
    mse_temp=zeros(1,num_stream);
    sinr_temp=zeros(1,num_stream);
    for n_d=1:num_stream

        % MSE computation
        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));
        mse_temp(1,n_d)=mse_temp_val;

        % signal power
        signal_pow=abs(ctranspose(rece_fil_ls(:,n_d))* ...
```

```
                    mix_mat(:,n_d))^2;

                % interference power
                interf_pow=0;
                for n_dd=1:num_stream
                    if n_dd==n_d
                        % pass
                    else
                        interf_pow=interf_pow+abs(ctranspose ...
                            (rece_fil_ls(:,n_d))*mix_mat(:,n_dd))^2;
                    end
                end

                % noise power
                noise_pow=abs(ctranspose(rece_fil_ls(:,n_d))* ...
                    noise_covar*rece_fil_ls(:,n_d));

                % SINR computation
                sinr_temp_val=signal_pow/(interf_pow+noise_pow);
                sinr_temp(1,n_d)=sinr_temp_val;
            end

            % average over all streams
            mse(m,n/train_base)=mean(mse_temp,2);
            sinr(m,n/train_base)=mean(sinr_temp,2);
        end
end

% data averaging, sample data extraction and decibel conversion
mse_ave=10*log10(mean(mse));
mse_sam=10*log10(mse(1,:));
sinr_ave=10*log10(mean(sinr));
sinr_sam=10*log10(sinr(1,:));

% training length axis
train_leng=zeros(1,train_leng_ran);
for m=1:train_leng_ran
    train_leng(1,m)=m*train_base;
end

end




ls_mse_sinr_vs_trainleng_bidirec_rr.m:

% LS estimated MSE & SINR vs Training length with reduced-rank

```

```
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)

% parameter extraction
[Nr,num_stream]=size(Gaussian_chan*nor_precoder);
% output container
mse=zeros(size(red_rank,2),ave_num,train_leng_ran);
sinr=zeros(size(red_rank,2),ave_num,train_leng_ran);

% for each reduced-rank
for rr_ind=1:size(red_rank,2)
    % assign reduced-rank
    rr=red_rank(rr_ind);

    % for each trial in averaging
    for m=1:ave_num

        % training sequence
        b=bidirec_trainseq_gen(2*num_ite+1,num_stream,train_base ...
            *train_leng_ran);
        % noise sequence
        noise=bidirec_noiseseq_gen(2*num_ite+1,Nr,train_base* ...
            train_leng_ran);

        % for each training length
        for n=train_base:train_base:train_base*train_leng_ran

            % initialized forward training
            b_dot=reshape(b(1,:,[1:n]),[num_stream,n]);
            noise_dot=reshape(noise(1,:,[1:n]),[Nr,n]);

            % initialized mixing matrix
            mix_mat=sqrt(pow)*Gaussian_chan*nor_precoder;

            % sample correlation matrix
            cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                       *ctranspose(mix_mat*b_dot+noise_dot)/n;
            % sample cross-correlation matrix
            cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                           *ctranspose(b_dot)/n;
            % reduced-rank LS estimated receive filter
            [rece_fil,¬,¬]=ls_rr_fil_mse_sinr_cal(mix_mat, ...
                cor_mat_ls,cro_cor_mat_ls,noise_covar,rr,false);
            % power normalization
            rece_fil_nor=mat_pow_nor(rece_fil);

            % for each number of bi-directional training
            for n_d=1:num_ite
```

```
                
                % backward training
                b_dot=reshape(b(2*n_d,:,[1:n]),[num_stream,n]);
                noise_dot=reshape(noise(2*n_d,:,[1:n]),[Nr,n]);

                % backward mixing matrix
                mix_mat=transpose(sqrt(pow)*Gaussian_chan)* ...
                    conj(rece_fil_nor);
                % sample correlation matrix
                cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                           *ctranspose(mix_mat*b_dot+noise_dot)/n;
                % sample cross-correlation matrix
                cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                               *ctranspose(b_dot)/n;
                % reduced-rank LS estimated precoder
                [precoder,¬,¬]=ls_rr_fil_mse_sinr_cal(mix_mat, ...
                    cor_mat_ls,cro_cor_mat_ls,noise_covar,rr,false);
                precoder=conj(precoder);
                % power normalization
                precoder_nor=mat_pow_nor(precoder);

                % forward training
                b_dot=reshape(b(2*n_d+1,:,[1:n]),[num_stream,n]);
                noise_dot=reshape(noise(2*n_d+1,:,[1:n]),[Nr,n]);

                % forward mixing matrix
                mix_mat=sqrt(pow)*Gaussian_chan*precoder_nor;
                % sample correlation matrix
                cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                           *ctranspose(mix_mat*b_dot+noise_dot)/n;
                % sample cross-correlation matrix
                cro_cor_mat_ls=(mix_mat*b_dot+noise_dot) ...
                               *ctranspose(b_dot)/n;
                % reduced-rank LS estimated receive filter
                [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);
                % power normalization
                rece_fil_nor=mat_pow_nor(rece_fil);
            end

            mse(rr_ind,m,n/train_base)=mse_val;
            sinr(rr_ind,m,n/train_base)=sinr_val;
        end
    end
end

% average data preallocation
mse_ave=zeros(size(red_rank,2),train_leng_ran);
```

```
sinr_ave=zeros(size(red_rank,2),train_leng_ran);

% averaging & decibel conversion
for m=1:size(red_rank,2)
    mse_ave(m,:)=10*log10(mean(squeeze(mse(m,:,:))));
    sinr_ave(m,:)=10*log10(mean(squeeze(sinr(m,:,:))));
end

% sample data preallocation
mse_sam=zeros(size(red_rank,2),train_leng_ran);
sinr_sam=zeros(size(red_rank,2),train_leng_ran);

% extraction & decibel conversion
for m=1:size(red_rank,2)
    mse_sam(m,:)=10*log10(squeeze(mse(m,1,:)));
    sinr_sam(m,:)=10*log10(squeeze(sinr(m,1,:)));
end

% training length axis
train_leng=zeros(1,train_leng_ran);
for m=1:train_leng_ran
    train_leng(1,m)=m*train_base;
end

end




ls_mse_sinr_vs_trainleng_fr.m:

% LS estimated MSE & SINR vs Training length with full-rank

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)

% parameter extraction
[Nr,num_stream]=size(mix_mat);

% statistical correlation matrix
cor_mat_op=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);

% output container
mse=zeros(ave_num,train_leng_ran);
sinr=zeros(ave_num,train_leng_ran);

% for each trial
for m=1:ave_num
```

```
% training sequence
b=bpsk_mat_gen(num_stream,train_base*train_leng_ran);
% Gaussian additive noise
noise=noise_mat_gen(Nr,train_base*train_leng_ran);

% for each training length
for n=train_base:train_base:train_base*train_leng_ran

    % sample correlation matrix
    cor_mat_ls=(mix_mat*b(:,[1:n])+noise(:,[1:n]))* ...
        ctranspose(mix_mat*b(:,[1:n])+noise(:,[1:n]))/n;
    % sample cross-correlation matrix
    cro_cor_mat_ls=(mix_mat*b(:,[1:n])+noise(:,[1:n]))* ...
        ctranspose(b(:,[1:n]))/n;
    % LS estimated receive filter
    rece_fil_ls_fr=cor_mat_ls\cro_cor_mat_ls;

    mse_temp=zeros(1,num_stream);
    sinr_temp=zeros(1,num_stream);
    for n_d=1:num_stream
        % MSE computation
        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));
        mse_temp(1,n_d)=mse_temp_val;

        % signal power
        signal_pow=abs(ctranspose(rece_fil_ls_fr(:,n_d))* ...
            mix_mat(:,n_d))^2;

        % interference power
        interf_pow=0;
        for n_dd=1:num_stream
            if n_dd==n_d
                % pass
            else
                interf_pow=interf_pow+abs(ctranspose ...
                    (rece_fil_ls_fr(:,n_d))*mix_mat(:,n_dd))^2;
            end
        end

        % noise power
        noise_pow=abs(ctranspose(rece_fil_ls_fr(:,n_d))* ...
            noise_covar*rece_fil_ls_fr(:,n_d));

        % SINR computation
```

```
            sinr_temp_val=signal_pow/(interf_pow+noise_pow);
            sinr_temp(1,n_d)=sinr_temp_val;
        end
        % average over all streams
        mse(m,n/train_base)=mean(mse_temp,2);
        sinr(m,n/train_base)=mean(sinr_temp,2);
    end
end

% averaging over trials & decibel conversion
mse_ave=10*log10(mean(mse));
sinr_ave=10*log10(mean(sinr));
% sample extracting & decibel conversion
mse_sam=10*log10(mse(1,:));
sinr_sam=10*log10(sinr(1,:));

% training length axis
train_leng=zeros(1,train_leng_ran);
for m=1:train_leng_ran
    train_leng(1,m)=m*train_base;
end

end




ls_mse_sinr_vs_trainleng_rr_un.m:

% LS estimated MSE & SINR vs Training length with ...
    reduced-rank and time-averaging Krylov subspace

function [mse_ave,mse_sam,sinr_ave,sinr_sam,train_leng] = ...
    ls_mse_sinr_vs_trainleng_rr_un(mix_mat,red_rank,ave_num, ...
    train_leng_ran,train_base,noise_covar)

% parameter extraction
[Nr,num_stream]=size(mix_mat);

% statistical correlation matrix
cor_mat_op=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);

mse=zeros(size(red_rank,2),ave_num,train_leng_ran);
sinr=zeros(size(red_rank,2),ave_num,train_leng_ran);

% for each reduced-rank
for rr_ind=1:size(red_rank,2)
    % assign reduced-rank
    rr=red_rank(rr_ind);
```

```
    % Krylov subspace preallocation
    krylov=zeros(Nr,rr);

    % for each trial
    for m=1:ave_num

        % training sequence
        b=bpsk_mat_gen(num_stream,train_base*train_leng_ran);
        % Gaussian additive noise
        noise=noise_mat_gen(Nr,train_base*train_leng_ran);

        % for each training length
        for n=train_base:train_base:train_base*train_leng_ran

            % sample correlation matrix
            cor_mat_ls=(mix_mat*b(:,[1:n])+noise(:,[1:n]))* ...
                ctranspose(mix_mat*b(:,[1:n])+noise(:,[1:n]))/n;

            % sample cross-correlation matrix
            cro_cor_mat_ls=(mix_mat*b(:,[1:n])+noise(:,[1:n]))* ...
                ctranspose(b(:,[1:n]))/n;

            mse_temp=zeros(1,num_stream);
            sinr_temp=zeros(1,num_stream);

            for n_d=1:num_stream
                % Krylov subspace
                for n_dot=1:rr
                    % Krylov subspace
                    krylov(:,n_dot)=cor_mat_ls^(n_dot-1)* ...
                        cro_cor_mat_ls(:,n_d);
                end

                rece_fil_ls=krylov*((ctranspose(krylov)*cor_mat_ls ...
                    *krylov) ...
                             \ctranspose(krylov))*cro_cor_mat_ls ...
                                 (:,n_d);

                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);

                mse_temp(1,n_d)=mse_temp_val;

                % signal power
```

```
                signal_pow=abs(ctranspose(rece_fil_ls)* ...
                    mix_mat(:,n_d))^2;

                % interference power
                interf_pow=0;
                for n_dd=1:num_stream
                    if n_dd==n_d
                        % pass
                    else
                        interf_pow=interf_pow+abs ...
                            (ctranspose(rece_fil_ls)*mix_mat(:,n_dd))^2;
                    end
                end

                % noise power
                noise_pow=abs(ctranspose(rece_fil_ls)* ...
                    noise_covar*rece_fil_ls);

                sinr_temp_val=signal_pow/(interf_pow+noise_pow);

                sinr_temp(1,n_d)=sinr_temp_val;
            end
            % average over streams
            mse_temp=mean(mse_temp,2);
            sinr_temp=mean(sinr_temp,2);

            mse(rr_ind,m,n/train_base)=mse_temp;
            sinr(rr_ind,m,n/train_base)=sinr_temp;
        end
    end
end

% average data preallocation
mse_ave=zeros(size(red_rank,2),train_leng_ran);
sinr_ave=zeros(size(red_rank,2),train_leng_ran);

% averaging & decibel conversion
for m=1:size(red_rank,2)
    mse_ave(m,:)=10*log10(mean(squeeze(mse(m,:,:))));
    sinr_ave(m,:)=10*log10(mean(squeeze(sinr(m,:,:))));
end

% sample data preallocation
mse_sam=zeros(size(red_rank,2),train_leng_ran);
sinr_sam=zeros(size(red_rank,2),train_leng_ran);

% extraction & decibel conversion
for m=1:size(red_rank,2)
    mse_sam(m,:)=10*log10(squeeze(mse(m,1,:)));
```

```
    sinr_sam(m,:)=10*log10(squeeze(sinr(m,1,:)));
end

% training length axis
train_leng=zeros(1,train_leng_ran);
for m=1:train_leng_ran
    train_leng(1,m)=m*train_base;
end

end




ls_rr_fil_mse_sinr_cal.m:

% LS estimated reduced-rank precoder & combiner, MSE and ...
    SINR calculator

function [fil,mse,sinr] = ...
    ls_rr_fil_mse_sinr_cal(mix_mat,cor_mat,cro_cor_mat,noise_covar,rr, ...
    flag)

% dimension extraction
[Nr,num_stream]=size(mix_mat);
% Krylov subspace preallocation
krylov=zeros(Nr,rr);
% filter preallocation
fil=zeros(Nr,num_stream);

% statistical correlation matrix
cor_mat_op=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);

% for each transmitted stream, for computing filters ...
    (precoders or combiners)
for m=1:num_stream

    % constructe Krylov subspace for the current transmitted ...
        stream
    for n=1:rr
        krylov(:,n)=cor_mat^(n-1)*cro_cor_mat(:,m);
    end

    % reduced-rank filter
    fil(:,m)=krylov*((ctranspose(krylov)*cor_mat*krylov) ...
             \ctranspose(krylov))*cro_cor_mat(:,m);
end

% need to compute MSE & SINR
```

```
if flag==true

    mse_temp=zeros(1,num_stream);
    sinr_temp=zeros(1,num_stream);

    % for each transmitted stream, for computing MSE and SINR
    for m=1:num_stream
        % MSE
        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));
        mse_temp(1,m)=mse_temp_val;

        % signal power
        signal_pow=abs(ctranspose(fil(:,m))*mix_mat(:,m))^2;

        % interference power
        interf_pow=0;
        for m_dot=1:num_stream
            if m_dot==m
                % pass
            else
                interf_pow=interf_pow+abs(ctranspose(fil(:,m)) ...
                    *mix_mat(:,m_dot))^2;
            end
        end

        % noise power
        noise_pow=abs(ctranspose(fil(:,m))*noise_covar*fil(:,m));

        % SINR
        sinr_temp_val=signal_pow/(interf_pow+noise_pow);
        sinr_temp(1,m)=sinr_temp_val;
    end

    % averaging over all transmitted streams
    mse=mean(mse_temp,2);
    sinr=mean(sinr_temp,2);

% no need to compute MSE & SINR
else
    mse=0;
    sinr=0;
end

end
```

ls_rr_fil_sumrate_in_cal.m:

```
% Interference network LS reduced-rank filter, sum rate ...
    calculator

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)
% dimension extraction
[Nr,num_stream]=size(mix_mat11);
% Krylov subspace preallocation
krylov1=zeros(Nr,rr);
krylov2=zeros(Nr,rr);
krylov3=zeros(Nr,rr);
% filter preallocation
fil1=zeros(Nr,num_stream);
fil2=zeros(Nr,num_stream);
fil3=zeros(Nr,num_stream);

% for each transmitted stream, for computing filters
for m=1:num_stream

    % constructe Krylov subspace for the current transmitted ...
        stream
    for n=1:rr
        krylov1(:,n)=cor_mat1^(n-1)*cro_cor_mat1(:,m);
        krylov2(:,n)=cor_mat2^(n-1)*cro_cor_mat2(:,m);
        krylov3(:,n)=cor_mat3^(n-1)*cro_cor_mat3(:,m);
    end

    % reduced-rank filters
    fil1(:,m)=krylov1*((ctranspose(krylov1)*cor_mat1*krylov1) ...
        ...
              \ctranspose(krylov1))*cro_cor_mat1(:,m);
    fil2(:,m)=krylov2*((ctranspose(krylov2)*cor_mat2*krylov2) ...
        ...
              \ctranspose(krylov2))*cro_cor_mat2(:,m);
    fil3(:,m)=krylov3*((ctranspose(krylov3)*cor_mat3*krylov3) ...
        ...
              \ctranspose(krylov3))*cro_cor_mat3(:,m);
end

% need to compute sum rate
```

```
if flag==true

    sumrate1=zeros(1,num_stream);
    sumrate2=zeros(1,num_stream);
    sumrate3=zeros(1,num_stream);

    % for each transmitted stream, for computing MSE and SINR
    for m=1:num_stream

        % signal power
        signal_pow1=abs(ctranspose(fil1(:,m))*mix_mat11(:,m))^2;
        signal_pow2=abs(ctranspose(fil2(:,m))*mix_mat22(:,m))^2;
        signal_pow3=abs(ctranspose(fil3(:,m))*mix_mat33(:,m))^2;

        % interference power
        interf_pow1=0;
        interf_pow2=0;
        interf_pow3=0;
        % interference from its own transmitter
        for m_dot=1:num_stream
            if m_dot==m
                % pass
            else
                interf_pow1=interf_pow1+abs(ctranspose(fil1(:,m)) ...
                    *mix_mat11(:,m_dot))^2;
                interf_pow2=interf_pow2+abs(ctranspose(fil2(:,m)) ...
                    *mix_mat22(:,m_dot))^2;
                interf_pow3=interf_pow3+abs(ctranspose(fil3(:,m)) ...
                    *mix_mat33(:,m_dot))^2;
            end
        end

        % interference from other transmitters
        for m_dot=1:num_stream
            interf_pow1=interf_pow1+abs(ctranspose(fil1(:,m)) ...
                *mix_mat21(:,m_dot))^2 ...
                                 +abs(ctranspose(fil1(:,m)) ...
                                     *mix_mat31(:,m_dot))^2;
            interf_pow2=interf_pow2+abs(ctranspose(fil2(:,m)) ...
                *mix_mat12(:,m_dot))^2 ...
                                 +abs(ctranspose(fil2(:,m)) ...
                                     *mix_mat32(:,m_dot))^2;
            interf_pow3=interf_pow3+abs(ctranspose(fil3(:,m)) ...
                *mix_mat13(:,m_dot))^2 ...
                                 +abs(ctranspose(fil3(:,m)) ...
                                     *mix_mat23(:,m_dot))^2;
        end

        % noise power
```

```
        noise_pow1=abs(ctranspose(fil1(:,m))*noise_covar*fil1(:,m));
        noise_pow2=abs(ctranspose(fil2(:,m))*noise_covar*fil2(:,m));
        noise_pow3=abs(ctranspose(fil3(:,m))*noise_covar*fil3(:,m));

        % sum rate
        sumrate1(1,m)=log2(1+signal_pow1/(interf_pow1+noise_pow1));
        sumrate2(1,m)=log2(1+signal_pow2/(interf_pow2+noise_pow2));
        sumrate3(1,m)=log2(1+signal_pow3/(interf_pow3+noise_pow3));
    end

    % network sum rate
    sumrate=sum(sumrate1)+sum(sumrate2)+sum(sumrate3);

% no need to compute sum rate
else
    sumrate=0;
end

end




ls_sumrate_vs_numite_bidirec_in_fr.m:

% LS estimated Sum rate vs Number of bi-directional training ...
    iteration with full-rank
% in interference network

function [sumrate_ave,sumrate_sam,num_ite_leng] = ...
    ls_sumrate_vs_numite_bidirec_in_fr(pow,Gaussian_chan_mat, ...
    nor_precoder_mat,ave_num,train_leng,num_ite_ran,num_ite_base, ...
    noise_covar)

% parameter extraction
N=size(Gaussian_chan_mat,1);
Nr=size(squeeze(Gaussian_chan_mat(1,1,:,:)),1);
num_stream=size(squeeze(nor_precoder_mat(1,:,:)),2);
% temporary container
sumrate1=zeros(1,num_stream);
sumrate2=zeros(1,num_stream);
sumrate3=zeros(1,num_stream);
% output container
sumrate=zeros(ave_num,num_ite_ran/num_ite_base);

% for each trial in averaging
for m=1:ave_num
    % training sequence
    b=in_bidirec_trainseq_gen(N,2*num_ite_ran+1,num_stream, ...
```

```
        train_leng);
    % noise sequence
    noise=in_bidirec_noiseseq_gen(N,2*num_ite_ran+1,Nr,train_leng);

    % for each number of bi-directional training iteration
    for n=num_ite_base:num_ite_base:num_ite_ran
        % initialized forward training
        b1_dot=reshape(b(1,1,:,:),[num_stream,train_leng]);
        b2_dot=reshape(b(2,1,:,:),[num_stream,train_leng]);
        b3_dot=reshape(b(3,1,:,:),[num_stream,train_leng]);
        noise1_dot=reshape(noise(1,1,:,:),[Nr,train_leng]);
        noise2_dot=reshape(noise(2,1,:,:),[Nr,train_leng]);
        noise3_dot=reshape(noise(3,1,:,:),[Nr,train_leng]);

        % receive filter updating, precoder --> receive filter
        % initialized mixing matrices
        mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:)) ...
            *squeeze(nor_precoder_mat(1,:,:));
        mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:)) ...
            *squeeze(nor_precoder_mat(2,:,:));
        mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:)) ...
            *squeeze(nor_precoder_mat(3,:,:));
        mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:)) ...
            *squeeze(nor_precoder_mat(1,:,:));
        mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
            *squeeze(nor_precoder_mat(2,:,:));
        mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
            *squeeze(nor_precoder_mat(3,:,:));
        mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
            *squeeze(nor_precoder_mat(1,:,:));
        mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
            *squeeze(nor_precoder_mat(2,:,:));
        mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
            *squeeze(nor_precoder_mat(3,:,:));
        % sample correlation matrices
        cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+mix_mat31* ...
            b3_dot+noise1_dot) ...
                    *ctranspose(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
                        mix_mat31*b3_dot+noise1_dot)/train_leng;
        cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+mix_mat32* ...
            b3_dot+noise2_dot) ...
                    *ctranspose(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
                        mix_mat32*b3_dot+noise2_dot)/train_leng;
        cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+mix_mat33* ...
            b3_dot+noise3_dot) ...
                    *ctranspose(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
                        mix_mat33*b3_dot+noise3_dot)/train_leng;
        % sample cross-correlation matrices
        cro_cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
```

```
    mix_mat31*b3_dot+noise1_dot) ...
                *ctranspose(b1_dot)/train_leng;
cro_cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
    mix_mat32*b3_dot+noise2_dot)  ...
                *ctranspose(b2_dot)/train_leng;
cro_cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
    mix_mat33*b3_dot+noise3_dot)  ...
                *ctranspose(b3_dot)/train_leng;
% LS estimated receive filters
rece_fil1=cor_mat1\cro_cor_mat1;
rece_fil2=cor_mat2\cro_cor_mat2;
rece_fil3=cor_mat3\cro_cor_mat3;
% power normalization
rece_fil1_nor=mat_pow_nor(rece_fil1);
rece_fil2_nor=mat_pow_nor(rece_fil2);
rece_fil3_nor=mat_pow_nor(rece_fil3);

% for bi-directional training with each number of ...
    iteration
for n_d=1:n
    % backward training
    b1_dot=reshape(b(1,2*n_d,:,:),[num_stream,train_leng]);
    b2_dot=reshape(b(2,2*n_d,:,:),[num_stream,train_leng]);
    b3_dot=reshape(b(3,2*n_d,:,:),[num_stream,train_leng]);
    noise1_dot=reshape(noise(1,2*n_d,:,:),[Nr,train_leng]);
    noise2_dot=reshape(noise(2,2*n_d,:,:),[Nr,train_leng]);
    noise3_dot=reshape(noise(3,2*n_d,:,:),[Nr,train_leng]);

    % backward mixing matrices
    % precoder updating, precoder <-- receive filter
    mix_mat11=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(1,1,:,:)))*conj(rece_fil1_nor);
    mix_mat12=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(2,1,:,:)))*conj(rece_fil2_nor);
    mix_mat13=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(3,1,:,:)))*conj(rece_fil3_nor);
    mix_mat21=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(1,2,:,:)))*conj(rece_fil1_nor);
    mix_mat22=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(2,2,:,:)))*conj(rece_fil2_nor);
    mix_mat23=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(3,2,:,:)))*conj(rece_fil3_nor);
    mix_mat31=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(1,3,:,:)))*conj(rece_fil1_nor);
    mix_mat32=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(2,3,:,:)))*conj(rece_fil2_nor);
    mix_mat33=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(3,3,:,:)))*conj(rece_fil3_nor);
    % sample correlation matrices
```

```
        cor_mat1=(mix_mat11*b1_dot+mix_mat12*b2_dot+ ...
            mix_mat13*b3_dot+noise1_dot) ...
                *ctranspose(mix_mat11*b1_dot+ ...
                    mix_mat12*b2_dot+mix_mat13*b3_dot+noise1_dot) ...
                    /train_leng;
        cor_mat2=(mix_mat21*b1_dot+mix_mat22*b2_dot+ ...
            mix_mat23*b3_dot+noise2_dot) ...
                *ctranspose(mix_mat21*b1_dot+ ...
                    mix_mat22*b2_dot+mix_mat23*b3_dot+noise2_dot) ...
                    /train_leng;
        cor_mat3=(mix_mat31*b1_dot+mix_mat32*b2_dot+ ...
            mix_mat33*b3_dot+noise3_dot) ...
                *ctranspose(mix_mat31*b1_dot+mix_mat32* ...
                    b2_dot+mix_mat33*b3_dot+noise3_dot) ...
                    /train_leng;
        % sample cross-correlation matrices
        cro_cor_mat1=(mix_mat11*b1_dot+mix_mat12*b2_dot+ ...
            mix_mat13*b3_dot+noise1_dot) ...
                     *ctranspose(b1_dot)/train_leng;
        cro_cor_mat2=(mix_mat21*b1_dot+mix_mat22*b2_dot+ ...
            mix_mat23*b3_dot+noise2_dot) ...
                     *ctranspose(b2_dot)/train_leng;
        cro_cor_mat3=(mix_mat31*b1_dot+mix_mat32*b2_dot+ ...
            mix_mat33*b3_dot+noise3_dot) ...
                     *ctranspose(b3_dot)/train_leng;
        % LS estimated precoders and power normalization
        precoder1=mat_pow_nor(conj(cor_mat1\cro_cor_mat1));
        precoder2=mat_pow_nor(conj(cor_mat2\cro_cor_mat2));
        precoder3=mat_pow_nor(conj(cor_mat3\cro_cor_mat3));

        % forward training
        b1_dot=reshape(b(1,2*n_d+1,:,:),[num_stream,train_leng]);
        b2_dot=reshape(b(2,2*n_d+1,:,:),[num_stream,train_leng]);
        b3_dot=reshape(b(3,2*n_d+1,:,:),[num_stream,train_leng]);
        noise1_dot=reshape(noise(1,2*n_d+1,:,:),[Nr,train_leng]);
        noise2_dot=reshape(noise(2,2*n_d+1,:,:),[Nr,train_leng]);
        noise3_dot=reshape(noise(3,2*n_d+1,:,:),[Nr,train_leng]);

        % forward mixing matrices
        % receive filter updating, precoder --> receive ...
            filter
        mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:)) ...
            *precoder1;
        mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:)) ...
            *precoder2;
        mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:)) ...
            *precoder3;
        mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:)) ...
            *precoder1;
```

```
            mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
                *precoder2;
            mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
                *precoder3;
            mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
                *precoder1;
            mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
                *precoder2;
            mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
                *precoder3;
            % sample correlation matrices
            cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+mix_mat31* ...
                b3_dot+noise1_dot) ...
                      *ctranspose(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
                          mix_mat31*b3_dot+noise1_dot)/train_leng;
            cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+mix_mat32* ...
                b3_dot+noise2_dot) ...
                      *ctranspose(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
                          mix_mat32*b3_dot+noise2_dot)/train_leng;
            cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+mix_mat33* ...
                b3_dot+noise3_dot) ...
                      *ctranspose(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
                          mix_mat33*b3_dot+noise3_dot)/train_leng;
            % sample cross-correlation matrices
            cro_cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
                mix_mat31*b3_dot+noise1_dot) ...
                          *ctranspose(b1_dot)/train_leng;
            cro_cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
                mix_mat32*b3_dot+noise2_dot)  ...
                          *ctranspose(b2_dot)/train_leng;
            cro_cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
                mix_mat33*b3_dot+noise3_dot)  ...
                          *ctranspose(b3_dot)/train_leng;
            % LS estimated receive filters
            rece_fil1=cor_mat1\cro_cor_mat1;
            rece_fil2=cor_mat2\cro_cor_mat2;
            rece_fil3=cor_mat3\cro_cor_mat3;
            % power normalization
            rece_fil1_nor=mat_pow_nor(rece_fil1);
            rece_fil2_nor=mat_pow_nor(rece_fil2);
            rece_fil3_nor=mat_pow_nor(rece_fil3);
        end

        % for each stream
        for n_d=1:num_stream

            % signal power
            signal_pow1=abs(ctranspose(rece_fil1(:,n_d))* ...
                mix_mat11(:,n_d))^2;
```

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

            % interference power
            interf_pow1=0;
            interf_pow2=0;
            interf_pow3=0;
            % interference from its own transmitter
            for n_dd=1:num_stream
                if n_dd==n_d
                    % pass
                else
                    interf_pow1=interf_pow1+abs(ctranspose ...
                        (rece_fil1(:,n_d))*mix_mat11(:,n_dd))^2;
                    interf_pow2=interf_pow2+abs(ctranspose ...
                        (rece_fil2(:,n_d))*mix_mat22(:,n_dd))^2;
                    interf_pow3=interf_pow3+abs(ctranspose ...
                        (rece_fil3(:,n_d))*mix_mat33(:,n_dd))^2;
                end
            end

            % interference from other transmitters
            for n_dd=1:num_stream
                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;
                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;
                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;
            end

            % noise power
            noise_pow1=abs(ctranspose(rece_fil1(:,n_d))* ...
                noise_covar*rece_fil1(:,n_d));
            noise_pow2=abs(ctranspose(rece_fil2(:,n_d))* ...
                noise_covar*rece_fil2(:,n_d));
            noise_pow3=abs(ctranspose(rece_fil3(:,n_d))* ...
                noise_covar*rece_fil3(:,n_d));

            % sum rate computation
            sumrate1(1,n_d)=log2(1+signal_pow1/(interf_pow1+ ...
```

```
                noise_pow1));
            sumrate2(1,n_d)=log2(1+signal_pow2/(interf_pow2+ ...
                noise_pow2));
            sumrate3(1,n_d)=log2(1+signal_pow3/(interf_pow3+ ...
                noise_pow3));
        end

        % network sum rate
        sumrate(m,n/num_ite_base)=sum(sumrate1)+sum(sumrate2)+ ...
            sum(sumrate3);
    end
end

% data averaging and extracting
sumrate_ave=mean(sumrate);
sumrate_sam=sumrate(1,:);

% number of iteration axis
num_ite_leng=zeros(1,num_ite_ran/num_ite_base);
for m=1:num_ite_ran/num_ite_base
    num_ite_leng(1,m)=m*num_ite_base;
end

end




ls_sumrate_vs_numite_bidirec_in_rr.m:

% LS estimated Sum rate vs Number of bi-directional training ...
    iteration with reduced-rank
% in interference network

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)

% parameter extraction
N=size(Gaussian_chan_mat,1);
Nr=size(squeeze(Gaussian_chan_mat(1,1,:,:)),1);
num_stream=size(squeeze(nor_precoder_mat(1,:,:)),2);
% temporary container
sumrate=zeros(size(red_rank,2),ave_num,num_ite_ran/num_ite_base);
% output data
sumrate_ave=zeros(size(red_rank,2),num_ite_ran/num_ite_base);
sumrate_sam=zeros(size(red_rank,2),num_ite_ran/num_ite_base);

```

```
% for every reduced-rank
for rr_ind=1:size(red_rank,2)
    % assign current reduced-rank
    rr=red_rank(rr_ind);

    % for each trial in averaging
    for m=1:ave_num
        % training sequence
        b=in_bidirec_trainseq_gen(N,2*num_ite_ran+1,num_stream, ...
            train_leng);
        % noise sequence
        noise=in_bidirec_noiseseq_gen(N,2*num_ite_ran+1,Nr, ...
            train_leng);

        % for each number of bi-directional training iteration
        for n=num_ite_base:num_ite_base:num_ite_ran

            % initialized forward training
            % receive filter updating, precoder --> receive ...
                filter
            b1_dot=reshape(b(1,1,:,:),[num_stream,train_leng]);
            b2_dot=reshape(b(2,1,:,:),[num_stream,train_leng]);
            b3_dot=reshape(b(3,1,:,:),[num_stream,train_leng]);
            noise1_dot=reshape(noise(1,1,:,:),[Nr,train_leng]);
            noise2_dot=reshape(noise(2,1,:,:),[Nr,train_leng]);
            noise3_dot=reshape(noise(3,1,:,:),[Nr,train_leng]);

            % initialized mixing matrices
            mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:)) ...
                *squeeze(nor_precoder_mat(1,:,:));
            mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:)) ...
                *squeeze(nor_precoder_mat(2,:,:));
            mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:)) ...
                *squeeze(nor_precoder_mat(3,:,:));
            mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:)) ...
                *squeeze(nor_precoder_mat(1,:,:));
            mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
                *squeeze(nor_precoder_mat(2,:,:));
            mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
                *squeeze(nor_precoder_mat(3,:,:));
            mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
                *squeeze(nor_precoder_mat(1,:,:));
            mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
                *squeeze(nor_precoder_mat(2,:,:));
            mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
                *squeeze(nor_precoder_mat(3,:,:));
            % sample correlation matrices
            cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
                mix_mat31*b3_dot+noise1_dot) ...
```

```
                *ctranspose(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
                    mix_mat31*b3_dot+noise1_dot)/train_leng;
        cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+mix_mat32* ...
            b3_dot+noise2_dot) ...
                *ctranspose(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
                    mix_mat32*b3_dot+noise2_dot)/train_leng;
        cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+mix_mat33* ...
            b3_dot+noise3_dot) ...
                *ctranspose(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
                    mix_mat33*b3_dot+noise3_dot)/train_leng;
        % sample cross-correlation matrices
        cro_cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
            mix_mat31*b3_dot+noise1_dot) ...
                 *ctranspose(b1_dot)/train_leng;
        cro_cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
            mix_mat32*b3_dot+noise2_dot)  ...
                 *ctranspose(b2_dot)/train_leng;
        cro_cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
            mix_mat33*b3_dot+noise3_dot)  ...
                 *ctranspose(b3_dot)/train_leng;
        % LS estimated receive filter and power ...
            normalization
        [rece_fil1,rece_fil2,rece_fil3,¬]= ...
            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,false);
        rece_fil1=mat_pow_nor(rece_fil1);
        rece_fil2=mat_pow_nor(rece_fil2);
        rece_fil3=mat_pow_nor(rece_fil3);

        % for bi-directional training with each number ...
            of iteration
        for n_d=1:n

            % backward training
            % precoder updating: precoder <-- receive filter
            b1_dot=reshape(b(1,2*n_d,:,:), ...
                [num_stream,train_leng]);
            b2_dot=reshape(b(2,2*n_d,:,:), ...
                [num_stream,train_leng]);
            b3_dot=reshape(b(3,2*n_d,:,:), ...
                [num_stream,train_leng]);
            noise1_dot=reshape(noise(1,2*n_d,:,:), ...
                [Nr,train_leng]);
            noise2_dot=reshape(noise(2,2*n_d,:,:), ...
```

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

    % backward mixing matrices
    mix_mat11=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(1,1,:,:)))*conj(rece_fil1);
    mix_mat12=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(2,1,:,:)))*conj(rece_fil2);
    mix_mat13=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(3,1,:,:)))*conj(rece_fil3);
    mix_mat21=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(1,2,:,:)))*conj(rece_fil1);
    mix_mat22=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(2,2,:,:)))*conj(rece_fil2);
    mix_mat23=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(3,2,:,:)))*conj(rece_fil3);
    mix_mat31=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(1,3,:,:)))*conj(rece_fil1);
    mix_mat32=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(2,3,:,:)))*conj(rece_fil2);
    mix_mat33=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(3,3,:,:)))*conj(rece_fil3);
    % sample correlation matrices
    cor_mat1=(mix_mat11*b1_dot+mix_mat12*b2_dot+ ...
        mix_mat13*b3_dot+noise1_dot) ...
             *ctranspose(mix_mat11*b1_dot+ ...
                mix_mat12*b2_dot+mix_mat13*b3_dot+ ...
                noise1_dot)/train_leng;
    cor_mat2=(mix_mat21*b1_dot+mix_mat22*b2_dot+ ...
        mix_mat23*b3_dot+noise2_dot) ...
             *ctranspose(mix_mat21*b1_dot+ ...
                mix_mat22*b2_dot+mix_mat23*b3_dot+ ...
                noise2_dot)/train_leng;
    cor_mat3=(mix_mat31*b1_dot+mix_mat32*b2_dot ...
        +mix_mat33*b3_dot+noise3_dot) ...
             *ctranspose(mix_mat31*b1_dot+ ...
                mix_mat32*b2_dot+mix_mat33*b3_dot+ ...
                noise3_dot)/train_leng;
    % sample cross-correlation matrices
    cro_cor_mat1=(mix_mat11*b1_dot+mix_mat12* ...
        b2_dot+mix_mat13*b3_dot+noise1_dot) ...
                 *ctranspose(b1_dot)/train_leng;
    cro_cor_mat2=(mix_mat21*b1_dot+mix_mat22* ...
        b2_dot+mix_mat23*b3_dot+noise2_dot) ...
                 *ctranspose(b2_dot)/train_leng;
    cro_cor_mat3=(mix_mat31*b1_dot+mix_mat32* ...
        b2_dot+mix_mat33*b3_dot+noise3_dot) ...
                 *ctranspose(b3_dot)/train_leng;
```

```
% LS estimated precoders
[precoder1,precoder2,precoder3,¬]= ...
    ls_rr_fil_sumrate_in_cal ...
      (mix_mat11,mix_mat12,mix_mat13,cor_mat1, ...
          cro_cor_mat1, ...
       mix_mat22,mix_mat21,mix_mat23,cor_mat2, ...
           cro_cor_mat2, ...
       mix_mat33,mix_mat31,mix_mat32,cor_mat3, ...
           cro_cor_mat3,noise_covar,rr,false);
precoder1=mat_pow_nor(conj(precoder1));
precoder2=mat_pow_nor(conj(precoder2));
precoder3=mat_pow_nor(conj(precoder3));

% forward training
% receive filter: precoder --> receive filter
b1_dot=reshape(b(1,2*n_d+1,:,:),[num_stream, ...
    train_leng]);
b2_dot=reshape(b(2,2*n_d+1,:,:),[num_stream, ...
    train_leng]);
b3_dot=reshape(b(3,2*n_d+1,:,:),[num_stream, ...
    train_leng]);
noise1_dot=reshape(noise(1,2*n_d+1,:,:),[Nr, ...
    train_leng]);
noise2_dot=reshape(noise(2,2*n_d+1,:,:),[Nr, ...
    train_leng]);
noise3_dot=reshape(noise(3,2*n_d+1,:,:),[Nr, ...
    train_leng]);

% forward mixing matrices
mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (1,1,:,:))*precoder1;
mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (2,1,:,:))*precoder2;
mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (3,1,:,:))*precoder3;
mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (1,2,:,:))*precoder1;
mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (2,2,:,:))*precoder2;
mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (3,2,:,:))*precoder3;
mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (1,3,:,:))*precoder1;
mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (2,3,:,:))*precoder2;
mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (3,3,:,:))*precoder3;
% sample correlation matrices
cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
```

```
                    mix_mat31*b3_dot+noise1_dot) ...
                         *ctranspose(mix_mat11*b1_dot+mix_mat21* ...
                            b2_dot+mix_mat31*b3_dot+noise1_dot)/ ...
                            train_leng;
                cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
                    mix_mat32*b3_dot+noise2_dot) ...
                         *ctranspose(mix_mat12*b1_dot+ ...
                            mix_mat22*b2_dot+mix_mat32*b3_dot+ ...
                            noise2_dot)/train_leng;
                cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
                    mix_mat33*b3_dot+noise3_dot) ...
                         *ctranspose(mix_mat13*b1_dot+ ...
                            mix_mat23*b2_dot+mix_mat33*b3_dot+ ...
                            noise3_dot)/train_leng;
                % sample cross-correlation matrices
                cro_cor_mat1=(mix_mat11*b1_dot+mix_mat21* ...
                    b2_dot+mix_mat31*b3_dot+noise1_dot) ...
                             *ctranspose(b1_dot)/train_leng;
                cro_cor_mat2=(mix_mat12*b1_dot+mix_mat22* ...
                    b2_dot+mix_mat32*b3_dot+noise2_dot)  ...
                             *ctranspose(b2_dot)/train_leng;
                cro_cor_mat3=(mix_mat13*b1_dot+mix_mat23* ...
                    2_dot+mix_mat33*b3_dot+noise3_dot)  ...
                             *ctranspose(b3_dot)/train_leng;
                % LS estimated receive filters
                [rece_fil1,rece_fil2,rece_fil3,sumrate_val]= ...
                    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,true);
                % power normalization
                rece_fil1=mat_pow_nor(rece_fil1);
                rece_fil2=mat_pow_nor(rece_fil2);
                rece_fil3=mat_pow_nor(rece_fil3);
            end
            sumrate(rr_ind,m,n/num_ite_base)=sumrate_val;
        end
    end
end

% averaging and extracting
for m=1:size(red_rank,2)
    sumrate_ave(m,:)=mean(squeeze(sumrate(m,:,:)));
    sumrate_sam(m,:)=squeeze(sumrate(m,1,:));
end
```

```
% number of iteration axis
num_ite_leng=zeros(1,num_ite_ran/num_ite_base);
for m=1:num_ite_ran/num_ite_base
    num_ite_leng(1,m)=m*num_ite_base;
end

end



ls_sumrate_vs_trainleng_bidirec_in_fr.m:

% LS estimated Sum rate vs Training length with full-rank
% in interference network

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)

% parameter extraction
N=size(Gaussian_chan_mat,1);
Nr=size(squeeze(Gaussian_chan_mat(1,1,:,:)),1);
num_stream=size(squeeze(nor_precoder_mat(1,:,:)),2);

% temporary container
sumrate=zeros(ave_num,train_leng_ran);

% for each trial in averaging
for m=1:ave_num
    % training sequence for each iteration
    b=in_bidirec_trainseq_gen(N,2*num_ite+1,num_stream, ...
        train_base*train_leng_ran);
    % noise sequence for each iteration
    noise=in_bidirec_noiseseq_gen(N,2*num_ite+1,Nr, ...
        train_base*train_leng_ran);

    % for each training length
    for n=train_base:train_base:train_base*train_leng_ran
        % initialized forward training
        % receive filter updating, precoder --> receive filter
        b1_dot=reshape(b(1,1,:,[1:n]),[num_stream,n]);
        b2_dot=reshape(b(2,1,:,[1:n]),[num_stream,n]);
        b3_dot=reshape(b(3,1,:,[1:n]),[num_stream,n]);
        noise1_dot=reshape(noise(1,1,:,[1:n]),[Nr,n]);
        noise2_dot=reshape(noise(2,1,:,[1:n]),[Nr,n]);
        noise3_dot=reshape(noise(3,1,:,[1:n]),[Nr,n]);
```

```

        % initialized mixing matrices
        mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:)) ...
            *squeeze(nor_precoder_mat(1,:,:));
        mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:)) ...
            *squeeze(nor_precoder_mat(2,:,:));
        mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:)) ...
            *squeeze(nor_precoder_mat(3,:,:));
        mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:)) ...
            *squeeze(nor_precoder_mat(1,:,:));
        mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
            *squeeze(nor_precoder_mat(2,:,:));
        mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
            *squeeze(nor_precoder_mat(3,:,:));
        mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
            *squeeze(nor_precoder_mat(1,:,:));
        mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
            *squeeze(nor_precoder_mat(2,:,:));
        mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
            *squeeze(nor_precoder_mat(3,:,:));

        % sample correlation matrices
        cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
            mix_mat31*b3_dot+noise1_dot) ...
                  *ctranspose(mix_mat11*b1_dot+ ...
                      mix_mat21*b2_dot+mix_mat31*b3_dot+noise1_dot)/n;
        cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
            mix_mat32*b3_dot+noise2_dot) ...
                  *ctranspose(mix_mat12*b1_dot+ ...
                      mix_mat22*b2_dot+mix_mat32*b3_dot+noise2_dot)/n;
        cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
            mix_mat33*b3_dot+noise3_dot) ...
                  *ctranspose(mix_mat13*b1_dot+ ...
                      mix_mat23*b2_dot+mix_mat33*b3_dot+noise3_dot)/n;
        % sample cross-correlation matrices
        cro_cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
            mix_mat31*b3_dot+noise1_dot) ...
                      *ctranspose(b1_dot)/n;
        cro_cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
            mix_mat32*b3_dot+noise2_dot)  ...
                      *ctranspose(b2_dot)/n;
        cro_cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
            mix_mat33*b3_dot+noise3_dot)  ...
                      *ctranspose(b3_dot)/n;
        % LS estimated receive filters
        rece_fil1=cor_mat1\cro_cor_mat1;
        rece_fil2=cor_mat2\cro_cor_mat2;
        rece_fil3=cor_mat3\cro_cor_mat3;
        % power normalization
```

```
        rece_fil1_nor=mat_pow_nor(rece_fil1);
        rece_fil2_nor=mat_pow_nor(rece_fil2);
        rece_fil3_nor=mat_pow_nor(rece_fil3);

        % for each number of bi-directional training
        for n_d=1:num_ite
            % backward training
            % precoder updating, precoder <-- receive filter
            b1_dot=reshape(b(1,2*n_d,:,[1:n]),[num_stream,n]);
            b2_dot=reshape(b(2,2*n_d,:,[1:n]),[num_stream,n]);
            b3_dot=reshape(b(3,2*n_d,:,[1:n]),[num_stream,n]);
            noise1_dot=reshape(noise(1,2*n_d,:,[1:n]),[Nr,n]);
            noise2_dot=reshape(noise(2,2*n_d,:,[1:n]),[Nr,n]);
            noise3_dot=reshape(noise(3,2*n_d,:,[1:n]),[Nr,n]);

            % backward mixing matrices
            mix_mat11=sqrt(pow)*transpose(squeeze ...
                (Gaussian_chan_mat(1,1,:,:)))*conj(rece_fil1_nor);
            mix_mat12=sqrt(pow)*transpose(squeeze ...
                (Gaussian_chan_mat(2,1,:,:)))*conj(rece_fil2_nor);
            mix_mat13=sqrt(pow)*transpose(squeeze ...
                (Gaussian_chan_mat(3,1,:,:)))*conj(rece_fil3_nor);
            mix_mat21=sqrt(pow)*transpose(squeeze ...
                (Gaussian_chan_mat(1,2,:,:)))*conj(rece_fil1_nor);
            mix_mat22=sqrt(pow)*transpose(squeeze ...
                (Gaussian_chan_mat(2,2,:,:)))*conj(rece_fil2_nor);
            mix_mat23=sqrt(pow)*transpose(squeeze ...
                (Gaussian_chan_mat(3,2,:,:)))*conj(rece_fil3_nor);
            mix_mat31=sqrt(pow)*transpose(squeeze ...
                (Gaussian_chan_mat(1,3,:,:)))*conj(rece_fil1_nor);
            mix_mat32=sqrt(pow)*transpose(squeeze ...
                (Gaussian_chan_mat(2,3,:,:)))*conj(rece_fil2_nor);
            mix_mat33=sqrt(pow)*transpose(squeeze ...
                (Gaussian_chan_mat(3,3,:,:)))*conj(rece_fil3_nor);
            % sample correlation matrices
            cor_mat1=(mix_mat11*b1_dot+mix_mat12*b2_dot ...
               +mix_mat13*b3_dot+noise1_dot) ...
                     *ctranspose(mix_mat11*b1_dot+ ...
                         mix_mat12*b2_dot+mix_mat13*b3_dot+ ...
                         noise1_dot)/n;
            cor_mat2=(mix_mat21*b1_dot+mix_mat22*b2_dot+ ...
               mix_mat23*b3_dot+noise2_dot) ...
                     *ctranspose(mix_mat21*b1_dot+ ...
                         mix_mat22*b2_dot+mix_mat23*b3_dot+ ...
                         noise2_dot)/n;
            cor_mat3=(mix_mat31*b1_dot+mix_mat32*b2_dot+ ...
               mix_mat33*b3_dot+noise3_dot) ...
                     *ctranspose(mix_mat31*b1_dot+ ...
                         mix_mat32*b2_dot+mix_mat33*b3_dot+ ...
```

```
                    noise3_dot)/n;
% sample cross-correlation matrices
cro_cor_mat1=(mix_mat11*b1_dot+mix_mat12*b2_dot ...
    +mix_mat13*b3_dot+noise1_dot) ...
            *ctranspose(b1_dot)/n;
cro_cor_mat2=(mix_mat21*b1_dot+mix_mat22*b2_dot ...
    +mix_mat23*b3_dot+noise2_dot) ...
            *ctranspose(b2_dot)/n;
cro_cor_mat3=(mix_mat31*b1_dot+mix_mat32*b2_dot ...
    +mix_mat33*b3_dot+noise3_dot) ...
            *ctranspose(b3_dot)/n;
% LS estimated precoders and power normalization
precoder1=mat_pow_nor(conj(cor_mat1\cro_cor_mat1));
precoder2=mat_pow_nor(conj(cor_mat2\cro_cor_mat2));
precoder3=mat_pow_nor(conj(cor_mat3\cro_cor_mat3));

% forward training
% receive filter updating, precoder --> receive ...
    filter
b1_dot=reshape(b(1,2*n_d+1,:,[1:n]),[num_stream,n]);
b2_dot=reshape(b(2,2*n_d+1,:,[1:n]),[num_stream,n]);
b3_dot=reshape(b(3,2*n_d+1,:,[1:n]),[num_stream,n]);
noise1_dot=reshape(noise(1,2*n_d+1,:,[1:n]),[Nr,n]);
noise2_dot=reshape(noise(2,2*n_d+1,:,[1:n]),[Nr,n]);
noise3_dot=reshape(noise(3,2*n_d+1,:,[1:n]),[Nr,n]);

% forward mixing matrices
mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (1,1,:,:))*precoder1;
mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (2,1,:,:))*precoder2;
mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (3,1,:,:))*precoder3;
mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (1,2,:,:))*precoder1;
mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (2,2,:,:))*precoder2;
mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (3,2,:,:))*precoder3;
mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (1,3,:,:))*precoder1;
mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (2,3,:,:))*precoder2;
mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (3,3,:,:))*precoder3;
% sample correlation matrices
cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
    mix_mat31*b3_dot+noise1_dot) ...
        *ctranspose(mix_mat11*b1_dot+ ...
```

```
                    mix_mat21*b2_dot+mix_mat31*b3_dot ...
                    +noise1_dot)/n;
        cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
            mix_mat32*b3_dot+noise2_dot) ...
                *ctranspose(mix_mat12*b1_dot+ ...
                    mix_mat22*b2_dot+mix_mat32*b3_dot+ ...
                    noise2_dot)/n;
        cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
            mix_mat33*b3_dot+noise3_dot) ...
                *ctranspose(mix_mat13*b1_dot+ ...
                    mix_mat23*b2_dot+mix_mat33*b3_dot+ ...
                    noise3_dot)/n;
        % sample cross-correlation matrices
        cro_cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
            mix_mat31*b3_dot+noise1_dot) ...
                    *ctranspose(b1_dot)/n;
        cro_cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
            mix_mat32*b3_dot+noise2_dot)  ...
                    *ctranspose(b2_dot)/n;
        cro_cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot ...
            +mix_mat33*b3_dot+noise3_dot)  ...
                    *ctranspose(b3_dot)/n;
        % LS estimated receive filters
        rece_fil1=cor_mat1\cro_cor_mat1;
        rece_fil2=cor_mat2\cro_cor_mat2;
        rece_fil3=cor_mat3\cro_cor_mat3;
        % power normalization
        rece_fil1_nor=mat_pow_nor(rece_fil1);
        rece_fil2_nor=mat_pow_nor(rece_fil2);
        rece_fil3_nor=mat_pow_nor(rece_fil3);
    end

    % data preallocation
    sumrate1=zeros(1,num_stream);
    sumrate2=zeros(1,num_stream);
    sumrate3=zeros(1,num_stream);

    for n_d=1:num_stream

        % signal power
        signal_pow1=abs(ctranspose(rece_fil1(:,n_d))* ...
            mix_mat11(:,n_d))^2;
        signal_pow2=abs(ctranspose(rece_fil2(:,n_d))* ...
            mix_mat22(:,n_d))^2;
        signal_pow3=abs(ctranspose(rece_fil3(:,n_d))* ...
            mix_mat33(:,n_d))^2;

        % interference power
        interf_pow1=0;
```

```
            interf_pow2=0;
            interf_pow3=0;
            % interference from its transmitter
            for n_dd=1:num_stream
                if n_dd==n_d
                    % pass
                else
                    interf_pow1=interf_pow1+abs(ctranspose ...
                        (rece_fil1(:,n_d))*mix_mat11(:,n_dd))^2;
                    interf_pow2=interf_pow2+abs(ctranspose ...
                        (rece_fil2(:,n_d))*mix_mat22(:,n_dd))^2;
                    interf_pow3=interf_pow3+abs(ctranspose ...
                        (rece_fil3(:,n_d))*mix_mat33(:,n_dd))^2;
                end
            end

            % interference from other transmitters
            for n_dd=1:num_stream
                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;
                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;
                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;
            end

            % noise power
            noise_pow1=abs(ctranspose(rece_fil1(:,n_d))* ...
                noise_covar*rece_fil1(:,n_d));
            noise_pow2=abs(ctranspose(rece_fil2(:,n_d))* ...
                noise_covar*rece_fil2(:,n_d));
            noise_pow3=abs(ctranspose(rece_fil3(:,n_d))* ...
                noise_covar*rece_fil3(:,n_d));

            % sum rate computation
            sumrate1(1,n_d)=log2(1+signal_pow1/(interf_pow1+ ...
                noise_pow1));
            sumrate2(1,n_d)=log2(1+signal_pow2/(interf_pow2+ ...
                noise_pow2));
            sumrate3(1,n_d)=log2(1+signal_pow3/(interf_pow3+ ...
                noise_pow3));
        end

```

```
        % network sum rate
        sumrate(m,n/train_base)=sum(sumrate1)+sum(sumrate2)+ ...
            sum(sumrate3);
    end
end

% data averaging, sample data extraction
sumrate_ave=mean(sumrate);
sumrate_sam=sumrate(1,:);

% training length axis
train_leng=zeros(1,train_leng_ran);
for m=1:train_leng_ran
    train_leng(1,m)=m*train_base;
end

end




ls_sumrate_vs_trainleng_bidirec_in_rr.m:

% LS estimated Sum rate vs Training length with reduced-rank in
% interference network

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)

% parameter extraction
N=size(Gaussian_chan_mat,1);
Nr=size(squeeze(Gaussian_chan_mat(1,1,:,:)),1);
num_stream=size(squeeze(nor_precoder_mat(1,:,:)),2);
% temporary container
sumrate=zeros(size(red_rank,2),ave_num,train_leng_ran);
% output data
sumrate_ave=zeros(size(red_rank,2),train_leng_ran);
sumrate_sam=zeros(size(red_rank,2),train_leng_ran);

% for each reduced-rank
for rr_ind=1:size(red_rank,2)
    % assign reduced-rank
    rr=red_rank(rr_ind);

    % for each trial in averaging
    for m=1:ave_num

```

```
        % training sequence for each iteration
        b=in_bidirec_trainseq_gen(N,2*num_ite+1,num_stream, ...
            train_base*train_leng_ran);
        % noise sequence for each iteration
        noise=in_bidirec_noiseseq_gen(N,2*num_ite+1,Nr, ...
            train_base*train_leng_ran);

        % for each training length
        for n=train_base:train_base:train_base*train_leng_ran

            % initialized forward training
            % receive filter updating, precoder --> receive ...
                filter
            b1_dot=reshape(b(1,1,:,[1:n]),[num_stream,n]);
            b2_dot=reshape(b(2,1,:,[1:n]),[num_stream,n]);
            b3_dot=reshape(b(3,1,:,[1:n]),[num_stream,n]);
            noise1_dot=reshape(noise(1,1,:,[1:n]),[Nr,n]);
            noise2_dot=reshape(noise(2,1,:,[1:n]),[Nr,n]);
            noise3_dot=reshape(noise(3,1,:,[1:n]),[Nr,n]);

            % initialized mixing matrices
            mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat ...
                (1,1,:,:))*squeeze(nor_precoder_mat(1,:,:));
            mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat ...
                (2,1,:,:))*squeeze(nor_precoder_mat(2,:,:));
            mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat ...
                (3,1,:,:))*squeeze(nor_precoder_mat(3,:,:));
            mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat ...
                (1,2,:,:))*squeeze(nor_precoder_mat(1,:,:));
            mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat ...
                (2,2,:,:))*squeeze(nor_precoder_mat(2,:,:));
            mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat ...
                (3,2,:,:))*squeeze(nor_precoder_mat(3,:,:));
            mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat ...
                (1,3,:,:))*squeeze(nor_precoder_mat(1,:,:));
            mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat ...
                (2,3,:,:))*squeeze(nor_precoder_mat(2,:,:));
            mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat ...
                (3,3,:,:))*squeeze(nor_precoder_mat(3,:,:));

            % sample correlation matrices
            cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
                mix_mat31*b3_dot+noise1_dot) ...
                    *ctranspose(mix_mat11*b1_dot+mix_mat21* ...
                        b2_dot+mix_mat31*b3_dot+noise1_dot)/n;
            cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
                mix_mat32*b3_dot+noise2_dot) ...
                    *ctranspose(mix_mat12*b1_dot+mix_mat22* ...
                        b2_dot+mix_mat32*b3_dot+noise2_dot)/n;
```

```
cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
    mix_mat33*b3_dot+noise3_dot) ...
        *ctranspose(mix_mat13*b1_dot+mix_mat23*b2_dot ...
            +mix_mat33*b3_dot+noise3_dot)/n;
% sample cross-correlation matrices
cro_cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
    mix_mat31*b3_dot+noise1_dot) ...
            *ctranspose(b1_dot)/n;
cro_cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
    mix_mat32*b3_dot+noise2_dot)  ...
            *ctranspose(b2_dot)/n;
cro_cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
    mix_mat33*b3_dot+noise3_dot)  ...
            *ctranspose(b3_dot)/n;
% reduced-rank LS estimated receive filters
[rece_fil1,rece_fil2,rece_fil3,¬]= ...
    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,false);
% power normalization
rece_fil1=mat_pow_nor(rece_fil1);
rece_fil2=mat_pow_nor(rece_fil2);
rece_fil3=mat_pow_nor(rece_fil3);

% for each number of bi-directional training
for n_d=1:num_ite

    % backward training
    % precoder updating: precoder <-- receive filter
    b1_dot=reshape(b(1,2*n_d,:,[1:n]),[num_stream,n]);
    b2_dot=reshape(b(2,2*n_d,:,[1:n]),[num_stream,n]);
    b3_dot=reshape(b(3,2*n_d,:,[1:n]),[num_stream,n]);
    noise1_dot=reshape(noise(1,2*n_d,:,[1:n]),[Nr,n]);
    noise2_dot=reshape(noise(2,2*n_d,:,[1:n]),[Nr,n]);
    noise3_dot=reshape(noise(3,2*n_d,:,[1:n]),[Nr,n]);

    % backward mixing matrices
    mix_mat11=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(1,1,:,:)))*conj(rece_fil1);
    mix_mat12=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(2,1,:,:)))*conj(rece_fil2);
    mix_mat13=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(3,1,:,:)))*conj(rece_fil3);
    mix_mat21=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(1,2,:,:)))*conj(rece_fil1);
```

```
            mix_mat22=sqrt(pow)*transpose(squeeze ...
                (Gaussian_chan_mat(2,2,:,:)))*conj(rece_fil2);
            mix_mat23=sqrt(pow)*transpose(squeeze ...
                (Gaussian_chan_mat(3,2,:,:)))*conj(rece_fil3);
            mix_mat31=sqrt(pow)*transpose(squeeze ...
                (Gaussian_chan_mat(1,3,:,:)))*conj(rece_fil1);
            mix_mat32=sqrt(pow)*transpose(squeeze ...
                (Gaussian_chan_mat(2,3,:,:)))*conj(rece_fil2);
            mix_mat33=sqrt(pow)*transpose(squeeze ...
                (Gaussian_chan_mat(3,3,:,:)))*conj(rece_fil3);
            % sample correlation matrices
            cor_mat1=(mix_mat11*b1_dot+mix_mat12*b2_dot ...
                +mix_mat13*b3_dot+noise1_dot) ...
                     *ctranspose(mix_mat11*b1_dot+ ...
                         mix_mat12*b2_dot+mix_mat13*b3_dot+ ...
                         noise1_dot)/n;
            cor_mat2=(mix_mat21*b1_dot+mix_mat22*b2_dot+ ...
                mix_mat23*b3_dot+noise2_dot) ...
                     *ctranspose(mix_mat21*b1_dot+ ...
                         mix_mat22*b2_dot+mix_mat23*b3_dot+ ...
                         noise2_dot)/n;
            cor_mat3=(mix_mat31*b1_dot+mix_mat32*b2_dot+ ...
                mix_mat33*b3_dot+noise3_dot) ...
                     *ctranspose(mix_mat31*b1_dot+ ...
                         mix_mat32*b2_dot+mix_mat33*b3_dot+ ...
                         noise3_dot)/n;
            % sample cross-correlation matrices
            cro_cor_mat1=(mix_mat11*b1_dot+mix_mat12* ...
                b2_dot+mix_mat13*b3_dot+noise1_dot) ...
                          *ctranspose(b1_dot)/n;
            cro_cor_mat2=(mix_mat21*b1_dot+mix_mat22* ...
                b2_dot+mix_mat23*b3_dot+noise2_dot) ...
                          *ctranspose(b2_dot)/n;
            cro_cor_mat3=(mix_mat31*b1_dot+mix_mat32* ...
                b2_dot+mix_mat33*b3_dot+noise3_dot) ...
                          *ctranspose(b3_dot)/n;
            % reduced-rank LS estimated precoders
            [precoder1,precoder2,precoder3,¬]= ...
                ls_rr_fil_sumrate_in_cal ...
                   (mix_mat11,mix_mat12,mix_mat13,cor_mat1, ...
                       cro_cor_mat1, ...
                    mix_mat22,mix_mat21,mix_mat23,cor_mat2, ...
                        cro_cor_mat2, ...
                    mix_mat33,mix_mat31,mix_mat32,cor_mat3, ...
                        cro_cor_mat3,noise_covar,rr,false);
            precoder1=mat_pow_nor(conj(precoder1));
            precoder2=mat_pow_nor(conj(precoder2));
            precoder3=mat_pow_nor(conj(precoder3));
```

```
% forward training
% receive filter: precoder --> receive filter
b1_dot=reshape(b(1,2*n_d+1,:,[1:n]),[num_stream,n]);
b2_dot=reshape(b(2,2*n_d+1,:,[1:n]),[num_stream,n]);
b3_dot=reshape(b(3,2*n_d+1,:,[1:n]),[num_stream,n]);
noise1_dot=reshape(noise(1,2*n_d+1,:,[1:n]),[Nr,n]);
noise2_dot=reshape(noise(2,2*n_d+1,:,[1:n]),[Nr,n]);
noise3_dot=reshape(noise(3,2*n_d+1,:,[1:n]),[Nr,n]);

% forward mixing matrices
mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (1,1,:,:))*precoder1;
mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (2,1,:,:))*precoder2;
mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (3,1,:,:))*precoder3;
mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (1,2,:,:))*precoder1;
mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (2,2,:,:))*precoder2;
mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (3,2,:,:))*precoder3;
mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (1,3,:,:))*precoder1;
mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (2,3,:,:))*precoder2;
mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (3,3,:,:))*precoder3;
% sample correlation matrices
cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
   mix_mat31*b3_dot+noise1_dot) ...
        *ctranspose(mix_mat11*b1_dot+mix_mat21* ...
           b2_dot+mix_mat31*b3_dot+noise1_dot)/n;
cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
   mix_mat32*b3_dot+noise2_dot) ...
        *ctranspose(mix_mat12*b1_dot+mix_mat22* ...
           b2_dot+mix_mat32*b3_dot+noise2_dot)/n;
cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
   mix_mat33*b3_dot+noise3_dot) ...
        *ctranspose(mix_mat13*b1_dot+mix_mat23* ...
           b2_dot+mix_mat33*b3_dot+noise3_dot)/n;
% sample cross-correlation matrices
cro_cor_mat1=(mix_mat11*b1_dot+mix_mat21*b2_dot+ ...
   mix_mat31*b3_dot+noise1_dot) ...
            *ctranspose(b1_dot)/n;
cro_cor_mat2=(mix_mat12*b1_dot+mix_mat22*b2_dot+ ...
   mix_mat32*b3_dot+noise2_dot)  ...
            *ctranspose(b2_dot)/n;
cro_cor_mat3=(mix_mat13*b1_dot+mix_mat23*b2_dot+ ...
```

```
                        mix_mat33*b3_dot+noise3_dot)  ...
                                 *ctranspose(b3_dot)/n;
                    % reduced-rank LS estimated receive filters
                    [rece_fil1,rece_fil2,rece_fil3,sumrate_val]= ...
                        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,true);
                    % power normalization
                    rece_fil1=mat_pow_nor(rece_fil1);
                    rece_fil2=mat_pow_nor(rece_fil2);
                    rece_fil3=mat_pow_nor(rece_fil3);
                end
                sumrate(rr_ind,m,n/train_base)=sumrate_val;
            end
        end
    end

    % averaging and extracting
    for m=1:size(red_rank,2)
        sumrate_ave(m,:)=mean(squeeze(sumrate(m,:,:)));
        sumrate_sam(m,:)=squeeze(sumrate(m,1,:));
    end

    % training length axis
    train_leng=zeros(1,train_leng_ran);
    for m=1:train_leng_ran
        train_leng(1,m)=m*train_base;
    end

    end




    mat_pow_nor.m:

    % Power normalization for each beamformer for each matrix

    function [mat] = mat_pow_nor(mat)

    % extract the number of streams
    num_stream=size(mat,2);

    % power normalization for each beamformer
```

```
for m=1:num_stream
    mat(:,m)=mat(:,m)/norm(mat(:,m),'fro');
end

end




noise_mat_gen.m:

% Gaussian additive noise generator

function noise_mat = noise_mat_gen(Nr,train_leng)

noise_mat=(1/sqrt(2))*(randn(Nr,train_leng)/10+ ...
    i*randn(Nr,train_leng)/10);

end




precoder_gen.m:

% Normalized precoder matrix

function precoder = precoder_gen(Nt,num_stream,pow)

% randomly generated precoder
precoder=rand(Nt,num_stream);

% power normalization for each beamformer
for m=1:num_stream
    precoder(:,m)=sqrt(pow)*precoder(:,m)/norm(precoder(:,m), ...
        'fro');
end

end




wiener_fr_mmse_sinr_cal.m:

% Wiener estimated full-rank MMSE and SINR calculator

function [mmse,sinr] = ...
    wiener_fr_mmse_sinr_cal(mix_mat,cor_mat,combiner)
```

```
% extract parameter
num_stream=size(mix_mat,2);

% temporary container
mmse_temp=zeros(1,num_stream);
sinr_temp=zeros(1,num_stream);

% calculate MMSE & SINR for each stream
for m=1:num_stream
    % MMSE
    mmse_temp(1,m)=1-abs(ctranspose(combiner(:,m))* ...
        cor_mat*combiner(:,m));
    % SINR
    sinr_temp(1,m)=abs(ctranspose(mix_mat(:,m))*inv ...
        (cor_mat-mix_mat(:,m)*ctranspose(mix_mat(:,m)))*mix_mat ...
        (:,m));
end

% average over all streams
mmse=mean(mmse_temp,2);
sinr=mean(sinr_temp,2);

end




wiener_mmse_sinr_fixnumite_bidirec_fr.m:

% Wiener (optimal) MMSE & SINR for fixed number of ...
    bi-directional
% optimization iteration with full-rank transmitter and receiver

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)

% parameter extraction
[Nr,num_stream]=size(Gaussian_chan*nor_precoder);
% output container
mmse=zeros(1,train_leng_ran);
sinr=zeros(1,train_leng_ran);

% initialized forward optimization
% initialized mixing matrix
mix_mat=sqrt(pow)*Gaussian_chan*nor_precoder;
% statistical correlation matrix
cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
```

```
% Wiener (optimal) receive filter
rece_fil=cor_mat\mix_mat;
% power normalization
rece_fil_nor=mat_pow_nor(rece_fil);

% for each bi-directional optimization
for m=1:num_ite
    % backward optimization
    mix_mat=transpose(sqrt(pow)*Gaussian_chan)* ...
        conj(rece_fil_nor);
    % statistical correlation matrix
    cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
    % Wiener (optimal) precoder
    precoder=conj(cor_mat\mix_mat);
    % power normalization
    precoder_nor=mat_pow_nor(precoder);

    % forward optimization
    mix_mat=sqrt(pow)*Gaussian_chan*precoder_nor;
    % statistical correlation matrix
    cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
    % Wiener (optimal) receive filter
    rece_fil=cor_mat\mix_mat;
    % power normalization
    rece_fil_nor=mat_pow_nor(rece_fil);
end

% data preallocation
mmse_temp=zeros(1,num_stream);
sinr_temp=zeros(1,num_stream);

% calculate MMSE & SINR
% for each stream
for n=1:num_stream
    % MMSE
    mmse_temp(1,n)=1-abs(ctranspose(rece_fil(:,n))*cor_mat* ...
        rece_fil(:,n));
    % SINR
    sinr_temp(1,n)=abs(ctranspose(mix_mat(:,n))*((cor_mat- ...
        mix_mat(:,n)*ctranspose(mix_mat(:,n)))\mix_mat(:,n)));
end

% averaging and decibel conversion
mmse_val=10*log10(mean(mmse_temp,2));
sinr_val=10*log10(mean(sinr_temp,2));

% training length axis
train_leng=zeros(1,train_leng_ran);
for m=1:train_leng_ran
```

```
    mmse(1,m)=mmse_val;
    sinr(1,m)=sinr_val;
    train_leng(1,m)=m*train_base;
end

end




wiener_mmse_sinr_fixednumite_bidirec_rr.m:

% Wiener (optimal) MMSE & SINR for fixed number of ...
    bi-directional optimization iteration with reduced-rank

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)

% parameter extraction
[Nr]=size(Gaussian_chan,1);
% temporary data preallocation
mmse_temp=zeros(1,size(red_rank,2));
sinr_temp=zeros(1,size(red_rank,2));
% data preallocation
mmse=zeros(size(red_rank,2),train_leng_ran);
sinr=zeros(size(red_rank,2),train_leng_ran);

% for each reduced-rank
for rr_ind=1:size(red_rank,2)
    % reduced-rank value
    rr=red_rank(rr_ind);

    % initialized forward optimization
    % initialized mixing matrix
    mix_mat=sqrt(pow)*Gaussian_chan*nor_precoder;
    % statistical correlation matrix
    cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
    % reduced-rank Wiener (optimal) receive filter
    [rece_fil,¬,¬]=wiener_rr_fil_mmse_sinr_cal(cor_mat, ...
        mix_mat,rr);
    % power normalization
    rece_fil_nor=mat_pow_nor(rece_fil);

    % for each bi-directional optimization
    for m=1:num_ite
        % backward optimization
        mix_mat=transpose(sqrt(pow)*Gaussian_chan)* ...
```

```
                conj(rece_fil_nor);
            % statistical correlation matrix
            cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
            % reduced-rank Wiener (optimal) precoder
            [precoder,¬,¬]=wiener_rr_fil_mmse_sinr_cal(cor_mat, ...
                mix_mat,rr);
            precoder=conj(precoder);
            % power normalization
            precoder_nor=mat_pow_nor(precoder);

            % forward optimization
            mix_mat=sqrt(pow)*Gaussian_chan*precoder_nor;
            % statistical correlation matrix
            cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
            % reduced-rank Wiener (optimal) receive filter
            [rece_fil,mmse_val,sinr_val]=wiener_rr_fil_mmse_sinr_cal ...
                (cor_mat,mix_mat,rr);
            % power normalization
            rece_fil_nor=mat_pow_nor(rece_fil);
        end

        % decibel conversion and assign data value to ...
            corresponding allocation
        mmse_temp(1,rr_ind)=10*log10(mmse_val);
        sinr_temp(1,rr_ind)=10*log10(sinr_val);
end

% training length axis
train_leng=zeros(1,train_leng_ran);
for m=1:train_leng_ran
    mmse(1,m)=mmse_temp(1,1);
    mmse(2,m)=mmse_temp(1,2);
    sinr(1,m)=sinr_temp(1,1);
    sinr(2,m)=sinr_temp(1,2);
    train_leng(1,m)=m*train_base;
end

end




wiener_mmse_sinr_fr.m:

% Wiener (optimal) estimated MMSE & SINR in full-rank

function [mmse,sinr] = wiener_mmse_sinr_fr(mix_mat,noise_covar)

% parameter extraction
```

```
[Nr,num_stream]=size(mix_mat);

% temporary container
mmse=zeros(1,num_stream);
sinr=zeros(1,num_stream);

% correlation matrix
cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
% full-rank receive filter
rece_fil=cor_mat\mix_mat;

for m=1:num_stream
    mmse(1,m)=1-abs(ctranspose(rece_fil(:,m))*cor_mat* ...
        rece_fil(:,m));
    sinr(1,m)=abs(ctranspose(mix_mat(:,m))*((cor_mat- ...
        mix_mat(:,m)*ctranspose(mix_mat(:,m)))\mix_mat(:,m)));
end

% averaging & decibel conversion
mmse=10*log10(mean(mmse,2));
sinr=10*log10(mean(sinr,2));

end
```

wiener_mmse_sinr_rr.m:

```
% Wiener (optimal) estimated MMSE & SINR in reduced-rank

function [mmse,sinr] = ...
    wiener_mmse_sinr_rr(mix_mat,red_rank,noise_covar)

% parameter extraction
[Nr]=size(mix_mat,1);

% temporary container
mmse=zeros(1,size(red_rank,2));
sinr=zeros(1,size(red_rank,2));

% correlation matrix
cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);

for rr_ind=1:size(red_rank,2)
    % reduced-rank
    rr=red_rank(1,rr_ind);
    % compute MMSE & SINR for each reduced-rank
    [¬,mmse(1,rr_ind),sinr(1,rr_ind)]= ...
```

```
        wiener_rr_fil_mmse_sinr_cal(cor_mat,mix_mat,rr);
end

% decibel conversion
mmse=10*log10(mmse);
sinr=10*log10(sinr);

end




wiener_mmse_sinr_vs_numite_bidirec_fr.m:

% Wiener (optimal) estimated (MMSE & SINR) vs Number of ...
    bi-directional optimization iteration with full-rank

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)

% parameter extraction
[Nr,num_stream]=size(Gaussian_chan*nor_precoder);

% output container
mmse=zeros(1,num_ite_ran/num_ite_base);
sinr=zeros(1,num_ite_ran/num_ite_base);

% temporary container
mmse_temp=zeros(1,num_stream);
sinr_temp=zeros(1,num_stream);

% for each number of bi-directional optimization iteration
for m=num_ite_base:num_ite_base:num_ite_ran
    % initialized forward training
    % initialized mixing matrix
    mix_mat=sqrt(pow)*Gaussian_chan*nor_precoder;
    % statistical correlation matrix
    cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
    % Wiener estimated receive filter
    rece_fil=cor_mat\mix_mat;
    % power normalization
    rece_fil_nor=mat_pow_nor(rece_fil);

    for m_d=1:m
        % backward optimization
        % mixing matrix
        mix_mat=transpose(sqrt(pow)*Gaussian_chan)* ...
            conj(rece_fil_nor);
```

```
        % statistical correlation matrix
        cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
        % Wiener estimated precoder
        precoder=conj(cor_mat\mix_mat);
        % power normalization
        precoder_nor=mat_pow_nor(precoder);

        % forward optimization
        % mixing matrix
        mix_mat=sqrt(pow)*Gaussian_chan*precoder_nor;
        % statistical correlation matrix
        cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
        % Wiener estimated receive filter
        rece_fil=cor_mat\mix_mat;
        % power normalization
        rece_fil_nor=mat_pow_nor(rece_fil);
    end

    % calculate MMSE & SINR
    % for each stream
    for n=1:num_stream
        % MMSE
        mmse_temp(1,n)=1-abs(ctranspose(rece_fil(:,n))* ...
            cor_mat*rece_fil(:,n));
        % SINR
        sinr_temp(1,n)=abs(ctranspose(mix_mat(:,n))*(( ...
            cor_mat-mix_mat(:,n)*ctranspose(mix_mat(:,n)))\ ...
            mix_mat(:,n)));
    end

    % average over streams
    mmse(1,m/num_ite_base)=mean(mmse_temp,2);
    sinr(1,m/num_ite_base)=mean(sinr_temp,2);
end

% decibel conversion
mmse=10*log10(mmse);
sinr=10*log10(sinr);

% number of iteration axis
num_ite_leng=zeros(1,num_ite_ran/num_ite_base);
for m=1:num_ite_ran/num_ite_base
    num_ite_leng(1,m)=m*num_ite_base;
end

end
```

```
wiener_mmse_sinr_vs_numite_bidirec_rr.m:

% Wiener (optimal) estimated (MMSE & SINR) vs Number of ...
    bi-directional optimization iteration with reduced-rank

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)

% parameter extraction
[Nr]=size(Gaussian_chan,1);
% output container
mmse=zeros(size(red_rank,2),num_ite_ran/num_ite_base);
sinr=zeros(size(red_rank,2),num_ite_ran/num_ite_base);

% for each reduced-rank
for rr_ind=1:size(red_rank,2)
    % assign current reduced-rank
    rr=red_rank(rr_ind);

    % for each number of bi-directional training iteration
    for m=num_ite_base:num_ite_base:num_ite_ran

        % initialized forward training
        % initialized mixing matrix
        mix_mat=sqrt(pow)*Gaussian_chan*nor_precoder;
        % statistical correlation matrix
        cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr);
        % Wiener estimated combiner
        [rece_fil,¬,¬]=wiener_rr_fil_mmse_sinr_cal(cor_mat, ...
            mix_mat,rr);
        % power normalization
        combiner_nor=mat_pow_nor(rece_fil);

        for m_d=1:m
            % backward optimization
            % mixing matrix
            mix_mat=transpose(sqrt(pow)*Gaussian_chan)*conj( ...
                combiner_nor);
            % statistical correlation matrix
            cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar* ...
                eye(Nr);
            % Wiener estimated precoder
            [precoder,¬,¬]=wiener_rr_fil_mmse_sinr_cal(cor_mat, ...
                mix_mat,rr);
            precoder=conj(precoder);
            % power normalization
            precoder_nor=mat_pow_nor(precoder);
```

```

            % forward optimization
            % mixing matrix
            mix_mat=sqrt(pow)*Gaussian_chan*precoder_nor;
            % statistical correlation matrix
            cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar* ...
                eye(Nr);
            % Wiener estimated combiner
            [rece_fil,mmse_val,sinr_val]= ...
                wiener_rr_fil_mmse_sinr_cal(cor_mat,mix_mat,rr);
            % power normalization
            combiner_nor=mat_pow_nor(rece_fil);
        end
        mmse(rr_ind,m/num_ite_base)=mmse_val;
        sinr(rr_ind,m/num_ite_base)=sinr_val;
    end
end

% decibel conversion
mmse=10*log10(mmse);
sinr=10*log10(sinr);

% number of iteration axis
num_ite_leng=zeros(1,num_ite_ran/num_ite_base);
for m=1:num_ite_ran/num_ite_base
    num_ite_leng(1,m)=m*num_ite_base;
end

end




wiener_mmse_sinr_vs_snr_fr.m:

% Wiener (optimal) estimated MMSE & SINR vs SNR with full-rank

function [mmse,sinr,snr] = ...
    wiener_mmse_sinr_vs_snr_fr(pow,Gaussian_chan,nor_precoder, ...
    noise_covar)

% parameter extraction
xleng=size(pow,2);
Nr=size(Gaussian_chan,1);
num_stream=size(nor_precoder,2);

% output container
mmse=zeros(1,xleng);
sinr=zeros(1,xleng);
```

```
snr=zeros(1,xleng);

% for each transmit power
for m=1:xleng
    % temporary MMSE & SINR container
    mmse_temp=zeros(1,num_stream);
    sinr_temp=zeros(1,num_stream);

    mix_mat=sqrt(pow(1,m))*Gaussian_chan*nor_precoder; ...
                                % mixing matrix
    cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr); ...
                          % correlation matrix
    rece_fil=cor_mat\mix_mat; ...
                                                          % ...
        receive filter

    % calculate MMSE & SINR
    % for each stream
    for n=1:num_stream
        % MMSE
        mmse_temp(1,n)=1-abs(ctranspose(rece_fil(:,n))* ...
            cor_mat*rece_fil(:,n));
        % SINR
        sinr_temp(1,n)=abs(ctranspose(mix_mat(:,n))*(( ...
            cor_mat-mix_mat(:,n)*ctranspose(mix_mat(:,n)))\ ...
            mix_mat(:,n)));
    end

    % average over streams
    mmse(1,m)=mean(mmse_temp,2);
    sinr(1,m)=mean(sinr_temp,2);
    snr(1,m)=pow(1,m)/noise_covar;
end

% decibal conversion
mmse=10*log10(mmse);
sinr=10*log10(sinr);
snr=10*log10(snr);

end




wiener_mmse_sinr_vs_snr_rr.m:

% Wiener (optimal) estimated MMSE & SINR vs SNR with ...
    reduced-rank

```

```
function [mmse,sinr,snr] = ...
    wiener_mmse_sinr_vs_snr_rr(pow,Gaussian_chan,nor_precoder, ...
    noise_covar,red_rank)

% parameter extraction
xleng=size(pow,2);
Nr=size(Gaussian_chan,1);
num_stream=size(nor_precoder,2);

% output container
mmse=zeros(size(red_rank,2),xleng);
sinr=zeros(size(red_rank,2),xleng);

% for each reduced rank
for rr_ind=1:size(red_rank,2)
    % reduced rank value
    rr=red_rank(rr_ind);

    for m=1:xleng
        krylov=zeros(Nr,rr); ...
                                                    % ...
            Krylov subspace
        mix_mat=sqrt(pow(1,m))*Gaussian_chan*nor_precoder; ...
                                   % mixing matrix
        cor_mat=mix_mat*ctranspose(mix_mat)+noise_covar*eye(Nr); ...
                            % correlation matrix

        mmse_temp=zeros(1,num_stream);
        sinr_temp=zeros(1,num_stream);
        % for each stream
        for n=1:num_stream
            % Krylov subspace
            for n_dot=1:rr
                krylov(:,n_dot)=cor_mat^(n_dot-1)*mix_mat(:,n);
            end

            % MMSE
            tri_dia_cor_mat=ctranspose(krylov)*cor_mat*krylov; ...
                               % tri-diagonal covariance matrix
            mmse_temp_val=1-abs(ctranspose(mix_mat(:,n))* ...
                krylov*(tri_dia_cor_mat\(ctranspose(krylov)))* ...
                mix_mat(:,n));
            mmse_temp(1,n)=mmse_temp_val;

            % SINR
            sinr_temp_val=abs(ctranspose(mix_mat(:,n))*krylov ...
                ...
                                 *((ctranspose(krylov)*(cor_mat- ...
                                     mix_mat(:,n)*ctranspose(mix_mat ...
```

```
                                        (:,n)))*krylov) ...
                                    \ctranspose(krylov))*mix_mat(:,n));
                sinr_temp(1,n)=sinr_temp_val;
            end

            % averaging over streams
            mmse(rr_ind,m)=mean(mmse_temp,2);
            sinr(rr_ind,m)=mean(sinr_temp,2);
        end
    end

    % decibel conversion
    mmse=10*log10(mmse);
    sinr=10*log10(sinr);
    snr=19*log10(pow/noise_covar);

    end




    wiener_rr_fil_mmse_sinr_cal.m:

    % Wiener (optimal) reduced-rank filter, MMSE and SINR calculator

    function [fil,mmse,sinr] = ...
        wiener_rr_fil_mmse_sinr_cal(cor_mat,mix_mat,rr)

    % dimension extraction
    [Nr,num_stream]=size(mix_mat);
    % Krylov subspace preallocation
    krylov=zeros(Nr,rr);
    % filter preallocation
    fil=zeros(Nr,num_stream);

    % temporary container
    mmse_temp=zeros(1,num_stream);
    sinr_temp=zeros(1,num_stream);

    % for each transmitted stream
    for m=1:num_stream

        % constructe Krylov subspace for the current transmitted ...
            stream
        for n=1:rr
            krylov(:,n)=cor_mat^(n-1)*mix_mat(:,m);
        end

        % reduced-rank filter
```

```
    fil(:,m)=krylov*((ctranspose(krylov)*cor_mat*krylov) ...
             \ctranspose(krylov))*mix_mat(:,m);

    % MSE
    tri_dia_cor_mat=ctranspose(krylov)*cor_mat*krylov; ...
                                   % tri-diagonal covariance matrix
    mmse_temp_val=1-abs(ctranspose(mix_mat(:,m))*krylov* ...
        (tri_dia_cor_mat\ctranspose(krylov))*mix_mat(:,m));
    mmse_temp(1,m)=mmse_temp_val;

    % SINR
    sinr_temp_val=abs(ctranspose(mix_mat(:,m))*krylov ...
                     *((ctranspose(krylov)*(cor_mat-mix_mat ...
                        (:,m)*ctranspose(mix_mat(:,m)))*krylov) ...
                        ...
                     \ctranspose(krylov))*mix_mat(:,m));

    sinr_temp(1,m)=sinr_temp_val;
end

% averaging over all transmitted streams
mmse=mean(mmse_temp,2);
sinr=mean(sinr_temp,2);

end




wiener_rr_fil_sumrate_in_cal.m:

% Interference network Wiener (optimal) reduced-rank filter, ...
    sum rate calculator

function [fil1,fil2,fil3,sumrate] = ...
    wiener_rr_fil_sumrate_in_cal(cor_mat1,mix_mat11, ...
    cor_mat2,mix_mat22, cor_mat3,mix_mat33,rr)

% dimension extraction
[Nr,num_stream]=size(mix_mat11);
% Krylov subspaces preallocation
krylov1=zeros(Nr,rr);
krylov2=zeros(Nr,rr);
krylov3=zeros(Nr,rr);
% filters preallocation
fil1=zeros(Nr,num_stream);
fil2=zeros(Nr,num_stream);
fil3=zeros(Nr,num_stream);

```

```
% temporary container
sumrate1=zeros(1,num_stream);
sumrate2=zeros(1,num_stream);
sumrate3=zeros(1,num_stream);

% for each transmitted stream
for m=1:num_stream

    % constructe Krylov subspaces for the current ... transmitted stream
    for n=1:rr
        krylov1(:,n)=cor_mat1^(n-1)*mix_mat11(:,m);
        krylov2(:,n)=cor_mat2^(n-1)*mix_mat22(:,m);
        krylov3(:,n)=cor_mat3^(n-1)*mix_mat33(:,m);
    end

    % reduced-rank filters
    fil1(:,m)=krylov1*((ctranspose(krylov1)*cor_mat1*krylov1) ... ...
              \ctranspose(krylov1))*mix_mat11(:,m);
    fil2(:,m)=krylov2*((ctranspose(krylov2)*cor_mat2*krylov2) ... ...
              \ctranspose(krylov2))*mix_mat22(:,m);
    fil3(:,m)=krylov3*((ctranspose(krylov3)*cor_mat3*krylov3) ... ...
              \ctranspose(krylov3))*mix_mat33(:,m);

    % sum rate
    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)));
    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)));
    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)));
end

% network sum rate
sumrate=sum(sumrate1)+sum(sumrate2)+sum(sumrate3);

end
```

```
wiener_sumrate_fixednumite_bidirec_in_fr.m:

% Wiener (optimal) Sum rate for fixed number of ...
    bi-directional optimization iteration with full-rank
% in interference network

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)

% parameter extraction
Nr=size(squeeze(Gaussian_chan_mat(1,1,:,:)),1);
num_stream=size(squeeze(nor_precoder_mat(1,:,:)),2);
% output data
sumrate=zeros(1,train_leng_ran);

% initialized forward optimization
% receive filter updating, precoder --> receive filter
% initialized mixing matrices
mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:))* ...
    squeeze(nor_precoder_mat(1,:,:));
mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:))* ...
    squeeze(nor_precoder_mat(2,:,:));
mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:))* ...
    squeeze(nor_precoder_mat(3,:,:));
mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:))* ...
    squeeze(nor_precoder_mat(1,:,:));
mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:))* ...
    squeeze(nor_precoder_mat(2,:,:));
mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:))* ...
    squeeze(nor_precoder_mat(3,:,:));
mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:))* ...
    squeeze(nor_precoder_mat(1,:,:));
mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:))* ...
    squeeze(nor_precoder_mat(2,:,:));
mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:))* ...
    squeeze(nor_precoder_mat(3,:,:));
% statistical correlation matrices
cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
        +mix_mat21*ctranspose(mix_mat21) ...
        +mix_mat31*ctranspose(mix_mat31) ...
        +noise_covar*eye(Nr);
cor_mat2=mix_mat12*ctranspose(mix_mat12) ...
        +mix_mat22*ctranspose(mix_mat22) ...
        +mix_mat32*ctranspose(mix_mat32) ...
```

```
        +noise_covar*eye(Nr);
cor_mat3=mix_mat13*ctranspose(mix_mat13) ...
        +mix_mat23*ctranspose(mix_mat23) ...
        +mix_mat33*ctranspose(mix_mat33) ...
        +noise_covar*eye(Nr);
% Wiener (optimal) receive filters with power normalization
rece_fil1=mat_pow_nor(cor_mat1\mix_mat11);
rece_fil2=mat_pow_nor(cor_mat2\mix_mat22);
rece_fil3=mat_pow_nor(cor_mat3\mix_mat33);

% for each bi-directional optimization
for m=1:num_ite
    % backward optimization
    % precoder updating, precoder <-- receive filter
    % mixing matrices
    mix_mat11=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
        (1,1,:,:)))*conj(rece_fil1);
    mix_mat12=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
        (2,1,:,:)))*conj(rece_fil2);
    mix_mat13=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
        (3,1,:,:)))*conj(rece_fil3);
    mix_mat21=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
        (1,2,:,:)))*conj(rece_fil1);
    mix_mat22=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
        (2,2,:,:)))*conj(rece_fil2);
    mix_mat23=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
        (3,2,:,:)))*conj(rece_fil3);
    mix_mat31=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
        (1,3,:,:)))*conj(rece_fil1);
    mix_mat32=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
        (2,3,:,:)))*conj(rece_fil2);
    mix_mat33=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
        (3,3,:,:)))*conj(rece_fil3);
    % statistical correlation matrices
    cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
            +mix_mat12*ctranspose(mix_mat12) ...
            +mix_mat13*ctranspose(mix_mat13) ...
            +noise_covar*eye(Nr);
    cor_mat2=mix_mat21*ctranspose(mix_mat21) ...
            +mix_mat22*ctranspose(mix_mat22) ...
            +mix_mat23*ctranspose(mix_mat23) ...
            +noise_covar*eye(Nr);
    cor_mat3=mix_mat31*ctranspose(mix_mat31) ...
            +mix_mat32*ctranspose(mix_mat32) ...
            +mix_mat33*ctranspose(mix_mat33) ...
            +noise_covar*eye(Nr);
    % Wiener (optimal) precoders with power normalization
    precoder1=mat_pow_nor(conj(cor_mat1\mix_mat11));
    precoder2=mat_pow_nor(conj(cor_mat2\mix_mat22));
```

```
    precoder3=mat_pow_nor(conj(cor_mat3\mix_mat33));

    % forward optimization
    % for receiver, precoder --> receive filter
    % mixing matrices
    mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:)) ...
        *precoder1;
    mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:)) ...
        *precoder2;
    mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:)) ...
        *precoder3;
    mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:)) ...
        *precoder1;
    mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
        *precoder2;
    mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
        *precoder3;
    mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
        *precoder1;
    mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
        *precoder2;
    mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
        *precoder3;
    % statistical correlation matrices
    cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
            +mix_mat21*ctranspose(mix_mat21) ...
            +mix_mat31*ctranspose(mix_mat31) ...
            +noise_covar*eye(Nr);
    cor_mat2=mix_mat12*ctranspose(mix_mat12) ...
            +mix_mat22*ctranspose(mix_mat22) ...
            +mix_mat32*ctranspose(mix_mat32) ...
            +noise_covar*eye(Nr);
    cor_mat3=mix_mat13*ctranspose(mix_mat13) ...
            +mix_mat23*ctranspose(mix_mat23) ...
            +mix_mat33*ctranspose(mix_mat33) ...
            +noise_covar*eye(Nr);
    % Wiener (optimal) receive filters with power normalization
    rece_fil1=mat_pow_nor(cor_mat1\mix_mat11);
    rece_fil2=mat_pow_nor(cor_mat2\mix_mat22);
    rece_fil3=mat_pow_nor(cor_mat3\mix_mat33);
end

% data preallocation
sumrate1=zeros(1,num_stream);
sumrate2=zeros(1,num_stream);
sumrate3=zeros(1,num_stream);

% for each stream
for m=1:num_stream
```

```
    % calculate sum rate
    sumrate1(1,m)=log2(1+abs(ctranspose(mix_mat11(:,m))* ...
        ((cor_mat1-mix_mat11(:,m)*ctranspose(mix_mat11(:,m))) ...
        \mix_mat11(:,m))));
    sumrate2(1,m)=log2(1+abs(ctranspose(mix_mat22(:,m))* ...
        ((cor_mat2-mix_mat22(:,m)*ctranspose(mix_mat22(:,m))) ...
        \mix_mat22(:,m))));
    sumrate3(1,m)=log2(1+abs(ctranspose(mix_mat33(:,m))* ...
        ((cor_mat3-mix_mat33(:,m)*ctranspose(mix_mat33(:,m))) ...
        \mix_mat33(:,m))));
end

% network sum rate
sumrate_val=sum(sumrate1)+sum(sumrate2)+sum(sumrate3);

% training length axis
train_leng=zeros(1,train_leng_ran);
for m=1:train_leng_ran
    sumrate(1,m)=sumrate_val;
    train_leng(1,m)=m*train_base;
end

end




wiener_sumrate_fixednumite_bidirec_in_rr.m:

% Wiener (optimal) Sum rate for fixed number of ...
    bi-directional optimization iteration with reduced-rank
% in interference network

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)

% parameter extraction
Nr=size(squeeze(Gaussian_chan_mat(1,1,:,:)),1);
% temporary container
sumrate_temp=zeros(1,size(red_rank,2));

% for each reduced-rank
for rr_ind=1:size(red_rank,2)
    % reduced-rank value
    rr=red_rank(rr_ind);

    % initialized forward optimization
```

```
    % receive filter updating, precoder --> receive filter
    % initialized mixing matrices
    mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:)) ...
        *squeeze(nor_precoder_mat(1,:,:));
    mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:)) ...
        *squeeze(nor_precoder_mat(2,:,:));
    mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:)) ...
        *squeeze(nor_precoder_mat(3,:,:));
    mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:)) ...
        *squeeze(nor_precoder_mat(1,:,:));
    mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
        *squeeze(nor_precoder_mat(2,:,:));
    mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
        *squeeze(nor_precoder_mat(3,:,:));
    mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
        *squeeze(nor_precoder_mat(1,:,:));
    mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
        *squeeze(nor_precoder_mat(2,:,:));
    mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
        *squeeze(nor_precoder_mat(3,:,:));
    % statistical correlation matrices
    cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
             +mix_mat21*ctranspose(mix_mat21) ...
             +mix_mat31*ctranspose(mix_mat31) ...
             +noise_covar*eye(Nr);
    cor_mat2=mix_mat12*ctranspose(mix_mat12) ...
             +mix_mat22*ctranspose(mix_mat22) ...
             +mix_mat32*ctranspose(mix_mat32) ...
             +noise_covar*eye(Nr);
    cor_mat3=mix_mat13*ctranspose(mix_mat13) ...
             +mix_mat23*ctranspose(mix_mat23) ...
             +mix_mat33*ctranspose(mix_mat33) ...
             +noise_covar*eye(Nr);
    % reduced-rank Wiener (optimal) receive filters
    [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);
    rece_fil1=mat_pow_nor(rece_fil1);
    rece_fil2=mat_pow_nor(rece_fil2);
    rece_fil3=mat_pow_nor(rece_fil3);

    % for each bi-directional optimization
    for m=1:num_ite
        % backward optimization
        % precoder updating, precoder <-- receive filter
        % mixing matrices
        mix_mat11=sqrt(pow)*transpose(squeeze ...
            (Gaussian_chan_mat(1,1,:,:)))*conj(rece_fil1);
        mix_mat12=sqrt(pow)*transpose(squeeze ...
```

```
    (Gaussian_chan_mat(2,1,:,:)))*conj(rece_fil2);
mix_mat13=sqrt(pow)*transpose(squeeze ...
    (Gaussian_chan_mat(3,1,:,:)))*conj(rece_fil3);
mix_mat21=sqrt(pow)*transpose(squeeze ...
    (Gaussian_chan_mat(1,2,:,:)))*conj(rece_fil1);
mix_mat22=sqrt(pow)*transpose(squeeze ...
    (Gaussian_chan_mat(2,2,:,:)))*conj(rece_fil2);
mix_mat23=sqrt(pow)*transpose(squeeze ...
    (Gaussian_chan_mat(3,2,:,:)))*conj(rece_fil3);
mix_mat31=sqrt(pow)*transpose(squeeze ...
    (Gaussian_chan_mat(1,3,:,:)))*conj(rece_fil1);
mix_mat32=sqrt(pow)*transpose(squeeze ...
    (Gaussian_chan_mat(2,3,:,:)))*conj(rece_fil2);
mix_mat33=sqrt(pow)*transpose(squeeze ...
    (Gaussian_chan_mat(3,3,:,:)))*conj(rece_fil3);
% statistical correlation matrices
cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
        +mix_mat12*ctranspose(mix_mat12) ...
        +mix_mat13*ctranspose(mix_mat13) ...
        +noise_covar*eye(Nr);
cor_mat2=mix_mat21*ctranspose(mix_mat21) ...
        +mix_mat22*ctranspose(mix_mat22) ...
        +mix_mat23*ctranspose(mix_mat23) ...
        +noise_covar*eye(Nr);
cor_mat3=mix_mat31*ctranspose(mix_mat31) ...
        +mix_mat32*ctranspose(mix_mat32) ...
        +mix_mat33*ctranspose(mix_mat33) ...
        +noise_covar*eye(Nr);
% reduced-rank Wiener (optimal) precoders
[precoder1,precoder2,precoder3,¬]= ...
    wiener_rr_fil_sumrate_in_cal(cor_mat1,mix_mat11, ...
    cor_mat2,mix_mat22, cor_mat3,mix_mat33,rr);
precoder1=mat_pow_nor(conj(precoder1));
precoder2=mat_pow_nor(conj(precoder2));
precoder3=mat_pow_nor(conj(precoder3));

% forward optimization
% for receiver, precoder --> receive filter
% mixing matrices
mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (1,1,:,:))*precoder1;
mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (2,1,:,:))*precoder2;
mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (3,1,:,:))*precoder3;
mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (1,2,:,:))*precoder1;
mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat ...
    (2,2,:,:))*precoder2;
```

```
        mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat ...
            (3,2,:,:))*precoder3;
        mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat ...
            (1,3,:,:))*precoder1;
        mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat ...
            (2,3,:,:))*precoder2;
        mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat ...
            (3,3,:,:))*precoder3;
        % statistical correlation matrices
        cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
                +mix_mat21*ctranspose(mix_mat21) ...
                +mix_mat31*ctranspose(mix_mat31) ...
                +noise_covar*eye(Nr);
        cor_mat2=mix_mat12*ctranspose(mix_mat12) ...
                +mix_mat22*ctranspose(mix_mat22) ...
                +mix_mat32*ctranspose(mix_mat32) ...
                +noise_covar*eye(Nr);
        cor_mat3=mix_mat13*ctranspose(mix_mat13) ...
                +mix_mat23*ctranspose(mix_mat23) ...
                +mix_mat33*ctranspose(mix_mat33) ...
                +noise_covar*eye(Nr);
        % reduced-rank Wiener (optimal) receive filters
        [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);
        rece_fil1=mat_pow_nor(rece_fil1);
        rece_fil2=mat_pow_nor(rece_fil2);
        rece_fil3=mat_pow_nor(rece_fil3);
    end

    sumrate_temp(1,rr_ind)=sumrate_val;
end

% training length axis
train_leng=zeros(1,train_leng_ran);
sumrate=zeros(size(red_rank,2),train_leng_ran);
for m=1:train_leng_ran
    sumrate(1,m)=sumrate_temp(1,1);
    sumrate(2,m)=sumrate_temp(1,2);
    train_leng(1,m)=m*train_base;
end

end




wiener_sumrate_vs_numite_bidirec_in_fr.m:

```

```
% Wiener (optimal) estimated Sum rate vs Number of ...
    bi-directional optimization iteration
% with full-rank in interference network

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)

% parameter extraction
Nr=size(squeeze(Gaussian_chan_mat(1,1,:,:)),1);
num_stream=size(squeeze(nor_precoder_mat(1,:,:)),2);
% temporary containers
sumrate1=zeros(1,num_stream);
sumrate2=zeros(1,num_stream);
sumrate3=zeros(1,num_stream);
% output data
sumrate=zeros(1,num_ite_ran/num_ite_base);

% for each number of bi-directional optimization iteration
for m=num_ite_base:num_ite_base:num_ite_ran
    % initialized forward training
    % receive filter updating, precoder --> receive filter
    % initialized mixing matrices
    mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:)) ...
        *squeeze(nor_precoder_mat(1,:,:));
    mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:)) ...
        *squeeze(nor_precoder_mat(2,:,:));
    mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:)) ...
        *squeeze(nor_precoder_mat(3,:,:));
    mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:)) ...
        *squeeze(nor_precoder_mat(1,:,:));
    mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
        *squeeze(nor_precoder_mat(2,:,:));
    mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
        *squeeze(nor_precoder_mat(3,:,:));
    mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
        *squeeze(nor_precoder_mat(1,:,:));
    mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
        *squeeze(nor_precoder_mat(2,:,:));
    mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
        *squeeze(nor_precoder_mat(3,:,:));
    % statistical correlation matrices
    cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
            +mix_mat21*ctranspose(mix_mat21) ...
            +mix_mat31*ctranspose(mix_mat31) ...
            +noise_covar*eye(Nr);
    cor_mat2=mix_mat12*ctranspose(mix_mat12) ...
            +mix_mat22*ctranspose(mix_mat22) ...
            +mix_mat32*ctranspose(mix_mat32) ...
```

```
            +noise_covar*eye(Nr);
    cor_mat3=mix_mat13*ctranspose(mix_mat13) ...
            +mix_mat23*ctranspose(mix_mat23) ...
            +mix_mat33*ctranspose(mix_mat33) ...
            +noise_covar*eye(Nr);
    % Wiener estimated receive filters with power normalization
    rece_fil1=mat_pow_nor(cor_mat1\mix_mat11);
    rece_fil2=mat_pow_nor(cor_mat2\mix_mat22);
    rece_fil3=mat_pow_nor(cor_mat3\mix_mat33);

    for m_d=1:m
        % backward optimization
        % precoder updating, precoder <-- receive filter
        % mixing matrices
        mix_mat11=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
            (1,1,:,:)))*conj(rece_fil1);
        mix_mat12=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
            (2,1,:,:)))*conj(rece_fil2);
        mix_mat13=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
            (3,1,:,:)))*conj(rece_fil3);
        mix_mat21=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
            (1,2,:,:)))*conj(rece_fil1);
        mix_mat22=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
            (2,2,:,:)))*conj(rece_fil2);
        mix_mat23=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
            (3,2,:,:)))*conj(rece_fil3);
        mix_mat31=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
            (1,3,:,:)))*conj(rece_fil1);
        mix_mat32=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
            (2,3,:,:)))*conj(rece_fil2);
        mix_mat33=sqrt(pow)*transpose(squeeze(Gaussian_chan_mat ...
            (3,3,:,:)))*conj(rece_fil3);
        % statistical correlation matrices
        cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
                +mix_mat12*ctranspose(mix_mat12) ...
                +mix_mat13*ctranspose(mix_mat13) ...
                +noise_covar*eye(Nr);
        cor_mat2=mix_mat21*ctranspose(mix_mat21) ...
                +mix_mat22*ctranspose(mix_mat22) ...
                +mix_mat23*ctranspose(mix_mat23) ...
                +noise_covar*eye(Nr);
        cor_mat3=mix_mat31*ctranspose(mix_mat31) ...
                +mix_mat32*ctranspose(mix_mat32) ...
                +mix_mat33*ctranspose(mix_mat33) ...
                +noise_covar*eye(Nr);
        % Wiener estimated precoders
        precoder1=mat_pow_nor(conj(cor_mat1\mix_mat11));
        precoder2=mat_pow_nor(conj(cor_mat2\mix_mat22));
        precoder3=mat_pow_nor(conj(cor_mat3\mix_mat33));
```

```

        % forward optimization
        % for receiver, precoder --> receive filter
        % mixing matrices
        mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:)) ...
            *precoder1;
        mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:)) ...
            *precoder2;
        mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:)) ...
            *precoder3;
        mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:)) ...
            *precoder1;
        mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
            *precoder2;
        mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
            *precoder3;
        mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
            *precoder1;
        mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
            *precoder2;
        mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
            *precoder3;
        % statistical correlation matrices
        cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
                +mix_mat21*ctranspose(mix_mat21) ...
                +mix_mat31*ctranspose(mix_mat31) ...
                +noise_covar*eye(Nr);
        cor_mat2=mix_mat12*ctranspose(mix_mat12) ...
                +mix_mat22*ctranspose(mix_mat22) ...
                +mix_mat32*ctranspose(mix_mat32) ...
                +noise_covar*eye(Nr);
        cor_mat3=mix_mat13*ctranspose(mix_mat13) ...
                +mix_mat23*ctranspose(mix_mat23) ...
                +mix_mat33*ctranspose(mix_mat33) ...
                +noise_covar*eye(Nr);
        % Wiener estimated receive filters with power ...
            normalization
        rece_fil1=mat_pow_nor(cor_mat1\mix_mat11);
        rece_fil2=mat_pow_nor(cor_mat2\mix_mat22);
        rece_fil3=mat_pow_nor(cor_mat3\mix_mat33);
    end

    % for each stream
    for n=1:num_stream
        % calculate sum rate
        sumrate1(1,n)=log2(1+abs(ctranspose(mix_mat11(:,n)) ...
            *((cor_mat1-mix_mat11(:,n)*ctranspose(mix_mat11(:,n))) ...
            \mix_mat11(:,n))));
        sumrate2(1,n)=log2(1+abs(ctranspose(mix_mat22(:,n)) ...
```

```
            *((cor_mat2-mix_mat22(:,n)*ctranspose(mix_mat22(:,n))) ...
            \mix_mat22(:,n))));
        sumrate3(1,n)=log2(1+abs(ctranspose(mix_mat33(:,n))* ...
            ((cor_mat3-mix_mat33(:,n)*ctranspose(mix_mat33(:,n))) ...
            \mix_mat33(:,n))));
    end

    % network sum rate
    sumrate(1,m/num_ite_base)=sum(sumrate1)+sum(sumrate2)+ ...
        sum(sumrate3);
end

% number of iteration axis
num_ite_leng=zeros(1,num_ite_ran/num_ite_base);
for m=1:num_ite_ran/num_ite_base
    num_ite_leng(1,m)=m*num_ite_base;
end

end




wiener_sumrate_vs_numite_bidirec_in_rr.m:

% Wiener (optimal) estimated Sum rate vs Number of ...
    bi-directional optimization iteration
% with reduced-rank in interference network

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)

% parameter extraction
Nr=size(squeeze(Gaussian_chan_mat(1,1,:,:)),1);
% output container
sumrate=zeros(size(red_rank,2),num_ite_ran/num_ite_base);

% for each reduced-rank
for rr_ind=1:size(red_rank,2)
    % assign current reduced-rank
    rr=red_rank(rr_ind);

    % for each number of bi-directional training iteration
    for m=num_ite_base:num_ite_base:num_ite_ran

        % initialized forward training
        % receive filter updating, precoder --> receive filter
        % initialized mixing matrices
```

```
        mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat(1,1,:,:)) ...
            *squeeze(nor_precoder_mat(1,:,:));
        mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat(2,1,:,:)) ...
            *squeeze(nor_precoder_mat(2,:,:));
        mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat(3,1,:,:)) ...
            *squeeze(nor_precoder_mat(3,:,:));
        mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat(1,2,:,:)) ...
            *squeeze(nor_precoder_mat(1,:,:));
        mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat(2,2,:,:)) ...
            *squeeze(nor_precoder_mat(2,:,:));
        mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat(3,2,:,:)) ...
            *squeeze(nor_precoder_mat(3,:,:));
        mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat(1,3,:,:)) ...
            *squeeze(nor_precoder_mat(1,:,:));
        mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat(2,3,:,:)) ...
            *squeeze(nor_precoder_mat(2,:,:));
        mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat(3,3,:,:)) ...
            *squeeze(nor_precoder_mat(3,:,:));
        % statistical correlation matrices
        cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
                +mix_mat21*ctranspose(mix_mat21) ...
                +mix_mat31*ctranspose(mix_mat31) ...
                +noise_covar*eye(Nr);
        cor_mat2=mix_mat12*ctranspose(mix_mat12) ...
                +mix_mat22*ctranspose(mix_mat22) ...
                +mix_mat32*ctranspose(mix_mat32) ...
                +noise_covar*eye(Nr);
        cor_mat3=mix_mat13*ctranspose(mix_mat13) ...
                +mix_mat23*ctranspose(mix_mat23) ...
                +mix_mat33*ctranspose(mix_mat33) ...
                +noise_covar*eye(Nr);
        % reduced-rank Wiener (optimal) receive filters
        [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);
        rece_fil1=mat_pow_nor(rece_fil1);
        rece_fil2=mat_pow_nor(rece_fil2);
        rece_fil3=mat_pow_nor(rece_fil3);

        for m_d=1:m

            % backward optimization
            % precoder updating, precoder <-- receive filter
            % mixing matrices
            mix_mat11=sqrt(pow)*transpose(squeeze ...
                (Gaussian_chan_mat(1,1,:,:)))*conj(rece_fil1);
            mix_mat12=sqrt(pow)*transpose(squeeze ...
                (Gaussian_chan_mat(2,1,:,:)))*conj(rece_fil2);
            mix_mat13=sqrt(pow)*transpose(squeeze ...
```

```
        (Gaussian_chan_mat(3,1,:,:)))*conj(rece_fil3);
    mix_mat21=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(1,2,:,:)))*conj(rece_fil1);
    mix_mat22=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(2,2,:,:)))*conj(rece_fil2);
    mix_mat23=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(3,2,:,:)))*conj(rece_fil3);
    mix_mat31=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(1,3,:,:)))*conj(rece_fil1);
    mix_mat32=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(2,3,:,:)))*conj(rece_fil2);
    mix_mat33=sqrt(pow)*transpose(squeeze ...
        (Gaussian_chan_mat(3,3,:,:)))*conj(rece_fil3);
    % statistical correlation matrices
    cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
            +mix_mat12*ctranspose(mix_mat12) ...
            +mix_mat13*ctranspose(mix_mat13) ...
            +noise_covar*eye(Nr);
    cor_mat2=mix_mat21*ctranspose(mix_mat21) ...
            +mix_mat22*ctranspose(mix_mat22) ...
            +mix_mat23*ctranspose(mix_mat23) ...
            +noise_covar*eye(Nr);
    cor_mat3=mix_mat31*ctranspose(mix_mat31) ...
            +mix_mat32*ctranspose(mix_mat32) ...
            +mix_mat33*ctranspose(mix_mat33) ...
            +noise_covar*eye(Nr);
    % reduced-rank Wiener (optimal) precoders
    [precoder1,precoder2,precoder3,¬]= ...
       wiener_rr_fil_sumrate_in_cal(cor_mat1,mix_mat11, ...
       cor_mat2,mix_mat22, cor_mat3,mix_mat33,rr);
    precoder1=mat_pow_nor(conj(precoder1));
    precoder2=mat_pow_nor(conj(precoder2));
    precoder3=mat_pow_nor(conj(precoder3));

    % forward optimization
    % for receiver, precoder --> receive filter
    % mixing matrices
    mix_mat11=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (1,1,:,:))*precoder1;
    mix_mat21=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (2,1,:,:))*precoder2;
    mix_mat31=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (3,1,:,:))*precoder3;
    mix_mat12=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (1,2,:,:))*precoder1;
    mix_mat22=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (2,2,:,:))*precoder2;
    mix_mat32=sqrt(pow)*squeeze(Gaussian_chan_mat ...
        (3,2,:,:))*precoder3;
```

```
                mix_mat13=sqrt(pow)*squeeze(Gaussian_chan_mat ...
                    (1,3,:,:))*precoder1;
                mix_mat23=sqrt(pow)*squeeze(Gaussian_chan_mat ...
                    (2,3,:,:))*precoder2;
                mix_mat33=sqrt(pow)*squeeze(Gaussian_chan_mat ...
                    (3,3,:,:))*precoder3;
                % statistical correlation matrices
                cor_mat1=mix_mat11*ctranspose(mix_mat11) ...
                        +mix_mat21*ctranspose(mix_mat21) ...
                        +mix_mat31*ctranspose(mix_mat31) ...
                        +noise_covar*eye(Nr);
                cor_mat2=mix_mat12*ctranspose(mix_mat12) ...
                        +mix_mat22*ctranspose(mix_mat22) ...
                        +mix_mat32*ctranspose(mix_mat32) ...
                        +noise_covar*eye(Nr);
                cor_mat3=mix_mat13*ctranspose(mix_mat13) ...
                        +mix_mat23*ctranspose(mix_mat23) ...
                        +mix_mat33*ctranspose(mix_mat33) ...
                        +noise_covar*eye(Nr);
                % reduced-rank Wiener (optimal) receive filters
                [rece_fil1,rece_fil2,rece_fil3,sumrate_val]= ...
                    wiener_rr_fil_sumrate_in_cal(cor_mat1,mix_mat11, ...
                    ...


                rece_fil1=mat_pow_nor(rece_fil1);
                rece_fil2=mat_pow_nor(rece_fil2);
                rece_fil3=mat_pow_nor(rece_fil3);
            end
            sumrate(rr_ind,m/num_ite_base)=sumrate_val;
        end
    end

    % number of iteration axis
    num_ite_leng=zeros(1,num_ite_ran/num_ite_base);
    for m=1:num_ite_ran/num_ite_base
        num_ite_leng(1,m)=m*num_ite_base;
    end

    end
```