University of Alberta

A Near-Optimal and Efficiently Parallelizable Detector for Multiple-Input Multiple-Output Wireless Systems

by

Arsene Fourier Pankeu Yomi

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science
in
Communications

Department of Electrical and Computer Engineering

© Arsene Fourier Pankeu Yomi
Spring 2012
Edmonton, Alberta

Permission is hereby granted to the University of Alberta Libraries to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only. Where the thesis is converted to, or otherwise made available in digital form, the University of Alberta will advise potential users of the thesis of these terms.

The author reserves all other publication and other rights in association with the copyright in the thesis and, except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatsoever without the author's prior written permission.

This thesis is dedicated to my parents, who provide me with continuous moral and material support in every steps of my life, even with the long distance that separates us. It is also dedicated to the Guedom family, who made me a member of their family in all regards, and without whom this thesis would have never been possible.

Abstract

Broadband Wireless Communications and Multiple-Input Multiple-Output (MIMO) systems have been the focus of much research over the past decade. A variety of MIMO detection algorithms have been proposed for detecting the data signals from the multiple received and demodulated baseband signals. Among the detectors, sphere decoding algorithms are known to be near-optimal but they are relatively complicated and have variable detection latencies, and are therefore inconvenient to implement. Also, the variable latency of most sphere decoder algorithms makes them difficult to implement efficiently on parallel hardware.

This thesis work evaluates several alternative MIMO detectors and proposes a near-optimal and efficiently parallelizable detector. The new MIMO detector has much lower computational complexity than a sphere decoder, and has a convenient parallel structure comprising multiple instances of the Vertical Bell Laboratories Layered Space Time (V-BLAST) MIMO detection scheme.

Acknowledgements

I would like to acknowledge the guidance and tremendous corrections work of my supervisor, Dr. Bruce Cockburn and the theoretical advice from Saeed Fouladi Fard. I would also like to acknowledge the financial support of iCORE and the Natural Sciences and Engineering Research Council (NSERC) of Canada. Finally, I would like to thank my VLSI and HCDC fellows, Russell Dodd, John Koob and all the others for their help and company from the first day.

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Results

List of Abbreviations

3GPP2: Third Generation Partnership Project 2

a^*: complex conjugate of the number a

$\mathbf{A}^{-1}$**:** Inverse of the square matrix A

ADSL: Asymmetric Digital Subscriber Line

AWGN: Additive White Gaussian Noise

B.B.: baseband

BER: Bit Error Rate

CDMA: Code-Division Multiple Access

d: diversity gain

Eb/No: Energy per bit to noise power spectral density ratio

F-BLAST: Fouladi Fard's detection scheme based on the V-BLAST detection scheme

FEC: Forward Error Control

FR-BLAST: family of reduced parallelism detectors based on the FBLAST detection scheme

H: *m*-by-*m* channel matrix, with complex entry normally distributed with zero mean and unit variance

$\mathbf{H}^{\mathbf{H}}$**:** Hermitian matrix, i.e., conjugate transpose of the matrix **H**

$\mathbf{I}_m$**:** *m*-by-*m* identity matrix

L: number of information bits

LLR: Log-Likelihood ratio

LTE: Long Term Evolution, a standard for wireless communications

m: number of antennas at each side of the radio link

min: minimum of a set of real numbers

M: size of the constellation signal

MAP: Maximum A Posteriori

MIMO: Multiple-Input Multiple-Output

ML: Maximum Likelihood

MLD: Maximum Likelihood Detection

MMSE: Minimum Mean Square Error

$\underline{\boldsymbol{n}}$**:** complex noise vector

n_r: number of receiver antennas

n_t: number of transmitter antennas

OFDM: Orthogonal Frequency Division Multiplexing (OFDM)

$\mathbf{P_e(SNR)}$**:** average error probability of the given scheme at a given SNR

QAM: Quadrature Amplitude Modulation

QoS: Quality of Service

r: code rate of the encoder

r_m: multiplexing gain

RF: radio frequency

RSC: Recursive Systematic Convolutional

$R_{x,i}$: i-th receiver antenna

$\underline{\hat{\boldsymbol{s}}}$**:** detected symbol vector of length m

$\underline{\boldsymbol{s}}$**:** transmitted symbol vector of length m

SER: Symbol Error Rate

SIC-MMSE: Soft-Interference Cancellation based on the MMSE criterion

SISO: Single-Input Single-Output

SNR: Signal-to-Noise-Ratio

ST-BICM: Space-Time Bit-Interleaved Coded Modulation

T: duration of a frame

TC: Turbo Codes

$T_{x,i}$: i-th transmitter antenna

$|v|$: absolute value of the number v

$||\boldsymbol{v}||$: norm of the vector $\boldsymbol{V}$

V-BLAST: Vertical Bell Laboratories Layered Space Time

W: size of the search window or restricted search space

$\underline{\boldsymbol{y}}$**:** received noisy symbol vector

ZF: Zero Forcing

I- Introduction

This thesis is concerned with low-complexity and efficient detectors for Multiple-Input Multiple-Output (MIMO) wireless communication. Of special practical interest are detectors that can exploit parallel hardware and that can be scaled up to handle a larger number of antennas and more complex signal constellations. The performance of proposed new MIMO detection algorithms is compared to the performance characteristics of conventional detectors with respect to their Bit Error Rate (BER) versus Signal-to-Noise Ratio (SNR) performance and their computational complexity. We first consider uncoded systems, and then extend the investigation to a Turbo-coded system model.

MIMO wireless technology is being used to provide both greater data throughput over the same radio bandwidth as well as greater robustness in the presence of channel noise and other impairments. However, the more information that we try to send (within the Shannon capacity limit for the MIMO system), the more complex will be the detector at the receiver side.

The main objective of this project was to improve the relatively simple and well-known V-BLAST MIMO detector through the use of parallelism to achieve near-optimal performance. We considered a reasonable amount of pararallelism to be 16, since orthogonal frequency division multiplexing (OFDM) systems uses around 64 subcarriers. Parallel signal processing hardware has the advantages of potentially simplifying the chip implementation, lowering the voltage thus saving power and ensuring more predictable timing. A parallel implementation may make it easier to share hardware among multiple similar datapaths, which is a

likely scenario in multicarrier wireless systems. They are some challenges that must be overcome in parallel architectures, such as the problem of distributing data for parallel processing and then later gathering and possibly combining the results. A performance study was done to derive the computational complexity and to determine the Symbol Error Rate (SER) or Bit Error Rate (BER) versus SNR characteristics over a simulated noisy channel. The performance of alternative parallel MIMO detection schemes was compared with that of conventional detection schemes, such the minimum mean square error (MMSE) detector [1] and the original V-BLAST detector [2].

In contrast to existing near-optimal but computationally-expensive detection schemes, such as the sphere detector [3] or the tree-based search detector [4], we were able to achieve similar performance using parallel structures. The thesis also provides a detailed comparison of the computational cost for the alternative MIMO detectors. This cost is expressed in terms of the required number of fundamental real-valued operations (e.g., additions, multiplications, reciprocals) as well as the minimum possible execution time (in terms of single-cycle operations) assuming arbitrary parallelism.

The main contributions of this thesis are:

- A first analysis of the complexity of Fouladi Fard's parallel V-BLAST algorithm, which we call F-BLAST.
- Simulation-based investigation of the performance of new restricted search window versions of F-BLAST.

- A simulation-based investigation of the performance of real-valued versions of F-BLAST
- An analysis of the computational cost of the various MIMO detectors that were considered.
- Asymptotic analysis of F-BLAST, FR-BLAST and the real-valued versions of F-BLAST.
- A simulation study of the performance of the new detectors used in combination with soft turbo decoding.

II- Background

2.1 MIMO Systems

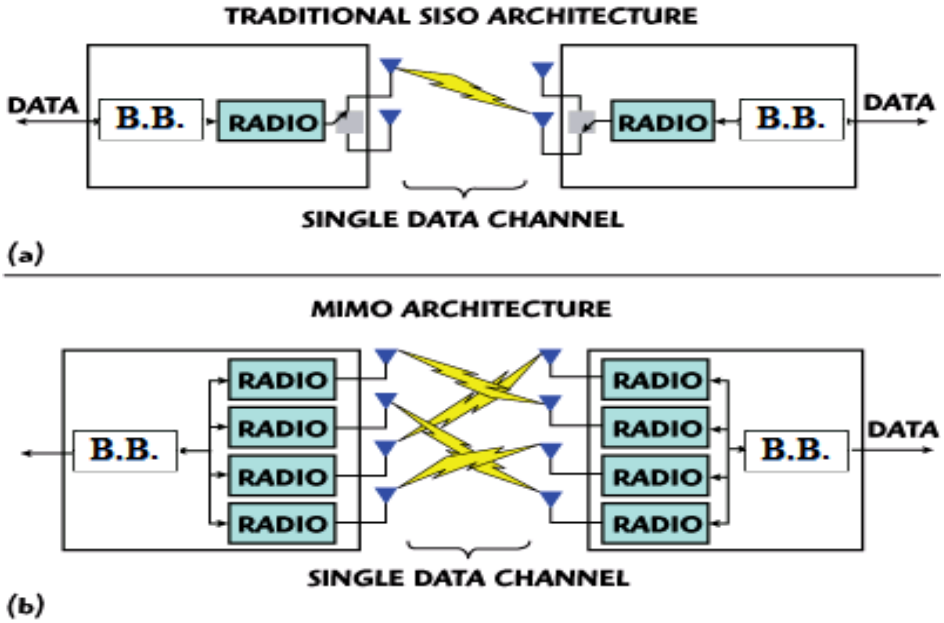


Figure 1 Radio Links Based on the (a) SISO and (b) MIMO Configurations [5]

In a conventional Single-Input Single-Output (SISO) communication system (see Figure 1a), there is a single transmitter and receiver at either end of the radio link. The transmitters and receivers contain a baseband (B.B.) processor as well as a radio frequency (RF) circuitry for each antenna (RADIO in Figure 1). We will assume that the B.B.-to-RF modulators in the transmitter and the RF-to-B.B demodulators in the receiver function perfectly without impairing the signals. In an ideal unobstructed communications channel, radio signals travel through free space along a single path from the transmit antenna to the receive antenna. Unfortunately, obstructions (such as buildings and natural terrain features) and propagation effects in the radio channel can create multipath effects such as

multiple reflected, refracted and scattered propagation paths. With multipath propagation, multiple copies of the transmitted signal arrive and get superimposed at the receiver antenna. Due to the inevitable differences in path lengths, the phases and amplitudes of these reflected signals are typically different from each other and from those of the possible direct line-of-sight path. Because of this, the signals at the receiver can combine constructively or destructively, causing position-dependant fluctuations in the received signal strength. These fluctuations can be very large (e.g. 30 dB or more) and will also change with time if the antennas move or if the environment changes. These position and time-dependant signal attenuations are called short-term fading [6]. Excessive fading effects can diminish the data throughput and could cause data loss. For transmission systems where the propagation effects can be determined only at the receiver, and under the assumption that each binary digit is equiprobable, the capacity (in bits of information/sec) of a SISO channel is given by Shannon's capacity theorem [7]:

$$C_{SISO} = B * \log_2(1 + \rho) \quad \text{bits of information/sec}$$

where B is the bandwidth of the channel and ρ is the average signal-to-noise-ratio. Shannon's theorem gives an upper limit on possible error-free data transmission. However the proof does not provide constructions that can achieve the limit.

In a conventional Multiple-Input Multiple-Output (MIMO) communications system (see Figure 1b), a radio link terminates at several antenna elements at both the transmitter and receiver. A baseband processor at the transmitter distributes the data over the multiple parallel tributaries and, optionally, inserts code bits for error control. At the receiver, the baseband

processor performs detection on the demodulated received signals and merges the recovered parallel bit streams into a single data stream. If coded bits were inserted at the transmitter, the receiver checks the values of the received data and code bits and possibly corrects errors in the data bits. MIMO technology has attracted attention in wireless communications since it offers significant increases in data throughput without requiring additional radio bandwidth or transmitted power. More specifically, MIMO technology provides higher spectral efficiency (more bits per second per hertz of bandwidth) and increased link reliability or diversity (greater robustness against fading).

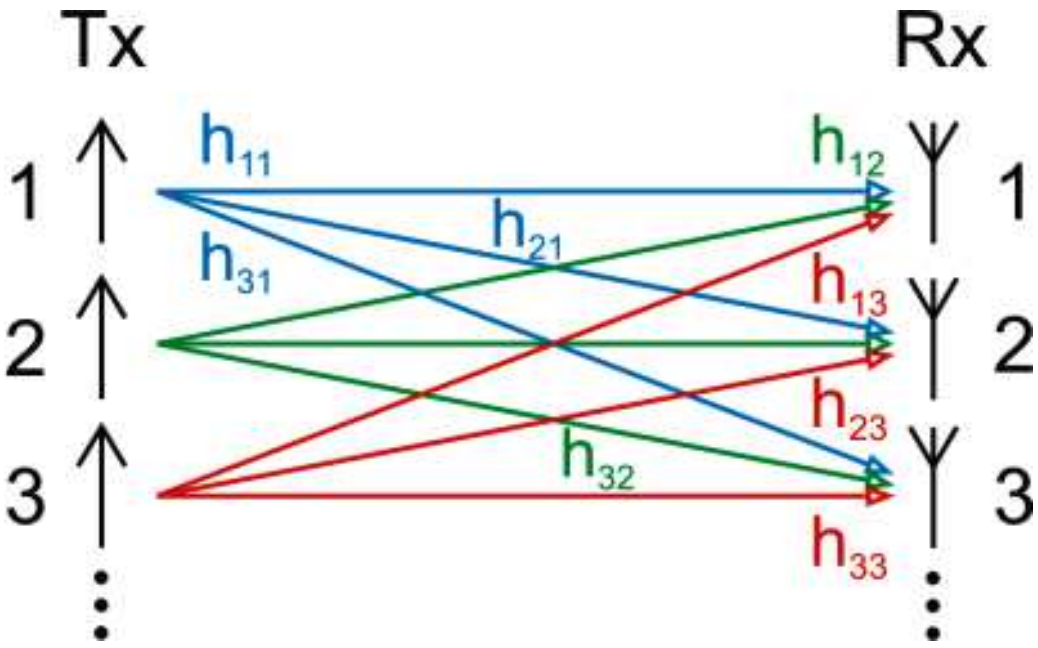


Figure 2 MIMO Channel Model [8]

The channel matrix **H** comprises the complex channel gains from each transmitting antenna to each receiving antenna. Each element $\boldsymbol{h_{ji}}$ of **H** is in general a complex vector that represents the discrete time channel impulse response between the *i*-th transmitter antenna and the *j*-th receiver antenna, as illustrated in

Figure 2. If the channel is flat fading, i.e., different frequency components of the signal experience the same magnitude of fading, then each element $\underline{h_{ji}}$ is a complex scalar. For a deterministic channel matrix, i.e., each element $\underline{h_{ji}}$ is known, without exploiting channel knowledge at the transmitter, the capacity of a MIMO channel is [7]:

$$C_{MIMO} = B * \log_2 (\det [\mathbf{I}_{n_r} + \frac{\rho}{n_t} * \underline{\mathbf{H}} * \underline{\mathbf{H}}^{\mathbf{H}}])$$

When n_t is large, $\frac{1}{n_t} * \underline{\mathbf{H}} * \underline{\mathbf{H}}^{\mathbf{H}} \cong \mathbf{I}_{n_r}$, where $\mathbf{I}_{n_r}$ denotes an $n_r \times n_r$ identity matrix.

In this special case [7]: $C_{MIMO} = m * B * \log_2 (1 + \rho) = m * C_{SISO}$ **bits/sec**

Here B is the bandwidth of the channel, ρ is the SNR, $n_t \geq 1$ and $n_r \geq 1$ are the number of transmit antennas and receive antennas respectively, and $\underline{\mathbf{H}}$ is the n_r-by-n_t channel matrix. The MIMO multiplexing gain m equals the minimum value of n_t and n_r. $\underline{\mathbf{H}}^{\mathbf{H}}$ denotes the Hermitian transpose of $\underline{\mathbf{H}}$, which is obtained by negating the imaginary part of each complex element of $\underline{\mathbf{H}}$ and then taking the transpose of the resulting matrix.

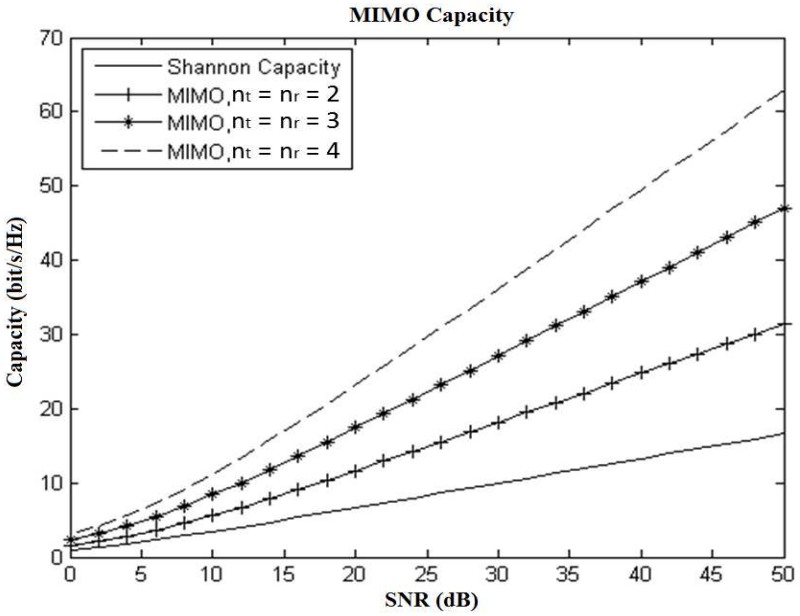


Figure 3 Average Capacity of Ideal MIMO (2×2, 3×3 and 4×4) and SISO (Conventional Shannon Capacity) Channels

The capacity expression implies that for a SISO system, 3 dB of extra signal power is needed for each extra bit per second of throughput at the maximum capacity limit. Also, as illustrated in Figure 3, the capacity of a MIMO system increases linearly with the minimum *m* of the number of transmit or receive antennas. An alternative view is that by providing multiple paths from the transmitter to the receiver, the effects of fading are mitigated on average and thus a larger effective SNR can be achieved while using a MIMO system.

Because of its advantages, MIMO technology has been adopted by all of the latest wireless standards such as the wireless local area network (WLAN) standard IEEE 802.11, used in Wi-Fi technologies; the wireless personal area network (WPAN) / Bluetooth - IEEE 802.15; and the metropolitan area network (MAN) which is branded as WiMax - IEEE 802.16 [9].

2.2 System Architecture

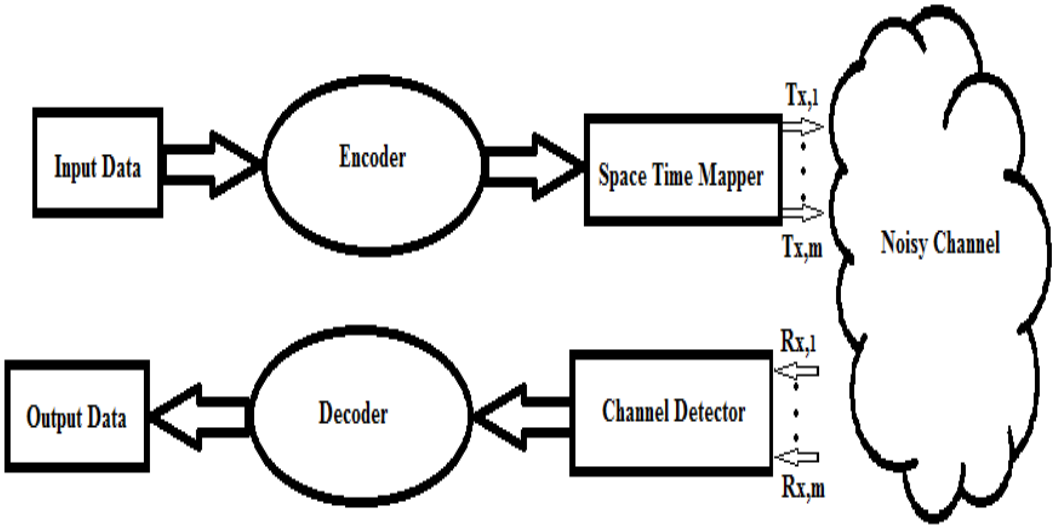


Figure 4 System Architecture

Figure 4 illustrates the architecture of the system that we used to model MIMO transmission and reception. Note that the system model is a conventional baseband model where the modulation step at the transmitter and the matching demodulation step at the receiver have both been omitted. Thus modulation and demodulation are assumed to occur without impairment. The serial stream of input data bits are encoded and then mapped to a space-time block of complex baseband symbols. The symbols are intermixed by convolution with the channel and then corrupted with additive white Gaussian noise (AWGN). The detector recovers complex symbols from the analog received signals and outputs blocks of soft data bits. Finally, the block of soft data bits is decoded and the resulting bits are merged into a serial stream of output data.

This work mainly focuses on the channel detector block shown in Figure 4. The channel detector is responsible for recovering a sequence of estimated

complex baseband symbols from the sampled baseband analog signals received from the antennas. We will first describe the features of the new class of detector and then analyze its performance. Finally, using a conventional Turbo Coding scheme, we will compare the BER performance of several alternative soft detectors. We will make the following assumptions:

- The input data or information bits or uncoded data is partitioned into blocks of length $L \gg 1$ containing randomly generated 0*s* and 1*s* with equal probability.
- If coding is used, the encoder is the parallel concatenation of two recursive systematic convolutional encoders of rate 1/2 [10]. The overall code rate is therefore 1/3 (r = 1/3), i.e., the output sequence length is tripled. The original information bits are interleaved with two equal-rate bits streams produced by the encoders.

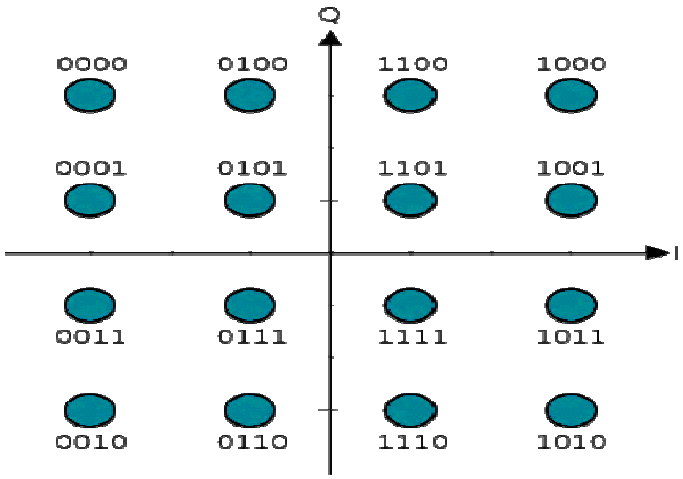


Figure 5 Constellation Diagram for Gray-Coded *M*-QAM (*M*=16) [11]

- The space-time mapper takes the block of encoded bits, and maps them into a block of symbols from a complex constellation containing $M \geq 2$ symbols, using Gray code, i.e., a binary numeral system where two successive numeral values differ in only one binary digit, improving error correction. The number M of available constellation symbols is typically 4, 8, 16, 64 or 256. We will be using standard complex M-ary Quadratic Amplitude Modulation (or M-QAM) constellations (see Figure 5). Sequences of symbols are grouped into frames that are sized so that the duration T of a frame, i.e., the number of samples in a frame, satisfies $T \geq 2 * m - 1$ (for near-optimal diversity-multiplexing trade-off [12]). Finally within each frame, the constellation symbols are grouped to form sample vectors of length m. Typically $m = 2$, 3 or 4.
- The samples are transmitted through a simulated noisy radio environment. To reduce the simulation workload, the channel matrix, $\underline{\mathbf{H}}$, is assumed to be invariant for the duration T of a frame (i.e., the number of symbols in a frame). Each element of $\underline{\mathbf{H}}$ is regenerated from a complex Gaussian distribution at the frame boundaries. Each scalar element $\underline{h_{ji}}$ represents the gain from the i-th transmitting antenna to the j-th receiving antenna, as illustrated in Figure 2. Scalar gains correspond to a frequency flat channel, where the duration of the impulse response of the channel is less than the symbol interval. We assume that the transmitter has no knowledge of the channel, but at the receiver side the channel matrix $\underline{\mathbf{H}}$ is perfectly estimated. In practice, the channel matrix $\underline{\mathbf{H}}$ is estimated using standard

methods that rely on fixed training symbol sequences that are inserted among the data symbols, reducing the code rate and/or throughput [13].

- The vectors of symbols transmitted at the same sample time through the channel are corrupted with Additive White Gaussian Noise (AWGN). The received noisy signal vector is then detected separately, producing a vector of either hard or soft bits. Hard bit information gives unqualified estimates (e.g., 0 or 1) of the received binary digit. Soft bit information gives estimates of binary digits along with probability information for each binary digit. The soft bits are typically log-likelihood ratios (LLRs) given to some finite bit precision (e.g., 4, 5 or 6 bits). The soft bits are output in blocks that correspond to the blocks of symbols produced in the transmitter by the space-time mapper. The multidimensional detected block of binary digits is then converted into a one-dimensional sequence of (hard or soft) bits.
- If coding was used in the transmitter, the block of received soft or hard bits is processed in the receiver by a decoder to recover the serial stream of estimated and corrected information bits. For example, in the case of a Turbo-coded system, the soft bits are processed iteratively using a standard soft decoding algorithm based on Maximum A Posteriori (MAP) Probability. The output of the decoder is a sequence of information bits with (hopefully) lowered BER.

The BER of the detector under consideration is computed by comparing the recovered data bits for each detection scheme to the originally transmitted data

bits. The performance of the detector is measured by determining the BER vs. SNR characteristic over a range of SNRs. A symbol encodes a sequence of $\log_2(M) \geq 1$ bits from an M-QAM constellation. Therefore at a given SNR, SER $\geq$ BER. Thus the BER vs. SNR characteristic provides the best overall measure of the performance of a coded system, i.e., the average number of bits in error from a received data block. The SER vs. SNR characteristic provides a more accurate measure of the performance of a symbol detector on its own, i.e., the average number of received symbols that have been detected with an error, without the benefit of an error-correcting code.

2.3 Conventional MIMO Detection Schemes

The simplified MIMO system model is illustrated in Figure 6.

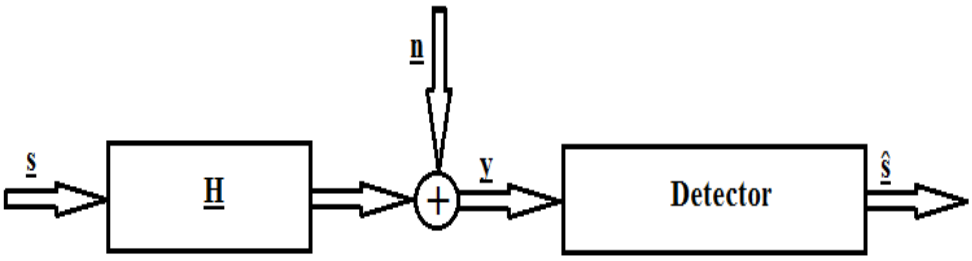


Figure 6 Simplified MIMO Detection Diagram

Here, $\underline{\mathbf{s}}$ is a transmitted symbol vector of length m, where $m = n_t = n_r$ is the assumed equal number of antennas at each end of the channel. Following standard practice, $\underline{\mathbf{H}}$ is an m-by-m channel matrix whose complex entries are normally-distributed with zero mean and unit variance. We assume that $\underline{\mathbf{H}}$ is constant for the duration of a frame, but is updated at frame boundaries in order to simulate a Rayleigh flat fading environment. The frame length T is adjusted empirically to achieve accurate simulated BER results. The noise vector $\underline{\boldsymbol{n}}$ is an AWGN vector of length m, whose coefficients are independent, normally-distributed complex variables with equal variance σ_n^2. $\underline{\boldsymbol{y}}$ is the corresponding received noisy signal vector, which can be expressed in the standard baseband discrete-time model as $\underline{\boldsymbol{y}} = \underline{\mathbf{H}} * \underline{\mathbf{s}} + \underline{\boldsymbol{n}}$ [14]. $\underline{\hat{\boldsymbol{s}}}$ is the detected signal vector of length m. The goal of the detector is to maximize the probability of the event '$\underline{\mathbf{s}} = \underline{\hat{\boldsymbol{s}}}$'. Various detectors have been proposed in the literature that range from the statistically-optimal (but prohibitively expensive) maximum likelihood (ML) detector to low-complexity detection schemes with relatively poor performance. We will focus on the

complexity, the accuracy and the delay (detection latency) to evaluate the different detectors.

2.3.1 Maximum Likelihood (ML) Detector [14]

Knowing that the transmitted symbols are drawn with equal probability from a known finite alphabet of size *M*, the ML detector selects the statistically most probable candidate from among the M^m possible transmitted symbol vectors. Intuitively, an optimal detector should return $\underline{\boldsymbol{s}} = \underline{\hat{\boldsymbol{s}}}$, the symbol vector whose conditional probability *Prob(*$\underline{s}$ *was sent |* $\underline{y}$ *is observed)* of having been sent is the largest, given the observed signal vector $\underline{\boldsymbol{y}}$:

$$\underline{\hat{s}} = \arg_{\max} [\text{Prob}(\underline{s} \text{ was sent} \mid \underline{y} \text{ is observed})]$$

$$= \arg_{\max} \left[\frac{\text{Prob}(\underline{y} \text{ is observed} \mid \underline{s} \text{ was sent}) * \text{Prob}(\underline{s} \text{ was sent})}{\text{Prob}(\underline{y} \text{ is observed})} \right], \text{ for all possible } \underline{s}.$$

This equality is known as the Maximum A Posteriori Probability (MAP). If we further assume that the probability *Prob(*$\underline{s}$ *was sent)* is constant for all $\underline{s} \in M^m$, i.e., we assume equiprobability in the transmitted $\underline{s}$ vectors, then the MAP detection rule can be written as:

$$\underline{\hat{s}} = \arg_{\max} [\text{Prob}(\underline{y} \text{ is observed} \mid \underline{s} \text{ was sent})], \text{ for all possible } \underline{s}.$$

A detector that returns an optimal solution satisfying this equation is called a Maximum Likelihood (ML) detector. Under the assumption that the additive channel noise is white and Gaussian-distributed (i.e., AWGN), we can express the ML detection problem as that of minimizing the squared Euclidean distance metric to a target vector $\underline{\boldsymbol{y}}$ over an M^m-dimensional finite discrete search set:

$\hat{\underline{s}} = \mathbf{arg_{min}}\ ||\underline{y} - \underline{\mathbf{H}} * \underline{s}||$, for all possible $\underline{s}$.

The pseudo-code for an ML detector is shown in Algorithm 1. After computing the error metric for all possible symbol vectors (Line 6), the detected symbol vector is the one with the minimum error metric (Line 11).

Algorithm 1 ML Detection Algorithm

---

1. **for** (every received symbol vector $\underline{y}$ in a block) **do**
2. **for** ($a_1 = 1$; $a_1 = a_1 + 1$; $a_1 < M + 1$) **do**
3. **for** ($a_2 = 1$; $a_2 = a_2 + 1$; $a_2 < M + 1$) **do**
4. **for** ($a_3 = 1$; $a_3 = a_3 + 1$; $a_3 < M + 1$) **do**

…

5. **for** ($a_m = 1$; $a_m = a_m + 1$; $a_m < M + 1$) **do**
6. CandidateError(s_{a1},…, s_{am}) = $\text{norm}_2(\underline{y} - \underline{\mathbf{H}} * [s_{a1};\ldots; s_{am}])$;
7. **end for**

…

8. **end for**
9. **end for**
10. **end for**
11. $\hat{\underline{s}}_{ML} = \arg_{min}$ [CandidateError($\underline{s}$)], for all $\underline{s}$ in M^m
12. **output** $\hat{\underline{s}}_{ML}$;
13. **end for**

2.3.2 Minimum Mean Square Error (MMSE) Detector

Due to its very great computational complexity, the ML detector is not often used in practice. To reduce the computational cost and to simplify the detection process, MIMO detectors using a conditioning matrix have been developed. The idea is to design a conditioning matrix $\underline{\mathbf{G}}$ such that: $\underline{\hat{\boldsymbol{s}}} = \mathbf{Q}(\underline{\mathbf{G}} * \underline{\boldsymbol{y}})$, where $\underline{\boldsymbol{y}} = \underline{\mathbf{H}} * \underline{\mathbf{s}} + \underline{\boldsymbol{n}}$ is the signal vector and Q(.) is a slicing function that returns a vector $\underline{\hat{\boldsymbol{s}}}$ of estimated symbols such that, for each element of '$\underline{\mathbf{G}} * \underline{\boldsymbol{y}}$' the corresponding element in $\underline{\hat{\boldsymbol{s}}}$ is the nearest (in Euclidean sense) constellation point. The Zero Forcing (ZF) conditioning matrix $\underline{\mathbf{G}}_{\mathbf{ZF}}$ aims to zero-out the inter-symbol interference (ISI) by setting $\underline{\hat{\boldsymbol{s}}}_{\mathbf{ZF}} = \mathbf{Q}(\underline{\mathbf{G}}_{\mathbf{ZF}} * \underline{\boldsymbol{y}})$, for a given $\underline{\boldsymbol{y}}$, $\underline{\mathbf{s}}$ and $\underline{\mathbf{H}}$. Note that ZF does not exploit knowledge of random additive noise $\underline{\boldsymbol{n}}$ in the signal. By contrast, the Minimum Mean square Error (MMSE) conditioning matrix is designed so that the expected error between $\underline{\hat{\boldsymbol{s}}}$ and $\underline{\mathbf{s}}$ satisfies the Minimum Mean Square Error criterion given that the noise $\underline{\boldsymbol{n}}$ is Gaussian-distributed. In Appendix 1 the following two expressions are derived

$$\underline{\mathbf{G}}_{\mathbf{ZF}} = (\underline{\mathbf{H}}^{\mathbf{H}} * \underline{\mathbf{H}})^{-1} * \underline{\mathbf{H}}^{\mathbf{H}}$$

$$\underline{\mathbf{G}}_{\mathbf{MMSE}} = (\underline{\mathbf{H}}^{\mathbf{H}} * \underline{\mathbf{H}} + (1 / \mathrm{SNR}) * \mathbf{I}_{\mathbf{m}})^{-1} * \underline{\mathbf{H}}^{\mathbf{H}}$$

where $\underline{\mathbf{H}}^{\mathbf{H}}$ is the Hermitian of $\underline{\mathbf{H}}$, i.e., the conjugate transpose of $\underline{\mathbf{H}}$. The SNR that is required in $\underline{\mathbf{G}}_{\mathbf{MMSE}}$ can be estimated using training symbols that are inserted at known positions among the data-carrying symbol sequence, or by using so-called blind noise statistics estimation techniques [16, 17]. Note that $\underline{\mathbf{G}}_{\mathbf{MMSE}}$ converges on $\underline{\mathbf{G}}_{\mathbf{ZF}}$ as SNR tends to infinity.

Given an accurate estimate of the SNR, MMSE detectors perform better than ZF detectors. Intuitively, ZF detectors tend to over-react to any additive channel noise, whereas MMSE detectors are optimized to minimize on average the effects of noise [18].

The pseudo-code of the MMSE detector is shown in Algorithm 2. After computing the MMSE output $\underline{\boldsymbol{y}}_{\text{MMSE}}$ from each receiver antenna (Line 3), the detector picks the closest symbol from the constellation with respect to the Euclidean distance (Lines 4 & 5; here $\mathbf{1}_{1,M}$ denotes a row vector containing M 1*s*, $\mathbf{1}_{m,1}$ denotes a column vector containing m 1*s*, and **ConstellationSymbolMatrix** is the row vector containing the signal complex alphabet). $\underline{\boldsymbol{y}}_{\text{MMSE}} * \mathbf{1}_{1,M}$ provides an m-by-M matrix whose i-th row contains a replication of the estimated position of the symbol transmitted on the i-th layer. $\mathbf{1}_{m,1} *$ **ConstellationSymbolMatrix** provides an m-by-M matrix whose i-th column contains a copy of the i-th symbol from the signal alphabet. Therefore, **Distance** is a matrix whose rows contain the distances of the estimated position of the transmitted symbol to each constellation symbol, and $\arg_{\min}$(**Distance**) returns the closest constellation symbol to the estimated position provided by the element of $\underline{\boldsymbol{y}}_{\text{MMSE}}$.

Algorithm 2 MMSE Detection Algorithm

---

1. $\underline{\mathbf{G}} = (\underline{\mathbf{H}}^{\mathbf{H}} * \underline{\mathbf{H}} + (1 / \text{SNR}) * \mathbf{I}_m)^{-1} * \underline{\mathbf{H}}^{\mathbf{H}}$;
2. **for** (every received symbol vector $\underline{\mathbf{y}}$ in a block) **do**
3. $\quad \underline{\boldsymbol{y}}_{\text{MMSE}} = \underline{\mathbf{G}} * \underline{\boldsymbol{y}}$; { *conditioning* }
4. $\quad$ **Distance** = $\left| \underline{\boldsymbol{y}}_{\text{MMSE}} * \mathbf{1}_{1,M} - \mathbf{1}_{m,1} *$ **ConstellationSymbolMatrix** $\right|$;

5. $[s_1;s_2\ldots;s_m] = \arg_{\min}(\textbf{Distance})$; { *slicing* }

6. **output** $\underline{\hat{\boldsymbol{s}}}_{\text{MMSE}} = [s_1;s_2\ldots;s_m]$;

7. **end for**

2.3.3 Vertical Bell Laboratories Layered Space Time (V-BLAST) Detector [19, 20]

The vertical Bell Laboratories layered space-time (V-BLAST) algorithm is a relatively low-complexity detection algorithm for the practical implementation of MIMO receivers. Its BER vs. SNR performance lies between that of ML and MMSE (See Figure 9).

The V-BLAST algorithm detects each symbol iteratively by using a serial decision feedback approach. The key idea in V-BLAST is to first detect the most powerful layer, i.e., the layer exhibiting the largest post-detection SNR, which is the layer corresponding to the column of $\underline{\mathbf{H}}$ which has the largest norm [20]. Detection of the first symbol exploits a linear equalizer, such as ZF or MMSE, which minimizes the expected interference from the other undetected symbols. We will assume that MMSE, which is more accurate in the presence of AWGN on average than ZF, is used to detect the first symbol. V-BLAST then regenerates the received signals given the channel matrix $\underline{\mathbf{H}}$ and after having subtracted away the additive interference produced by the first detected symbol. It then proceeds with the detection of the second most powerful, transmitted symbol since it has already removed the effects of the first symbol, and so forth. Note that the channel

matrix **H** and the corresponding MMSE conditioning matrix **G** must be deflated (reduced by one in size, in one dimension) after each detection iteration to reflect the disappearance of each detected symbol. The resulting vector should contain less interference for the yet-to-be-detected symbols. Without loss of generality, let s_1 denote the symbol with maximum strength (i.e., the symbol transmitted through the layer experiencing the largest post-detection SNR). Similarly, s_2 will denote the symbol with the second largest strength, etc. Thus the weaker symbols are detected only after having subtracted away the interference contributions due to the more powerful symbols (s_1, s_2, ...). Unfortunately, a weakness in V-BLAST is that an error in the detection of any symbol will amplify the interference noise and likely propagate to detection errors in subsequent symbols, and this cascade of errors will degrade the performance of the detector.

After ordering the layers according to their estimated strength (i.e., estimated post-detection signal-to-noise ratio), the V-BLAST detection scheme proceeds in three steps at each iteration (except the first one, which does not require an interference nulling step, and the last one, which does not require a symbol cancellation step). For the *j*-th iteration:

(Step 1) *Nulling*: Vector $\mathbf{y}_j$ contains interference from the still undetected symbols s_{j+1},..., s_m. However, this interference can be minimized by multiplying $\mathbf{y}_j$ by the *nulling vector* $\mathbf{q}_j$, which is the *j*-th row of **G** (i.e., the MMSE conditioning matrix corresponding to the deflated channel matrix **H** corresponding to the *j*-th iteration).

(Step 2) *Slicing*: Symbol $\underline{s}_j$ is detected by selecting the symbol $\underline{s}'$ that minimizes the complex scalar difference $\| \boldsymbol{g}_j * \boldsymbol{y}_j - \underline{s}_j \|$ over all M possible symbols $\underline{s}_j$ in the constellation.

(Step 3) *Cancellation*: Vector $\underline{\boldsymbol{y}}_{j+1}$ is computed by subtracting the predicted interference $\underline{\mathbf{H}} * [\underline{s}_1, \underline{s}_2, ..., \underline{s}_j, 0, ..., 0]$ from $\underline{\boldsymbol{y}}$.

The pseudo-code of the MMSE-V-BLAST detector is shown in Algorithm 3. As previously mentioned, the detector iteratively detects the layers according to their strength. This is equivalent to sorting the columns of $\underline{\mathbf{H}}$ with respect to their norm, i.e., the column which has the largest norm corresponds to the layer with the largest post-detection SNR. Similarly, as shown on line 5, the layers can be ordered by sorting the rows of $\underline{\mathbf{G}}$ ($\underline{\boldsymbol{g}}_j$, where $j = 1 \ldots (m - t)$, and t is the number of layers already detected), i.e., the layer with the largest post-detection SNR corresponds to the row of $\underline{\mathbf{G}}$ which has the smallest norm.

Algorithm 3 MMSE V-BLAST Detection Algorithm

---

1. $\underline{\hat{\mathbf{H}}} = \underline{\mathbf{H}}$;
2. **for** (every received symbol vector $\underline{\boldsymbol{y}}$ in a block) **do**
3. **for** ($i = 1$; $i = i + 1$; $i < m + 1$) **do**
4. $\underline{\mathbf{G}} = (\underline{\hat{\mathbf{H}}}^{\mathbf{H}} * \underline{\hat{\mathbf{H}}} + (1 / \text{SNR}) * \mathbf{I}_{(m - i + 1)})^{-1} * \underline{\hat{\mathbf{H}}}^{\mathbf{H}}$;
5. $O(i) = k = \min_j \| \underline{\boldsymbol{g}}_j \|^2$; { Ordering }
6. $\underline{\boldsymbol{g}}_i = \underline{\mathbf{G}}(k,:)$; { Extract nulling vector from $\underline{\mathbf{G}}$ }
7. $\hat{\underline{s}}_k = \underline{\boldsymbol{g}}_k * \underline{\boldsymbol{y}}$; { Nulling }
8. $\underline{s}_k = Q(\hat{\underline{s}}_k)$; { Slicing }

9. **if** ($i < m$) **then**

10. $\underline{\boldsymbol{y}} = \underline{\boldsymbol{y}} - \underline{\boldsymbol{h}}_k * \underline{s}_k$; { Cancellation }

11. **end if**

12. $\underline{\hat{\mathbf{H}}} = \underline{\hat{\mathbf{H}}} \setminus \underline{\boldsymbol{h}}_k$; { Deflation }

13. **end for**

14. **output** $\underline{\hat{\boldsymbol{s}}}_{\text{VBLAST}} = [s_1; s_2 \ldots; s_m]$;

15. **end for**

2.4 Fouladi Fard's Parallel Detection Scheme

Due to its high complexity, ML detection is impractical for real systems. Thus, researchers have investigated many sub-optimal but much more economical and hence practical MIMO detectors, such as ZF, MMSE and V-BLAST. However, simulations readily show that the V-BLAST detector provides far from optimal performance although its performance exceeds that of the ZF and MMSE detectors. The weakness in V-BLAST is that the first symbol detected does not benefit from interference cancellation. Also, at the symbol cancellation step, detections errors can occur and these errors enhance the apparent subsequent interference and thus cause detection errors for the following symbols.

Fouladi Fard's detection scheme (which actually rediscovered the Parallel Detector scheme described in 2002 by Yuan Li and Zhi-Quan Luo [21]) is based on the insight that the performance of the V-BLAST detector is limited by the detection of the strongest layer [22]. Detecting the strongest layer can be made more reliable by applying computation to speculatively subtract away interference from one of the other layers. By making this one layer the weakest layer, one can improve the joint detection of the strongest and the weakest symbols, and subsequent improve the detection of all other layers. To improve the estimate of the strongest symbol, the new algorithm starts with the weakest layer and exhaustively considers all possible candidate weakest transmitted symbol values from the constellation. For each hypothesized first symbol for the weakest layer, conventional V-BLAST is then applied to detect the remaining $m - 1$ symbols. The detection of the strongest layer should then experience less interference from

the weakest layer for the case where the correct weakest symbol has been chosen. Thus the strongest and weakest symbols are detected jointly. We will refer to Fouladi Fard's detection scheme as F-BLAST.

The pseudo-code of F-BLAST is shown in Algorithm 4. The algorithm cancels the contribution of a tentative candidate symbol $\underline{s}_k^j$ from the weakest layer k of the noisy received signal $\underline{\boldsymbol{y}}$, where $j = 1, \ldots, M$, and M is the cardinality of the constellation (Line 5 to 7). The remaining layers are detected according to the original V-BLAST scheme (Line 8). Then, an error metric $\varepsilon_j = ||\underline{\mathbf{H}} * \underline{\boldsymbol{s}}_j - \underline{\boldsymbol{y}}||^2$ for the tentative symbol vector $\underline{\boldsymbol{s}}^j$ is computed, where $\underline{\boldsymbol{s}}^j = [\underline{s}_1^j ; \underline{s}_2^j ; \ldots ; \underline{s}_m^j]$ is the detected symbol vector. After proceeding for all tentative weakest symbol candidates in the constellation, the detector picks the one symbol vector $\underline{\hat{\boldsymbol{s}}}$ with the smallest error metric ε_j.

In terms of complexity, V-BLAST requires m nulling steps (i.e., vector multiplications), m slicing steps (i.e., symbol comparisons), and $m - 1$ cancellation steps (i.e., symbol vector multiplications and vector subtractions), to detect every transmitted symbol vector. The computational complexity of F-BLAST is increased by roughly M compared to V-BLAST as each of the M sub-detectors requires one fewer nulling and slicing operation for the worst symbol. The numbers of nulling, slicing and cancellation steps used in V-BLAST are thus increased by M, some of which are shared (a detailed analysis is provided in Section IV). It is important to note that the M sub-detectors in the proposed scheme can operate independently and, therefore, an M-fold parallel implementation of sub-detectors provides the same symbol detection throughput

as in the V-BLAST technique. A tree-structured output circuit can rapidly select the symbol vector with the least error ε_j. For a more compact implementation, one could implement only one instance of a sub-detector and then time multiplex it among other *M - 1* sub-detectors at the expense of lowering the symbol detection throughput.

Algorithm 4 F-BLAST Detection Algorithm

---

1. $\underline{\mathbf{G}} = (\underline{\mathbf{H}}^{H} * \underline{\mathbf{H}} + (1 / SNR) * \mathbf{I}_{(m - i + 1)})^{-1} * \underline{\mathbf{H}}^{H}$;
2. ε_{Best} = *LargeNumber*;
3. **for** (every received symbol vector $\underline{\boldsymbol{y}}$ in a block) **do**
4. O(1) = $k = \max_j ||\underline{\boldsymbol{g}_j}||^2$; { Ordering }
5. **for** (every symbol from the constellation) **do**
6. $\underline{s_k}$ = *CurrentConstellationSymbol*;
7. $\underline{\boldsymbol{y}} = \underline{\boldsymbol{y}} - \underline{\boldsymbol{h}_k} * \underline{\boldsymbol{s}_k}$; { Cancellation }
8. *Original V-BLAST with MMSE equalizer on the m - 1 remaining layers*
9. ***CurrentCandidateSymbolVector*** = $[s_1; s_2 \ldots; s_m]$;
10. $\varepsilon_j = ||\underline{\mathbf{H}} * \underline{\boldsymbol{CurrentCandidateSymbolVector}} - \underline{\boldsymbol{y}}||^2$;
11. **if** ($\varepsilon_j < \varepsilon_{Best}$) **then**
12. ***BestCandidate*** = ***CurrentCandidateSymbolVector***;
13. $\varepsilon_{Best} = \varepsilon_j$;
14. **end if**
15. **end for**

16. **output** $\hat{\underline{s}}_{\text{FBLAST}} = \underline{\boldsymbol{BestCandidate}}$;

17. **end for**

Figure 7 shows the similarity and the parallelizable structure of F-BLAST compared to V-BLAST. On this figure, two successive transmitted symbol vectors y_I and y_{I+1} are being detected using V-BLAST and F-BLAST. Each layer is represented by a shaded square, whose brightness is relative to its strength (i.e., its SNR). Finally, among the *M* (cardinality of the constellation) candidate symbol vectors, the symbol vector detected by F-BLAST is highlighted.

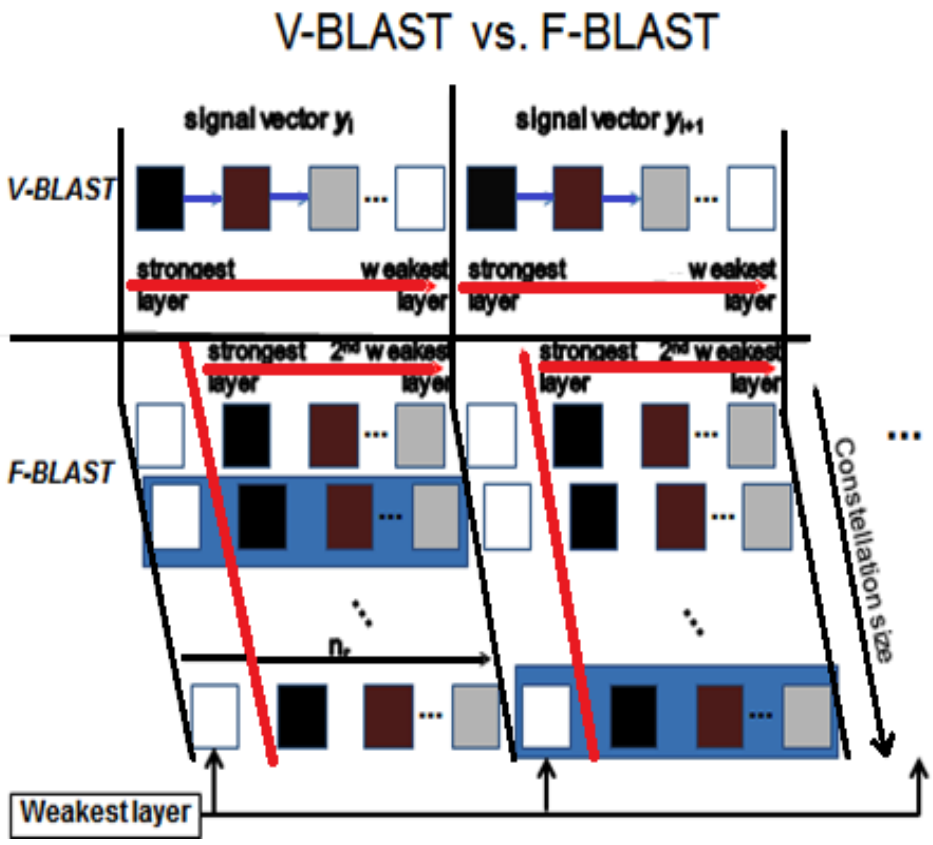


Figure 7 V-BLAST vs. F-BLAST [23]

Figure 8 shows the SER performance for three MIMO configurations (4×4 (a), 6×6 (b) and 8×8 (c)). When performing the exhaustive search on the weakest layer, the F-BLAST detector appears to achieve near-optimal performance. Here F(W*i*)-BLAST designates the F-BLAST detector that runs the exhaustive search on the *i*-th weakest layer.

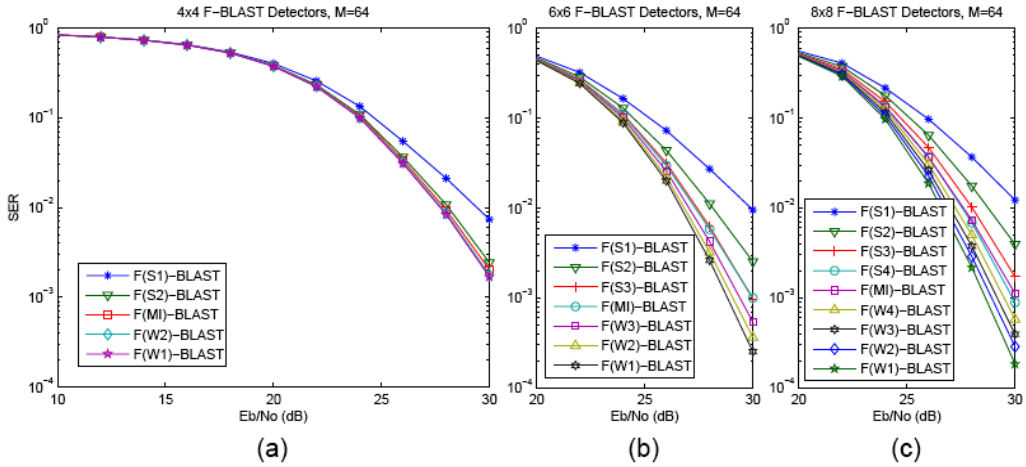


Figure 8 SER vs. SNR for F-BLAST for Different Parallel-search Layers and Increasing Numbers of Antennas [24]

Likewise, F(S*i*)-BLAST designates the F-BLAST detector which runs the exhaustive search on the *i*-th strongest layer. Note that F-BLAST tries to limit the error propagation by reducing the interference noise from the weakest layer, thus increasing the confidence on the important first decision made on the strongest layer. Given this motivation, we also tried to process the exhaustive search on the layer having the greatest interference on the strongest layer designated by F(MI)-BLAST. The performance of F(MI)-BLAST was not found to be as good, however, as F(W1)-BLAST, as illustrated in Figure 8.

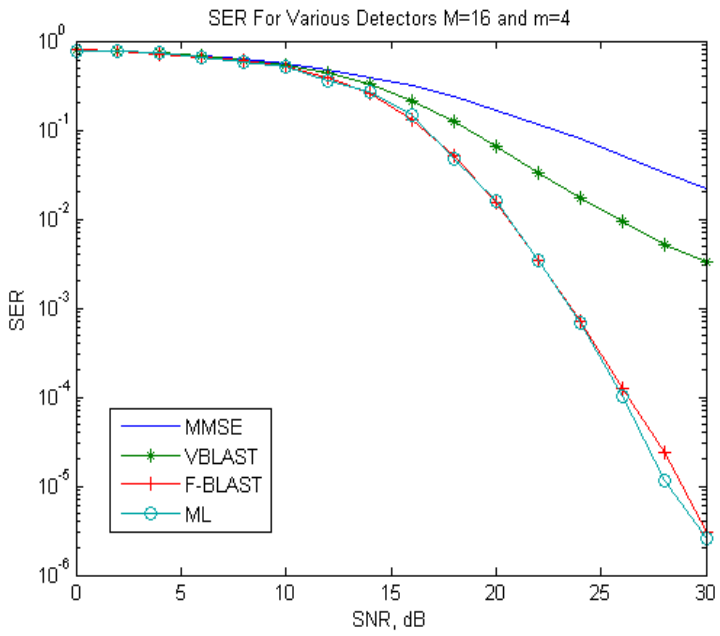


Figure 9 SER of Alternative Detection Schemes for a 4×4 16-QAM MIMO System over a Rayleigh Fading Channel

Figure 9 shows the simulated SER performance of the four detectors that have been reviewed in this section. Here F-BLAST denotes the same thing as F(W1)-BLAST, where the exhaustively searched layer is the weakest layer. Note the remarkable performance of F-BLAST when detecting 16-QAM symbols in a 4×4 MIMO system, which is equivalent to simultaneously running 16 3×4 V-BLAST detectors. The SER performance of F-BLAST very closely matches the optimal performance of ML. F-BLAST can be easily implemented for the practical MIMO detection of signals with small symbol constellations. However, for larger constellations, such as 64-QAM or 256-QAM, the exhaustive search parallelism of F-BLAST becomes increasingly impractical and, indeed, this drawback motivates the research reported in this thesis.

III- Proposed Detection Scheme

3.1 Key Ideas

As stated in the previous section, the linearly growing complexity of F-BLAST makes it impractical to detect MIMO signals with large constellations (i.e., greater than 16-QAM). Our goal was to keep key ideas from the F-BLAST scheme but to limit the parallelism, say, to a maximum of 16 or 32 so that large constellations such as 256-QAM could be detected. In addition, the simple parallelism in F-BLAST has potential advantages in Orthogonal Frequency-Division Multiplexing (OFDM) receivers, where a pool of hardware resources could be shared among the subcarriers. With this limit on the parallelism, it may become practical to use the MMSE equalizer and the Ordering and Successive Interference Cancellation (OSIC) method (e.g., BLAST) that is being used in the industry [25]. However, challenges remain in the optimal choice of the restricted search set and in the optimal ordering of the layers during detection.

3.1.1 Definition of the Restricted Search Set

The definition of the restricted search set will definitely impact the performance of the proposed detector. Ideally, we want the new detector's decisions to be identical to the decision that would be produced by F-BLAST detection. Intuitively, the larger that the searched window is within the full constellation, the closer to F-BLAST should be the error rate performance. Unfortunately a larger search space will require more computation, and therefore

more energy, which is a limited resource in battery-powered communication devices. For simplicity, the search window within a constellation is positioned around a constellation point that is more easily estimated. We used an MMSE equalizer to define the center of the window search because it gives better results than the ZF equalizer with only slightly more computation. Using a V-BLAST estimator to determine the search window center was not found to give significant benefits over MMSE given the additional computational cost. At present, there is no tractable theoretical basis for optimally constructing the search window. Therefore an empirical method was used, which corresponds roughly to constructing the search window of size W as the MMSE estimated symbol on the searched layer together with the W - 1 nearest symbols (in a Euclidean sense) in that layer. The precise shape of the search window for each symbol position $\underline{s}_X$ was optimized empirically by simulation experiments. Specifically, we collected histograms for the (assumed near-optimal) F-BLAST estimate given that the MMSE estimate was $\underline{s}_X$ for all possible values of $\underline{s}_X$. For each histogram, a search window of size W was constructed by selecting the W most likely F-BLAST decisions (that is, near-optimal decisions) for each s_X. The M windows were then stored in look-up tables (the number of tables can be reduced significantly by exploiting constellation symmetry). Figure 10 shows the ten unique search windows for W = 8 and 16 for M = 64. The number of windows has been reduced from 64 to ten in this figure by exploiting all possible symmetries (about the diagonals, the vertical and horizontal axis).

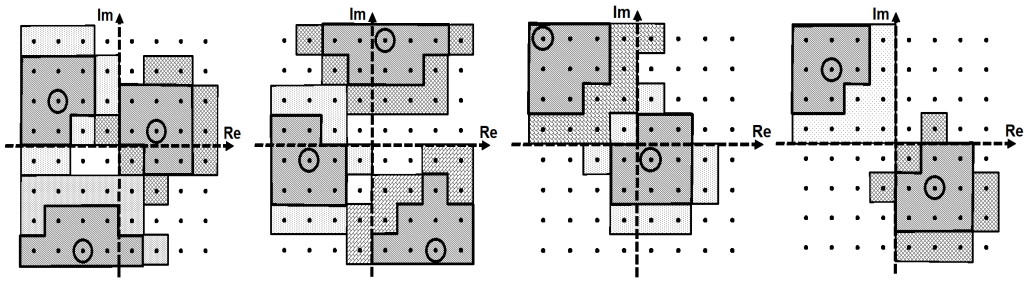


Figure 10 Search Windows of Size $W = 8$ (Darker Shading) and 16 (Darker and Lighter Shading) for the 64-QAM Constellation [24]

3.1.2 Layer Ordering

As for the F-BLAST detection scheme, the proposed detector first performs a parallel search within one chosen searched layer, and then the original V-BLAST detection scheme is applied to the remaining layers. While the V-BLAST scheme starts the detection on the strongest layer, the F-BLAST scheme starts the detection on the weakest layer (by guessing all possible symbol values in parallel) in an attempt to reduce the interference on the strongest layer. When the selected layer is not exhaustively searched, our simulations have shown that the best choice of layer is not necessarily the weakest. The best ordering method will be discussed in more detail in the next section.

3.2 Simulations and Results

To simplify the discussion, we will use **S*i*** to denote the *i*-th strongest layer, i.e., the layer corresponding to the column of **H** with the *i*-th strongest strength (the *i*-th largest norm among all columns of **H**). Similarly, we will use **W*i*** to denote the *i*-th weakest layer, i.e., the layer corresponding to the column of **H** with the *i*-th weakest strength (the *i*-th smallest norm among all columns of **H**). Finally, we will use **MI** to denote the layer having the greatest expected interference on the strongest layer, i.e., the layer with the largest coefficient $\underline{\boldsymbol{h}}_{ji}$ in *i*-th column of **H** corresponding to the strongest layer.

A simulation study was conducted using MATLAB implementations for several different MIMO system configurations (m = 4, 6 or 8), while detecting several large constellation signals (M-QAM with M = 64, 128 or 256). We used a block-based data partitioning for simulation afficiency. Typically, $2 * 10^4$ blocks are simulated together, with each block containing 10 frames each, for accuracy of the average number of binary digits in error. To reduce the calculation effort, the channel matrix was set to be constant for the duration of a frame. The frame size was set to ensure accurate modelling of Rayleigh fading. Specifically, $2 * M$ sample vectors were transmitted during each frame. To ensure statistical accuracy in the error measurements, a minimum of 10,000 symbols in error were simulated.

In order to pick the best starting layer, a simulation study of the family of new detectors was conducted. The simulation results are shown in Figure 11. Here FR(W*i*,*W*)-BLAST designates the FR-BLAST detector which runs W parallel

searches in layer **W*i***. Likewise FR(S*i*,*W*)-BLAST designates the FR-BLAST detector which runs *W* parallel

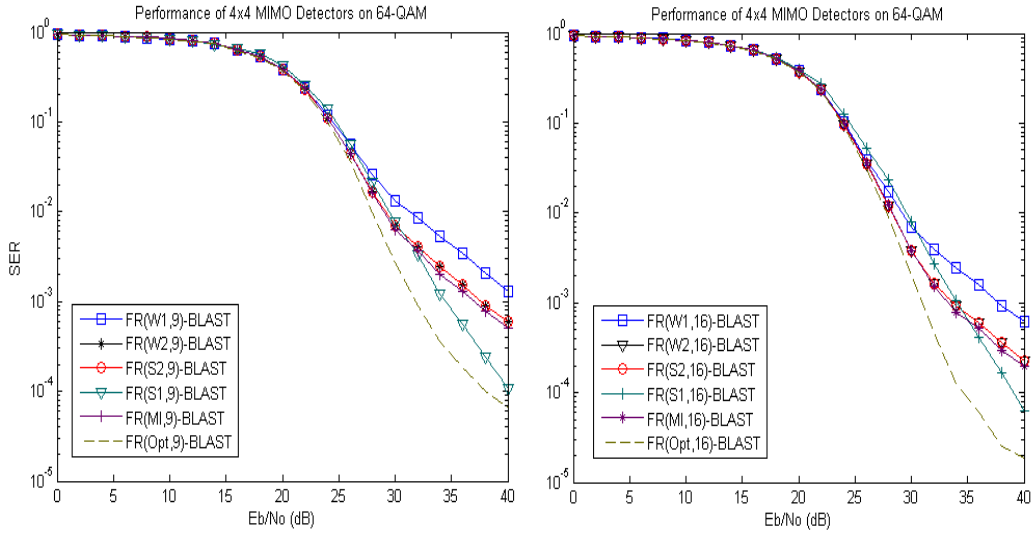


Figure 11 SER vs. SNR for FR-BLAST of Various Reduced Search Windows

searches in layer **S*i***, and FR(MI,*W*)-BLAST designates the FR-BLAST detector which runs *W* parallel searches in layer **MI**. We observe that the relative performance of the various detectors depends on the SNR. Note that at high SNR the best starting layer is **S1** whereas for intermediate SNR values both **S2** and **MI** outperform the other layer choices. FR(Opt,*W*)-BLAST refers to the optimal detector that uses a window size *W*, i.e., the detector which picks the one best layer for each frame of sample vectors. This detection algorithm produces the best-case upper limit for FR-BLAST detectors that are limited to searching *W* parallel choices on a dynamically-chosen search layer. The decisions of FR(Opt,*W*)-BLAST were stored as a best-case reference for further study. Observe that FR(Opt,16)-BLAST clearly outperforms the performance of the other detectors for SNR values greater than 34 dB.

One question that arises is the possibility of optimally selecting the first layer to be detected for an improved FR-BLAST detector and thus hopefully approach the performance of FR(Opt, *W*)-BLAST. To be practical, the layer selection rule would have to be both simple and accurate. In order to answer this question, a statistical study was conducted. We used the stored decisions of the optimal FR-BLAST detector and analysed them using discriminant analysis routines from the Statistical Package for the Social Sciences (SPSS) [26]. The SPSS provides powerful routines for data clustering and discriminant analysis. The method and the results are presented in Appendix 2. Unfortunately, SPSS was unable to find an effective linear rule for selecting the parallel search layer. For picking the best starting layer, the optimal layer selector model proposed by SPSS only made the right decision about 25% of the time. This corresponds to randomly selecting that layer from among the four candidate layers.

The next phase of this work studied the performance of the detectors in an encoded scheme. The BER/SER performance was generally studied at low SNR, therefore we decided to focus our study of the FR-BLAST detection scheme while restricting the search to within only the **S2** or **MI** layers, i.e., the layers which show the best performance for SNR values less than 30 dB. Unfortunately, experimental results show that the **MI layer** is different from **S2** more than 2/3 of the time. Therefore, at this time, we once again found that there is no simple criterion for picking the best starting layer for FR-BLAST.

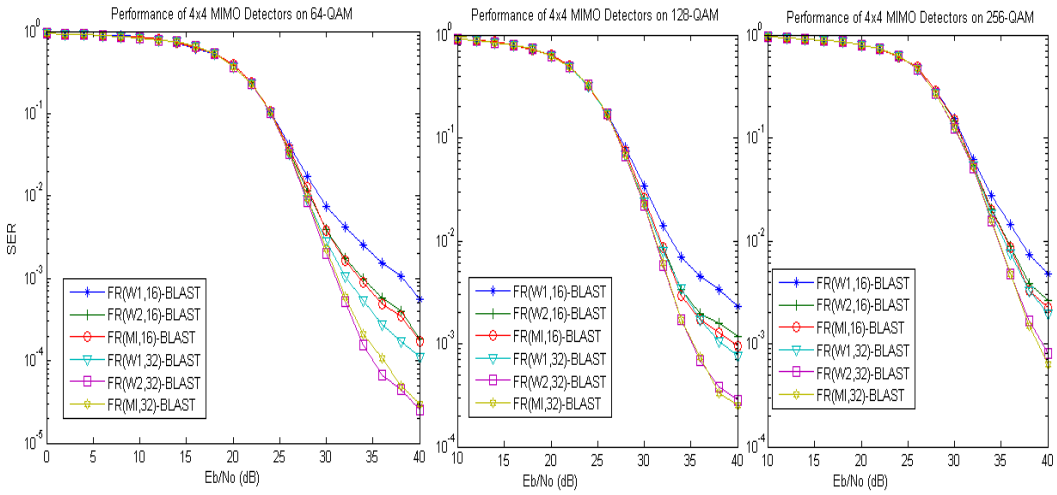


Figure 12 SER vs. SNR for FR-BLAST for Various Reduced Search Windows and Signal Constellations

Figure 12 summarizes the performance of the FR-BLAST scheme in a 4×4 MIMO configuration when detecting 64-, 128- and 256-QAM signals. It shows that the SER vs. SNR performance of FR-BLAST increases with the size of the search window, i.e., increasing the size of the window search will lower the SER. Further, the performance of FR-BLAST is still very good for larger constellation signals.

Figure 13 confirms that the performance characteristics of the new family detectors (FR-BLAST) are better than that of MMSE and V-BLAST for all SNR values and for three different constellations (64, 128 and 256). In addition, FR-BLAST experiences saturation in performance at the larger SNRs. That is for the larger SNRs, the slope of its performance characteristic (i.e., the diversity order) reduces from that of the near-optimal F-BLAST (-2) to that of V-BLAST, MMSE and ZF (-1). Thus FR-BLAST clearly has worse performance than that of near-optimal F-BLAST.

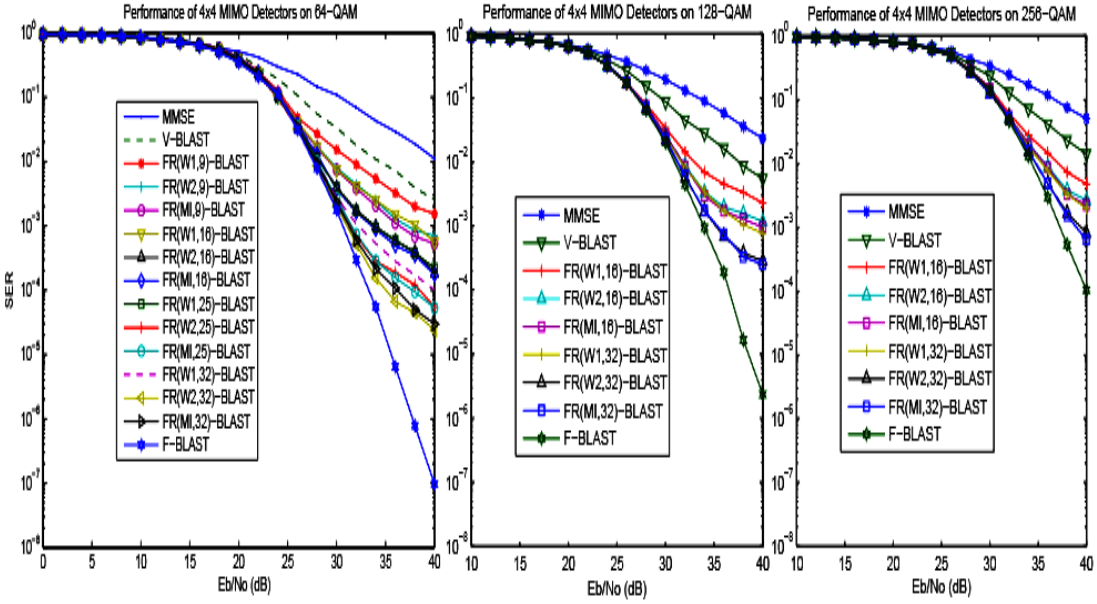


Figure 13 SER vs. SNR for MMSE, V-BLAST, FR-BLAST of Various Reduced Search Windows, and F-BLAST [24]

FR-BLAST is an interesting alternative detection scheme that achieves good performance in terms of error rate, relatively low implementation complexity, low computational complexity (derived in Chapter IV) and an attractive parallelizable structure. This could be attractive for MIMO detectors for small to moderate SNRs, offering performance that lies between V-BLAST and the near-optimal (but very expensive) F-BLAST and sphere-decoding-based detectors.

The pseudo-code of FR-BLAST is shown in Algorithm 5. Apart from the definition of the restricted search window (Line 5 and Line 6), the pseudo-code of FR-BLAST is very similar to that of F-BLAST.

Algorithm 5 FR-BLAST Detection Algorithm

---

1. $\underline{\mathbf{G}} = (\underline{\hat{\mathbf{H}}}^{H} * \underline{\hat{\mathbf{H}}} + (1 / SNR) * \mathbf{I}_{(m - i + 1)})^{-1} * \underline{\hat{\mathbf{H}}}^{H}$;

2. ε_{Best} = *LargeNumber*;

3. **for** (every received symbol vector $\underline{\boldsymbol{y}}$ in a block) **do**

4. O(1) = $k = \max_j \|\underline{\boldsymbol{g}}_j\|^2$; { Ordering }

5. $\underline{s}_X = \underline{\boldsymbol{g}}_k * \underline{\boldsymbol{y}}$; { Find center of the restricted search set }

6. $\underline{\boldsymbol{Subset}}$ = **Table**($\underline{s}_X$, *W*); { Constructing the search window from a look-up table}

7. **for** (every symbol from the restricted search set) **do**

8. $\underline{s}_k$ = *CurrentConstellationSymbol*;

9. $\underline{\boldsymbol{y}} = \underline{\boldsymbol{y}} - \underline{\boldsymbol{h}}_k * \underline{\boldsymbol{s}}_k$; { Cancellation }

9. *Original V-BLAST with MMSE equalizer on the m - 1 remaining layers*

10. **return** $\underline{\boldsymbol{CurrentCandidateSymbolVector}} = [\underline{s}_1;\underline{s}_2\ldots;\underline{s}_m]$;

11. $\varepsilon_j = \|\underline{\mathbf{H}} * \underline{\boldsymbol{CurrentCandidateSymbolVector}} - \underline{\boldsymbol{y}}\|^2$;

12. **if** ($\varepsilon_j < \varepsilon_{Best}$) **then**

13. $\underline{\boldsymbol{BestCandidate}} = \underline{\boldsymbol{CurrentCandidateSymbolVector}}$;

14. **end if**;

15. $\varepsilon_{Best} = \varepsilon_j$;

16. **end for**

17. **output** $\underline{\hat{\boldsymbol{s}}}_{\mathrm{FRBLAST}} = \underline{\boldsymbol{BestCandidate}}$;

18. **end for**

3.3 Alternative Detectors

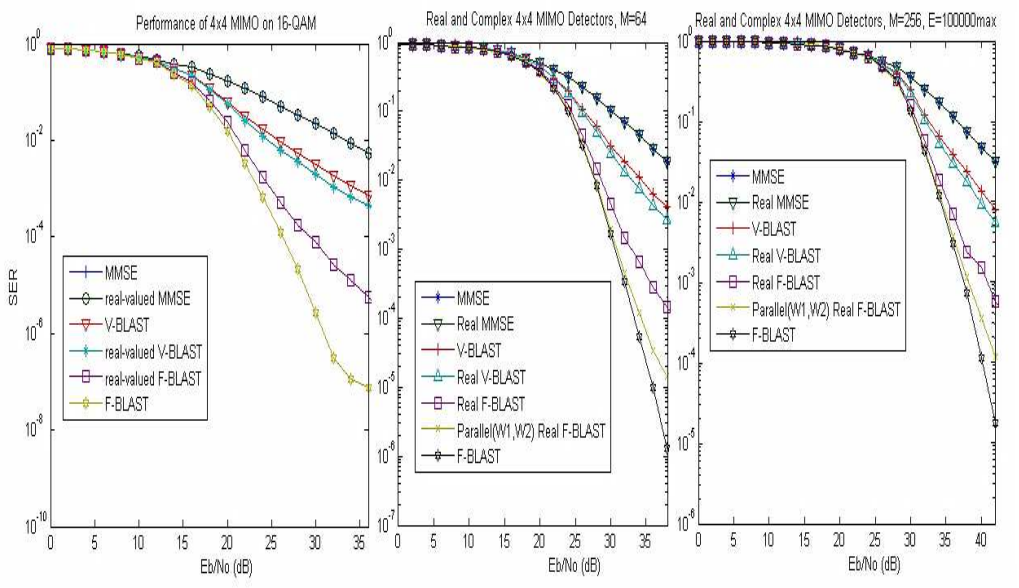


Figure 14 Performance of the Real-valued F-BLAST Detector

The detectors described in the previous section use complex-valued sampled signals, channel matrices and noise coefficients. In this section we will instead study detectors in a real-valued equivalent model. The real-valued detectors have an improved error rate performance compared to the traditional V-BLAST detection scheme. The improvement stems from the greater degrees of freedom afforded by having twice as many symbol layers that can be ordered more effectively [27].

Complex-valued detection uses the following equation to describe a transmission over a MIMO system: $\underline{\boldsymbol{y}} = \underline{\mathbf{H}} * \underline{\boldsymbol{s}} + \underline{\boldsymbol{n}}$. In contrast, the real-valued equivalent model is based on the following equation [27]:

$$\boldsymbol{y}^{R} = \mathbf{H}^{R} * \boldsymbol{s}^{R} + \boldsymbol{n}^{R},$$

where $\boldsymbol{y}^{R} = [\text{real}(\underline{\boldsymbol{y}}), \text{imag}(\underline{\boldsymbol{y}})]^{T}$,

$$\mathbf{H}^{R} = \begin{bmatrix} \text{real}(\underline{\mathbf{H}}) & -\text{imag}(\underline{\mathbf{H}}) \\ \text{imag}(\underline{\mathbf{H}}) & \text{real}(\underline{\mathbf{H}}) \end{bmatrix},$$

$$\boldsymbol{s}^{R} = [\text{real}(\underline{\boldsymbol{s}}), \text{imag}(\underline{\boldsymbol{s}})]^{T},$$

and $\boldsymbol{n}^{R} = [\text{real}(\underline{\boldsymbol{n}}), \text{imag}(\underline{\boldsymbol{n}})]^{T}$

Here the notations real($\underline{\boldsymbol{z}}$) and imag($\underline{\boldsymbol{z}}$) denote the real and the imaginary vector components of the complex-valued vector $\underline{\boldsymbol{z}}$.

For example, with $\underline{\boldsymbol{z}}$ = [1 + .5 * j, 5 + 3 * j], real($\underline{\boldsymbol{z}}$) = [1, 5] and imag($\underline{\boldsymbol{z}}$) = [.5, 3].

The real-valued equivalent model has effectively twice the number of antennas at each side of the radio link. This leads to a bigger channel matrix: the number of components in each dimension is doubled. Figure 14 compares the performance of the F-BLAST scheme while using the real-valued equivalent model compared with the complex-valued MMSE and the V-BLAST detectors. Note that the real-valued equivalent model uses only one dimension of the signal constellation, i.e., while doing a real-valued detection, the number of symbols is $\sqrt{M}$. Thus the real-valued F-BLAST detection scheme uses $\sqrt{M}$-fold parallelism instead of *M*. That is a significant reduction in the amount of parallelism for $M \geq 64$ and this flexibility could be used to make trade-offs at a circuit design level. An improved real-valued detector that uses 2 * $\sqrt{M}$ fold parallelism was also studied. This second real-valued MIMO detector simply performs the real-valued F(W1)-BLAST and the real-valued F(W2)-BLAST and picks the best output vector at the end, we will refer to it as Parallel(W1,W2) Real F-BLAST. This last detector benefits from selection diversity [28] and has improved BER performance.

IV- Computational Complexity Results

4.1 Assumptions

When comparing the cost of alternative algorithms, it is important to accurately count the number of elementary operations (i.e., real number additions, real number multiplications, and real number reciprocals) that are required to detect each received symbol. In order to derive the exact cost, in terms of the number of elementary operations for the various MIMO detectors considered in this project, let us first make some reasonable simplifying assumptions.

We will assume that memory allocation does not require any elementary operations. Sufficient memory is assumed to have been pre-allocated for the decoder calculations. Also, initializing a matrix, i.e., defining each initial element of a matrix, does not require any elementary operations. In general, all variables are assumed to be allocated statically at initialization and thus do not require any further run time.

In addition, we will assume that computing the negation of a real or complex number, i.e., a to $-a$, and computing the complex conjugation operation, i.e. a to a^*, do not require significant run time. In the same spirit, we will assume that computing the transpose of a matrix or the Hilbert transform of a matrix will not require any elementary operations. Any changes in sign can be merged into the next arithmetic operation without extra cost.

Also, the arithmetic division "*a/b*" will be assumed to be accomplished by multiplying '*a*' by the reciprocal of '*b*' so that the relatively expensive division operation is replaced by a (fast) multiplication and an optimized reciprocal.

As noted above, we will associate computational cost with three main elementary operations:

- real-valued addition or subtraction, denoted by '+'
- real-valued multiplication, denoted by '*'
- real-valued reciprocal of an argument *N*, denoted by '1/*N*'

All three operations are assumed to be performed in the real-valued domain. The vast majority of processors cannot directly handle complex arithmetic natively, so our computational complexity results will be presented in terms of real-valued operations.

Finally, we will consider the possibility of parallel operations that exploit the possible presence of parallel arithmetic hardware units. It can safely be assumed that maximum parallelism is now going to be affordable using the reconfigurable logic and arithmetic units of large field-programmable gate arrays (FPGAs). One operation cycle, which we will denote by '*c*', is the time, usually equal in the instruction set of a modern computer, to perform one addition or one subtraction. Frequently, modern computers will in fact be able to compute a multiplication in the same amount of time as an addition by using a combined multiplier-accumulator in the arithmetic data-path.

4.2 General Results for Basic Operations

The total number of elementary operations is a useful metric that is roughly proportional to the energy required by the computation. On the other hand, the minimum number of required consecutive operations (assuming maximum hardware parallelism) gives a measure of the best-case (i.e., minimum) time complexity. A real reciprocal function requires roughly four to ten cycles, depending on the algorithm and the required bit width, thus more than one parallel real addition or multiplication can be performed in parallel with one real reciprocal. However for simplicity we will assume that parallel real additions, real multiplications or real reciprocals all require only one cycle.

Let $f_i(x,y,z)$ and $g_i(x,y,z)$ denote the number of real-valued operations and operation cycles, respectively, associated with parameters x, y and z. Table 1 summarizes the number of elementary operations and operation cycles required for basic operations that are used in the detection algorithms. A detailed derivation can be found in Appendix 3.

Table 1 General Complexity Results

***i*. Operation**	$\mathbf{f}_i$			$\mathbf{g}_i$
	Real Additions	**Real Multiplications**	**Real Reciprocals**	**Minimum Cycles**
1. Real number addition	1	0	0	1
2. Real number multiplication	0	1	0	1
3. Real number reciprocal	0	0	1	1
4. Square absolute value of real number	0	1	0	1
5. *m*-by-*n* real-valued matrix addition	$m*n$	0	0	1
6. Addition of a set containing *n* real numbers	n-1	0	0	$\lceil \log_2 n \rceil$
7. Minimum of a set containing *n* real numbers	n-1	0	0	$\lceil \log_2 n \rceil$
8. Square norm of a real-valued column vector of length *n*	n-1	n	0	1+$\lceil \log_2 n \rceil$
9. Multiplication of an *m*-by-*p* and a *p*-by-*n* real-valued matrices	$m*n*(p-1)$	$m*n*p$	0	1+$\lceil \log_2 p \rceil$
10. Inverse of an *n*-by-*n* real-valued matrix	$n^2(n-1)$	$n^2(n-1)$	n^2	$3n$
11. MMSE conditioning matrix from a 2*n*-by-2*n* real-valued channel matrix	$n[8n^2-n-1]$	$4n^2(2n+1)$	n^2+1	6+7n+2* $\lceil \log_2 n \rceil$
12. MMSE conditioning matrix from a *m*-by-*n* real-valued deflated channel matrix	$m(m-1)(n+m)$	$m^2(n+m-1)$	m^2	3m+1+ $\lceil \log_2 m \rceil$
13. Complex number addition	2	0	0	1
14. Complex number multiplication	2	4	0	2
15. Complex number reciprocal	1	4	1	4
16. Square absolute value of a complex number	1	2	0	2
17. *m*-by-*n* complex-valued matrix addition	$2m*n$	0	0	1
18. Addition of a set containing *n* complex numbers	$2n$-2	0	0	$\lceil \log_2 n \rceil$
19. Square norm of a complex-valued column vector of length *n*	$2n$-1	$2n$	0	2+$\lceil \log_2 n \rceil$
20. Multiplication of an *m*-by-*p* complex-valued matrix by a *p*-by-*n* complex-valued matrix	$m*n(4p-2)$	$4m*n*p$	0	2+$\lceil \log_2 p \rceil$

21. Inverse of an *n*-by-*n* complex-valued matrix	$n^2(4n-3)$	$4n^3$	n^2	$7n$
22. MMSE conditioning matrix from an *n*-by-*n* complex-valued channel matrix	$n[8n^2-n-1]$	$4n^2(2n+1)$	n^2+1	$6+7n+2*\lceil \log_2 n \rceil$
23. MMSE conditioning matrix from an *m*-by-*n* complex-valued deflated channel matrix	$n(4m^2-3m+4mn-2n)$	$4m^2(m+n)$	m^2	$7m+2+\lceil \log_2 m \rceil$

4.3 Computational Complexity of the Detectors

4.3.1 Real-valued Detection

The real-valued equivalent model has twice as many antennas, i.e., 2*m* instead of *m*, and the constellation size is equal to $\sqrt{M}$ along one dimension instead of *M* across two dimensions. Typical values of *m* are 2, 3 and 4, while typical values of *M* are 4, 16 and 64.

Complexity Result 1 MMSE detection on 2*m* real-equivalent layers requires:

(a) $m^2+1 \approx m^2$ **real reciprocals**

(b) $8m^3+8m^2+2m\sqrt{M} \approx 8m^3+2m\sqrt{M}$ **real multiplications**

(c) $8m^3+3m^2-4m+2m\sqrt{M} \approx 8m^3+2m\sqrt{M}$ **real additions**

(d) $7m+10+\lceil \log_2 m \rceil+\lceil \log_2 \sqrt{M} \rceil \approx 7m+\lceil \log_2 \sqrt{M} \rceil$ **parallel cycles**

Proof: The number of arithmetic operations contributed by each line is as follows:

Line 1

Computing the MMSE conditioning matrix will require $f_{11}(2m)$ operations and $g_{11}(2m)$ cycles. With $f_i(2m)$ representing the number of additions and multiplications for the *i*-th basic operation as presented in table 1; likewise $g_i(2m)$ represents the number of minimum cycles for the *i*-th basic operation.

Line 3

With (*m*, *p*, *n*) = (2*m*, 2*m*, 1), computing **G** * ***y*** requires $f_9(2m,1,2m)$ operations and $g_9(2m,1,2m)$ cycles.

Line 4

$\boldsymbol{y}_{\text{mmse}}$ * $\mathbf{1}_{1,M}$ – $\mathbf{1}_{m,1}$ * **ConstellationSymbolMatrix**, can be seen as memory allocation, so this calculation requires no operations. With (m, n) = ($2m$, $\sqrt{M}$), computing $\boldsymbol{y}_{\text{mmse}}$ * $\mathbf{1}_{1,M}$ – $\mathbf{1}_{m,1}$ * **ConstellationSymbolMatrix** requires $f_5(2m, \sqrt{M})$ operations and $g_5(2m, \sqrt{M})$ cycles. To compute the absolute square norm of $2m\sqrt{M}$ real-valued numbers, we require $2m\sqrt{M}$ * f_4 operations and one cycle.

Line 5

We have to compute the minimum of $2m$ sets of real numbers of length $\sqrt{M}$ each, thus, $2m$ * $f_7(\sqrt{M})$ operations and $g_7(\sqrt{M})$ cycles are required.

Q.E.D.

Complexity Result 2 V-BLAST detection on $2m$ - 1 real-equivalent remaining layers requires:

(a) $1/3(8m^3-6m^2+m) \approx 8/3m^3$ real reciprocals

(b) $1/3(28m^4-32m^3+17m^2-17m)+(2m-1)\sqrt{M} \approx 28/3m^4+2m\sqrt{M}$ real multiplications

(c) $1/3(28m^4-28m^3+17m^2-35m+4)+(4m-2)\sqrt{M} \approx 28/3m^4+4m\sqrt{M}$ real additions

(d) $6m^2+13m-11+(4m-3)\lceil \log_2 m \rceil+(2m-1)\left\lceil \log_2 \sqrt{M} \right\rceil+2\sum_{d=1}^{2m-1}\lceil \log_2 d \rceil \approx 6m^2+2m\left\lceil \log_2 \sqrt{M} \right\rceil$ parallel cycles

Note: Here, we assumed that the detection on the first layer has already been performed.

Proof: The number of arithmetic operations contributed by each line is as follows:

On the 2*m*-1 remaining layers, **H** is considered to be an 2*m*-by-*d* complex matrix, where 2*m*-*d* is the number of layers already deflated.

Line 4

With (*n*, *m*) = (2*m*, *d*), the computation of **G** requires $f_{12}(2m,d)$ operations and $g_{12}(2m,d)$ cycles.

Line 5

In order to proceed with the right layer, we need to compute the minimum of a set of *d* real numbers; this requires $f_7(d)$ operations and $g_7(d)$ cycles. Except on the last layer to be detected, this step is omitted.

Line 7

With (*m*, *n*, *p*) = (1, 1, 2*m*), nulling will require $f_9(1,1,2m)$ operations and $g_9(1,1,2m)$ cycles.

Line 8

In order to pick the right symbol from the constellation. First we need to compute the distance from each constellation symbol, i.e., with (*m*, *n*) =($\sqrt{M}$, 1), $f_5(\sqrt{M},1)$ operations and $g_5(\sqrt{M},1)$ cycles are required. Then we need to compute $\sqrt{M}$ square absolute value, requiring, $\sqrt{M}$ * f_4 operations and g_4 cycles. Finally, we pick the minimum of a set of $\sqrt{M}$ real numbers. This requires $f_7(\sqrt{M})$ operations and $g_7(\sqrt{M})$ cycles.

Line 10

With (*m*, *n*, *p*) = (2*m*, 1, 1), computing $\boldsymbol{h}_k$ * $\boldsymbol{s}_k$ requires $f_9(2m,1,1)$ operations and $g_9(2m,1,1)$ cycles. Now, computing $\boldsymbol{y} - \boldsymbol{h}_k * \boldsymbol{s}_k$ requires $f_5(2m,1)$ operations and $g_5(2m,1)$ cycles. However on the last layer to be

detected, the ordering of the layers is not required since there is only one layer.

Line 12

No operations nor time in cycles are required to deflate the channel matrix **H**.

Q.E.D.

Complexity Result 3 V-BLAST detection on 2*m* real-equivalent layers requires:

(a) $1/3(8m^3-3m^2+m+3) \approx 8/3m^3$ **real reciprocals**

(b) $1/3(28m^4-8m^3+41m^2-5m)+(2m+2)\sqrt{M} \approx 28/3m^4+2m\sqrt{M}$ **real multiplications**

(c) $1/3(28m^4-4m^3+26m^2-35m-5)+(4m-2)\sqrt{M} \approx 28/3m^4+4m\sqrt{M}$ **real additions**

(d) $6m^2+20m+2+(4m+2)\lceil \log_2 m \rceil+2m\left\lceil \log_2 \sqrt{M} \right\rceil+2\sum_{d=1}^{2m-1}\lceil \log_2 d \rceil \approx 6m^2+2m\left\lceil \log_2 \sqrt{M} \right\rceil$ **parallel cycles**

Proof: The number of arithmetic operations contributed by each line is as follows:

Line 5

Ordering requires the computation of 2*m* times 1-by-2*m* real-valued vector norms, thus 2*m* * $f_8(2m)$ operations and $g_8(2m)$ cycles are required.

Line 3 to Line 13

On the first layer to be detected,

Line 4

The computation of **G** requires $f_{11}(2m)$ operations and $g_{11}(2m)$ cycles.

Line 5

In order to proceed on the right layer, we need to compute the minimum of a set of $2m$ real numbers. This requires $f_7(2m)$ operations and $g_7(2m)$ cycles.

Line 7

With $(m, n, p) = (1, 1, 2m)$, nulling will require $f_9(1,1,2m)$ operations and $g_9(1,1,2m)$ cycles.

Line 8

In order to pick the right symbol from the constellation, first we need to compute the distance from each constellation symbol, i.e., with $(m, n) = (\sqrt{M}, 1)$, $f_5(\sqrt{M},1)$ operations and $g_5(\sqrt{M},1)$ cycles are required. Then we need to compute $\sqrt{M}$ square absolute values i.e., $\sqrt{M}$ * f_4 operations and g_4 cycles. Finally, we pick the minimum of a set of $\sqrt{M}$ real number; this requires $f_7(\sqrt{M})$ operations and $g_7(\sqrt{M})$ cycles.

Line 10

With $(m, n, p) = (2m, 1, 1)$, computing $\boldsymbol{h_k} * \boldsymbol{s_k}$ requires $f_9(2m,1,1)$ operations and $g_9(2m,1,1)$ cycles. Computing $\boldsymbol{y} - \boldsymbol{h_k} * \boldsymbol{s_k}$ requires $f_5(2m,1)$ operations and $g_5(2m,1)$ cycles.

Finally, the detection of the first layer will require m^2+1 ($1/N$), $8m^3+8m^2+4m+\sqrt{M}$ (*), $8m^3+3m^2+3m+2\sqrt{M}-3$ (+), and $13+7m+\lceil \log_2 m \rceil+\left\lfloor \log_2 \sqrt{M} \right\rfloor$ (c). On the $2m-1$ remaining layers, the complexity is given by **Complexity Result 2**.

Q.E.D.

Complexity Result 4 F-BLAST detection on 2*m* real-equivalent layers requires:

(a) $m^2+1+\sqrt{M}/3(8m^3-6m^2+m) \approx 8/3m^3\sqrt{M}$ **real reciprocals**

(b) $8m^3+8m^2+\sqrt{M}/3(28m^4-32m^3+29m^2-5m)+(2m-1)M \approx 28/3m^4\sqrt{M}+2mM$ **real multiplications**

(c) $8m^3+3m^2-3m-1+\sqrt{M}/3(28m^4-32m^3+29m^2-23m+1)+(4m-1)M \approx 28/3m^4\sqrt{M}+4mM$ **real additions**

(d) $6m^2+20m+4+(4m+2)\lceil \log_2 m \rceil+2m\lceil \log_2 \sqrt{M} \rceil+2\sum_{d=1}^{2m-1}\lceil \log_2 d \rceil \approx 6m^2+2m\lceil \log_2 \sqrt{M} \rceil$ **parallel cycles**

Proof: The number of arithmetic operations contributed by each line is as follows:

Line 1

The computation of **G** requires $f_{11}(2m)$ operations and $g_{11}(2m)$ cycles.

Line 4

Ordering requires computing 2*m* times 1-by-2*m* real-valued vector norms, thus $2m * f_8(2m)$ operations and $g_8(2m)$ cycles are required. In order to identify on the right layer, we need to compute the minimum of a set of 2*m* real numbers; this requires $f_7(2m)$ operations and $g_7(2m)$ cycles.

Line 5 to Line 10

For each $\sqrt{M}$ symbol from the constellation, the following lines can be done in parallel.

Line 7

With (*m*, *n*, *p*) = (2*m*, 1, 1), computing $\boldsymbol{h}_k * \boldsymbol{s}_k$ requires $\sqrt{M} * f_9(2m,1,1)$ operations and $g_9(2m,1,1)$ cycles. Computing $\boldsymbol{y} - \boldsymbol{h}_k * \boldsymbol{s}_k$ requires $\sqrt{M} * f_5(2m,1)$ operations and $g_5(2m,1)$ cycles.

Line 8

The detection of the 2*m*-1 remaining real-equivalent layers will require $\sqrt{M}$ times the operations provided in **Complexity Result 2**, and the same number of run cycles, assuming maximum parallelism.

Line 10

With (*m*, *n*, *p*) = (2*m*, 1, 2*m*), computing **H** * ***s*** requires $\sqrt{M}$ * $f_9(2m,1,2m)$ operations and $g_9(2m,2m,1)$ cycles. Now, computing ***y*** – **H** * ***s*** requires $f_5(2m,1)$ operations and $g_5(2m,1)$ cycles. Finally, computing epsilon, i.e., the square norm of ***y*** – **H** * ***s***, requires $\sqrt{M}$ * $f_8(2m)$ operations and $g_8(2m)$ cycles.

Line 11 to 14

In order to pick the best candidate vector, we need to compute the minimum of a set of $\sqrt{M}$ real numbers; this will require $f_7(\sqrt{M})$ operations and $g_7(\sqrt{M})$ cycles.

Q.E.D.

Table 2 shows that the real-valued V-BLAST detector requires fewer than ten times more real-valued operations and time in cycles than the real-valued MMSE detector, in both the 3×3 or a 4×4 MIMO configurations, regardless of the modulation used (16-QAM, 64-QAM or 256-QAM). Figure 14 shows that this increase in the number of computations leads to a reduction of a factor ten in terms of BER compared to that of the real-valued MMSE detector, regardless of the modulation used and for SNR greater than 35 dB. Note that, while the ratio of the number of real-valued operations required by the real-valued V-BLAST detector over that required by the real-valued MMSE detector is approximately the same for the three different modulations, the number of real-valued operations required by the real-valued F-BLAST detector is approximately doubled when M is quadrupled, and it is approximately tripled compare to that of the real-valued V-BLAST detector for $M = 16$. Also, due to its parallel structure, the real-valued F-BLAST detector requires approximately the same amount of time, in operation cycles, than the real-valued V-BLAST detector. However, Figure 14 shows that the real-valued F-BLAST's BER is at least 100 times lower than that of the real-valued MMSE detector, for SNR greater than 35 dB, regardless of the modulation used.

Therefore, the real-valued F-BLAST detector achieves a better BER vs. SNR performance than the real-valued V-BLAST detector, without increasing the required time in cycles (assuming parallel hardware), and with only a relatively small increase in the number of required real-valued operations (about four times

and ten times more operations are required for 16-QAM and 256-QAM respectively).

Table 2 Computational Complexity of Real-valued MIMO Detection Algorithms

Note: The numbers in brackets give counts relative to the MMSE detector with the corresponding value of *m*.

- *m* = 3 complex layers, or *m* = 6 for the real-valued equivalent model

Scheme	*M*	Real Multiplications	Real Additions	Real Reciprocals	Time in Cycles (Parallel Hardware)
MMSE	**16**	312	255	10	45
	64	336	279	10	46
	256	384	327	10	47
V-BLAST	**16**	834 (2.7)	802 (3.1)	65 (6.5)	172 (3.8)
	64	866 (2.6)	842 (3.0)	65 (6.5)	178 (3.9)
	256	930 (2.4)	922 (2.8)	65 (6.5)	184 (4.0)
F-BLAST	**16**	2,580 (8.3)	2,539 (10.0)	230 (23.0)	174 (3.9)
	64	5,020 (14.9)	5,196 (18.6)	450 (45.0)	180 (3.9)
	256	10,380 (27.0)	11,566 (35.4)	890 (89.0)	186 (4.0)

- *m* = 4 complex layers, or *m* = 8 for the real-valued equivalent model

Scheme	*M*	Real Multiplications	Real Additions	Real Reciprocals	Time in Cycles (Parallel Hardware)
MMSE	**16**	672	576	17	52
	64	704	608	17	53
	256	768	672	17	54
V-BLAST	**16**	2,471 (3.7)	2,451 (4.2)	157 (9.2)	258 (5.0)
	64	2,511 (3.6)	2,507 (4.1)	157 (9.2)	266 (5.0)
	256	2,591 (3.4)	2,619 (3.9)	157 (9.2)	274 (5.1)
F-BLAST	**16**	8,187 (12.2)	8,111 (14.1)	577 (33.9)	260 (5.0)
	64	15,941 (22.6)	16,155 (26.6)	1,137 (66.9)	268 (5.0)
	256	32,123 (41.8)	33,683 (50.1)	2,257 (132.8)	276 (5.1)

4.3.2 Complex-valued Detection

Complex-valued arithmetic is not directly supported in most computers, but it is implementable using custom arithmetic units, as could be synthesized in FPGA designs. Typically the support for complex arithmetic is coordinated in software and the real and imaginary parts are computed using multiple machine language instructions on the hardware [28].

Complexity Result 5 ML detection on m layers requires:

(a) 0 real reciprocals

(b) $\boldsymbol{M^m(4m^2+2m) \approx 4m^2M^m}$ **real multiplications**

(c) $\boldsymbol{M^m(4m^2+2m)-1 \approx 4m^2M^m}$ **real additions**

(d) $\boldsymbol{5+2\lceil \log_2 m \rceil + \lceil \log_2 M^m \rceil \approx \lceil \log_2 M^m \rceil}$ **parallel cycles**

Proof: The number of arithmetic operations contributed by each line is as follows:

Line 1 to **Line 6**

We have M^m iterations of **candidateError** to compute, which can be done in parallel. First, with $(m, p, n) = (m, m, 1)$, computing $\underline{\mathbf{H}} * [\underline{s}_{a1}; \underline{s}_{a2} \ldots ; \underline{s}_{am}]$ for all candidates requires $M^m * f_{20}(m,m,1)$ operations and $g_{20}(m,m,1)$ cycles. Then, with $(m, n) = (m, 1)$, computing $\underline{\boldsymbol{y}} - \underline{\mathbf{H}} * [\underline{s}_{a1}; \underline{s}_{a2} \ldots ; \underline{s}_{am}]$ for all candidates requires $M^m * f_{17}(m,1)$ operations and $g_{17}(m,1)$ cycles. Finally, with $n = m$, computing $\|\underline{\boldsymbol{y}} - \underline{\mathbf{H}} * [\underline{s}_{a1}; \underline{s}_{a2} \ldots ; \underline{s}_{am}]\|^2$ requires $M^m * f_{19}(m)$ operations and $g_{19}(m)$ cycles.

Line 11

Computing the minimum of a set of M^m real number requires $f_7(M^m)$ operations and $g_7(M^m)$ cycles.

Q.E.D.

Complexity Result 6 MMSE detection on m layers requires:

(a) $m^2+1 \approx m^2$ **real reciprocals**

(b) $8m^3+8m^2+2mM \approx 8m^3+2mM$ **real multiplications**

(c) $8m^3+3m^2-3m+3mM+M-1 \approx 8m^3+3mM$ **real additions**

(d) $7m+11+3\lceil \log_2 m \rceil+\lceil \log_2 M \rceil \approx 7m+\lceil \log_2 M \rceil$ **parallel cycles**

Proof: The number of arithmetic operations contributed by each line is as follows:

Line 1

Computing the MMSE conditioning matrix will require $f_{22}(m)$ operations and $g_{22}(m)$ cycles.

Line 3

With $(m, n, p) = (m, 1, m)$, computing $\underline{\mathbf{G}} * \underline{\mathbf{y}}$ requires $f_{20}(m,1,m)$ operations and $g_{20}(m,1,m)$ cycles.

Line 4

First, $\underline{\mathbf{y}}_{\mathbf{mmse}} * \mathbf{1}_{1,M}$ and $\mathbf{1}_{m,1} *$ **ConstellationSymbolMatrix**, can be seen as memory allocation, so they require no operations. Then with $(m, n) = (m, M)$, computing $\underline{\mathbf{y}}_{\mathbf{mmse}} * \mathbf{1}_{1,M} - \mathbf{1}_{m,1} *$ **ConstellationSymbolMatrix** requires $f_{17}(n,M)$ operations and $g_{17}(n,M)$ cycles. Finally, we have to compute the absolute square norm of mM complex-valued numbers, thus $mM * f_{16}$ operations and two cycles are required.

Line 5

We have to compute the minimum of m sets of real number of length M, thus, $m * f_7(M)$ operations and $g_7(M)$ cycles are required.

Q.E.D.

Complexity Result 7 V-BLAST detection on m - 1 remaining layers requires:

(a) $\mathbf{1/6(2m^3-3m^2+m) \approx 1/3m^3}$ **real reciprocals**

(b) $\mathbf{7/3m^4-3m^3+18m^2-12m+(4m-2)M \approx 7/3m^4+4mM}$ **real multiplications**

(c) $\mathbf{7/3m^4-7m^3+50/3m^2-17m+2+(4m-4)M \approx 7/3m^4+4mM}$ **real additions**

(d) $\mathbf{7/2m^2+13/2m-13-\lceil \log_2 m - 1 \rceil+(m-1)(\lceil \log_2 m \rceil+\lceil \log_2 M \rceil)+2\sum_{d=1}^{m-1}\lceil \log_2 d \rceil}$

$\mathbf{\approx 7/2m^2+m\lceil \log_2 M \rceil}$ **parallel cycles**

Note: Here, we assumed that the detection on the first layer has already been accomplished.

Proof: The number of arithmetic operations contributed by each line is as follows:

On the m - 1 remaining layers, $\underline{\mathbf{H}}$ is considered to be an m-by-d complex matrix where m-d is the number of layers already deflated.

Line 4

With $(n, m) = (m, d)$, the computation of $\underline{\mathbf{G}}$ requires $f_{23}(m,d)$ operations and $g_{23}(m,d)$ cycles.

Line 5

In order to identify the right layer, we need to compute the minimum of a set of m real numbers and, this requires $f_7(d)$ operations and $g_7(d)$ cycles. Note that on the last layer to be detected, the ordering of the layers is not required.

Line 7

With $(m, n, p) = (1, 1, m)$, nulling will require $f_{20}(1,1,m)$ operations and $g_{20}(1,1,m)$ cycles.

Line 8

We need to pick the right symbol from the constellation. First we need to compute the distance from each constellation symbol, i.e., with (m, n) = (M, 1), $f_{17}(M,1)$ operations and $g_{17}(M,1)$ cycles are required.

We then need to compute M complex square absolute values, i.e., $M * f_{16}$ operations and g_{16} cycles. Finally, we pick the minimum of a set of M real numbers; this requires $f_7(M)$ operations and $g_7(M)$ cycles.

Line 10

With (m, n, p) (m, 1, 1), computing $\underline{\boldsymbol{h}}_k * \underline{\boldsymbol{s}}_k$ requires $f_{20}(m,1,1)$ operations and $g_{20}(m,1,1)$ cycles. Now, computing $\underline{\boldsymbol{y}} - \underline{\boldsymbol{h}}_k * \underline{\boldsymbol{s}}_k$ requires $f_{17}(m,1)$ operations and $g_{17}(m,1)$ cycles. Except on the last layer to be detected.

Line 12

Neither operations nor time in cycles are required to deflate the channel matrix $\underline{\mathbf{H}}$.

Q.E.D.

Complexity Result 8 V-BLAST detection on m layers requires:

(a) $\mathbf{1/6(2m^3+3m^2+m+6) \approx 1/3m^3}$ **real reciprocals**

(b) $\mathbf{7/3m^4+5m^3+24m^2-4m+4mM \approx 7/3m^4+4mM}$ **real multiplications**

(c) $\mathbf{7/3m^4+m^3+53/3m^2-10m-1+4mM \approx 7/3m^4+4mM}$ **real additions**

(d) $\mathbf{21/2m^2+13/2m+3-\lceil \log_2 m-1 \rceil+(m+4)\lceil \log_2 m \rceil+m\lceil \log_2 M \rceil+2\sum_{d=1}^{m-1}\lceil \log_2 d \rceil \approx 21/2m^2+m\lceil \log_2 M \rceil}$ **parallel cycles**

Proof: The number of arithmetic operations contributed by each line is as follows:

Line 5

Ordering requires computing m times 1-by-m complex-valued vector norms, thus $m * f_{19}(m)$ operations and $g_{19}(m)$ cycles are required.

Line 3 to Line 13

On the first layer to be detected,

Line 4

The computation of $\underline{\mathbf{G}}$ requires $f_{22}(m)$ operations and $g_{22}(m)$ cycles.

Line 5

In order to proceed on the right layer, we need to compute the minimum of a set of m real numbers. This requires $f_7(m)$ operations and $g_7(m)$ cycles.

Line 7

With $(m, p, n) = (1, m, 1)$, nulling will require $f_{20}(1, m, 1)$ operations and $g_{20}(1, m, 1)$ cycles.

Line 8

The right symbol needs to be picked from the constellation. First we need to compute the distance from each constellation symbol, i.e., with $(m, n) = (M , 1)$, $f_{17}(M,1)$ operations and $g_{17}(M,1)$ cycles are required. Then, we need to compute M complex square absolute values, i.e., $M * f_{16}$ operations and g_{16} cycles. Finally, we need to pick the minimum of a set of M real numbers, this requires $f_7(M)$ operations and $g_7(M)$ cycles.

Line 10

With (m, n, p) = (m, 1, 1), computing $\underline{\boldsymbol{h}}_k$ * $\underline{\boldsymbol{s}}_k$ requires $f_{20}(m,1,1)$ operations and $g_{20}(m,1,1)$ cycles. Now, computing $\underline{\boldsymbol{y}} - \underline{\boldsymbol{h}}_k$ * $\underline{\boldsymbol{s}}_k$ requires $f_{17}(m,1)$ operations and $g_{17}(m,1)$ cycles.

Finally, the detection on the first layer will require m^2+1 ($1/N$), $8m^3+6m^2+8m+2M$ (*), $8m^3+m^2+7m+4M-3$ (+), $7m+16+\lceil \log_2 M \rceil+5\lceil \log_2 m \rceil$ (c).

On the m-1 remaining layers, the complexity is given by **Complexity Result 7**.

Q.E.D.

Complexity Result 9 F-BLAST detection on m layers requires:

(a) $\boldsymbol{m^2+1+M/6(2m^3-3m^2+m) \approx 1/3m^3M}$ **real reciprocals**

(b) $\boldsymbol{8m^3+6m^2+(7/3m^4-3m^3+22m^2-6m)M+(4m-2)M^2 \approx 7/3m^4M+4mM^2}$ **real multiplications**

(c) $\boldsymbol{8m^3+m^2-m-2+(7/3m^4-7m^3+62/3m^2-11m+2)M+(4m-4)M^2 \approx 7/3m^4M+4mM^2}$ **real additions**

(d) $\boldsymbol{7/2m^2+27/2m+3-\lceil \log_2 m-1 \rceil+(m+5)\lceil \log_2 m \rceil+m\lceil \log_2 M \rceil+2\sum_{d=1}^{m-1}\lceil \log_2 d \rceil \approx 7/2m^2+m\lceil \log_2 M \rceil}$ **parallel cycles**

Proof: The number of arithmetic operations contributed by each line is as follows:

Line 1

The computation of $\underline{\mathbf{G}}$ requires $f_{22}(m)$ operations and $g_{22}(m)$ cycles.

Line 3

Ordering requires computing m times 1-by-m complex-valued vector norms, thus m * $f_{19}(m)$ operations and $g_{19}(m)$ cycles are required. In order to identify the right

layer, we need to compute the minimum of a set of m real numbers; this requires $f_7(m)$ operations and $g_7(m)$ cycles.

Line 5 to Line 10

For each M symbol from the constellation, this can be done in parallel.

Line 7

With $(m, n, p) = (m, 1, 1)$, computing $\underline{\boldsymbol{h}}_{\mathbf{k}} * \underline{\boldsymbol{s}}_{\mathbf{k}}$ requires $M * f_{20}(m,1,1)$ operations and $g_{20}(m,1,1)$ cycles. Now, computing $\underline{\boldsymbol{y}} - \underline{\boldsymbol{h}}_{\mathbf{k}} * \underline{\boldsymbol{s}}_{\mathbf{k}}$ requires $M * f_{17}(m,1)$ operations and $g_{17}(m,1)$ cycles.

Line 8

The detection of the m-1 remaining layers will require M times the operations provided in **Complexity Result 7**, and the same number of run cycles, assuming maximum parallelism.

Line 10

With $(m, n, p) = (m, 1, m)$, computing $\underline{\mathbf{H}} * \underline{\boldsymbol{S}}$ requires $M * f_{20}(m, 1, m)$ operations and $g_{20}(m,m,1)$ cycles. Now, computing $\underline{\boldsymbol{y}} - \underline{\mathbf{H}} * \underline{\boldsymbol{S}}$ requires $M * f_{17}(m,1)$ operations and $g_{17}(m,1)$ cycles. Finally, computing epsilon, i.e., the square norm of $\underline{\boldsymbol{y}} - \underline{\mathbf{H}} * \underline{\boldsymbol{S}}$, requires $M * f_{19}(m)$ operations and $g_{19}(m)$ cycles.

Line 11 to 14

In order to pick the best candidate vector, we need to compute the minimum of a set of M real numbers; this will require $f_7(M)$ operations and $g_7(M)$ cycles.

Q.E.D.

Complexity Result 10 FR-BLAST detection on *m* layers requires:

(a) $m^2+1+W/6(2m^3-3m^2+m) \approx 1/3m^3W$ **real reciprocals**

(b) $8m^3+6m^2+4m(7/3m^4-3m^3+22m^2-6m)W+(4m-2)MW \approx 7/3m^4W+4mMW$ **real multiplications**

(c) $8m^3+m^2+3m-4+(7/3m^4-7m^3+62/3m^2-11m+2)M+(4m-4)MW \approx 7/3m^4M+4mMW$ **real additions**

(d) $7/2m^2+27/2m+5-\lceil \log_2 m-1 \rceil+(m+6)\lceil \log_2 m \rceil+m\lceil \log_2 M \rceil+2\sum_{d=1}^{m-1}\lceil \log_2 d \rceil \approx 7/2m^2+m\lceil \log_2 M \rceil$ **parallel cycles**

Proof: The number of arithmetic operations contributed by each line is as follows:

The algorithm is very similar to Fouladi Fard's algorithm, except the fact that instead of searching in the entire constellation for the first symbol, a restricted search in a subset (search space containing *W* symbol, see Line 5 and 6) is performed.

Line 5

With (*m*, *n*, *p*) = (1, 1, *m*), computing $\underline{\boldsymbol{g}}_k * \underline{\boldsymbol{y}}$ requires $M * f_{20}(1,1, m)$ operations and $g_{20}(1,1, m)$ cycles.

Line 6

No operations are required as it is equivalent to reading coefficients from a pre-defined matrix.

Q.E.D.

Table 3 confirms that ML is not suitable for practical implementation, due to the large number of real-valued operations that it requires. In addition, it shows that V-BLAST requires about twice as many real-valued operations and time in cycles than MMSE, for both the 3×3 and 4×4 MIMO configurations, regardless of the modulation used (16-QAM, 64-QAM or 256-QAM). Figure 13 shows that this increase in the number of computations leads to BER approximately 10 times lower, in comparison to that of MMSE, regardless of the modulation used and for SNR greater than 30 dB. Also, while the ratio of the number of real-valued operations for V-BLAST over that of MMSE is approximately the same for the three different modulations, the number of real-valued operations required by F-BLAST is about ten times greater when M is quadrupled. Due to its parallel structure, F-BLAST requires approximately the same number of cycles than V-BLAST. However, Figure 13 shows that F-BLAST exhibits a BER at least 10 times lower (about 40 times lower with 16-QAM and 64-QAM, for SNR = 40 dB).

For search window sizes W = 8, 16 or 32, the gain achieved by FR-BLAST in terms of BER vs. SNR in comparison to that of MMSE, lies between 1 dB and 2 dB. The computational complexity of FR-BLAST is about W times greater than that of V-BLAST, and considerably smaller than that of F-BLAST. Observe that for the special case where W = 1 and W = M, FR-BLAST is equivalent to V-BLAST and F-BLAST respectively, thus one can predict that the performance characteristic of FR-BLAST and its computational complexity will lie between that of V-BLAST and F-BLAST, depending on the value of W.

In conclusion, FR-BLAST achieves a better BER versus SNR performance than V-BLAST, without increasing the required time in cycles, but at the expense of additional computations proportional to the size of the restricted search set. Interestingly the size of the search set, and hence the degree of parallelism, can be adjusted to control the performance and the required power.

Table 3 Computational Complexity of Complex-valued MIMO Detection Algorithms

Note: The numbers in brackets give counts relative to the MMSE detector with the corresponding value of *m*.

- $m = 3$

Scheme	M	Real Multiplications	Real Additions	Real Reciprocals	Time in Cycles (Parallel Hardware)
ML	**16**	172,032	172,031	0	21
	64	11,010,048	11,010,047	0	27
	256	704,643,072	704,643,071	0	33
MMSE	**16**	384 (1.0)	393 (1.0)	10 (1.0)	42 (1.0)
	64	672 (1.0)	873 (1.0)	10 (1.0)	44 (1.0)
	256	1,824 (1.0)	2,793 (1.0)	10 (1.0)	46 (1.0)
V-BLAST	**16**	720 (1.9)	536 (1.4)	15 (1.5)	144 (3.5)
	64	1,296 (2.0)	1,112 (1.3)	15 (1.5)	150 (3.4)
	256	3,600 (2.0)	3,416 (1.3)	15 (1.5)	156 (3.4)
F-BLAST	**16**	7,438 (19.4)	4,748 (15.1)	90 (9)	104 (2.5)
	64	59,662 (88.8)	42,908 (49.2)	330 (33)	110 (2.5)
	256	729,358 (399.9)	564,188 (202.1)	1,290 (129)	116 (2.6)
FR-BLAST $W = 8$	**64**	7,706 (11.5)	5,566 (6.4)	58 (5.8)	114 (2.6)
	256	23,066 (12.7)	17,854 (6.4)	58 (5.8)	120 (2.7)
FR-BLAST $W = 16$	**64**	15,130 (22.6)	10,902 (12.5)	106 (10.6)	114 (2.6)
	256	45,850 (25.2)	35,478 (12.8)	106 (10.6)	120 (2.7)
FR-BLAST $W = 32$	**64**	29,978 (44.7)	21,574 (24.8)	202 (20.2)	114 (2.6)
	256	91,418 (50.2)	70,726 (25.4)	202 (20.2)	120 (2.7)

- $m = 4$

Scheme	M	Real Multiplications	Real Additions	Real Reciprocals	Time in Cycles (Parallel Hardware)
ML	**16**	4,718,592	4,718,591	0	25
	64	1,207,959,552	1,207,959,551	0	33
	256	309,237,645,312	309,237,645,311	0	41
MMSE	**16**	768 (1.0)	755 (1.0)	17(1.0)	49 (1.0)
	64	1,152 (1.0)	1,379 (1.0)	17(1.0)	51 (1.0)
	256	2,688 (1.0)	3,875 (1.0)	17(1.0)	53 (1.0)
V-BLAST	**16**	1,542 (2.1)	1,159 (1.6)	31 (1.9)	231 (5.7)
	64	2,310 (2.1)	1,927 (1.4)	31 (1.9)	239 (4.7)
	256	5,382 (2.1)	4,999 (1.3)	31 (1.9)	247 (4.7)
F-BLAST	**16**	15,926 (34.6)	10,602 (14.1)	241 (14.2)	149 (3.7)
	64	104,886 (91.1)	77,706 (56.4)	913 (53.8)	157 (3.1)
	256	1,105,846 (411.5)	899,082 (232.1)	3,601 (211.9)	165 (3.2)
FR-BLAST *W* = 8	**64**	13,659 (11.9)	10,184 (7.4)	137 (8.1)	161 (3.4)
	256	35,163 (13.1)	28,616 (7.4)	137 (8.1)	169 (3.1)
FR-BLAST *W* = 16	**64**	26,694 (23.2)	19,832 (14.4)	257 (15.2)	161 (3.4)
	256	69,702 (26.0)	56,696 (14.7)	257 (15.2)	169 (3.1)
FR-BLAST *W* = 32	**64**	52,763 (45.9)	39,128 (28.4)	497 (29.3)	161 (3.4)
	256	138,779 (51.6)	112,856 (29.2)	497 (29.3)	169 (3.2)

Thus assuming that we can afford a detector that is three times slower than MMSE, we recommend the use of FR-BLAST with a search window containing 16 symbols, since it only increases the number of arithmetical operations by a factor of ten regardless of the modulation scheme, for a reduction of the BER higher than ten. In other words, by slowing down a little bit the detection process, and consuming ten times more power, the detector can make ten times less error.

V- Asymptotic Analysis

5.1 Assumptions

We assume that the channel is unknown at the transmitter side, but known or perfectly estimated at the receiver side. Error correcting codes are not used.

Let n_R and n_T be the number of antennas at the receiver and transmitter side, respectively. In this work we assume that $n_R = n_T = m \geq 1$. As before, M denotes the number of points in the complex symbol constellation.

The channel matrix coefficients and the components of the Additive White Gaussian Noise are assumed to be circularly Gaussian. The channel matrix, **H**, is assumed to be constant for the duration T of a frame. For this work, sequences of symbols are grouped into frames such that the duration T of a frame, i.e., the number of samples in a frame, satisfies $\boldsymbol{T \geq 2 * m - 1}$ [12]. This choice has been shown through experience to give acceptable channel modeling accuracy with reduced computational complexity.

5.2 Asymptotic Performance Analysis

5.2.1 Definitions

- The *multiplexing gain* is the multiplicative gain in capacity (with respect to a SISO channel) achieved by distributing the main data streams into multiple parallel data streams. The multiplexing gain, r_m, is given by $r_m = \min(n_R, n_T)$. The multiplexing gain r_m of all the detection schemes presented in this work is equal to the MIMO gain, i.e., $r_m = m$ [30].
- The *diversity order* expresses how fast the average error probability decreases with respect to increasing SNR [31]. Let d denote the diversity order, then

  $d = - \lim_{snr \to \infty} \log P_e(SNR) / \log SNR$, where $P_e(SNR)$ is the average error probability of the scheme. The error probability is usually measured by the bit error rate.
- The *diversity gain* expresses how the diversity order increases additively over that of the SISO system which is one [31].

  The ML detection scheme experiences full diversity order, i.e., $d_{ML} = n_R = m$ [32]. A reduced complexity scheme, such as the MMSE detector, experiences full multiplexing gain while reducing significantly the computational complexity, but at the cost of severe loss in performance. At high SNR, $d_{MMSE} = n_R - n_T + 1 = 1$ [31]. The V-BLAST detection scheme has a diversity gain of zero, which is independent of the strength (i.e., the estimated post-detection SNR) of the first layer to be detected,

thus $d_{\mathrm{VBLAST}} = n_R - n_T + 1 = 1$ [33]. As, one might expect, the diversity gain of V-BLAST is limited by the MMSE detection of the first symbol, which does not benefit from interference cancellation from the other symbols. Even though the fall-in P_e(SNR) is the same for both MMSE and V-BLAST, the BER of V-BLAST is much lower than that of MMSE, as shown in Figure 14.

5.2.2 Asymptotic Analysis of F-BLAST

Asymptotic Result 1 The diversity order d_{FBLAST} of F-BLAST is 2.

Proof:

Let $P_{e,\mathrm{FBLAST}}$ (SNR) denote the average error probability of the F-BLAST scheme at a given SNR. The *law of total probability* leads to:

$$P_{e,\mathrm{FBLAST}}(\mathrm{SNR}) = P(E \mid \bar{E}_1) * P(\bar{E}_1) + P(E \mid E_1) * P(E_1) \quad (1)$$

Here, for a given SNR, $P(E_1)$ is the average error probability for the decision on the first layer to be detected; $P(\bar{E}_1)$ is the average probability of a correct decision on the first layer to be detected; $P(E \mid \bar{E}_1)$ is the average error probability of the F-BLAST scheme given that a correct decision had been made on the first layer to be detected (i.e., an error occurred after the first symbol was detected correctly, so no error propagated from the detection of the first symbol); and $P(E \mid E_1)$ is the average error probability of the F-BLAST scheme given that an erroneous decision had been made on the first layer to be detected (i.e., an error in the detection of the first symbol propagates to an error in the detection of the second symbol).

$\bar{E}_1$ is the complementary event of E_1, thus

$$P(\bar{E}_1) = 1 - P(E_1) \quad (2)$$

From (1) and (2), we obtain

$$P_{e,FBLAST}(SNR) = P(E \mid \bar{E}_1) + \left(P(E \mid E_1) - P(E \mid \bar{E}_1)\right) * P(E_1) \quad (3)$$

When the SNR is large [32],

$$P_{e,VBLAST}(SNR) \cong \frac{C_{VBLAST}}{SNR^{n_R - n_T + 1}} \quad (4)$$

where $P_{e,VBLAST}$ (SNR) is the average error probability of the V-BLAST scheme at the given SNR, and C_{VBLAST} is a positive constant.

Consider a scheme that always makes an erroneous decision on the first layer to be detected and then proceeds with the original V-BLAST detection over the *m* - 1 remaining layers. Its average error probability is an upper bound on the error probability of V-BLAST on an *m*-by-*m* system [33]. Note that a scheme that always makes a correct decision on the first layer to be detected and then proceeds with original V-BLAST detection over the *m* - 1 remaining layers has an error probability equivalent to that of V-BLAST on an (*m*-1)-by-*m* MIMO system since the second layer to be detected will experience one less interferer [33]. Thus with (4) and for large SNR we obtain

$$P(E \mid \bar{E}_1) \cong \frac{C_{VBLAST}}{SNR^{n_R - n_T + 1}} \quad , \; n_R = m \text{ and } n_T = m - 1$$

$$P(E \mid \bar{E}_1) \cong \frac{C_{VBLAST}}{SNR^{2}} \quad (5)$$

and $$P(E \mid E_1) \cong \frac{C_{VBLAST}}{SNR^{n_R - n_T + 1}} \quad , n_R = n_T = m$$

$$P(E \mid E_1) \cong \frac{C_{VBLAST}}{SNR^{1}} \quad (6)$$

Let $\underline{s}_1$ and $\hat{\underline{s}}_1$ denote the transmitted and detected symbols, respectively, on the first layer to be detected. Let $E_{\underline{H}}\{\}$ denote the expected value operator over the channel matrix $\underline{\mathbf{H}}$. Then by definition [34]:

$$P(E_1) = E_{\underline{H}}\{\sum_{\hat{\underline{s}}_1 \neq \underline{s}_1} (P(\hat{\underline{s}}_1 \neq \underline{s}_1 \mid \underline{\mathbf{H}}) * P(\underline{s}_1))\} \quad (7)$$

For large SNR, and with C_{ML} a positive constant [31]:

$$E_{\underline{H}}\{P(\hat{\underline{s}}_1 \neq \underline{s}_1 \mid \underline{\mathbf{H}})\} \cong \frac{C_{ML}}{SNR^{n_R}} \quad (8)$$

With $n_R = m$, and assuming equiprobable symbols $P(\underline{s}_1) = 1 / M$. Substituting (8) into (7) for large SNR yields

$$P(E_1) \cong \sum_{\hat{\underline{s}}_1 \neq \underline{s}_1} \frac{C_{ML}}{M * SNR^m} \quad (9)$$

$$P(E_1) \cong \frac{(M-1) * C_{ML}}{M * SNR^m} \quad (10)$$

Thus, for large SNR, (3), (5), (6) and (10) yield

$$P_{e,FBLAST}(SNR) \cong \left(\frac{C_{VBLAST}}{SNR^1} - \frac{C_{VBLAST}}{SNR^2}\right) * \frac{(M-1) * C_{ML}}{M * SNR^m} + \frac{C_{VBLAST}}{SNR^2}$$

By factoring, we obtain

$$P_{e,FBLAST}(SNR) \cong \frac{C_{VBLAST}}{SNR^2} * \left(1 + \frac{M-1}{M} * \frac{C_{ML}}{SNR^{m-1}} * \left(1 - \frac{1}{SNR}\right)\right)$$

Thus, for large SNR and $m > 1$,

$$\log(P_{e,FBLAST}(SNR)) \cong \log(C_{VBLAST}) - 2\log(SNR) + \log\left(1 + \frac{M-1}{M} * \frac{C_{ML}}{SNR^{m-1}} * \left(1 - \frac{1}{SNR}\right)\right)$$

In the limit as the SNR becomes large we obtain

$$\lim_{SNR \to \infty} \log P_{e,FBLAST}(SNR) \Big/ \log SNR = -2 \quad (11)$$

Q.E.D.

5.2.3 Asymptotic Analysis of FR-BLAST

Asymptotic Result 2 The diversity order $d_{FR(W)\text{-}BLAST}$ of FR-BLAST is one.

Proof:

Let $\underline{\boldsymbol{S}}_W$ denote the restricted search set containing W M-QAM symbols, as defined in Section 3.1.1., and let $P_{e,FR(W)BLAST}(SNR)$ denote the average error probability of the FR-BLAST scheme which runs W parallel searches, at a given SNR. The *law of total probability* leads to:

$$P_{e,FR(W)BLAST}(SNR) = P(E \mid \underline{s}_1 \in \underline{\boldsymbol{S}}_W)*P(\underline{s}_1 \in \underline{\boldsymbol{S}}_W) + P(E \mid \overline{\underline{\boldsymbol{s}}_1 \in \underline{\boldsymbol{S}}_W})*P(\overline{\underline{\boldsymbol{s}}_1 \in \underline{\boldsymbol{S}}_W}) \quad (12)$$

Here, for a given SNR, $P(\overline{\underline{\boldsymbol{s}}_1 \in \underline{\boldsymbol{S}}_W})$ is the average probability of not having within the restricted search set the symbol transmitted on the first layer; $P(\underline{s}_1 \in \underline{\boldsymbol{S}}_W)$ is the average probability of having within the search set the symbol transmitted on the first layer; $P(E \mid \overline{\underline{\boldsymbol{s}}_1 \in \underline{\boldsymbol{S}}_W})$ is the average error probability of FR(W)-BLAST given that the restricted search set does not include the symbol transmitted on the first layer; and $P(E \mid \underline{s}_1 \in \underline{\boldsymbol{S}}_W)$ is the average error probability of the FR(W)-BLAST given that the restricted search set includes the symbol transmitted on the first layer.

By definition:

$$P(\underline{s}_1 \in \underline{\boldsymbol{S}}_W) = \frac{W}{M} \quad (13)$$

and from (2) and (13),

$$P(\overline{\underline{\boldsymbol{s}}_1 \in \underline{\boldsymbol{S}}_W}) = \frac{M-W}{M} \quad (14)$$

If the restricted search set does not include the symbol transmitted on the first layer to be detected, then FR(W)-BLAST will make an erroneous decision on

this layer and this error will propagate through to the detection of the m - 1 remaining layers. But, after the detection of $\hat{\underline{s}}_1$, FR(W)-BLAST runs the original V-BLAST detection on the m - 1 remaining layers. Therefore from (6) and for large SNR

$$P(E \mid \overline{\underline{s}_1 \in \underline{\boldsymbol{S}}_W}) = P(E \mid E_1) \cong C_{VBLAST} * \frac{1}{SNR^2} \quad (15)$$

If the restricted search set includes the symbol transmitted on the first layer to be detected, from (3) and the *law of total probability*,

$$P(E|\underline{s}_1 \in \underline{\boldsymbol{S}}_W) = P(E|(\bar{E}_1, \underline{s}_1 \in \underline{\boldsymbol{S}}_W)) + \left[P(E|(E_1, \underline{s}_1 \in \underline{\boldsymbol{S}}_W)) - P(E|(\bar{E}_1, \underline{s}_1 \in \underline{\boldsymbol{S}}_W))\right] * P(E_1, \underline{s}_1 \in \underline{\boldsymbol{S}}_W) \quad (16)$$

After the decision on the first layer to be detected, both the F-BLAST and FR(W)-BLAST detectors are identical. Therefore, from (5) and (6):

$$P(E \mid (\bar{E}_1, \underline{s}_1 \in \underline{\boldsymbol{S}}_W)) = P(E \mid \bar{E}_1) \cong \frac{C_{VBLAST}}{SNR^2} \quad (17)$$

$$P(E \mid (E_1, \underline{s}_1 \in \underline{\boldsymbol{S}}_W)) = P(E \mid E_1) \cong \frac{C_{VBLAST}}{SNR^2} \quad (18)$$

Similarly, to the derivation of $P(E_1)$ in Section 5.2.2., from (9) we obtain

$$P(E \mid \underline{s}_1 \in \underline{\boldsymbol{S}}_W) \cong \sum_{\hat{\underline{s}}_1 \neq \underline{s}_1} \frac{C_{ML}}{M * SNR^m}$$

There are W - 1 symbols different from $\underline{s}_1$ in the restricted search set, therefore

$$P(E \mid \underline{s}_1 \in \underline{\boldsymbol{S}}_W) \cong \frac{(W-1) * C_{ML}}{M * SNR^m} \quad (19)$$

Thus, for large SNR, (17), (18), (19) and (20) yield to

$$P(E \mid \underline{s}_1 \in \underline{\boldsymbol{S}}_W) \cong \left(\frac{C_{VBLAST}}{SNR^2} - \frac{C_{VBLAST}}{SNR^2}\right) * \frac{(W-1) * C_{ML}}{M * SNR^m} + \frac{C_{VBLAST}}{SNR^2}$$

Finally, we have

$$P(E \mid \underline{s}_1 \in \underline{\boldsymbol{S}}_W) \cong \frac{C_{VBLAST}}{SNR^2} * \left(1 + \frac{W-1}{M} * \frac{C_{ML}}{SNR^{m-1}} * \left(1 - \frac{1}{SNR}\right)\right) \quad (20)$$

From (13), (14), (15), (16) and (20) we obtain

$$P_{e,FR(W)BLAST} \cong \frac{W}{M} * \left(\frac{C_{VBLAST}}{SNR^2} * \left(1 + \frac{W-1}{M} * \frac{C_{ML}}{SNR^{m-1}} * \left(1 - \frac{1}{SNR}\right)\right)\right) + \frac{(M-W) * C_{VBLAST}}{M * SNR^2}$$

By factorization, it is straightforward to obtain for large SNR

$$P_{e,FR(W)BLAST}(SNR) \cong \frac{(M-W) * C_{VBLAST}}{M * SNR^2} * \left(1 + \frac{W}{(M-W) * SNR^2} + \frac{W * (W-1) * C_{ML}}{(M-W) * SNR^m} * \left(1 - \frac{1}{SNR}\right)\right)$$

Thus, for large SNR and $m > 1$,

$$\log(P_{e,\,FR(W)BLAST}(SNR)) \cong \log\left(\frac{(M-W) * C_{VBLAST}}{M}\right) - \log(SNR) + \log\left(1 + \frac{W}{(M-W) * SNR^2} + \frac{W * (W-1) * C_{ML}}{(M-W) * SNR^m} * \left(1 - \frac{1}{SNR}\right)\right)$$

In the limit as the SNR becomes large we obtain

$$\lim_{SNR \to \infty} \log P_{e,FR(W)BLAST}(SNR) \Big/ \log SNR = -1 \qquad (21)$$

Q.E.D.

5.2.4 Asymptotic Analysis of the Real-valued F-BLAST

Asymptotic Result 3 The diversity order d_{FBLAST^R} of real-valued F-BLAST is two.

Proof:

Recall that the real-valued equivalent detection scheme is based on the following model (Section 3.3):

$$\boldsymbol{y}^R = \mathbf{H}^R * \boldsymbol{s}^R + \boldsymbol{n}^R,$$

where $\boldsymbol{y}^R = [\text{real}(\underline{\boldsymbol{y}}), \text{imag}(\underline{\boldsymbol{y}})]^T$, $\mathbf{H}^R = \begin{bmatrix} \text{real}(\underline{\mathbf{H}}) & -\text{imag}(\underline{\mathbf{H}}) \\ \text{imag}(\underline{\mathbf{H}}) & \text{real}(\underline{\mathbf{H}}) \end{bmatrix}$, $\boldsymbol{s}^R = [\text{real}(\underline{\boldsymbol{s}}), \text{imag}(\underline{\boldsymbol{s}})]]^T$, and $\boldsymbol{n}^R = [\text{real}(\underline{\boldsymbol{n}}), \text{imag}(\underline{\boldsymbol{n}})]]^T$.

The diversity order of V-BLAST is limited by the diversity achieved by the first layer to be detected. Furthermore, this layer experiences a diversity order similar to that of MMSE. The diversity order of V-BLAST was derived in [33] and the average error probability of V-BLAST for large SNR is provided in (4). Hence, in the 2*m*-by-2*m* MIMO real equivalent model (here $n_r = n_t = 2m$), the average error probability of MMSE for large SNR can be approximated by [35]:

$$P_{e,VBLAST}{}^{R}(\mathrm{SNR}) \cong \frac{C_{VBLAST}}{\mathrm{SNR}^{n_R - n_T + 1}} \tag{22}$$

An approximation of the average probability of the typical error event for large SNR is [34]:

$$E_{H^R}\{P(\hat{s}_1 \neq s_1 \mid \mathbf{H}^{\mathbf{R}})\} \cong \frac{C_{ML}}{\mathrm{SNR}^{\frac{n_R}{2}}} \tag{23}$$

Recall, the expression of the average probability of F-BLAST derived in Section 5.2.2:

$$P_{e,FBLAST}{}^{R}(\mathrm{SNR}) = P(E \mid \bar{E}_1) + \left(P(E \mid E_1) - P(E \mid \bar{E}_1)\right) * P(E_1) \tag{24}$$

Likewise, using the 2*m*-by-2*m* real-valued equivalent model, we obtain for large SNR:

$P(E \mid \bar{E}_1) \cong \frac{C_{VBLAST}}{\mathrm{SNR}^{n_R - n_T + 1}}$, $n_R = 2m$ and $n_T = 2m - 1$

$$P(E \mid \bar{E}_1) \cong \frac{C_{VBLAST}}{\mathrm{SNR}^{2}} \tag{25}$$

and $P(E \mid E_1) \cong \frac{C_{VBLAST}}{\mathrm{SNR}^{n_R - n_T + 1}}$, $n_R = n_T = 2m$

$$P(E \mid E_1) \cong \frac{C_{VBLAST}}{\mathrm{SNR}^{1}} \tag{26}$$

With $n_R = 2m$, $P(s_1) = \frac{1}{\sqrt{M}}$, (7) and (23), for large SNR, we can evaluate $P(E_1)$ as

$$P(E_1) \cong \sum_{\hat{s}1 \neq s1} \frac{C_{ML}}{\sqrt{M} * \mathrm{SNR}^{\frac{2m}{2}}} \tag{27}$$

$$P(E_1) \cong \frac{(\sqrt{M}-1) * C_{ML}}{\sqrt{M} * SNR^{\frac{2m}{2}}} \quad (28)$$

Thus, for large SNR, (24), (25), (26) and (28) yield

$$P_{e,FBLAST^R}(SNR) \cong (\frac{C_{VBLAST}}{SNR^1} - \frac{C_{VBLAST}}{SNR^2}) * \frac{(\sqrt{M}-1) * C_{ML}}{\sqrt{M} * SNR^{\frac{2m}{2}}} + \frac{C_{VBLAST}}{SNR^1}$$

By factoring, it is straightforward to obtain

$$P_{e,FBLAST^R}(SNR) \cong \frac{C_{VBLAST}}{SNR^2} * (1 + \frac{(\sqrt{M}-1) * C_{ML}}{\sqrt{M} * SNR^{\frac{2m-2}{2}}} * (1 - \frac{1}{SNR^1}))$$

Thus, for large SNR and $m > 1$

$$\log(P_{e,FBLAST^R}(SNR)) \cong \log(C_{VBLAST}) - 2*\log(SNR) + \log(1 + \frac{(\sqrt{M}-1) * C_{ML}}{\sqrt{M} * SNR^{\frac{2m-2}{2}}} * (1 - \frac{1}{SNR^1}))$$

Finally, in the limit for large SNR

$$\lim_{SNR \to \infty} \log P_{e,FBLAST^R}(SNR) \Big/ \log SNR = -2 \quad (29)$$

Q.E.D.

Corollary to Asymptotic Result 3 The diversity order $d_{F(W1,W2)BLAST^R}$ of the real-valued F(**W1**,**W2**)-BLAST is 2.5.

Proof:

The real-valued F(**W1**,**W2**)-BLAST is the detector that selects the best symbol vector between (1) the real-valued detector that searches the real component of the weakest layer, and (2) the real-valued detector that searches the complex component of the weakest layer. Such a system should benefit from 'selection diversity', thus assuming reasonable independence in the two detectors,

the diversity gain for a two-branch selection diversity should be 1.5 [36].

Therefore, by definition, the diversity order of such detector is 2.5.

Q.E.D.

One could extend this idea to obtain selection benefits by search on greater than the first two real layers, to say three or four layers. However the computational cost will rapidly become prohibitive.

5.2.5 Simulation Study

We calculated the slope of the performance characteristic of various schemes studied in this work and illustrated in Figure 8, 9, 11, 13 and 14. The results are summarized in Table 4. Since, each curve was drawn using the average number of symbols in error for a given SNR, then stored for future analysis, we basically picked two points from the tail of the performance characteristic to compute its slope, using the following formula:

$$slope = \frac{\log_{10}(SER_1) - \log_{10}(SER_2)}{\frac{SNR_{dB1}}{10} - \frac{SNR_{dB2}}{10}}$$

Here SER_i and SNR_{dBi} are the Y-axis and X-axis coordinate, respectively, of a point from the tail of the performance characteristic.

Table 4 Experimentally Measured Tail Slope

- Figure 8: M = 64, SNR_1 = 28 dB and SNR_2 = 30 dB

Detector	F-BLAST 4×4	F-BLAST 6×6	F-BLAST 8×8
slope	- 3.6	- 5.1	- 5.4

- Figure 9: M = 16, SNR_1 = 26 dB and SNR_2 = 28 dB

Detector	ML	F-BLAST	V-BLAST	MMSE
slope	- 5.0	- 4.8	- 1.0	- 1.0

- Figure 11: M = 64

	slope	
Detector	SNR_1 = 28 dB and SNR_2 = 30 dB	SNR_1 = 28 dB and SNR_2 = 30 dB
FR(OPT,9)-BLAST	- 2.8	- 1.0
FR(W2,9)-BLAST	- 2.0	- 1.0
FR(S1,9)-BLAST	- 2.2	- 1.7
FR(OPT,16)-BLAST	- 2.8	- 1.0
FR(W2,16)-BLAST	- 1.9	- 1.0
FR(S1,16)-BLAST	- 2.2	- 1.7

- Figure 13: SNR_1 = 34 dB, SNR_2 = 36 dB

	slope		
Detector	M = 64	M = 128	M = 256
FR-BLAST(W2,32)	- 1.8	- 2.0	- 2.3
FR-BLAST(W2,16)	- 1.1	- 1.2	- 1.8
F-BLAST	- 4.5	- 3.5	- 3.7
V-BLAST	- 0.9	- 1.2	- 1.3
MMSE	- 0.8	- 0.9	- 1.0

- Figure 14: SNR_1 = 34 dB, SNR_2 = 36 dB

	slope		
Detector	M = 16	M = 64	M = 256
MMSE	- 1.00	- 1.0	- 0.9
V-BLAST	- 1.03	- 1.2	- 1.1
real F-BLAST	- 1.66	- 1.9	- 2.1
F-BLAST	- 2.13	- 3.6	- 3.0
Parallel (W1,W2) real F-BLAST	x	-2.7	-3.0

These tables show that for SNR ranging from 26 dB to 40 dB, neither FR-BLAST (with relatively large window sizes), real F-BLAST, F-BLAST nor ML's

performance characteristic has reached their theoretical asymptotic behaviour, i.e., the absolute value of the tail's slope is not approximately equal to the predicted diversity order of the scheme. This, can be explained by the fact that, the SNR values considered here are not large enough, as illustrated in Figure 15. Also, we can observe that the larger the constellation, the slower the absolute value of the slope takes to reach its theoretical value, i.e., the diversity. Consistently with this trend, the larger the window size *W* of FR-BLAST, the slower the absolute value of the slope takes to reach its theoretical value.

In conclusion, as the search window's size increases, the performance characteristic of the proposed detector is improved for SNR values of interest, i.e., 0 dB to 40 dB. As derived above, the diversity order for FR-BLAST does not depend on the size of the search window.

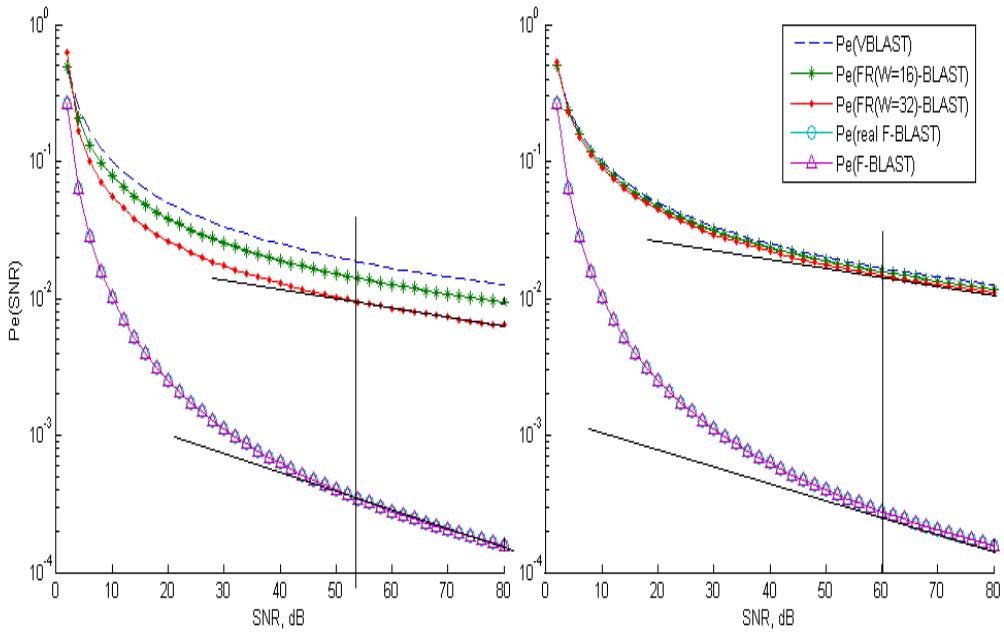


Figure 15 Approximation of the Average Error Probability for Large SNR Values

VI- Performance Study of a Turbo Decoder

In practical wireless systems, to greatly enhance the effective quality of the channel, blocks of information data are encoded at the transmitter side of the radio link, and decoded at the receiver side of the radio link using an error control algorithm, such as the well-known Turbo Codes (TC) [37]. The performance of TCs approaches the Shannon limit, and they have been adopted by the next generation of 3GPP2 / CDMA 2000 Wireless Communication Systems [38].

6.1 Turbo Codes

TC are based on the parallel concatenation of two Recursive Systematic Convolutional (RSC) codes separated by an interleaver [39]. The turbo decoding principle calls for an iterative algorithm involving two component decoders that exchange information in order to improve the error correction performance with increasing numbers of decoding iterations [37].

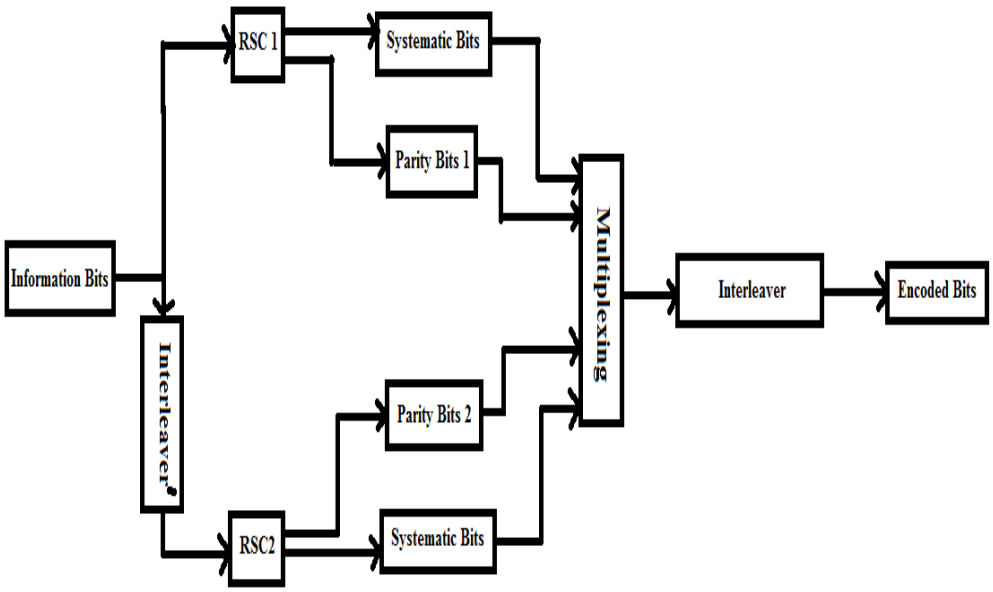


Figure 16 Turbo Encoder Diagram

Figure 16 illustrates the structure of a turbo encoder. The information bits are grouped to form a block of fixed-length information data bits to be encoded. The encoder is formed by two RSC encoders that operate in parallel. Each encoder generates a sequence of n_1 and n_2 coded bits, respectively, from a sequence of i information bits, producing an overall code rate of 'i / (n_1 + n_2 - i)'. To enhance the performance of the decoder, the second encoder processes the information bits in a different order, i.e., the information bits are interleaved or scrambled to obtain a decorrelated version of the same information. The coded bits sequence is comprised of parity bits and systematic bits (original information bits). Finally, the encoders' outputs are multiplexed and interleaved using a predefined interleaver to strengthen the code.

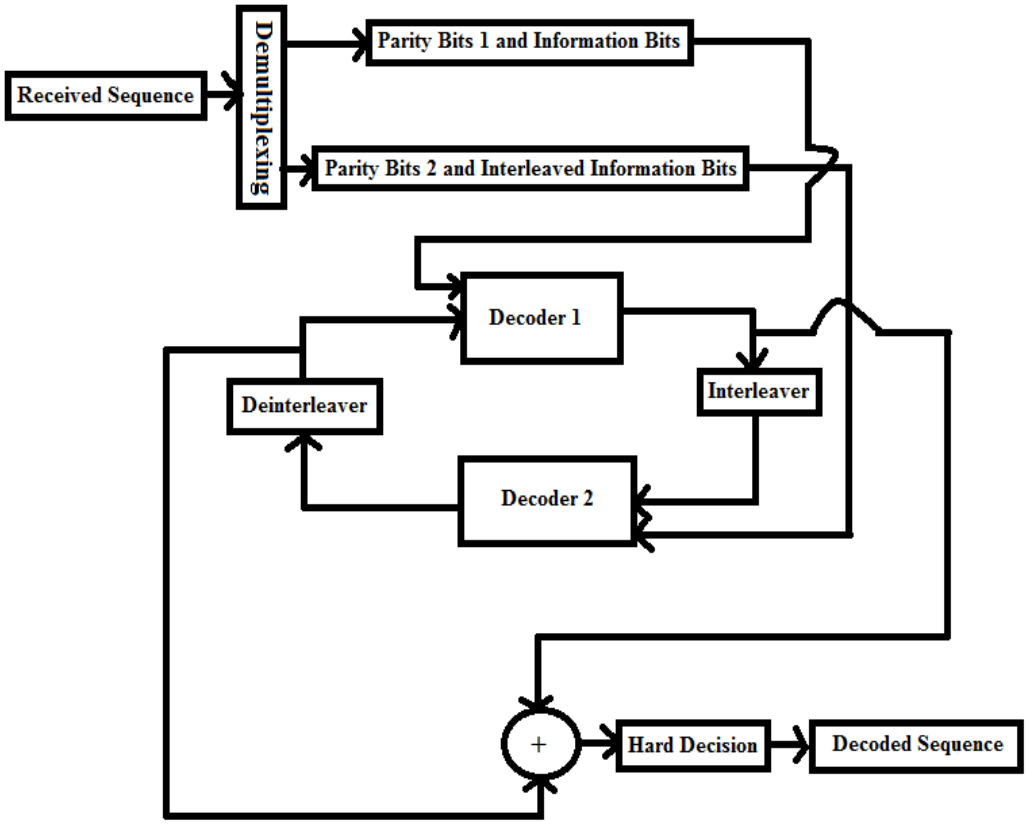


Figure 17 Turbo Decoder Diagram

Figure 17 illustrates the structure of a turbo decoder. The received sequence is comprised of soft information bits and soft parity bits. The soft bits give estimates of bit values along with probability information for each bit. They are typically log-likelihood ratios (LLRs) given to some finite bit precision (e.g., 3, 4, 5 or 6 bits). Maximum Likelihood Detection (MLD) principle, i.e., the comparison of the probability of a received soft bit being a ‘one’ or a ‘zero’, is used to decode TC. The decoder is formed by two Maximum A-posteriori Probability (MAP) decoders that have knowledge of the lattice structure of the encoders. Each MAP decoder receives the original soft information bits and one of the two streams of soft parity bits, and produces a (hopefully) more accurate sequence of soft information and soft parity bits. After a certain number of iterations, the outputs of the decoders are compared by addition, i.e., the LLRs corresponding to the *i*-th bits from the block of soft information bits generated by the first and the second component APP decoder, respectively, are summed to increase or decrease the probability of the *i*-th information bit being a zeros or a one. Finally, a hard decision is made to recover the original sequence of information bits.

6.2 Turbo Codes for MIMO Systems [40]

In [38], the author discusses several applications of TC. In this section, we will briefly present three major applications that retain our attention.

6.2.1 TC Design in Asymmetric Digital Subscriber Line (ADSL)

Asymmetric Digital Subscriber Line (ADSL) is presently the main technology of broadband wireline communications. This transmission model associates a TC with a multicarrier modulation such as QAM [40]. Figure 18 illustrates the structure of an ADSL modem.

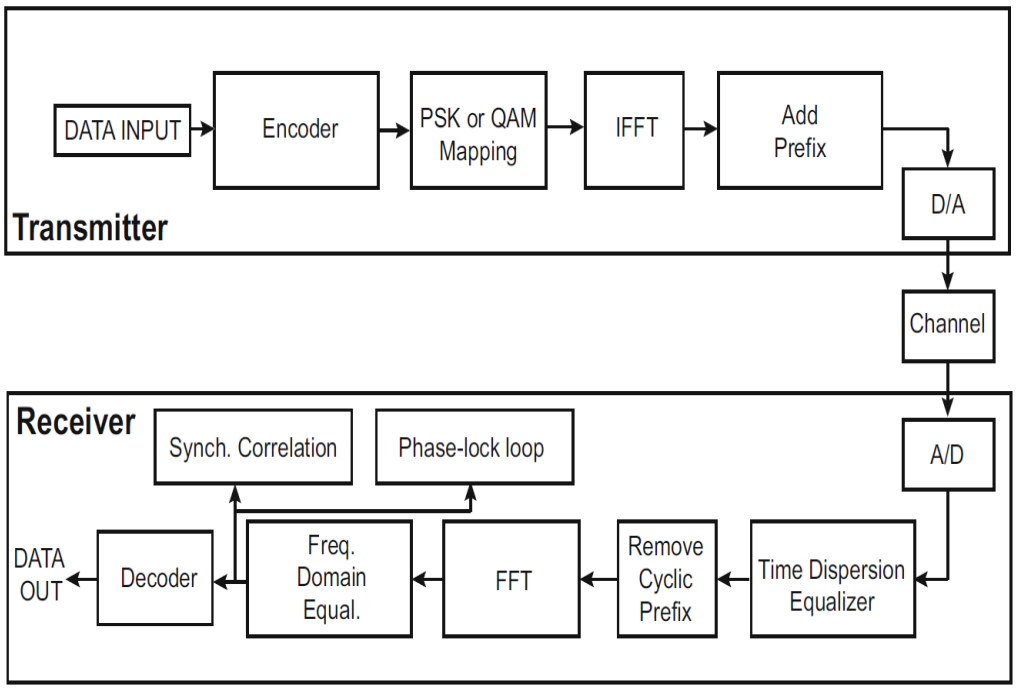


Figure 18 Block Diagram of an ADSL Modem [41]

Observe that there is no direct feedback from the decoder to the encoder. For this application, in order to reduce the effect of impulse noise that corrupts the signal in the twisted pair channel, the original information bits sequence is interleaved. This step tends to break up blocks of erroneous bits to create isolated erroneous bits that can be more easily corrected. The Gray code mapping [42] is the most common assignment of the bits to an M-QAM constellation, where the systematic bits are assigned to the least significant bits and parity bits are assigned to the remaining more significant bits. The turbo decoder utilizes a MAP algorithm for soft decoding, as described in Section 6.1. Since the delay is an important parameter in ADSL transmission, the main focus here is designing the interleavers. Large interleavers are not affordable, consequently medium-sized interleavers which provide sufficiently good error rate performance are of interest [41].

6.2.2 Iterative Decoding for Wireless Communications [43]

Another application of TCs can be found in multiple antenna and Code-Division Multiple Access (CDMA) channels. The CDMA channels are coded independently from each others, whereas the multiple antenna channel is synchronous and its sub-streams can be jointly coded. In many applications, the fading channel can be modeled by the Rayleigh model [44]. For an optimal usage of those channels in high SNR regions, large signal constellations should be used. However, the transmitted signals are correlated by the channel and due to the cardinality of the constellation, it is impractical to search over all possible

candidate signals. A solution to this problem is to create single-stream channels or layers and use iterative demodulation and decoding.

In Iterative Decoding, a soft-output APP decoder plays the role of an outer code and the channel plays the role of an inner code, the first decoder sending back *a posteriori* probabilities of each bit iteratively to the second decoder. Figure 19 illustrates the structure of such a decoder for a CDMA system.

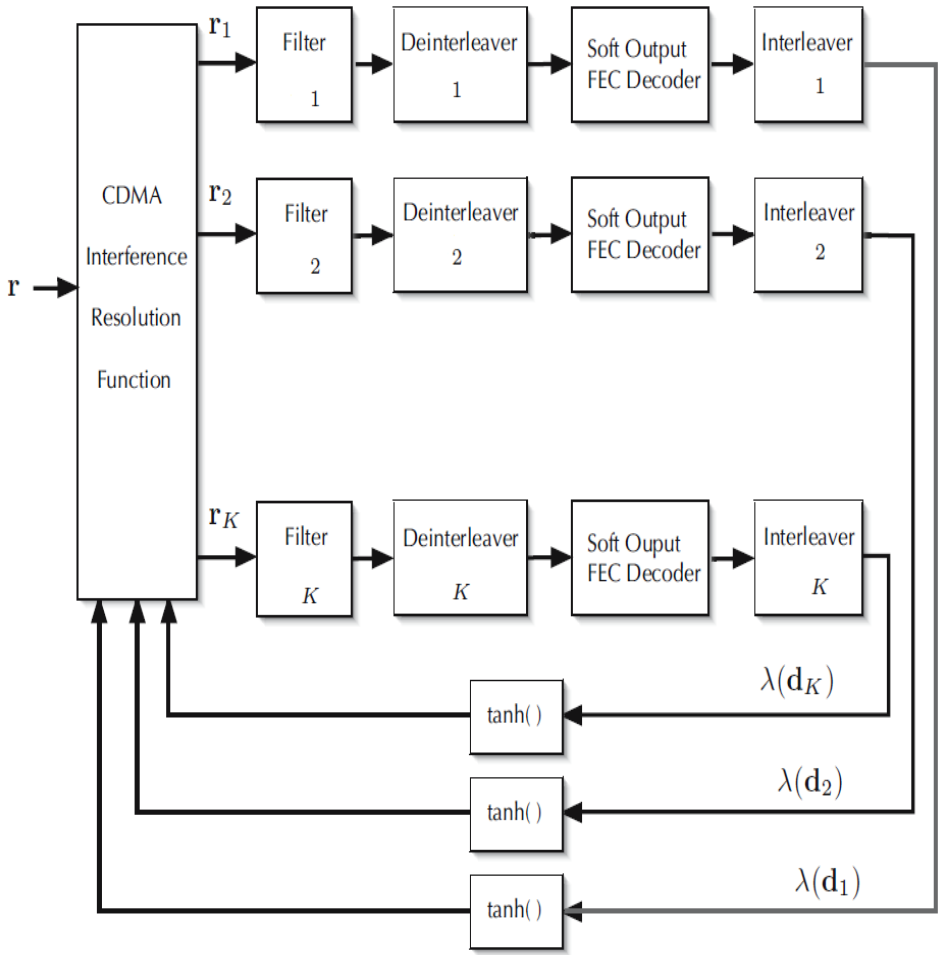


Figure 19 Iterative Multiuser Decoder with Soft Information Exchange [43]

Here FEC stands for Forward Error Control, and $\lambda(d_i)$ represents the symbol-wise log-likelihood ratio from the *i*-th layer.

The complexity of this decoder grows only linearly with the number of layers, consequently a practical usage can be found for large constellation signals. This technique is being applied in high-speed MIMO wireless communications [45], which is the topic of the next section.

6.2.3 High-speed MIMO Wireless Communications [45]

Turbo-MIMO is a class of MIMO systems based on the principle of turbo processing. An example is the Space-Time Bit-Interleaved Coded Modulation (ST-BICM) architecture, which provides very good performance with receivers using iterative detection and decoding, such as the Minimum Mean-Squared based Soft-Interference Cancellation (SIC-MMSE).

In the ST-BICM architecture, the detector and the channel decoder use soft data at both the inputs and outputs. Since detection and channel decoding are processed separately, the complexity of the detectors grows linearly with the number of receiver antennas. The more iterations in the detector/decoder loop, the better is the error rate performance. What is more, it has been proven that the performance of ST-BICM exceeds that of encoded non-iterative MIMO systems such as V-BLAST [45].

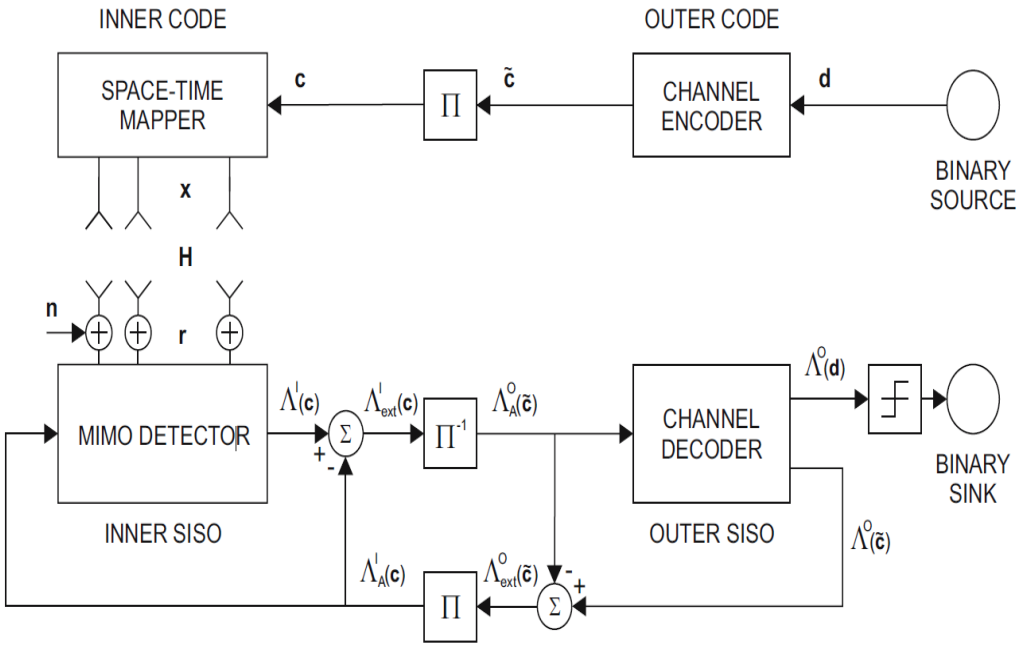


Figure 20 Block Diagram of a MIMO System Employing ST-BICM and an Iterative Receiver [45]

In Figure 20, ***d*** is the information bits sequence, $\tilde{\boldsymbol{c}}$ the output of the encoder, ***c*** the interleaved sequence of encoded bits, ***x*** the symbol vector, **H** the MIMO channel, ***r*** the received signal, ***n*** the additive noise vector, Λ^X_{ext} the extrinsic LLRs, Λ^X_{A} the *a priori* LLRs, and Λ^X the *a posteriori* LLRs, with ^{X} representing I and O, i.e., inner decoder and outer decoder, respectively

Figure 20 illustrates the structure of the ST-BICM MIMO scheme. The transmitted information bits sequence is first encoded, then interleaved and finally converted into parallel substreams (i.e., one FEC block is used to encode the original information bits). Each substream is mapped onto a sequence of constellation symbols that is transmitted from a separate antenna [45]. The decoding process is separated into two stages:

- The inner decoder, i.e., the detector, generates extrinsic LLRs for the received coded bits sequence and makes them available as *a priori* information to the outer decoder, after deinterleaving.
- The outer decoder, generally a turbo decoder such as the one described in Section 6.1, uses the *a priori* information and generates extrinsic information on both the coded bits and the information bits and feeds back the extrinsic information on the coded bits to the inner decoder after interleaving.

These stages are repeated until a pre-defined criterion is achieved, then hard decisions are made to compute the decoded bits sequence.

The iterative receiver generally improves the soft decisions after iteration. APP decoders are optimal for the inner decoder [45], but they are impractical due to their higher computational complexity, thus sub-optimal detectors such as SIC-MMSE [47] are of interest.

In SIC-MMSE, when detecting the symbol transmitted through one layer, this layer is assumed to receive interference from the n_t-1 remaining layers and the additive noise. *A priori* information available as an input, is used to estimate and cancel interference from the remaining layers, and to supress the residual interference and noise given the MMSE criterion. In [48], a family of detectors (B-Chase) based on the parallel decoder (PD) [21] scheme, which was discovered at the very end of the research project, presents a comparison study of such detection scheme in an iterative decoding architecture.

6.3 System Model

The proposed Turbo is illustrated in Figure 21. We used Simulink to model (we stringed together function blocks) and simulate all of the systems in the transmission chain associated with various soft detectors as described below.

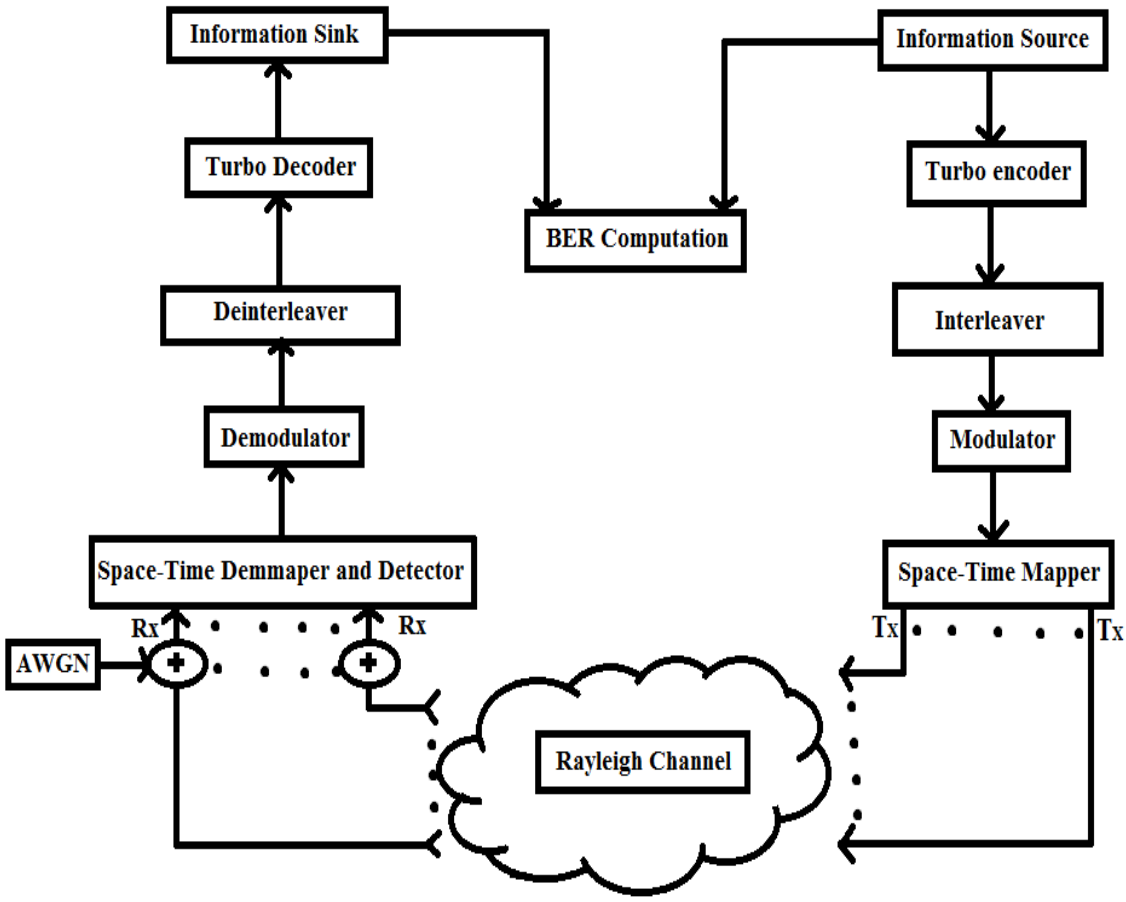


Figure 21 Block Diagram of the Turbo Decoder Model

The number of blocks, the encoder's interleaver size, and the trellis structure of the encoder have been chosen according to the parameters provided in Berou's original paper [37].

We modeled the transmission of a sequence of equiprobable 0*s* and 1*s*, in 128 blocks containing 65536 (256 * 256) information bits each. The turbo

encoder block contains a parallel concatenation of two RSC rate-1/2 encoders, whose outputs are defined by the generator polynomials 37 and 21 in octal, with a constraint length of 5, for an overall code rate of 1/3, as described in Section 6.1. The modulator accepts the encoded sequence of bits and converts them into a sequence of *M*-QAM signals (*M* = 16, 64 or 256). These constellation signals are grouped to form sample vectors of length *m* (for the purposes of simulation). Each component of a sample is associated with a different transmitter antenna. The block of sample vectors is divided into frames of length 2**M*, which are then transmitted through a noisy environment assumed to be constant for the duration of the frame in order to model Rayleigh fading.

At the receiver side, the transmitted signal vector is corrupted with AWGN, and the block of received signal vectors is detected using a soft-detector. The soft detector produces soft information bits that are fed into the turbo decoder. Thus, an implementation of the equivalent soft-output detector is required. A direct implementation of the soft output information is very complex. [49] proposes a simple approach that reduces the complexity without loss of performance. The idea is to demap the received signal into soft bits which have the same sign as provided by a hard detector and whose absolute value indicates the reliability of the decision [49]. All the hard detectors mentioned in this work use a linear filter (following the MMSE criterion) in order to estimate the position on the constellation diagram, of the symbol transmitted on the current detected layer, then a slicer function, picks the closest symbol from the constellation, finally hard bits are produced following the gray mapping of the constellation

symbols. In the case, soft-outputs are of interest, [49] proposed to use decision regions for the real and the imaginary part of the estimated position on the constellation diagram ($\underline{b} = b_Q + i * b_I$, as illustrated in Figure 22).

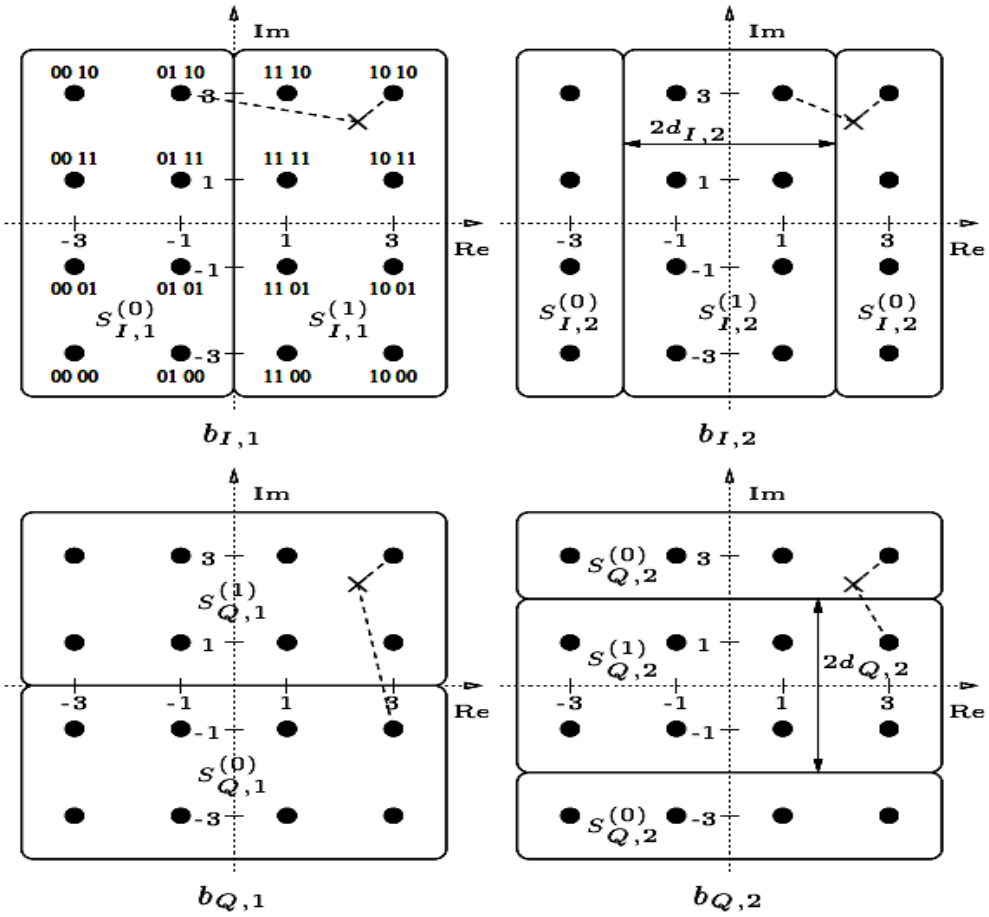


Figure 22 Partition of the 16-QAM constellation [49]

($S^{(0)}_{I;k}$; $S^{(1)}_{I;k}$) for the component in phase b_I, and ($S^{(0)}_{Q;k}$; $S^{(1)}_{Q;k}$) for component in quadrature b_Q, in the case of the 16-QAM constellation. In practical, the soft-outputs associated with the component in phase are defined by:

$$S_{I,1} = \begin{cases} b_I \\ 2 * (b_I - 1) \\ 2 * (b_I + 1) \end{cases}$$

$$S_{I,2} = - | b_I | + 2$$

The $S_{Q,k}$ functions for the in quadrature component are the same, with b_I replaced by b_Q.

Finally, the turbo decoder generates the output information bits as described in Section 6.1.

This system model uses a single turbo encoder since all antenna signals employ the same modulation. The proposed family of Turbo F-BLAST detectors does not have an iterative processing between the soft detector and the turbo decoder in order to reduce the processing delay. Note that the iterative processing is being replaced by the parallel search on the first layer to be detected.

The next section presents a fair comparison among all the corresponding soft detectors.

6.4 Results

We simulated the same sequence of information bits in a Rayleigh fading environment through a 4×4 MIMO system, using different modulation schemes, such as 16-QAM, 64-QAM and 256-QAM.

6.4.1 16-QAM

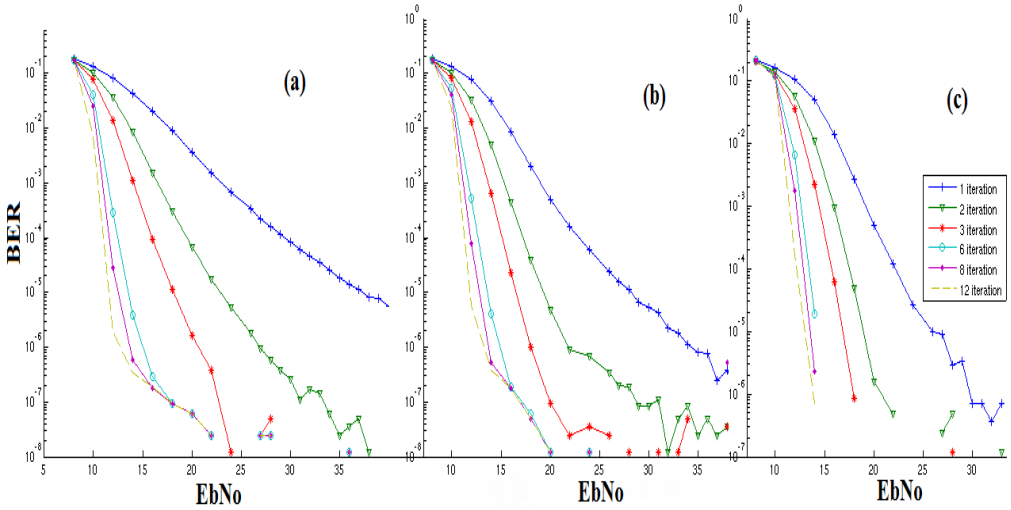


Figure 23 BER vs. EbNo of a 16-QAM Turbo MIMO Model Associated with Various Soft Detectors: MMSE (a), Real-Valued F-BLAST (b), and F-BLAST (c)

Figure 23 shows that after a certain number of iterations, the turbo model associated with various soft detectors seems to exhibit the same BER performance vs. the bit energy to noise ratio (EbNo). This performance limit on the BER vs. EbNo characteristic is known as the turbo cliff and is represented by a sudden drop of the BER [50]. Also, observe that for a small number of iterations (three or four), the soft F-BLAST and the soft real-valued F-BLAST detectors have BER performance which exceeds that of MMSE. But, if a system can afford up to 12 iterations, a soft MMSE detector provides good performance, as shown in Figure 24.

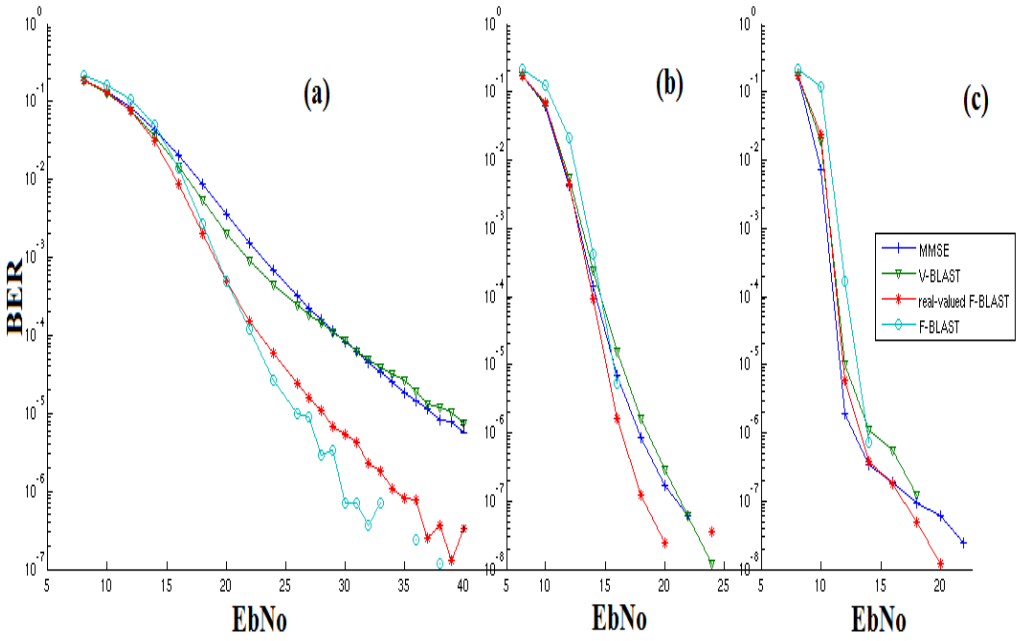


Figure 24 BER vs. EbNo (dB) of a 16-QAM Turbo MIMO Model Associated with Various Soft Detectors for: one iteration (a), four iterations (b), and twelve iterations (c)

These observations are highlighted in Table 5.

	One iteration				Four iterations				12 iterations			
BER	mmse	vb	real fb	fb	mmse	vb	real fb	fb	mmse	vb	real fb	fb
10^{-2}	18	17	16	17	12	12	12	13	10	10	10	11
10^{-4}	30	30	23	22	14	15	14	15	11	11	11	12
10^{-6}	/	/	34	30	17	18	16	/	14	13	12.5	14

Table 5 Corresponding EbNo (dB) for Selected BERs for Various Soft Detectors in a 4×4 16-QAM Turbo MIMO System

Here ‘vb’, ‘real fb’ and ‘fb’ stand for V-BLAST, real-valued F-BLAST and F-BLAST, respectively.

The soft F-BLAST detector uses a search window containing only 16 constellation signals (i.e., the entire constellation), thus it is very encouraging for further research involving a better design of the interleaver, and extensions to iterative decoding between the turbo decoder and the soft detector. While transmitting and detecting 16-QAM signals, the real-valued detector, which has a lower computational complexity, exhibits approximately the same performance compared to that of the complex-valued detection, thus real-valued detection should be of interest for transmission over a turbo MIMO system.

6.4.2 64-QAM

The top chart of Figure 25, i.e., (a), (b), (c) and (d), showing BER plots for 64-QAM signals leads to the same conclusions as for 16-QAM signals. The soft detectors used for the turbo MIMO model exhibits approximately the same performance after a certain number of iterations (eg. four or more). But the soft F-BLAST and the soft real-valued F-BLAST detectors show better performance for fewer than four iterations of the turbo decoder. Further, Figure 25 & 26 show that the family of soft FR-BLAST detectors have very close performance with either a search window of nine or 16 constellation symbols. However, when starting with the second weakest layer, the BER performance is for some reason worse, as shown in Figure 27. Figure 27 also shows that the family of soft FR-BLAST detectors provides good performance with either a search window size of 9 or 16, starting with any layers, but the second weakest one in term of strength. Note that the performance of the family of soft FR-BLAST detectors is a little better than that of the soft real-valued F-BLAST detector. Thus, in order to reduce the

computational complexity of the receiver, and for turbo MIMO systems that require fewer iterations in order to reduce the power consumption and/or the recovery time, the family of soft FR-BLAST detectors may be of interest for the detection scheme.

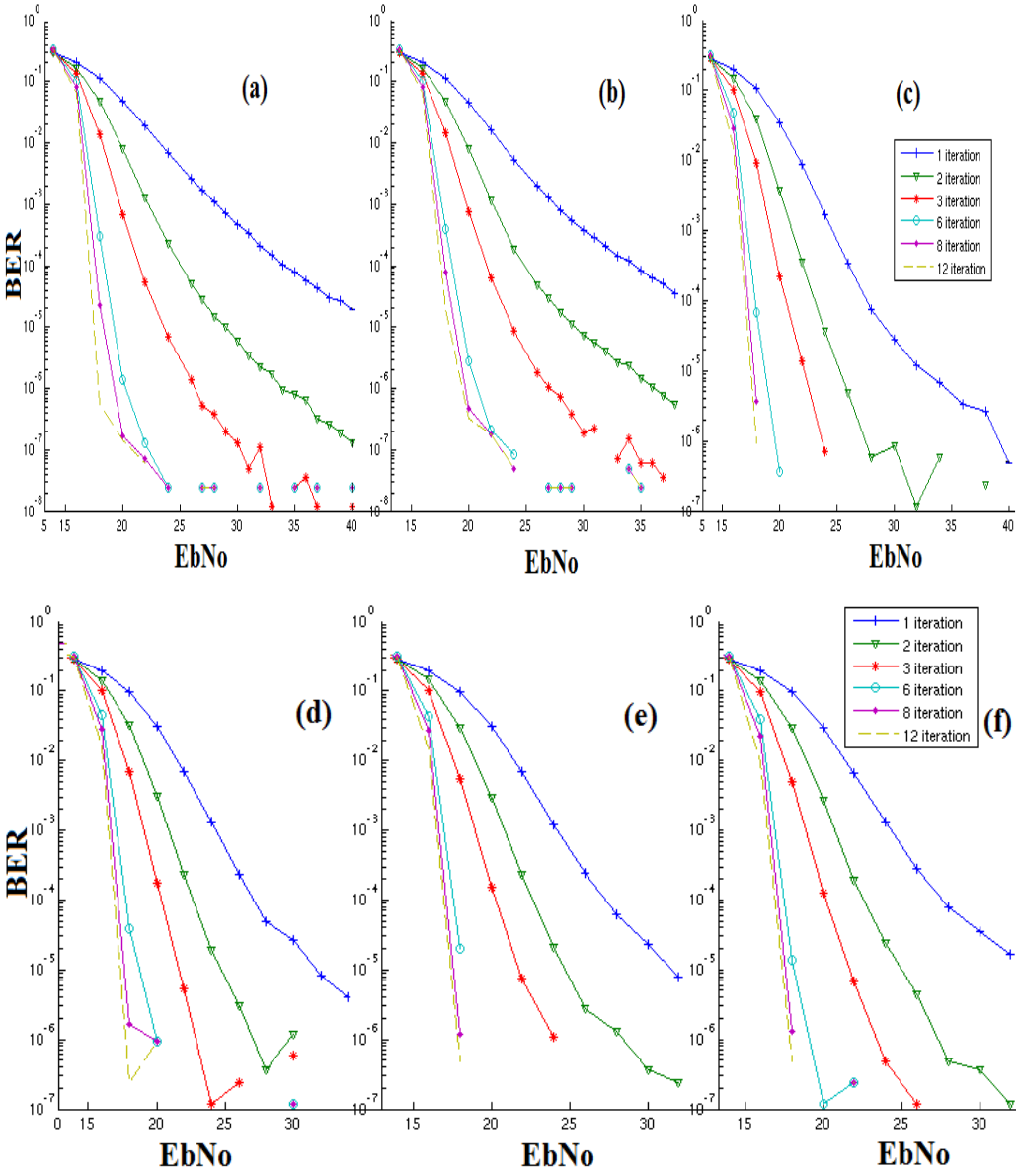


Figure 25 BER vs. EbNo of a 64-QAM Turbo MIMO Model Associated with Various Soft Detectors, MMSE (a), V-BLAST (b), Real-Valued F-BLAST (c), FR(9,S1)-BLAST (d), FR(9,S2)-BLAST (e), FR(9,W2)-BLAST (f)

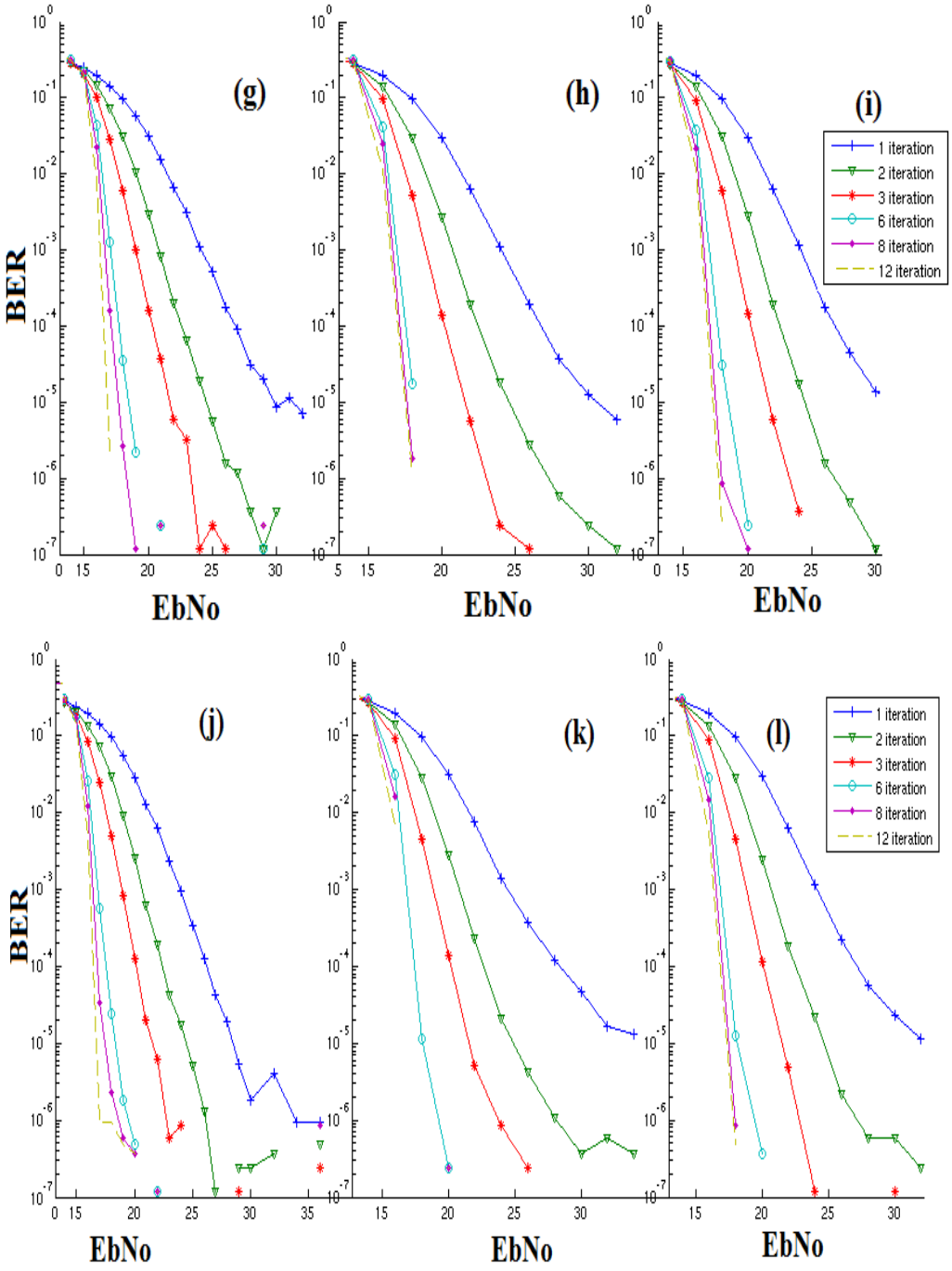


Figure 26 BER vs. EbNo of a 64-QAM Turbo MIMO Model Associated with Various Soft Detectors, FR(16,S1)-BLAST (g), FR(16,S2)-BLAST (h), FR(16,W2)-BLAST (i), F-BLAST (j), FR(9,W1)-BLAST (k), and FR(16,W1)-BLAST (l)

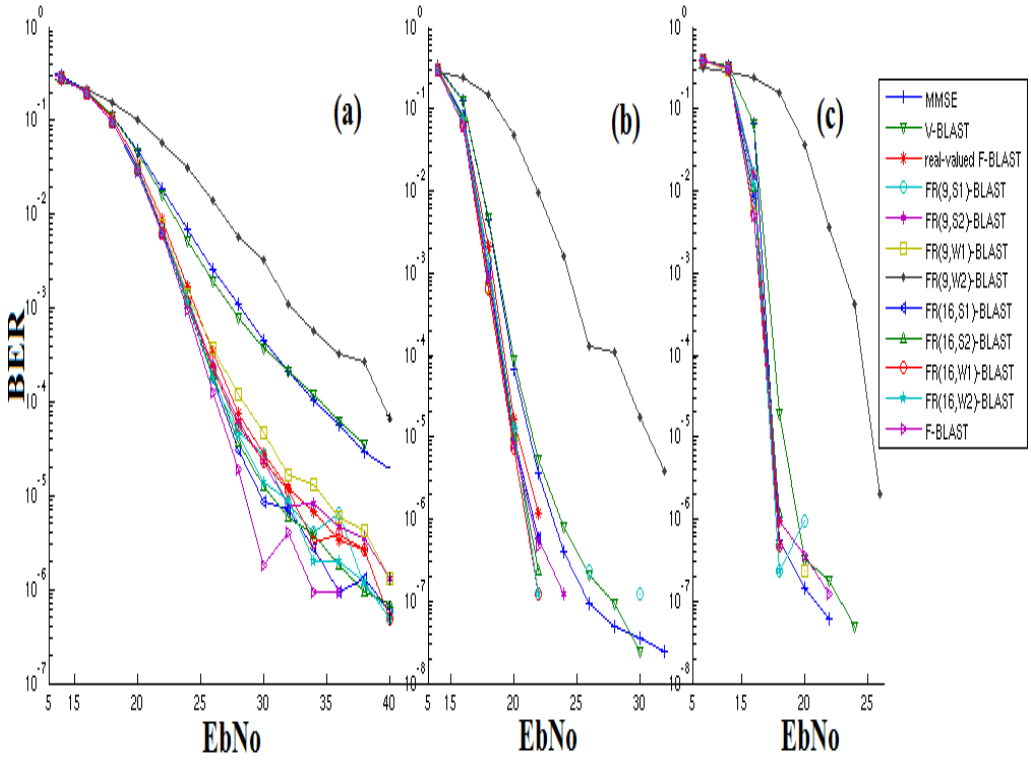


Figure 27 BER vs. EbNo (dB) of a 64-QAM Turbo MIMO Model Associated with Various Soft Detectors for: one iteration (a), four iterations (b), and twelve iterations (c)

6.4.3 256-QAM

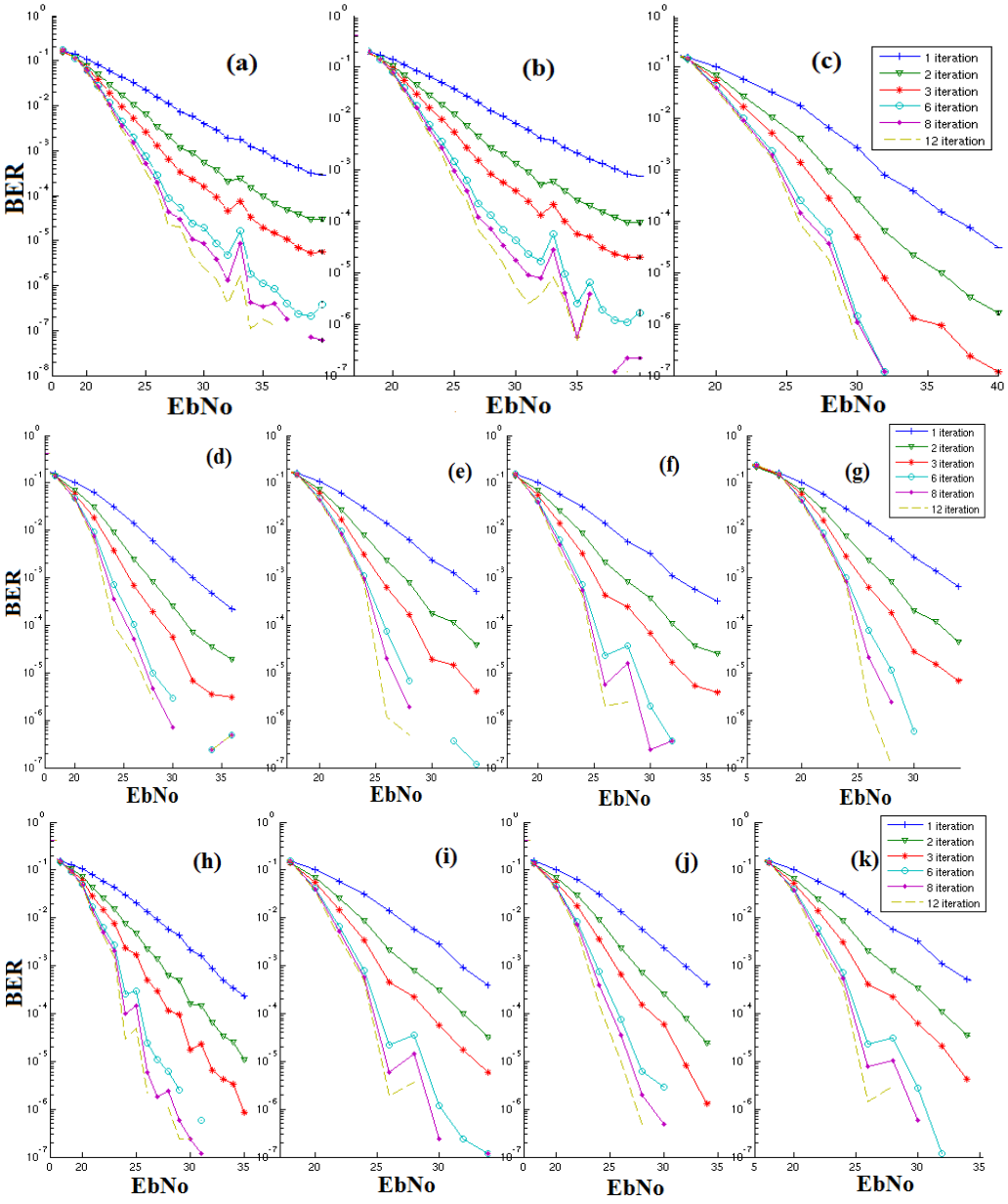


Figure 28 BER vs. EbNo of a 256-QAM Turbo MIMO Model Associated with Various Soft Detectors, MMSE (a), V-BLAST (b), Real-Valued F-BLAST (c), FR(9,S1)-BLAST (d), FR(9,S2)-BLAST (e), FR(9,W2)-BLAST (f), FR(9,W1)-BLAST (g), FR(16,S1)-BLAST (h), FR(16,S2)-BLAST (i), FR(16,W2)-BLAST (j), and FR(16,W1)-BLAST (k)

Figure 28 shows that the soft detectors presented here have approximately the same performance with respect to BER. After four iterations, their BER performance characteristics seem to have reached the turbo cliff limit. Figure 29 shows that for EbNo values less than 30 dB, the soft detectors provide very similar results, but above 30 dB it is not clear which one is better since insufficient error statistics led to eratic and hence unreliable behaviour of the BER performance characteristic. Simulation times were unfortunately become very long (e.g. many days).

A simple soft MMSE detector seems to provide a BER performance very close if not better than that of more complicated soft detectors, such as the family of soft FR-BLAST detectors. We think that MMSE with the current parameters will benefit more from the effects of turbo decoding. Nevertheless, a simulation study with more information bits might be required to accurately access the performance of the different soft detectors for large Eb/No values.

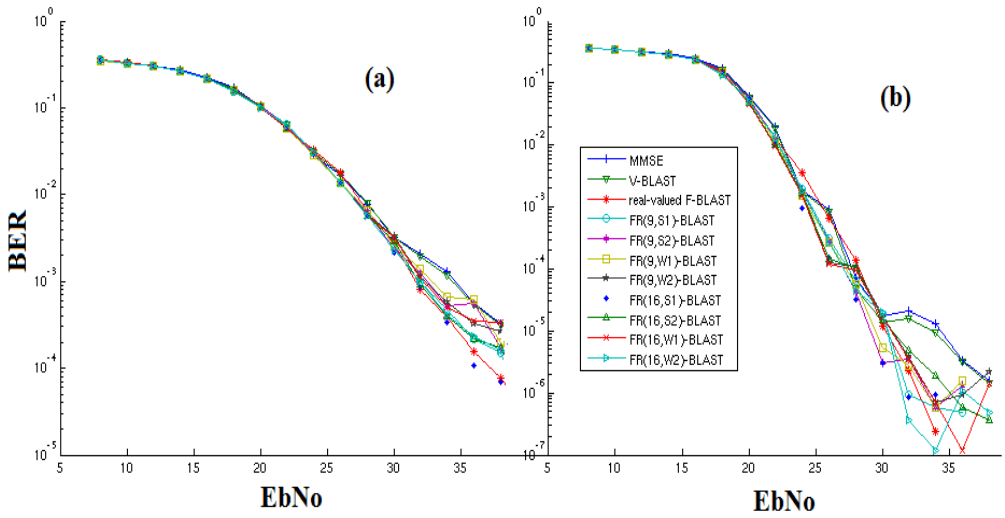


Figure 29 BER vs. EbNo (dB) of a 256-QAM Turbo MIMO Model Associated with Various Soft Detectors for: one iteration (a), and four iterations (b)

VII- Conclusions and Future Directions

7.1 Conclusions

New standards, such as LTE, require high-speed, small and power-efficient devices at both ends of a radio communication link. This thesis presents a new family of detectors that tries to meet this demand.

We first reviewed a detection scheme, F-BLAST, which has been proven to give optimal performance for SNR values ranging from 0 dB to 40 dB, in an uncoded MIMO system. But due to its higher computational complexity, we proposed and investigated a new family of detectors (FR-BLAST) and a detection scheme based on real-valued decomposition (real-valued F-BLAST) that are computationally efficient, near-optimal in term of BER performance, and should be of great interest for larger constellations. The main idea being the use of limited parallelism to improve the error rate performance, and to reduce the recovering time, for a device using such technique.

A study of the computational complexity of the proposed detectors (FR-BLAST and real-valued F-BLAST) and those found in the literature (MMSE, V-BLAST, F-BLAST and ML), confirms that the real-valued F-BLAST scheme requires fewer arithmetical operations (multiplications and additions), but many more reciprocals. Thanks to its parallelizable structure, FR-BLAST does not require many cycles even when using the target number of 16 parallel computational threads. Besides, theoretical analysis shows that the diversity order of the new schemes is identical to that of MMSE and V-BLAST, although the

BER performance characteristic does not indicate such behaviour at a relatively high SNR region (up to 40 dB) when detecting large constellation signals such as 16-QAM, 64-QAM, 128-QAM and 256-QAM. Motivated by these results, we investigated the combination of the novel detection schemes with a simple turbo MIMO system without iterative exchange of information between the soft detector and the turbo decoder.

Unfortunately, the results obtained are not as promising for the new detectors as for the transmission of uncoded information. We observed remarkable performance for our soft detectors for fewer than four iterations of the turbo decoder when detecting 16-QAM and 64-QAM constellation signals. But for 256-QAM constellation signals, and in general for more than four iterations, a simple MMSE soft detector seems to be good enough, for the proposed turbo MIMO model. The effect of the Turbo Code dominates the performance of the detectors.

At this point, further modifications and implementations are to be investigated to have a fair comparison of the various schemes presented for the purpose of this work.

7.2 Future Directions

Further investigation should focus on the turbo coded scheme and Low-Density Parity Check (LDPC) since TCs and LDPC codes are widely used in the industry, and since they provide so much coding gain error for MMSE detectors. The design of the interleavers and turbo encoders, that are suitable for Successive Interference Cancellation (SIC) systems, is of great interest. Also, since iterative decoding has proven to have excellent performance for detectors using the MMSE criterion, one might consider investigating such a system architecture. In addition, in order to reduce the simulation time and to present results that are suitable for practical usage, an implementation on hardware and the use of an industrial fading model with a simulated Gaussian noise generator should be considered.

References

[1] B. C. Levy, *Principles of Signal Detection and Parameter Estimation*, Springer, 2008, ch. 4, pp. 113-130.

[2] Wolniansky P.W., Foschini G.J., Golden G.D., Valenzuela R.A., "*V-BLAST: An architecture for realizing very high data rates over the rich-scattering wireless channel*," International Symposium on Signals, Systems and Electronics (ISSSE'98), 1998, pp. 295-300.

[3] Hassibi B. and Vikalo H., *On the Expected Complexity of Sphere Decoding*, Conference Record of the 35th Asilomar Conference on Signals, Systems and Computers, 2001, vol. 2, pp. 1051-1055.

[4] Wenjie Jiang, Yusuke Asai, and Shuji Kubota, *Tree Search Based Approximate Maximum Likelihood Detector for Spatial Multiplexing Systems*, The 18th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (IMRC'07), 2007, pp. 1-5.

[5] Ron Rausch and Mark Buffo, *Developing Strategies for MIMO Testing*, Microwave journal, March 2009, vol. 52, no. 3, pp. 94.

[6] Benny Bing, *Broadband Wireless Access*, Kluwer Academic Publishers, ch. 1, pp. 9-10.

[7] Mohinder Jankiraman, *Space-Time Codes and MIMO Systems*, Artech House Publishers, 1st ed., August 2004, pp. 23-32.

[8] Benjamin Baumgartner, *MIMO*, Wikimedia Commons, November 2005.

[9] Bob O'Hara, *IEEE 802 Working Group & Executive Committee Study Group Home Pages*, IEEE 802 Working Group, July 2009.

[10] S. Benedetto and G. Montorsi, *Design of Parallel Concatenated Convolutional Codes*, IEEE Transactions on Communications, 1996, vol. 44, no. 5, pp. 591-600.

[11] Splash, *Constellation diagram for Gray coded 16-QAM*, Wikimedia Commons, October 2006.

[12] Lizhong Zheng and David N. C. Tse, *Diversity and Multiplexing: A Fundamental Tradeoff in Multiple-Antenna Channels*, IEEE Transactions on Information Theory, May 2003, vol. 49, no. 5, pp. 1073-1096.

[13] J. H. Kotecha and A. M. Sayeed, *Training Signal Design for Optimal Estimation of Correlated MIMO Channels*, IEEE Transactions on Signal Processing, 2004, vol. 55, no. 2, pp. 546-557.

[14] J.G. Proakis, *Digital Communications*, New York: McGraw-Hill, 4th edition, 2001, ch. 7.

[15] B. C. Levy, *Principles of Signal Detection and Parameter Estimation*, Springer, 2008, ch. 4, pp. 113-130.

[16] Manzoor, Rana Shahid Gani, Regina Jeoti, Varun Kamel, Nidal Asif, Muhammad, *Implementation of FFT Using Discrete Wavelet Packet Transform (DWPT) and Its Application to SNR Estimation in OFDM Systems*, International Symposium on Information technology (ITSim'08), 2008, vol. 4, pp. 1-6.

[17] Norman C. Beaulieu, Andrew S. Toms and David R. Pauluzzi, *Comparison of Four SNR Estimators for QPSK Modulation*, IEEE Communications Letters, February 2000, vol. 4, no. 2, pp. 43-45.

[18] Cheng Wang, Edward K. S. Au, Ross D. Murch, Wai Ho Mow, Roger S. Cheng and Vincent Lau, *On the Performance of the MIMO Zero-Forcing Receiver in the Presence of Channel Estimation Error*, IEEE Transactions on Wireless Communications, March 2007, vol. 6, no. 3.

[19] Wolniansky P.W., Foschini G.J., Golden G.D., Valenzuela R.A., "*V-BLAST: An architecture for realizing very high data rates over the rich-scattering wireless channel*," International Symposium on Signals, Systems and Electronics (ISSSE'98), 1998, pp. 295-300.

[20] G. D. Golden, J. G. Foschini, R. A. Valenzuela, and P. W. Wolniansky, *Detection Algorithm and Initial Laboratory Results using V-BLAST Space-Time Communication Architecture*, Electronics Letters, January 1999, vol. 35, no. 1, pp. 14-15.

[21] Yaun Li and Zhi-Quan Lo, *Parallel Detection for V-BLAST System*, International Conference on Communications, 2002, vol. 1, pp. 340-344.

[22] S. Fouladi Fard, A. Alimohammad, and B. F. Cockburn, *Improved MIMO Detection Algorithm with Near-Optimal Performance*, IET Electronics Letters, 18th June 2009, vol. 45, no. 13, pp. 675-677.

[23] Arsene Pankeu Yomi, Bruce Cockburn, *Near-Optimal and Efficient Multiple-Input Multiple-Output Detectors for Large Constellations*, Banff

Summer School 2010 on Communication and Information Theory, Presentations Report, 2010.

[24] Arsene Pankeu Yomi, Bruce Cockburn, *Enhanced MIMO Detection with Parallel V-BLAST*, IEEE *Pacific Rim Conference* on Communications, Computers and Signal Processing (PACRIM'11), 2011, pp. 702-707.

[25] Jeffrey G. Andrews, *Interference Cancellation for Cellular Systems: A Contemporary Overview*, The University of Texas Austin, IEEE Wireless Communications, April 2005, vol. 2, no. 2, pp. 1284-1536.

[26] Allan Agresti, *An Introduction to Categorical Data analysis*, John Wiley and Sons, 1996, ch. 25.

[27] R.F.H. Fischer and C. Windpassinger, *Real versus Complex-Valued Equalization in V-BLAST Systems*, IEEE Electronic Letters, 6th March 2003, vol. 39, no. 5, pp. 470-471.

[28] H. H. Beverage and H. O. Peterson, *Diversity Receiving System of R.C.A. Communications, Inc., for Radiotelegraphy*, Proceedings of the IRE, April 1931, vol. 19, no. 4, pp. 531-561.

[29] Georges B. Arfken and Hans J. Weber, *Mathematical Methods for Physicists*, sixth edition, 2005, Elsevier Academic Press, p. 179 & 186.

[30] Andrea Goldsmith, *Wireless Communications*, Cambridge University Press, ch. 10.5, pp. 335-337.

[31] Ezio Biglieri, Robert Calderbank, Anthony Constantinides, Andrea Goldsmith, Arogyaswami Paulraj, H. Vincent Poor, *MIMO Wireless Communications*, Cambridge University Press, ch. 4.2.3, pp. 145-146.

[32] John R. Barry, Edward A. Lee, David G. Messerschmitt, *Digital Communication*, Springer, ch. 11.5, pp. 545-548.

[33] Yi Jiang, Xiayu Zheng and Jian Li, *Asymptotic Performance Analysis of V-BLAST*, IEEE GLOBECOM, 2005, pp. 3882-3996.

[34] W. J. Choi, R. Negi, and J. M. Cioffi, *Combined ML and DFE Decoding for the V-BLAST System*, IEEE International Conference on Communications (ICCC'00), June 2000, vol. 3, pp. 25-29.

[35] G. Kalyana Krishnan and V. Umapathi Reddy, *High Performance Low Complexity Receiver for V-BLAST*, IEEE workshop 8th on Signal Processing Advances in Wireless Communications, 2007, pp. 1-5.

[36] D. G. Brennan, *Linear Diversity Combining Techniques*, Procedings of the IRE, vol. 47, pp. 1075-1102.

[37] C. Berrou, A. Glavieux, P. Thitimajshima, *Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes*, Proc. ICC'93, Geneva, May 1993, pp. 1064–1070.

[38] Third Generation Partnership Project 2 (3GPP2), *Physical Layer Standard for CDMA2000 Spread Spectrum Systems, Release D*, 3GPP2 C.S0002-D, Version 1.0, February 2004.

[39] Claude Berrou, *Codes and Turbo Codes*, Springer, ch. 5.2.

[40] Keattisak Spripimanwat, *TURBO CODE APPLICATIONS: A Journey from a Paper to Realization*, Springer, 2005.

[41] Keattisak Spripimanwat, *TURBO CODE APPLICATIONS: A Journey from a Paper to Realization*, Springer, 2005, ch. 4, pp. 67-79.

[42] K Meena, *Principles of Digital Electronics*, PHI Learning Pvt. Ltd., 2009, pp. 34-37.

[43] Keattisak Spripimanwat, *TURBO CODE APPLICATIONS: A Journey from a Paper to Realization*, Springer, 2005, ch. 6, pp. 123-124 & 127-139.

[44] David Tse, Pramod Viswanath, *Fundamentals of Wireless Communication*, Cambridge University Press, 2005, ch 2.4, pp 34-37.

[45] Keattisak Spripimanwat, *TURBO CODE APPLICATIONS: A Journey from a Paper to Realization*, Springer, 2005, ch. 9, pp. 223-228 & 233-234.

[46] G. D. Golden, J. G. Foschini, R. A. Valenzuela, and P. W. Wolniansky, *Detection Algorithm and Initial Laboratory Results using V-BLAST Space-Time Communication Architecture*, Electronics Letters, 1999, vol. 35, pp. 14–15.

[47] M. Sellathurai and S. Haykin, *Turbo-BLAST for Wireless Communications: Theory and Experiments, IEEE Trans. Signal Processing*, October 2002, vol. 50, no. 20, pp. 2538–2546.

[48] Deric W. Waters and John R. Barry, *The Chase Family of Detection Algorithms for Multiple-Input Multiple-Output Channels*, IEEE Transactions On Signal Processing, February 2008, vol. 56, no. 2, pp. 739-747.

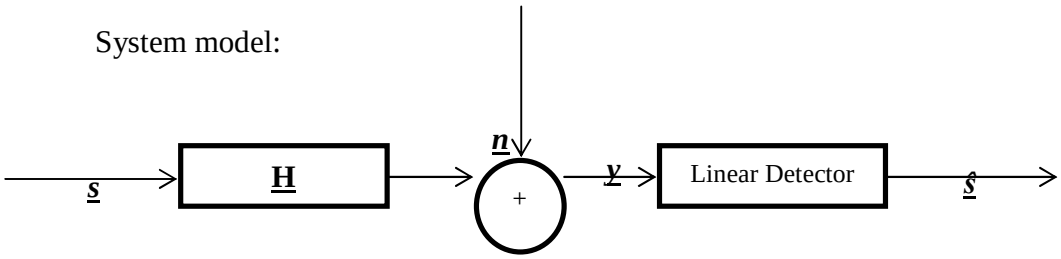
[49] Filippo Tosato, Paola Bisaglia, *Simplified Soft-Output Demapper for Binary Interleaved COFDM with Application to HIPERLAN/2*, Imaging Systems Laboratory, HP Laboratories Bristol, HPL-2001-246, October 2001.

[50] Christian B. Schlegel, Lance C. Perez, *Trellis and Turbo Coding*, John Wiley & Sons, 2004, ch. 10.1.

Appendices

1. Minimum Mean Square Error (MMSE) Conditioning Matrix [14]

System model:



$$\underline{\boldsymbol{y}}^{T} = \underline{\boldsymbol{s}}^{T} * \underline{\mathbf{H}} + \underline{\boldsymbol{n}}^{T} \quad \text{Or} \quad \underline{\boldsymbol{y}} = \underline{\mathbf{H}}^{\mathbf{T}} * \underline{\boldsymbol{s}} + \underline{\boldsymbol{n}} \quad \text{Or} \quad \underline{\boldsymbol{y}}^{H} = \underline{\boldsymbol{s}}^{H} * \mathbf{H}^{*} + \underline{\boldsymbol{n}}^{H} \quad (1)$$

Where:

- $\underline{\boldsymbol{y}}$ is the received noisy signal vector
- $\underline{\boldsymbol{s}}$ is the transmitted signal vector
- $\underline{\boldsymbol{n}}$ is an AWGN vector of length *m*
- $\underline{\hat{\boldsymbol{s}}}$ is the detected signal vector of length *m*
- $\underline{\mathbf{H}}$ is the m-by-m channel matrix

The goal is to find a linear approximation $\underline{\hat{\boldsymbol{s}}}$, of the transmitted symbol vector $\underline{\boldsymbol{s}}$ given the received symbol vector $\underline{\boldsymbol{y}}$, such that:

$$\underline{\hat{\boldsymbol{s}}} = \underline{\mathbf{G}}^{\mathbf{T}} * \underline{\boldsymbol{y}} + \underline{\boldsymbol{b}} \quad (2)$$

Where $\underline{\mathbf{G}}^{\mathbf{T}}$ and $\underline{\boldsymbol{b}}$ are constants to be determined. The channel matrix $\underline{\mathbf{H}}$ is assumed to be known or perfectly estimated.

Let $\underline{\boldsymbol{e}} = \underline{\boldsymbol{s}} - \underline{\hat{\boldsymbol{s}}}$ the error vector, $\underline{\mathbf{A}} = \underline{\mathbf{G}}^{\mathrm{T}}$, and E{.} the expected value operator.

$$\underline{\boldsymbol{m}}_{\boldsymbol{e}} = \underline{\boldsymbol{m}}_{\boldsymbol{s}} - \underline{\boldsymbol{m}}_{\hat{\boldsymbol{s}}} \quad (3)$$

and from (2) and the linearity of the mean:

$$\underline{\boldsymbol{m}}_{\hat{\boldsymbol{s}}} = \underline{\mathbf{A}} * \underline{\boldsymbol{m}}_{\boldsymbol{y}} + \underline{\boldsymbol{b}} \quad (4)$$

thus,

$$\underline{\boldsymbol{b}} = \underline{\boldsymbol{m}}_{\hat{s}} - \underline{\mathbf{A}} * \underline{\boldsymbol{m}}_{y} \quad (5)$$

Minimizing the error vector is equivalent to minimizing its mean, i.e., $\underline{\boldsymbol{m}}_{e} = \mathbf{0}$, thus $\underline{\boldsymbol{m}}_{s} - \underline{\boldsymbol{m}}_{\hat{s}} = \mathbf{0}$, and $\underline{\boldsymbol{m}}_{s} = \underline{\boldsymbol{m}}_{\hat{s}}$. But by definition $\underline{\boldsymbol{m}}_{s} = \underline{\boldsymbol{m}}_{y} = \mathbf{0}$, therefore

$$\underline{\boldsymbol{b}} = \mathbf{0} - \underline{\mathbf{A}} * \mathbf{0} = \mathbf{0} \quad (6)$$

Consider the following equations:

$$\underline{\boldsymbol{e}} - \underline{\boldsymbol{m}}_{e} = (\underline{\boldsymbol{s}} - \underline{\hat{\boldsymbol{s}}}) - (\underline{\boldsymbol{m}}_{s} - \underline{\mathbf{A}} * \underline{\boldsymbol{m}}_{y} + \underline{\boldsymbol{b}})$$

$$= (\underline{\boldsymbol{s}} - \underline{\boldsymbol{m}}_{s}) - \underline{\mathbf{A}}\,(\underline{\boldsymbol{y}} - \underline{\boldsymbol{m}}_{y})$$

$$\underline{\boldsymbol{e}} - \underline{\boldsymbol{m}}_{e} = [\mathbf{I}_{m} \quad -\underline{\mathbf{A}}] * \begin{bmatrix} \underline{\boldsymbol{s}} - \underline{\boldsymbol{m}}_{s} \\ \underline{\boldsymbol{y}} - \underline{\boldsymbol{m}}_{y} \end{bmatrix} \quad (7)$$

$$\underline{\mathbf{K}}_{\mathbf{e}} = E\{(\underline{\boldsymbol{e}} - \underline{\mathbf{m}}_{\mathbf{e}})\,(\underline{\boldsymbol{e}} - \underline{\boldsymbol{m}}_{e})^{H}\} \quad (8)$$

From (7) and (8) we obtain:

$$\underline{\mathbf{K}}_{\mathbf{e}} = E\{([\mathbf{I}_{m} \quad -\underline{\mathbf{A}}] * \begin{bmatrix} \underline{\boldsymbol{s}} - \underline{\boldsymbol{m}}_{s} \\ \underline{\boldsymbol{y}} - \underline{\boldsymbol{m}}_{y} \end{bmatrix}) * ([\mathbf{I}_{m} \quad -\underline{\mathbf{A}}] * \begin{bmatrix} \underline{\boldsymbol{s}} - \underline{\boldsymbol{m}}_{s} \\ \underline{\boldsymbol{y}} - \underline{\boldsymbol{m}}_{y} \end{bmatrix})^{H}\} \quad (9)$$

Define $\underline{\mathbf{K}}$ as:

$$\underline{\mathbf{K}} = E\{\begin{bmatrix} \underline{\boldsymbol{s}} - \underline{\boldsymbol{m}}_{s} \\ \underline{\boldsymbol{y}} - \underline{\boldsymbol{m}}_{y} \end{bmatrix} \begin{bmatrix} \underline{\boldsymbol{s}} - \underline{\boldsymbol{m}}_{s} \\ \underline{\boldsymbol{y}} - \underline{\boldsymbol{m}}_{y} \end{bmatrix}^{H}\} \quad (10)$$

$$\underline{\mathbf{K}} = \begin{bmatrix} \underline{\mathbf{K}}_{s} & \underline{\mathbf{K}}_{sy} \\ \underline{\mathbf{K}}_{ys} & \underline{\mathbf{K}}_{y} \end{bmatrix} (11)$$

Where $\underline{\mathbf{K}}_{ys}$, $\underline{\mathbf{K}}_{sy}$ and $\underline{\mathbf{K}}_{y}$ can be defined as:

$$\underline{\mathbf{K}}_{sy} = \underline{\mathbf{K}}_{ys} = E\{(\underline{\boldsymbol{s}} - \underline{\boldsymbol{m}}_{s}) * (\underline{\boldsymbol{y}} - \underline{\boldsymbol{m}}_{y})^{H}\} \quad (12)$$

$$\underline{\mathbf{K}}_{y} = E\{(\underline{\boldsymbol{y}} - \underline{\boldsymbol{m}}_{y}) * (\underline{\boldsymbol{y}} - \underline{\boldsymbol{m}}_{y})^{H}\} \quad (13)$$

Thus from (9), (10) and (11):

$$\underline{\mathbf{K}}_e = [\mathbf{I}_m \quad -\underline{\mathbf{A}}] * \begin{bmatrix} \underline{\mathbf{K}}_s & \underline{\mathbf{K}}_{sy} \\ \underline{\mathbf{K}}_{ys} & \underline{\mathbf{K}}_y \end{bmatrix} * [\mathbf{I}_m \quad -\underline{\mathbf{A}}]^H$$

$$= [(\underline{\mathbf{K}}_s - \underline{\mathbf{A}} * \underline{\mathbf{K}}_{ys}) \quad (\underline{\mathbf{K}}_{sy} - \underline{\mathbf{A}} * \underline{\mathbf{K}}_y)] * [\mathbf{I}_m \quad -\underline{\mathbf{A}}]^H$$

$$= \underline{\mathbf{K}}_s - \underline{\mathbf{A}} * \underline{\mathbf{K}}_{ys} - \underline{\mathbf{K}}_{sy} * \underline{\mathbf{A}}^{\mathbf{H}} + \underline{\mathbf{A}} * \underline{\mathbf{K}}_y * \underline{\mathbf{A}}^{\mathbf{H}}$$

$$\underline{\mathbf{K}}_e = (\underline{\mathbf{A}} - \underline{\mathbf{K}}_{sy} * \underline{\mathbf{K}}_y^{-1}) * \underline{\mathbf{K}}_y * (\underline{\mathbf{A}} - (\underline{\mathbf{K}}_y^{-1} * \underline{\mathbf{K}}_{ys})^H)^H + \underline{\mathbf{K}}_s - \underline{\mathbf{K}}_{sy} * \underline{\mathbf{K}}_y^{-1} * \underline{\mathbf{K}}_{ys} \quad (14)$$

Because $\underline{\mathbf{K}}_s - \underline{\mathbf{K}}_{sy} * \underline{\mathbf{K}}_y^{-1} * \underline{\mathbf{K}}_{ys}$ is constant, Minimizing $\underline{\mathbf{K}}_e$ is equivalent to minimizing $(\underline{\mathbf{A}} - \underline{\mathbf{K}}_{sy} * \underline{\mathbf{K}}_y^{-1}) * \underline{\mathbf{K}}_y * (\underline{\mathbf{A}} - (\underline{\mathbf{K}}_y^{-1} * \underline{\mathbf{K}}_{ys})^H)^H$, thus from (12) we obtain the following equation:

$$(\underline{\mathbf{A}} - \underline{\mathbf{K}}_{sy} * \underline{\mathbf{K}}_y^{-1}) * \underline{\mathbf{K}}_y * (\underline{\mathbf{A}} - (\underline{\mathbf{K}}_y^{-1} * \underline{\mathbf{K}}_{ys})^H)^H = \mathbf{0}$$

$$\underline{\mathbf{A}} = \underline{\mathbf{K}}_{sy} * \underline{\mathbf{K}}_y^{-1} \text{ or } \underline{\mathbf{A}} = (\underline{\mathbf{K}}_y^{-1} * \underline{\mathbf{K}}_{ys})^H) \quad (15)$$

From (4), (12) and $E\{\underline{\boldsymbol{s}} * \underline{\boldsymbol{n}}^{\boldsymbol{H}}\} = 0$ (the signal vector is independent from the distributed noise)

$$\underline{\mathbf{K}}_{sy} = E\{(\underline{\boldsymbol{s}} - \underline{\boldsymbol{m}}_s) * (\underline{\boldsymbol{y}} - \underline{\boldsymbol{m}}_y)^H\}$$

$$= E\{\underline{\boldsymbol{s}} * (\underline{\mathbf{H}}^T * \underline{\boldsymbol{s}} + \underline{\boldsymbol{n}})^H\}$$

$$= E\{\underline{\boldsymbol{s}} * \underline{\boldsymbol{s}}^{\boldsymbol{H}} * (\underline{\mathbf{H}}^{\mathbf{T}})^H\} + E\{\underline{\boldsymbol{s}} * \underline{\boldsymbol{n}}^{\boldsymbol{H}}\},$$

$$= E\{\underline{\boldsymbol{s}} * \underline{\boldsymbol{s}}^{\boldsymbol{H}}\} * (\underline{\mathbf{H}}^{\mathbf{T}})^H$$

$$= \underline{\mathbf{K}}_s * (\underline{\mathbf{H}}^{\mathbf{T}})^H, \ \underline{\mathbf{K}}_s = \sigma^2_s * \mathbf{I}_m$$

$$\underline{\mathbf{K}}_{sy} = \sigma^2_s * (\underline{\mathbf{H}}^{\mathbf{T}})^H \quad (16)$$

$$\underline{\mathbf{K}}_y = E\{(\underline{\boldsymbol{y}} - \underline{\boldsymbol{m}}_y) * (\underline{\boldsymbol{y}} - \underline{\boldsymbol{m}}_y)^H\}$$

$$= E\{(\underline{\mathbf{H}}^{\mathbf{T}} * \underline{\boldsymbol{s}} + \underline{\boldsymbol{n}}) * (\underline{\boldsymbol{s}}^{\boldsymbol{H}} * \underline{\mathbf{H}} + \underline{\boldsymbol{n}})^H\}$$

$$= \underline{\mathbf{H}}^T * E\{\underline{\boldsymbol{s}} * \underline{\boldsymbol{s}}^{\boldsymbol{H}}\} * \underline{\mathbf{H}}^* + E\{\underline{\boldsymbol{n}} * \underline{\boldsymbol{n}}^{\boldsymbol{H}}\}, \ E\{\underline{\boldsymbol{n}} * \underline{\boldsymbol{n}}^{\boldsymbol{H}}\} = \sigma^2_n * \mathbf{I}_m$$

$$\underline{\mathbf{K}}_y = \sigma^2_s * \underline{\mathbf{H}}^{\mathbf{T}} * \underline{\mathbf{H}}^* + \sigma^2_n * \mathbf{I}_m \quad (17)$$

Finally, from (15), (16) and (17) we get :

$$\underline{\mathbf{A}} = \sigma^2_s * (\underline{\mathbf{H}}^{\mathbf{T}})^H * (\sigma^2_s * \underline{\mathbf{H}}^{\mathbf{T}} * \underline{\mathbf{H}}^* + \sigma^2_n * \mathbf{I}_m)^{-1} \quad (18)$$

With SNR = σ^2_s / σ^2_n, (18) yield to

$$\underline{\mathbf{G}} = \underline{\mathbf{A}}^{\mathbf{T}} = ((\underline{\mathbf{H}}^{\mathbf{T}} * \underline{\mathbf{H}}^{*})^{T} + (1 / SNR) * \mathbf{I}_{m})^{-1} * ((\underline{\mathbf{H}}^{\mathbf{T}})^{H})^{T} \quad (19)$$

Given that $(\underline{\mathbf{A}} * \underline{\mathbf{B}})^{T} = \underline{\mathbf{B}}^{\mathbf{T}} * \underline{\mathbf{A}}^{\mathbf{T}}$, $(\underline{\mathbf{A}}^{*})^{T} = \underline{\mathbf{A}}^{\mathbf{H}}$ and $((\underline{\mathbf{H}}^{\mathbf{T}})^{H})^{T} = \underline{\mathbf{H}}^{\mathbf{H}}$, from (19), we derivate the expression the MMSE conditioning matrix $\underline{\mathbf{G}}$ as:

$$\underline{\mathbf{G}} = (\underline{\mathbf{H}}^{\mathbf{H}} * \underline{\mathbf{H}} + (1 / SNR) * \mathbf{I}_{m})^{-1} * \underline{\mathbf{H}}^{\mathbf{H}}$$

2. Statistical Study of the Family of FR-BLAST First Detected layer [26]

Figure 11 shows that a significant improvement can be achieved if the detection starts on the right layer. The performance achieved by FR(Opt)-BLAST, i.e., the optimal proposed detector which picks the best layer for each frame of sample vectors, are clearly better than that of the family of detectors. However this decision is not trivial, thus we ran a statistical study to define a decision model.

The goal of this study was to build a statistical model to determine the best starting layer for an FR-BLAST detector, based on the size of the window search W (9 or 16 are a typical values), the MIMO gain m (typically 4 for this work), the constellation size M (64-QAM constellation signal here), the norms of each column of the channel matrix $\mathbf{H}$, and the gains in the column of $\mathbf{H}$ with the largest strength.

Several models (linear, quadratic, cubic, logarithmic and exponential) were studied, unfortunately the best model derivated by the statistical analysis tool (SPSS 18) was accurate only 25 % of the time, which can be seen as a random decision for practical implementation.

The results are presented below:

- e.g. 1: result for the linear model, $W = 9$

Classification Results[b,c]

		v10	Predicted Predicted starting layer				
			1	2	3	4	Total
Original	Count	1	12805	20759	26498	31733	91795
		2	12809	21333	26501	31791	92434
		3	12597	21170	26647	32020	92434
		4	12742	21132	26469	32043	92386
	%	1	13.9	22.6	28.9	34.6	100.0
		2	13.9	23.1	28.7	34.4	100.0
		3	13.6	22.9	28.8	34.6	100.0
		4	13.8	22.9	28.7	34.7	100.0
Cross-validated[a]	Count	1	12658	20808	26538	31791	91795
		2	12839	21123	26599	31873	92434
		3	12631	21272	26444	32087	92434
		4	12817	21222	26537	31810	92386
	%	1	13.8	22.7	28.9	34.6	100.0
		2	13.9	22.9	28.8	34.5	100.0
		3	13.7	23.0	28.6	34.7	100.0
		4	13.9	23.0	28.7	34.4	100.0

b. 25.2% of original grouped cases correctly classified.

c. 24.9% of cross-validated grouped cases correctly classified.

- e.g. 2 : Result for the combined linear and quadratic model, $W = 9$

Classification Results[b,c]

		v10	Predicted starting layer				
			1	2	3	4	Total
Original	Count	1	9824	22078	23623	36270	91795
		2	9814	22589	23837	36194	92434
		3	9723	22333	24059	36319	92434
		4	9871	22200	23793	36522	92386
	%	1	10.7	24.1	25.7	39.5	100.0
		2	10.6	24.4	25.8	39.2	100.0
		3	10.5	24.2	26.0	39.3	100.0
		4	10.7	24.0	25.8	39.5	100.0
Cross-validated[a]	Count	1	9691	22122	23643	36339	91795
		2	9837	22385	23946	36266	92434
		3	9737	22441	23862	36394	92434
		4	9933	22260	23858	36335	92386
	%	1	10.6	24.1	25.8	39.6	100.0
		2	10.6	24.2	25.9	39.2	100.0
		3	10.5	24.3	25.8	39.4	100.0
		4	10.8	24.1	25.8	39.3	100.0

b. 25.2% of original grouped cases correctly classified.
c. 25.0% of cross-validated grouped cases correctly classified.

e.g. 3 : Result for the combined linear, quadratic and logarithmic model, $W = 16$

Classification Results[b,c]

		v10	Predicted Group Membership				Total
			1	2	3	4	
Original	Count	1	20236	43728	31797	23211	118972
		2	20219	44170	31407	22785	118581
		3	20152	43845	31743	23166	118906
		4	19939	43447	31707	23258	118351
	%	1	17.0	36.8	26.7	19.5	100.0
		2	17.1	37.2	26.5	19.2	100.0
		3	16.9	36.9	26.7	19.5	100.0
		4	16.8	36.7	26.8	19.7	100.0
Cross-validated[a]	Count	1	19768	43835	32051	23318	118972
		2	20331	43960	31474	22816	118581
		3	20395	43933	31351	23227	118906
		4	20037	43489	31765	23060	118351
	%	1	16.6	36.8	26.9	19.6	100.0
		2	17.1	37.1	26.5	19.2	100.0
		3	17.2	36.9	26.4	19.5	100.0
		4	16.9	36.7	26.8	19.5	100.0

b. 25.1% of original grouped cases correctly classified.

c. 24.9% of cross-validated grouped cases correctly classified.

3. General Results for Basic Operations

1. Real-valued Operations

Let a_1 and a_2 denote two real numbers.

- Real number addition - f_1, g_1:

$a_1 + a_2$ requires one real addition and one cycle, so $\mathbf{f_1 = 1\ (+)}$ and $\mathbf{g_1 = 1\ (}\boldsymbol{c}\mathbf{)}$, i.e., one real-valued addition and one operation cycle, respectively.

- Real number multiplication - f_2, g_2:

$a_1 * a_2$ requires one real multiplication and one cycle, so $\mathbf{f_2 = 1\ (*)}$ and $\mathbf{g_2 = 1\ (}\boldsymbol{c}\mathbf{)}$.

- Real number reciprocal - f_3, g_3:

$1 / a_1$ requires one real reciprocal and one cycle, so $\mathbf{f_3 = 1\ (1/}\boldsymbol{N}\mathbf{)}$ and $\mathbf{g_3 = 1\ (}\boldsymbol{c}\mathbf{)}$.

- Square absolute value of a real number - f_4, g_4:

$|a_1|^2 = a_1^* a_1$ requires one real multiplication and one cycle, so $\mathbf{f_4 = 1\ (*)}$ and $\mathbf{g_4 = 1\ (}\boldsymbol{c}\mathbf{)}$.

- *m*-by-*n* real-valued matrix addition - $f_5(m,n)$, $g_5(m,n)$:

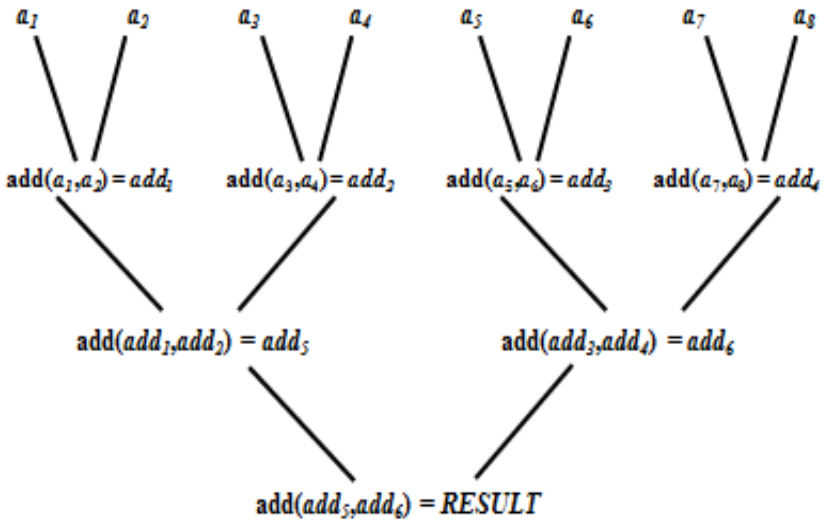
Consider two *m*-by-*n* real-valued matrices $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$ with $0 < i < m + 1$ and $0 < j < n + 1$. Adding each element at the *i*-th row and the *j*-th column of **A**, i.e., $a_{ij,}$ for $i = 1 \ldots m$ and $j = 1 \ldots n$, to the corresponding element from **B** will require $n * m$ real-valued additions, which could be performed in one cycle on parallel hardware. Thus the real-valued matrix addition will require $m * n * f_1$ operations and one g_1 cycle. Therefore, $\mathbf{f_5(}\boldsymbol{m,n}\mathbf{) = }\boldsymbol{mn}\ \mathbf{(+)}$ and $\mathbf{g_5(}\boldsymbol{m,n}\mathbf{) = 1\ (}\boldsymbol{c}\mathbf{)}$.

- Addition of a set containing *n* real numbers - $f_6(n)$, $g_6(n)$:

Considering two real numbers a_1 and a_2, computing the sum of these numbers will require one real addition and one cycle. Now, if we have a set of three numbers, we could first find the result from the addition of two of them, add_1, and then the result of the addition of add_1, and the remaining number. Basically, such a linear addition of a set of *n* real numbers will require *n* - 1 real additions and *n* - 1 cycles.

However there are more complicated, but faster, tree-based element summing algorithms for large sets of numbers. These algorithms are of no benefit for small sets.

Figure 1 illustrates the tree-based element summing procedure for *n* = 8. Consider a set containing *n* real numbers $a_1 \ldots a_n$. Observe that the height of a binary tree containing *n* elements in the leaves is $\lceil \log_2 n \rceil$. Note also that the number of non-leaf nodes, which is the number of additions, is *n* - 1 for a binary tree with *n* leaf nodes. Therefore, $\mathbf{f_6(n) = n - 1}$ **(+)** and $\mathbf{g_6(n) = \lceil \log_2 n \rceil}$ **(*c*)**.



$n = 8$

number of cycles $= 3 = \lceil \log_2 n \rceil$

number of real additions $= 7 = n - 1$

Figure 1 Tree-based Summing Algorithm to Add All elements in a Set of Size *n*

- Minimum of a set containing *n* real numbers - $f_7(n)$, $g_7(n)$:

To find the minimum of two numbers a_1 and a_2, we first compute the difference $diff = a_1 - a_2$. If $diff > 0$ then the minimum is a_2; otherwise, the minimum is a_1. Now, if we have a set $\{a_1, a_2, a_3\}$ containing three numbers, we will first find the minimum of the first two of them, $\min(a_1, a_2)$, and then the minimum of $\min(a_1, a_2)$, and the remaining third number a_3. It is straightforward to determine that computing the minimum of a set containing *n* real numbers is equivalent in computational complexity to adding together all the elements in a set of *n* real numbers. Therefore, $f_7(n) = f_6(n)$ and $g_7(n) = g_6(n)$. So,

$f_7(n) = n - 1$ (+) and **$g_7(n) = \lceil \log_2 n \rceil$ (*c*)**.

- Square norm of a real-valued column vector of length n - $f_8(n)$, $g_8(n)$:

Considering a vector column of n elements $\boldsymbol{v} = (v_1, v_2\ldots, v_n)$,

$$||\boldsymbol{v}||^2 = ||(v_1, v_2\ldots, v_n)||^2 = |v_1|^2 + |v_2|^2 + \ldots + |v_n|^2$$

Computing $|v_i|^2$ requires f_4 operations and g_4 cycles for each element v_i. But these independent operations can be performed in parallel. Thus, for all n components $n * f_4$ operations and g_4 cycles will be needed. Computing the sum of n squared absolute values will require $f_6(n)$ operations and $g_6(n)$ cycles. Therefore, computing the square norm of an n-element column vector requires $n * f_4 + f_6(n)$ operations and $g_4 + g_6(n)$ cycles. So,

$\mathbf{f_8(n) = n\ (*)}$ and $\mathbf{n - 1\ (+)}$, and $\mathbf{g_8(n) = 1 + \lceil \log_2 n \rceil\ (c)}$.

- Multiplication of an m-by-p real-valued and a p-by-n real-valued matrices - $f_9(m,n,p)$, $g_9(m,n,p)$: [29]

Considering an m-by-p and an p-by-n real-valued matrices $\mathbf{A} = [a_{ik}]$ and $\mathbf{B} = [b_{kj}]$, with $0 < i < m + 1$, $0 < j < m + 1$ and $0 < k < p + 1$.

Let, $\mathbf{C} = \mathbf{A} * \mathbf{B} = [c_{ij}]$, with $0 < i < m + 1$ and $0 < j < n + 1$

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}$$

Computing $a_{ik} * b_{kj}$ will require f_2 operations and g_2 cycles. Thus for the p terms in the sum, there will be $p * f_2$ operations and g_2 cycles. The computation for all $m * n$ product matrix coefficients will thus require $m * n * p * f_2$ operations and g_2 cycles.

We also need to compute the sum of a set of p real numbers for each coefficient, which will require $f_6(p)$ operations and $g_6(p)$ cycles, for all $m * n$ matrix coefficients, so $m * n * f_6(p)$ operations and $g_6(p)$ cycles are required.

Therefore, computing the product of an m-by-p real-valued matrix and a p-by-n real-valued matrix will require $m * n * p * f_2 + m * n * f_6(p)$ operations and $g_2 + g_6(p)$ cycles.

$f_9(m,n,p) = mnp$ (*) and **$mn(p - 1)$ (+), and $g_9(m,n,p) = 1 + \lceil \log_2 p \rceil$ (c)**.

- Inverse of an n-by-n real-valued matrix - $f_{10}(n)$, $g_{10}(n)$:

Considering $\mathbf{A} = [a_{ij}]$ for $0 < i < n + 1$ and $0 < j < n + 1$.

Let $\mathbf{B} = \mathbf{A}^{-1}$. To derive **B**, we will consider the Gauss-Jordan Matrix Inversion Method, which has the advantage of requiring fewer operations for large matrices than LU decomposition [29].

Consider the augmented matrix [**A**|**I**], with matrix **A** in the left side and the identity matrix of size n in the right side. By performing basic operations (one row multiplied by a scalar, or one row replaced by the original row minus a multiple of another row [29]) between the rows of **A** and the identity matrix, we will transform [**A**|**I**] to [**I**|**B**], where the identity matrix **I** is on the left side and matrix **B** on the right side.

Thus, on one side of the augmented matrix, the transformation will lead to the identity matrix. So, we will only consider operations on the other side, i.e., on n coefficients rather than the $2 * n$ coefficients of the augmented matrix.

Consider the operations on the i-th row, i.e., $\boldsymbol{R}_i$:

- First, we replace $\boldsymbol{R}_i$ by $\boldsymbol{R}_i$ / R_{ii}, where R_{ii} is the i-th coefficient on the i-th row. This will require f_3 operations and g_3 cycles per element. Considering maximum possible parallelism and n candidates, n * f_3 operations and g_3 cycles will be required.
- Next, we replace all the remaining rows (n - 1 in total) by $-R_{ji}$ * $\boldsymbol{R}_i$ + $\boldsymbol{R}_j$. Here we are transforming the j-th row; this will require f_2 operations and g_2 cycles, then f_1 operations and g_1 cycles per candidate. Thus considering full parallelism, n candidates and n - 1 rows, n * (n - 1) * (f_1 + f_2) operations and (g_1 + g_2) cycles will be required.

Therefore, for $0 < i < n + 1$,

$\mathbf{f_{10}(n) = n^2}$ **(1/*N*)**, $\boldsymbol{n^2(n-1)}$ **(*)** and $\boldsymbol{n^2(n-1)}$ **(+),** and $\mathbf{g_{10}(n) = 3n}$ **(*c*)**.

- Minimum Mean Square Error conditioning (MMSE) matrix from a 2n-by-2n real-valued channel matrix - $f_{11}(n,n)$, $g_{11}(n,n)$: [Appendix 1]

Note: For the purpose of this work, we are only interested in the equivalent real-valued matrices built from complex-valued matrices. Given an n-by-n complex-valued matrix $\underline{\mathbf{C}}$, the real-valued equivalent matrix is $\mathbf{C^{(R)}} = \begin{bmatrix} \mathbf{real}(\underline{\mathbf{C}}) & -\mathbf{imag}(\underline{\mathbf{C}}) \\ \mathbf{imag}(\underline{\mathbf{C}}) & \mathbf{real}(\underline{\mathbf{C}}) \end{bmatrix}$.

Thus, considering an n-by-n complex-valued conditioning matrix, $\underline{\mathbf{G}}$, the corresponding real-valued MMSE conditioning matrix, $G^{(R)}$ is defined as follows:

$$\mathbf{G^{(R)}} = \begin{bmatrix} \mathbf{real}(\mathbf{G}) & -\mathbf{imag}(\mathbf{G}) \\ \mathbf{imag}(\mathbf{G}) & \mathbf{real}(\mathbf{G}) \end{bmatrix}, \text{ where } \underline{\mathbf{G}} = (\underline{\mathbf{C}}^{\mathbf{H}} * \underline{\mathbf{C}} + (1 / \mathrm{SNR}) * \mathbf{I}_n)^{-1} * \underline{\mathbf{C}}^{\mathbf{H}}$$

Computing $\mathbf{H}^H$ from $\mathbf{H}$ will require no operations and no cycles.

It can be shown that the matrix product $\underline{\mathbf{C}}^{\mathbf{H}} * \underline{\mathbf{C}}$ is a symmetrical matrix. So instead of computing all n^2 coefficients, we do not need to compute the $n\ (n + 1) / 2$ coefficients above the main diagonal. Thus this computation will require $n\ (n + 1)\ (2n - 1)$ real additions, $2n^2\ (n + 1)$ real multiplications and $2 + \lceil \log_2 n \rceil$ run cycles.

Computing 1 / SNR will require f_3 operations and g_3 cycles.

Computing 1 / SNR $* \mathbf{I}_n$ will require no operations and no cycles as it is equivalent to replacing the diagonal elements of $\mathbf{I}_n$ by 1 / SNR.

Computing $\underline{\mathbf{C}}^{\mathbf{H}} * \underline{\mathbf{C}} + 1$ / SNR $* \mathbf{I}_n$ will require n real additions and one cycle as it is equivalent to an addition by a real constant on all diagonal coefficient of $\underline{\mathbf{C}}^{\mathbf{H}} * \underline{\mathbf{C}}$.

Computing $(\underline{\mathbf{C}}^{\mathbf{H}} * \underline{\mathbf{C}} + (1 / \text{SNR}) * \mathbf{I}_n)^{-1}$ will require $f_{21}(n)$ operations and $g_{21}(n)$ cycles.

Again, due to the symmetrical product, computing $(\underline{\mathbf{C}}^{\mathbf{H}} * \underline{\mathbf{C}} + (1 / \text{SNR}) * \mathbf{I}_n)^{-1} * \underline{\mathbf{C}}^{\mathbf{H}}$, will require $n\ (n + 1)\ (2n - 1)$ real additions, $2n^2\ (n + 1)$ real multiplications and $2 + \lceil \log_2 n \rceil$ run cycles.

Given the complex-valued conditioning matrix, $\underline{\mathbf{G}}$, we can build the real-valued conditioning matrix, $\mathbf{G}^{(\mathbf{R})}$, without further operations.

Therefore, $\mathbf{f_{11}(2n) = n^2 + 1\ \ (1/N),\ 4n^2(2n + 1)\ \ (*)}$ and $\mathbf{n[8n^2 - n - 1]\ (+)}$, and $\mathbf{g_{11}(2n) = 6 + 7n + 2\lceil \log_2 n \rceil\ (c)}$.

- Minimum Mean Square Error (MMSE) conditioning matrix from an *m*-by-*n* real-valued deflated channel matrix- $f_{12}(m,n)$, $g_{12}(m,n)$:

We will share computations from the calculation of the initial MMSE conditioning matrix.

Considering **D**, the *n*-by-*m* deflated version of the *n*-by-*n* channel matrix **H** ($m < n$),

$\mathbf{temp}_1 = \mathbf{H}^{\mathbf{H}} * \mathbf{H}$ is the matrix whose coefficient on the *i*-th row and *j*-th column is $\boldsymbol{h}_i^{H} * \boldsymbol{h}_j$, where $\boldsymbol{h}_i$ is the *i*-th column of **H**. Thus, with $\mathbf{temp_1} = \mathbf{H}^{\mathbf{H}} * \mathbf{H} + 1 / \text{SNR} * \mathbf{I}_n$ and $\mathbf{temp_2} = \mathbf{D}^{\mathbf{H}} * \mathbf{D} + 1 / \text{SNR} * \mathbf{I}_m$. Here $\mathbf{temp_2}$ is a deflated version of $\mathbf{temp_1}$, and its computation requires no run time.

$\mathbf{temp_2}$ is an *m*-by-*m* matrix, thus $\mathbf{temp_2}^{-1}$ will require $f_{10}(m)$ operations and $g_{10}(m)$ cycles

$\mathbf{G} = \mathbf{temp_2}^{-1} * \mathbf{D}^{\mathbf{H}}$, with $(m, p, n) = (m, m, n)$. Computing **G** requires $f_9(m,m,n)$ operations and $g_9(m,m,n)$ cycles.

Therefore $\mathbf{f_{12}(m,n) = m^2 (1/N) \text{ and } m^2(n + m - 1) (*)}$ and $\boldsymbol{m(m - 1)(m + n)}$ **(+)**, and $\mathbf{g_{12}(m,n) = 3m + 1 + \lceil \log_2 n \rceil (c)}$.

2. Complex-valued Operations

Let $\underline{c}_1$ and $\underline{c}_2$ denote two complex numbers such that $\underline{c}_1 = a_1 + j * b_1$ and $\underline{c}_2 = a_2 + j * b_2$.

- Complex number addition - f_{13}, g_{13}:

$\underline{c}_1 + \underline{c}_2 = a_1 + a_2 + j * (b_1 + b_2)$ requires two real additions and one cycle. So, $\mathbf{f_{13} = 2\ (+)}$ and $\mathbf{g_{13} = 1\ (c)}$.

- Complex number multiplication - f_{14}, g_{14}:

$\underline{c}_1 * \underline{c}_2 = a_1 * a_2 - b_1 * b_2 + j * (a_1 * b_2 + b_1 * a_2)$ requires two real additions, four real multiplications and two cycles. So, $\mathbf{f_{14} = 2\ (+)}$ and $\mathbf{4\ (*)}$, and $\mathbf{g_{14} = 2\ (c)}$.

- Complex number reciprocal - f_{15}, g_{15}:

$1 / \underline{c}_1 = (a_1 - j * b_1) / (a_1^2 + b_1^2) = a_1 / (a_1^2 + b_1^2) - j * b_1 / (a_1^2 + b_1^2)$ requires one real addition, four real multiplications, one real reciprocal and four cycles. So, $\mathbf{f_{15} = 1\ (1/N),\ 1\ (+)}$ and $\mathbf{4\ (*)}$, and $\mathbf{g_{15} = 4\ (c)}$.

- Square absolute value of a complex number - f_{16}, g_{16}:

$|\underline{c}_1|^2 = a_1^2 + b_1^2$ requires one real addition, two real multiplications and two cycles. So, $\mathbf{f_{16} = 2\ (*),\ 1\ (+)}$, and $\mathbf{g_{16} = 2\ (c)}$.

- *m*-by-*n* complex-valued matrix addition - $f_{17}(m,n)$, $g_{17}(m,n)$:

Considering two *m*-by-*n* complex-valued matrices $\underline{\mathbf{A}} = [\underline{a}_{ij}]$ and $\underline{\mathbf{B}} = [\underline{b}_{ij}]$ with $0 < i < m + 1$ and $0 < j < n + 1$, adding the term in the *i*-th row and *j*-th column of $\underline{\mathbf{A}}$, i.e., $\underline{a}_{ij}$, for $i = 1 \ldots m$ and $j = 1 \ldots n$, to the one from $\underline{\mathbf{B}}$ will require one complex-valued addition. Because there are $n * m$ elements in each matrix, the complex matrix addition will require $m * n * f_{14}$ operations and g_{14} cycles.

Therefore, $\mathbf{f_{17}(m,n) = 2mn\ (+)}$ and $\mathbf{g_{17}(m,n) = 1\ (c)}$.

- Addition of a set containing *n* complex numbers - $f_{18}(n)$, $g_{18}(n)$:

In this case, the only difference with the real-valued computation is that we are doing a group of complex additions, and thus the number of real additions will be doubled.

Therefore, $\mathbf{f_{18}(n) = 2n - 2\ (+)}$ and $\mathbf{g_{18}(n) = \lceil \log_2 n \rceil\ (c)}$.

- Square norm of a complex-valued column vector of length *n* - $f_{19}(n)$, $g_{19}(n)$:

Considering a vector column of *n* elements $\underline{\mathbf{v}} = (\underline{v}_1, \underline{v}_2 \ldots, \underline{v}_n)$,

$$||\underline{\mathbf{v}}||^2 = ||(\underline{v}_1, \underline{v}_2 \ldots, \underline{v}_n)||^2 = |\underline{v}_1|^2 + |\underline{v}_2|^2 + \ldots + |\underline{v}_n|^2$$

Computing $|\underline{v}_i|^2$ requires f_{16} operations and g_{16} cycles. Thus, for all dimensions $n * f_{16}$ operations and g_{16} cycles will be required.

Computing the sum of *n* squared absolute value will require $f_6(n)$ operations and $g_6(n)$ cycles.

Therefore, computing the square norm of a *n*-elements column vector require $n * f_{16} + f_6(n)$ operations and $g_{16} + g_6(n)$ cycles.

$\mathbf{f_{19}(n) = 2n\ (*)}$ and $\mathbf{2n - 1\ (+)}$, and $\mathbf{g_{19}(n) = 2 + \lceil \log_2 n \rceil\ (c)}$.

- Multiplication of an *m*-by-*p* complex-valued matrix by a *p*-by-*n* complex-valued matrix - $f_{20}(m,n,p)$, $g_{20}(m,n,p)$: [29]

Consider an *m*-by-*p* complex-valued matrix and a *p*-by-*n* complex-valued matrix $\underline{\mathbf{A}} = [\underline{a}_{ik}]$ and $\underline{\mathbf{B}} = [\underline{b}_{kj}]$, with $0 < i < m + 1$, $0 < j < m + 1$ and $0 < k < p + 1$.

Let $\underline{\mathbf{C}} = \underline{\mathbf{A}} * \underline{\mathbf{B}} = [\underline{c}_{ij}]$, with $0 < i < m + 1$ and $0 < j < n + 1$

$$\underline{c}_{ij} = \sum_{k=1}^{p} \underline{a}_{ik} \underline{b}_{kj}.$$

Computing $\underline{a}_{ik} * \underline{b}_{kj}$ will require f_{14} operations and g_{14} cycles. Thus for the p-th terms in the sum, there will be $p * f_{14}$ operations and g_{14} cycles. The computation for all $m * n$ matrix coefficients will require $m * n * p * f_{14}$ operations and g_{14} cycles.

We also need to compute the sum of a set of p complex numbers for each coefficient, which will require $f_{18}(p)$ operations and $g_{18}(p)$ cycles. Thus for all $m * n$ matrix coefficients, $m * n * f_{18}(p)$ operations and $g_{18}(p)$ cycles are required. Therefore, computing the product of an m-by-p and a p-by-n matrices will require $m * n * p * f_{14} + m * n * f_{18}(p)$ operations and $g_{14} + f_{14}(p)$ cycles.

$f_{20}(m,n,p) = 4mnp$ (*) and **$mn(4p - 2)$ (+)**, and **$g_{20}(m,n,p) = 2 + \lceil \log_2 p \rceil$ (*c*)**.

- Inverse of an n-by-n complex-valued matrix - $f_{21}(n)$, $g_{21}(n)$: [29]

Consider $\underline{\mathbf{A}} = [\underline{a}_{ij}]$ for $0 < i < n + 1$ and $0 < j < n + 1$.

Let $\underline{\mathbf{B}} = \underline{\mathbf{A}}^{-1}$. To derive $\underline{\mathbf{B}}$, we will once again consider the Gauss-Jordan Matrix Inversion Method.

Let put $\underline{\mathbf{A}}$ in the left side and the identity matrix of size n in the right side of the augmented matrix $[\underline{\mathbf{A}}|\mathbf{I}]$. By operating basic operations (one multiplied by a scalar or one row replaced by the original row minus a multiple of another row [29]) between the rows of $\underline{\mathbf{A}}$ and the identity matrix, we will obtain the augmented matrix $[\mathbf{I}|\underline{\mathbf{B}}]$ which has the identity matrix on the left side and matrix $\underline{\mathbf{B}}$ on the right side.

Thus, on one side of the augmented matrix, the transformation will lead to the identity matrix. So, we will only consider operations on the other side, i.e., on *n* coefficients rather than the 2 * *n* coefficients of the augmented matrix.

Consider the operations on the *i*-th row, i.e., $\boldsymbol{R}_i$:

- First, we replace $\underline{R}_i$ by $\underline{\boldsymbol{R}}_i$ / $\underline{R}_{ii}$, where $\underline{R}_{ii}$ is the *i*-th coefficient on the *i*-th row. This will require f_{15} operations and g_{15} cycles per candidate. Considering maximum possible parallelism and *n* candidates, *n* * f_{15} operations and g_{15} cycles will be required.
- Next, we replace all the remaining rows (*n* - 1 in total) by - $\underline{R}_{ji}$ * $\underline{\boldsymbol{R}}_i$+ $\underline{\boldsymbol{R}}_j$. Here we are transforming the *j*-th row; this will require f_{14} operations and g_{14} cycles, then f_{13} operations and g_{13} cycles per candidate. Thus considering full parallelism, *n* candidates and *n* - 1 rows, *n* * (*n* - 1) * (f_{14} + f_{13}) operations and (g_{14} + g_{13}) cycles will be required.

Therefore, $0 < i < n + 1$,

$\mathbf{f_{21}(n) = n^2\ (1/N)}$, $\mathbf{4n^3\ (*)}$ and $\mathbf{n^2\ (4n - 3)\ (+)}$, and $\mathbf{g_{21}(n) = 7n\ (c)}$.

- Minimum Mean Square Error conditioning matrix from an *n*-by-*n* complex-valued channel matrix- $f_{22}(n)$, $g_{22}(n)$: [Appendix 1]

Refering to the computation of a real-valued MMSE conditioning matrix,

$\mathbf{f_{22}(n) = n^2 + 1\ (1/N)}$, $\mathbf{4n^2(2n + 1)\ (*)}$ and $\mathbf{n(8n^2 - n - 1)\ (+)}$,

and $\mathbf{g_{22}(n) = 6 + 7 * n + 2 * \lceil \log_2 n \rceil\ (c)}$.

- Minimum Mean Square Error conditioning matrix from an *m*-by-*n* complex-valued deflated channel matrix- $f_{23}(m,n)$, $g_{23}(m,n)$:

We will share computations from the calculation of the initial MMSE conditioning matrix.

Considering **D**, the *m*-by-*n* deflated version of the *m*-by-*m* channel matrix **H** ($n < m$),

$\mathbf{temp_1} = \mathbf{H}^H * \mathbf{H}$ is the matrix whose coefficient on the *i*-th row and *j*-th column is $\boldsymbol{h}_i^H * \boldsymbol{h}_j$, where $\boldsymbol{h}_i$ is the *i*-th column of **H**. Thus, $\mathbf{temp_1} = \mathbf{H}^H * \mathbf{H} + 1 / \text{SNR} * \mathbf{I}_n$ and $\mathbf{temp_2} = \mathbf{D}^H * \mathbf{D} + 1 / \text{SNR} * \mathbf{I}_m$.

It can be shown that $\mathbf{temp_2}$ is a deflated version of $\mathbf{temp_1}$, thus it will require no run time.

$\mathbf{temp_2}$ is an *m*-by-*m* matrix, thus $\mathbf{temp_2}^{-1}$ will require $f_{21}(m)$ operations and $g_{21}(m)$ cycles

$\mathbf{G} = \mathbf{temp_2}^{-1} * \mathbf{D}^H$, with ($m$, p, n) = (m, m, n), so computing **G** requires $f_{20}(m,m,n)$ operations and $g_{20}(m,m,n)$ cycles.

Therefore $\mathbf{f_{23}(m,n)} = m^2 (1/N)$, $4m^2(n + m)$ (*) and $n[4m^2 - 3m + 4mn - 2n]$ (+),

and $\mathbf{g_{23}(m,n)} = 7m + 2 + \lceil \log_2 m \rceil$ (c).

4. Matlab Scripts for the Hard Detectors

1- Main Function (Complex-Valued Detection)

```
%================================================================
========
% This program was written by Arsene Pankeu Yomi, University of
Alberta
% It was designed in order to evaluate the SER of different MIMO
detectors
% in a typical Rayleigh channel; with the help of Amir Alimohammad
% and Fouladi Fard in the begining.
%================================================================
========

clear;clc;close all;
%%%% attention 'save' and 'change' the name of the saved data
%%%% FR-BLAST will work only for 64Q 128Q and 256Q
%    the search set for constellation greater than 256Q is not
provided
%%%% QAM from modulation is not valid for 128-QAM

% main parameters
NumbBlock = 10^4; % each block contains    2 * ConstellationSize
                  %                         * NUmber of frame in
a block
                  %                         samples
CalcSc    = 0;    % 1-> SER     0-> BER
ModSc     = 2;    % 1->16-qam   2->64-qam   3->128-qam
                  % 4->256-qam  5->512-qam  6->1024-qam
N         = 4;    % number of antennas; n_t = n_r = N
NumFrameInABlock   = 10;
MaxSnr    = 40;
MinSnr    = 0;
StepSnr   = 2;    % 1 or 2 e.g: stepSnr=2 --[MinSnr, MinSnr+1,
MinSnr+2...]

SnrDb     = MinSnr:StepSnr:MaxSnr;
SNR       = 10.^(SnrDb/10);
LSnr      = length(SNR);

% The minimum # of error is the same for all or decreases as SNR
increases
%  MatOfErrs = 100 * [1 .5 .5 .4 .4 .3 .3 .2 .2 .15 .15 .1 .1 .09
.09 .08 .08 .07 .07 .06 .06 .05 .05 .04 .04 .03 .03 .02 0.02 0.01
0.01];
 MatOfErrs = 100000 * ones(1,MaxSnr + 1);

% Launch detection
% '1' In order to launch detection scheme otherwise '0'
DoMlDetect        = 0;   % Maximum Likelihood detector
DoMmseDetect      = 1;   % MMSE detector
DoVBlastDetect    = 1;   % V-BLAST detector
```

```
DoFBlastDetect    = 0;   % F-BLAST detector; starting layer
indicated below
DoFRBlastDetect   = 0;   % FR-BLAST; starting layer indicated
below
SizeOfSphere      = 9;   % number of symbol in the search space

% Initialization of SER
if (DoMlDetect == 1)
    if (CalcSc == 1)
        SerMl     = zeros(1,LSnr);
    else
        BerMl     = zeros(1,LSnr);
    end
end

if (DoMmseDetect == 1)
    if (CalcSc == 1)
        SerMmse   = zeros(1,LSnr);
    else
        BerMmse   = zeros(1,LSnr);
    end
end

if (DoVBlastDetect == 1)
    if (CalcSc == 1)
        SerVBlast = zeros(1,LSnr);
    else
        BerVBlast = zeros(1,LSnr);
    end
end

if (DoFBlastDetect == 1)
    if (CalcSc == 1)
        SerFBlast = zeros(1,LSnr);
    else
        BerFBlast = zeros(1,LSnr);
    end
end

if (DoFRBlastDetect == 1)
    if (CalcSc == 1)
       SerFRBlast = zeros(1,LSnr);
    else
       BerFRBlast = zeros(1,LSnr);
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Symbol Matrix, Constant and sample generation %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[SymbMat SignalVar ConstSize NumbBitPerPt StatMat] =
Modulation(NumFrameInABlock,NumbBlock,N,ModSc);

[TotNumOfSampleInABlock FrameSize SentSymb] =
SampGen(NumFrameInABlock,NumbBlock,N,ConstSize,NumbBitPerPt,SymbMa
t);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Computation of SER for various detector %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for snr = 1:LSnr
    MinNoErrs          = MatOfErrs(SnrDb(snr) + 1);
    NoiseVar           = (N * SignalVar) / SNR(snr);

    if (DoMlDetect == 1)
        NumbErrMl      = 0;
    end

    if (DoMmseDetect == 1)
        NumbErrMmse    = 0;
    end

    if (DoVBlastDetect == 1)
        NumbErrVBlast  = 0;
    end

    if (DoFBlastDetect == 1)
        NumbErrFBlast  = 0;
    end

    if (DoFRBlastDetect == 1)
        NumbErrFRBlast = 0;
    end

    Continue           = 1;
    BlockCntr          = 0;
    % the following line counts the number of block used for the
current snr and verifies that MinNoErrs had been reached
    while ((Continue == 1) && ((BlockCntr < NumbBlock)))
          Continue          = 0;

          ContinueMl        = 0;
          ContinueMmse      = 0;
          ContinueVBlast    = 0;
          ContinueFBlast    = 0;
          ContinueFRBlast   = 0;

          BlockCntr         = BlockCntr + 1;

          % channel generation
          MIMOchResponse    = zeros(N,N,TotNumOfSampleInABlock);
```

```
          for ii = 1:NumFrameInABlock
              % generating CHANNEL
              H  = (randn(N) + i*randn(N)) * sqrt(0.5);
              for ii1 = 1:N
                  for ii2 = 1:N
                      MIMOchResponse(ii1, ii2, (1+(ii-
1)*FrameSize:ii*FrameSize)) = H(ii1, ii2); % one channel response
per frame
                  end
              end
          end

          % block of frames
          Block = SentSymb(:,(1+TotNumOfSampleInABlock*(BlockCntr-
1)):(BlockCntr*TotNumOfSampleInABlock));

          for frameCntr = 1:NumFrameInABlock

              %%%%%%%%%%
              % Channel %
              %%%%%%%%%%

              H     = MIMOchResponse(:, :, 1+(frameCntr-
1)*FrameSize);
              N_    =
sqrt(0.5)*(randn(N,FrameSize)+i*randn(N,FrameSize));
              W     = N_*sqrt(NoiseVar);
              % S is the current frame
              S     = Block(:,1+FrameSize*(frameCntr-
1):frameCntr*FrameSize);
              Y     = zeros(N, FrameSize);
              for ii = 1:FrameSize
                  % Y=H*S+W
Y(:,ii)=MIMOchResponse(:,:,ii+(frameCntr-1)*FrameSize) * S(:,ii) +
W(:,ii);
              end

              %%%%%%%%%%%%%%%%%%%%
              % Detection process %
              %%%%%%%%%%%%%%%%%%%%

              % MMSE matrix
              G = inv(H' * H + (1 / SNR(snr)) * eye(N)) * H';
              [SortedLayer IndMI] = SortCol(H,N);

              %%%% ML detection %%%%
              if (DoMlDetect == 1)
                  for vectorCntr = 1:FrameSize
                      [YMl] =
Ml(H,ConstSize,N,Y(:,vectorCntr),SymbMat);
                      % error calculation
                      if (CalcSc == 1)
                        [YMlDetect] = Ser(YMl,S(:,vectorCntr));
                      else
```

```
                        [YMlDetect] =
Ber(ConstSize,N,YMl,SymbMat,S(:,vectorCntr));
                    end
                    % counts the number of symbol in error in
the block
                    NumbErrMl     = NumbErrMl + YMlDetect;
                end
                if (NumbErrMl < MinNoErrs)
                   ContinueMl = 1;
                else
                   ContinueMl = 0;
                end
            end

            %%%% MMSE detection %%%%
            if (DoMmseDetect == 1)
                for vectorCntr = 1:FrameSize
                   [YMmse] =
Mmse(ConstSize,N,Y(:,vectorCntr),SymbMat,G);
                    % error calculation
                    if (CalcSc == 1)
                      [MmseDetect] = Ser(YMmse,S(:,vectorCntr));
                    else
                      [MmseDetect] =
Ber(ConstSize,N,YMmse,SymbMat,S(:,vectorCntr));
                    end
                     % counts the number of symbol in error in
the block
                     NumbErrMmse   = NumbErrMmse + MmseDetect;
                end
                if (NumbErrMmse < MinNoErrs)
                   ContinueMmse = 1;
                else
                   ContinueMmse = 0;
                end
            end

            %%%% V-BLAST detection %%%%
            if (DoVBlastDetect == 1)
                for vectorCntr = 1:FrameSize
                    % initialize position of the undetected
symbols
                    Pos = 1:N;
                    % initialize detected symbol vector
                    YVB = zeros(N,1);

 [YVBlast] =
VBlast(H,ConstSize,N,Y(:,vectorCntr),SymbMat,Pos,YVB,SNR,snr);
                    % error calculation
                    if (CalcSc == 1)
                      [VBlastDetect] =
Ser(YVBlast,S(:,vectorCntr));
                    else
                      [VBlastDetect] =
Ber(ConstSize,N,YVBlast,SymbMat,S(:,vectorCntr));
                    end
```

```
                    % Counts the number of symbol in error in
the block
                    NumbErrVBlast    = NumbErrVBlast +
VBlastDetect;
                end
                if (NumbErrVBlast < MinNoErrs)
                   ContinueVBlast = 1;
                else
                   ContinueVBlast = 0;
                end
            end

            %%%% F-BLAST detection %%%%
            if (DoFBlastDetect == 1)
                for vectorCntr = 1:FrameSize
       [YFBlast] =
FBlast(H,ConstSize,N,Y(:,vectorCntr),SymbMat,SortedLayer(1),SNR,sn
r);
                    % error calculation
                    if (CalcSc == 1)
                      [FBlastDetect] =
Ser(YFBlast,S(:,vectorCntr));
                    else
                      [FBlastDetect] =
Ber(ConstSize,N,YFBlast,SymbMat,S(:,vectorCntr));
                    end
                    % Counts the number of symbol in error in
the block
                    NumbErrFBlast    = NumbErrFBlast +
FBlastDetect;
                end
                if (NumbErrFBlast < MinNoErrs)
                   ContinueFBlast = 1;
                else
                   ContinueFBlast = 0;
                end
            end

            %%%% FR-BLAST detection %%%%
            if (DoFRBlastDetect == 1)
                for vectorCntr = 1:FrameSize
[YFRBlast] =
FRBlast(H,ConstSize,N,Y(:,vectorCntr),SymbMat,G,SortedLayer(1),SNR
,snr,SizeOfSphere,StatMat);
                    % error calculation
                    if (CalcSc == 1)
                      [FRBlastDetect] =
Ser(YFRBlast,S(:,vectorCntr));
                    else
                      [FRBlastDetect] =
Ber(ConstSize,N,YFRBlast,SymbMat,S(:,vectorCntr));
                    end
                     % Counts the number of symbol in error in
the block
                     NumbErrFRBlast   = NumbErrFRBlast +
FRBlastDetect;
```

```
                end
                if (NumbErrFRBlast   < MinNoErrs)
                   ContinueFRBlast   = 1;
                else
                   ContinueFRBlast   = 0;
                end
            end

          end % end frameCntr

          Intermediate = ContinueMl + ContinueMmse +
ContinueVBlast + ContinueFBlast + ContinueFRBlast;
          if (Intermediate == 0)
              Continue = 0;
          else
              Continue = 1;
          end

    end % while continue

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % SER computation for the current snr value %
    %         to be print on the screen         %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 clc;
 if DoMlDetect,
    if (CalcSc == 1)
        SerMl(snr)   = NumbErrMl / BlockCntr / FrameSize /
NumFrameInABlock / N
    else
        BerMl(snr)   = NumbErrMl / BlockCntr / FrameSize /
NumFrameInABlock / N / NumbBitPerPt
    end
 end

 if DoMmseDetect,
    if (CalcSc == 1)
        SerMmse(snr)   = NumbErrMmse / BlockCntr / FrameSize /
NumFrameInABlock / N
    else
        BerMmse(snr)   = NumbErrMmse / BlockCntr / FrameSize /
NumFrameInABlock / N / NumbBitPerPt
    end
 end

 if DoVBlastDetect,
    if (CalcSc == 1)
        SerVBlast(snr)   = NumbErrVBlast / BlockCntr / FrameSize /
NumFrameInABlock / N
    else
        BerVBlast(snr)   = NumbErrVBlast / BlockCntr / FrameSize /
NumFrameInABlock / N / NumbBitPerPt
    end
 end
```

```
 if DoFBlastDetect,
    if (CalcSc == 1)
        SerFBlast(snr)   = NumbErrFBlast / BlockCntr / FrameSize /
NumFrameInABlock / N
    else
        BerFBlast(snr)   = NumbErrFBlast / BlockCntr / FrameSize /
NumFrameInABlock / N / NumbBitPerPt
    end
 end

 if DoFRBlastDetect,
    if (CalcSc == 1)
        SerFRBlast(snr)   = NumbErrFRBlast / BlockCntr / FrameSize
/ NumFrameInABlock / N
    else
        BerFRBlast(snr)   = NumbErrFRBlast / BlockCntr / FrameSize
/ NumFrameInABlock / N / NumbBitPerPt
    end
 end

% change the name of the saved data in order to avoid confusion

%%Ser
%save('SavedData','SerMl','SerMmse','SerVBlast','SerFBlast','SerFR
Blast')

%Ber
save('SavedData_MMSE_VB_B10000_E50000','BerMmse','BerVBlast')

end % end for snr

%%%%%%%%%%%%%%%%%%%%%
% SER semilogy Plot %
%%%%%%%%%%%%%%%%%%%%%

%%%%    remove unlaunched detection !!!
if (CalcSc == 1)
    figure(1)
    semilogy(SnrDb,SerMmse,'--',SnrDb,SerVBlast,'*-
',SnrDb,SerFRBlast,'o-',SnrDb,SerFBlast,'+-');
    % enter proper constellation size
    title('SER For Various Detectors M=64');
    % enter proper search space size and starting layer
    legend('MMSE','V-BLAST','FR-BLAST W=9 L=1','F-BLAST L=1');
    xlabel('SNR, dB');
    ylabel('SER');
else
    figure(1)
    semilogy(SnrDb,BerMmse,'--',SnrDb,BerVBlast,'*-
',SnrDb,BerFRBlast,'o-',SnrDb,BerFBlast,'+-');
    % enter proper constellation size
    title('BER For Various Detectors M=64');
    % enter proper search space size and starting layer
    legend('MMSE','V-BLAST','FR-BLAST W=9 L=1','F-BLAST L=1');
```

```
    xlabel('SNR, dB');
    ylabel('BER');
end
```

2- Main Function (Real-Valued) Detection

```
%=================================================================
========
% This program was written by Arsene Pankeu Yomi, University of
Alberta
% It was designed in order to evaluate the SER of different MIMO
detectors
% in a typical Rayleigh channel; with the help of Amir Alimohammad
% and Fouladi Fard in the begining.
%=================================================================
========

clear;clc;close all;
%%%% attention 'save' and 'change' the name of the saved data
%%%% will work only for 16Q 64Q 128Q and 256Q

% main parameters
NumbBlock   = 10^4; % each block contains    2 * ConstellationSize
                    %                        * NUmber of frame in
a block
                    %                      samples
CalcSc      = 0;    % 1-> SER     0-> BER
ModSc       = 1;    % 1->16-qam   2->64-qam   3->256-qam 4->1024-
qam
N           = 4;    % number of antennas; n_t = n_r = N
NumFrameInABlock   = 10;
MaxSnr      = 40;
MinSnr      = 0;
StepSnr     = 2;  % 1 or 2 e.g: stepSnr=2 --[MinSnr, MinSnr+1,
MinSnr+2...]

SnrDb       = MinSnr:StepSnr:MaxSnr;
SNR         = 10.^(SnrDb/10);
LSnr        = length(SNR);

% The minimum # of error is the same for all or decreases as SNR
increases
%  MatOfErrs = 100 * [1 .5 .5 .4 .4 .3 .3 .2 .2 .15 .15 .1 .1 .09
.09 .08 .08 .07 .07 .06 .06 .05 .05 .04 .04 .03 .03 .02 0.02 0.01
0.01];
 MatOfErrs  = 25000 * ones(1,MaxSnr + 1);

% Launch detection
% '1' In order to launch detection scheme otherwise '0'

% Complex detection
DoMmseDetect          = 1;   % MMSE detector
DoVBlastDetect        = 1;   % V-BLAST detector
```

```
DoFBlastDetect        = 1;   % F-BLAST detector; starting layer to
be indicated below

% Real detection
DoRealMmseDetect      = 1;   % MMSE detector
DoRealVBlastDetect    = 1;   % V-BLAST detector
DoRealFBlastDetect    = 1;   % F-BLAST detector; starting layer to
be indicated below

% Initialization of SER

if (DoMmseDetect == 1)
    if (CalcSc == 1)
        SerMmse   = zeros(1,LSnr);
    else
        BerMmse   = zeros(1,LSnr);
    end
end

if (DoVBlastDetect == 1)
    if (CalcSc == 1)
        SerVBlast = zeros(1,LSnr);
    else
        BerVBlast = zeros(1,LSnr);
    end
end

if (DoFBlastDetect == 1)
    if (CalcSc == 1)
        SerFBlast = zeros(1,LSnr);
    else
        BerFBlast = zeros(1,LSnr);
    end
end

if (DoRealMmseDetect == 1)
    if (CalcSc == 1)
        SerRealMmse   = zeros(1,LSnr);
    else
        BerRealMmse   = zeros(1,LSnr);
    end
end

if (DoRealVBlastDetect == 1)
    if (CalcSc == 1)
        SerRealVBlast = zeros(1,LSnr);
    else
        BerRealVBlast = zeros(1,LSnr);
    end
end

if (DoRealFBlastDetect == 1)
    if (CalcSc == 1)
        SerRealFBlast = zeros(1,LSnr);
    else
```

```
        BerRealFBlast = zeros(1,LSnr);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Symbol Matrix, Constant and sample generation %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[SymbMat SignalVar ConstSize NumbBitPerPt Motif RealConstSize] =
Modulation(NumFrameInABlock,NumbBlock,N,ModSc);

[TotNumOfSampleInABlock FrameSize SentSymb] =
SampGen(NumFrameInABlock,NumbBlock,N,ConstSize,NumbBitPerPt,SymbMa
t);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Computation of SER for various detector %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for snr = 1:LSnr
    MinNoErrs          = MatOfErrs(SnrDb(snr) + 1);
    NoiseVar           = (N * SignalVar) / SNR(snr);

    if (DoMmseDetect == 1)
        NumbErrMmse        = 0;
    end

    if (DoVBlastDetect == 1)
        NumbErrVBlast      = 0;
    end

    if (DoFBlastDetect == 1)
        NumbErrFBlast      = 0;
    end

    if (DoRealMmseDetect == 1)
        NumbErrRealMmse    = 0;
    end

    if (DoRealVBlastDetect == 1)
        NumbErrRealVBlast  = 0;
    end

    if (DoRealFBlastDetect == 1)
        NumbErrRealFBlast  = 0;
    end

    Continue           = 1;
    BlockCntr          = 0;
    % the following line counts the number of block used for the
current snr and verifies that MinNoErrs had been reached
    while ((Continue == 1) && ((BlockCntr < NumbBlock)))
          Continue              = 0;

          ContinueMmse          = 0;
          ContinueVBlast        = 0;
```

```
          ContinueFBlast        = 0;
          ContinueRealMmse      = 0;
          ContinueRealVBlast    = 0;
          ContinueRealFBlast    = 0;

          BlockCntr         = BlockCntr + 1;

          % channel generation
          MIMOchResponse    = zeros(N,N,TotNumOfSampleInABlock);
          for ii = 1:NumFrameInABlock
              % generating CHANNEL
              H  = (randn(N) + i*randn(N)) * sqrt(0.5);
              for ii1 = 1:N
                  for ii2 = 1:N
                      MIMOchResponse(ii1, ii2, (1+(ii-
1)*FrameSize:ii*FrameSize)) = H(ii1, ii2); % one channel response
per frame
                  end
              end
          end

          % block of frames
          Block = SentSymb(:,(1+TotNumOfSampleInABlock*(BlockCntr-
1)):(BlockCntr*TotNumOfSampleInABlock));

          for frameCntr = 1:NumFrameInABlock

              %%%%%%%%%%
              % Channel %
              %%%%%%%%%%

              % complex value
              H   = MIMOchResponse(:, :, 1+(frameCntr-
1)*FrameSize);
              N_  =
sqrt(0.5)*(randn(N,FrameSize)+i*randn(N,FrameSize));
              W   = N_*sqrt(NoiseVar);
              % S is the current frame
              S   = Block(:,1+FrameSize*(frameCntr-
1):frameCntr*FrameSize);
              Y   = zeros(N, FrameSize);
              for ii = 1:FrameSize
                  % Y=H*S+W
Y(:,ii)=MIMOchResponse(:,:,ii+(frameCntr-1)*FrameSize) * S(:,ii) +
W(:,ii);
              end

              % real value
              RealH          = [real(H) -imag(H);imag(H) real(H)];

              RealY          = zeros(2*N, FrameSize);
              RealS          = zeros(2*N, FrameSize);
              RealW          = zeros(2*N, FrameSize);
              for ii = 1:FrameSize
                  TempRealY  = zeros(N, FrameSize);
```

```
                TempImagY  = zeros(N, FrameSize);
                TempRealS  = zeros(N, FrameSize);
                TempImagS  = zeros(N, FrameSize);
                TempRealW  = zeros(N, FrameSize);
                TempImagW  = zeros(N, FrameSize);
                for hh = 1:N
                    TempRealY(hh,ii) = real(Y(hh,ii));
                    TempImagY(hh,ii) = imag(Y(hh,ii));
                    TempRealS(hh,ii) = real(S(hh,ii));
                    TempImagS(hh,ii) = imag(S(hh,ii));
                    TempRealW(hh,ii) = real(W(hh,ii));
                    TempImagW(hh,ii) = imag(W(hh,ii));
                end



                RealY(:,ii) = [TempRealY(:,ii);TempImagY(:,ii)];
                RealS(:,ii) = [TempRealS(:,ii);TempImagS(:,ii)];
                RealW(:,ii) = [TempRealW(:,ii);TempImagW(:,ii)];
            end

            %%%%%%%%%%%%%%%%%%%%%
            % Detection process %
            %%%%%%%%%%%%%%%%%%%%%

            % MMSE matrix

            % complex value
            G = inv(H' * H + (1 / SNR(snr)) * eye(N)) * H';
            [SortedLayer IndMI] = SortCol(H,N);

            % real value
        RealG = inv(RealH' * RealH + (1 / SNR(snr)) * eye(2*N))
* RealH';
            [RealSortedLayer RealIndMI] = SortCol(RealH,2*N);

            %%%% MMSE detection %%%%
            if (DoMmseDetect == 1)
                for vectorCntr = 1:FrameSize
                   [YMmse] =
Mmse(ConstSize,N,Y(:,vectorCntr),SymbMat,G);
                    % error calculation
                    if (CalcSc == 1)
                      [MmseDetect] = Ser(YMmse,S(:,vectorCntr));
                    else
            [MmseDetect] =
Ber(ConstSize,N,YMmse,SymbMat,S(:,vectorCntr));
                    end
                     % counts the number of symbol in error in
the block
                     NumbErrMmse   = NumbErrMmse + MmseDetect;
                end
                if (NumbErrMmse < MinNoErrs)
                   ContinueMmse = 1;
                else
```

```
                    ContinueMmse = 0;
                end
            end

            %%%% V-BLAST detection %%%%
            if (DoVBlastDetect == 1)
                for vectorCntr = 1:FrameSize
                    % initialize position of the undetected
symbols
                    Pos = 1:N;
                    % initialize detected symbol vector
                    YVB = zeros(N,1);

 [YVBlast] =
VBlast(H,ConstSize,N,Y(:,vectorCntr),SymbMat,Pos,YVB,SNR,snr);
                    % error calculation
                    if (CalcSc == 1)
                      [VBlastDetect] =
Ser(YVBlast,S(:,vectorCntr));
                    else
         [VBlastDetect] =
Ber(ConstSize,N,YVBlast,SymbMat,S(:,vectorCntr));
                    end
                    % Counts the number of symbol in error in
the block
                    NumbErrVBlast    = NumbErrVBlast +
VBlastDetect;
                end
                if (NumbErrVBlast < MinNoErrs)
                   ContinueVBlast = 1;
                else
                   ContinueVBlast = 0;
                end
            end

            %%%% F-BLAST detection %%%%
            if (DoFBlastDetect == 1)
                for vectorCntr = 1:FrameSize
       [YFBlast] =
FBlast(H,ConstSize,N,Y(:,vectorCntr),SymbMat,SortedLayer(1),SNR,sn
r);
                    % error calculation
                    if (CalcSc == 1)
                      [FBlastDetect] =
Ser(YFBlast,S(:,vectorCntr));
                    else
         [FBlastDetect] =
Ber(ConstSize,N,YFBlast,SymbMat,S(:,vectorCntr));
                    end
                    % Counts the number of symbol in error in
the block
                    NumbErrFBlast    = NumbErrFBlast +
FBlastDetect;
                end
                if (NumbErrFBlast < MinNoErrs)
                   ContinueFBlast = 1;
```

```
                    else
                       ContinueFBlast = 0;
                    end
                end

                %%%% REAL MMSE detection %%%%
                if (DoRealMmseDetect == 1)
                    for vectorCntr = 1:FrameSize
     [RealYMmse] =
Mmse(RealConstSize,2*N,RealY(:,vectorCntr),Motif,RealG);

                        % complex transformation
                        YReal = zeros(N,1);
                        for ii = 1:N
                          YReal(ii) = RealYMmse(ii)+ i *
RealYMmse(N+ii);
                        end

                        % error calculation
                        if (CalcSc == 1)
                          [RealMmseDetect] =
Ser(YReal,S(:,vectorCntr));
                        else
          [RealMmseDetect] =
Ber(ConstSize,N,YReal,SymbMat,S(:,vectorCntr));
                        end
                         % counts the number of symbol in error in
the block
                        NumbErrRealMmse   = NumbErrRealMmse +
RealMmseDetect;
                    end
                    if (NumbErrRealMmse < MinNoErrs)
                       ContinueRealMmse = 1;
                    else
                       ContinueRealMmse = 0;
                    end
                end

                %%%% REAL V-BLAST detection %%%%
                if (DoRealVBlastDetect == 1)
                    for vectorCntr = 1:FrameSize
                        % initialize position of the undetected
symbols
                        Pos = 1:2*N;
                        % initialize detected symbol vector
                        YVB = zeros(2*N,1);

 [RealYVBlast] =
VBlast(RealH,RealConstSize,2*N,RealY(:,vectorCntr),Motif,Pos,YVB,S
NR,snr);

                         % complex transformation
                        YReal = zeros(N,1);
                        for ii = 1:N
```

```
                        YReal(ii) = RealYVBlast(ii) + i *
RealYVBlast(N+ii);
                    end

                    % error calculation
                    if (CalcSc == 1)
                   [RealVBlastDetect] =
Ser(YReal,S(:,vectorCntr));
                    else
 [RealVBlastDetect] =
Ber(ConstSize,N,YReal,SymbMat,S(:,vectorCntr));
                    end
                    % Counts the number of symbol in error in
the block
               NumbErrRealVBlast    = NumbErrRealVBlast +
RealVBlastDetect;
                end
                if (NumbErrRealVBlast < MinNoErrs)
                   ContinueRealVBlast = 1;
                else
                   ContinueRealVBlast = 0;
                end
            end

            %%%% REAL F-BLAST detection %%%%
            if (DoRealFBlastDetect == 1)
                for vectorCntr = 1:FrameSize
       [RealYFBlast] =
FBlast(RealH,RealConstSize,2*N,RealY(:,vectorCntr),Motif,RealSorte
dLayer(1),SNR,snr);

                    % complex transformation
                    YReal = zeros(N,1);
                    for ii = 1:N
                      YReal(ii) = RealYFBlast(ii)+ i *
RealYFBlast(N+ii);
                    end

                    % error calculation
                    if (CalcSc == 1)
                   [RealFBlastDetect] =
Ser(YReal,S(:,vectorCntr));
                    else
 [RealFBlastDetect] =
Ber(ConstSize,N,YReal,SymbMat,S(:,vectorCntr));
                    end
                    % Counts the number of symbol in error in
the block
               NumbErrRealFBlast    = NumbErrRealFBlast +
RealFBlastDetect;
                end
                if (NumbErrRealFBlast < MinNoErrs)
                   ContinueRealFBlast = 1;
                else
                   ContinueRealFBlast = 0;
```

```
                end
            end


        end % end frameCntr

        Intermediate = ContinueMmse + ContinueVBlast +
ContinueFBlast + ContinueRealVBlast + ContinueRealFBlast +
ContinueRealMmse;
        if (Intermediate == 0)
            Continue = 0;
        else
            Continue = 1;
        end

    end % while continue

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % SER computation for the current snr value %
    %         to be print on the screen         %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


 clc;
 if DoMmseDetect,
    if (CalcSc == 1)
        SerMmse(snr)         = NumbErrMmse / BlockCntr / FrameSize
/ NumFrameInABlock / N
    else
        BerMmse(snr)         = NumbErrMmse / BlockCntr / FrameSize
/ NumFrameInABlock / N / NumbBitPerPt
    end
 end

 if DoVBlastDetect,
    if (CalcSc == 1)
        SerVBlast(snr)       = NumbErrVBlast / BlockCntr /
FrameSize / NumFrameInABlock / N
    else
        BerVBlast(snr)       = NumbErrVBlast / BlockCntr /
FrameSize / NumFrameInABlock / N / NumbBitPerPt
    end
 end

 if DoFBlastDetect,
    if (CalcSc == 1)
        SerFBlast(snr)       = NumbErrFBlast / BlockCntr /
FrameSize / NumFrameInABlock / N
    else
        BerFBlast(snr)       = NumbErrFBlast / BlockCntr /
FrameSize / NumFrameInABlock / N / NumbBitPerPt
    end
 end

 if DoRealMmseDetect,
```

```
    if (CalcSc == 1)
        SerRealMmse(snr)     = NumbErrRealMmse / BlockCntr /
FrameSize / NumFrameInABlock / N
    else
        BerRealMmse(snr)     = NumbErrRealMmse / BlockCntr /
FrameSize / NumFrameInABlock / N / NumbBitPerPt
    end
 end

 if DoRealVBlastDetect,
    if (CalcSc == 1)
        SerRealVBlast(snr)   = NumbErrRealVBlast / BlockCntr /
FrameSize / NumFrameInABlock / N
    else
        BerRealVBlast(snr)   = NumbErrRealVBlast / BlockCntr /
FrameSize / NumFrameInABlock / N / NumbBitPerPt
    end
 end

 if DoRealFBlastDetect,
    if (CalcSc == 1)
        SerRealFBlast(snr)   = NumbErrRealFBlast / BlockCntr /
FrameSize / NumFrameInABlock / N
    else
        BerRealFBlast(snr)   = NumbErrRealFBlast / BlockCntr /
FrameSize / NumFrameInABlock / N / NumbBitPerPt
    end
 end

% change the name of the saved data in order to avoid confusion

%%Ser
%save('SavedData','SerMmse','SerVBlast','SerFBlast','SerRealMmse',
'SerRealVBlast','SerRealFBlast')

%Ber
save('SavedData','BerMmse','BerVBlast','BerFBlast','BerRealMmse','
BerRealVBlast','BerRealFBlast')

end % end for snr

%%%%%%%%%%%%%%%%%%%%%
% SER semilogy Plot %
%%%%%%%%%%%%%%%%%%%%%

%%%%    remove unlaunched detection !!!
if (CalcSc == 1)
    figure(1)
    semilogy(SnrDb,SerMmse,'--',SnrDb,SerRealMmse,'.-
',SnrDb,SerRealVBlast,'o-',SnrDb,SerVBlast,'+-
',SnrDb,SerRealFBlast,'s-',SnrDb,SerFBlast,'*-');
    % enter proper constellation size
    title('SER For Various Detectors M=64');
    legend('MMSE','REAL MMSE','REAL V-BLAST','V-BLAST','REAL F-
BLAST','F-BLAST');
```

```
    xlabel('SNR, dB');
    ylabel('SER');
else
    figure(1)
    semilogy(SnrDb,BerMmse,'--',SnrDb,BerRealMmse,'.-
',SnrDb,BerRealVBlast,'o-',SnrDb,BerVBlast,'+-
',SnrDb,BerRealFBlast,'s-',SnrDb,BerFBlast,'*-');
    % enter proper constellation size
    title('BER For Various Detectors M=64');
    legend('MMSE','REAL MMSE','REAL V-BLAST','V-BLAST','REAL F-
BLAST','F-BLAST');
    xlabel('SNR, dB');
    ylabel('BER');
end
```

3- SER

```
function [Error] = Ser(E,Q)
% This function returns number of Symbol in error in detected
symbol vector
% Usage:
%        [YMmse] = Ser(E,F,G)
% Input:
%             E = noisy transmitted vector Y
%             Q = transmitted vector S
% Output:
%         Error : number of symbol in error

Error  = nnz(E - Q);

end
```

4- BER

```
function [Er] = Ber(b,c,E,F,Q)
% This function returns number of bit in error in detected symbol
vector
% Usage:
%        [Er] = Ber(b,c,E,F,Q)
% Input:
%             c = number of receiver antennas (same as
transmitter)
%             b = constellation size
%             E = noisy transmitted vector Y
%             F = matrice of symbol from the constellation
%             Q = transmitted vector S
% Output:
%         Er : number of bit in error

% Initialization
PosNoisyTrans       = zeros(1,c);
```

```
PosTrans            = zeros(1,c);

for i = 1:c
    % position of the noisy transmitted vector
    [val PosNoisyTrans(i)] = min(E(i) * ones(b,1) - F);

    % position of the transmitted vector
    [val PosTrans(i)]      = min(Q(i) * ones(b,1) - F);
end
Er = nnz(dec2bin(PosNoisyTrans - 1,log2(b))-dec2bin(PosTrans -
1,log2(b)));

end
```

5- Modulation

```
function [SymbMat SignalVar ConstSize NumbBitPerPt StatMat] =
Modulation(a,b,c,d)
% This function returns uncorellated Modulated symbols.
% Usage:
%        [SymbMat SignalVar ConstSize NumbBitPerPt StatMat] =
Modulation(a,b,c,d)
% Input:
%
%             a = number of frame in a block
%             b = number of blocks
%             c = number of transmitter antennas
%             d = Modulation scheme : 1 ->16qam    2 ->64qam   3 -
>128qam
%                                     4 ->256qam   5 ->512qam  6 -
>1024qam
% Output:
%
% SignalVar              : Average Symbol energy (E{x[i]^2})
% SymbMat                : Constellation matrice
% ConstSize              : Number of point in the constellation
% NumbBitPerPt           : Number of bits per point
% StatMat                : Subset for the restricted search


         if     (d==1)
                M              = 16;
                NumbBitPerPt   = 4 ;
                SignalVar      = 10;

         elseif (d==2)
                M              = 64;
                NumbBitPerPt   = 6 ;
                SignalVar      = 42;
                load SymetricalSearchSpace64Q.mat;
```

```
         elseif (d==3)
                M              = 128;
                NumbBitPerPt   = 7 ;
                SignalVar      = 82;
                load SymetricalSearchSpace128Q.mat;

         elseif (d==4)
                M              = 256;
                NumbBitPerPt   = 8 ;
                SignalVar      = 170;
                load SymetricalSearchSpace256Q.mat;

        elseif (d==5)
                M              = 512;
                NumbBitPerPt   = 9 ;
                SignalVar      = 330;

        elseif (d==6)
                M              = 1024;
                NumbBitPerPt   = 10 ;
                SignalVar      = 682;
         end

         x          = [0:M-1];
         %matlab function for gray mapping with QAM constellation
         SymbMat    =
modulate(modem.qammod('M',M,'SymbolOrder','Gray'),x);
         SymbMat    = SymbMat.';
         ConstSize  = M;
         %SignalVar  = SymbMat' * SymbMat / ConstSize

         %FrameSize  = 2*ConstSize;

         % a sample is a received vector, i.e a set of N symbols
         NSymbol    = b * 2 * ConstSize * a * c;

         %generate bits
         SentBits   = randint(1,NSymbol*NumbBitPerPt);

         %generate symbols gray mapping
         Symb       = zeros(1,NSymbol);
         for r = 1:NSymbol
             t = 0;
             for zz = 1:NumbBitPerPt
                 t  = t+2^(zz-1)*SentBits(zz+(r-1)*NumbBitPerPt);
             end
             Symb(r) = SymbMat(t+1);
         end
end
```

6- FixedSearSet

```
function [Subset] = FixedSearchSet(A,b,c,d)
% This function returns searchset for the noisiest layer, 64QAM
% Usage:
%        [Subset] = FixedSearchSet(a,b,c,d)
% Input:
%             A = Matrix of statistic
%             b = Mmse noisiest estimate
%             c = size of the fixed search set
%             d = ConstSize
%
%
% Output:
%    Subset          : set of candidate symbol

    [symb indx]             = min(abs(b * ones(1, d) - A(1,:)));
    A(indx+1,indx)          = 0;
    Subset                  = [b];
    for kk = 1:c-1
        [value position]    = max(A(indx+1,:));
        A(indx+1,position)  = 0;
        Subset              = [Subset,A(1,position)];
    end
end
```

7- SampGen

```
function [TotNumOfSampleInABlock FrameSize SentSymb] =
SampGen(a,b,c,d,e,F)
% This function returns uncorellated Modulated symbols.
% Usage:
%        [SymbMat SignalVar ConstSize Symb StatMat] =
Modulation(a,b,c,d)
% Input:
%
%             a = number of frame in a block
%             b = number of blocks
%             c = number of transmitter antennas
%             d = Constsize
%             e = Number of bits per constellation point
%             F = Matrix of constelaltion symbol
% Output:
%
% TotNumOfSampleInABlock : Number of sample in a block
% Frame Size             : Size of a frame of sample vector
% SentSymb               : matrix of sample vector

FrameSize              = 2 * d;
TotNumOfSampleInABlock = FrameSize * a;

% generate samples - SentSymb - vector of N symbols
NSymbol                = b * TotNumOfSampleInABlock * c;
```

```
%generate bits
SentBits              = randint(1,NSymbol*e);

%generate symbols gray mapping
Symb                  = zeros(1,NSymbol);
for r = 1:NSymbol
        t             = 0;
    for zz = 1:e
        t             = t+2^(zz-1)*SentBits(zz+(r-1)*e);
    end
    Symb(r)           = F(t+1);
end

% zeros(N,NSymbol/N)
SentSymb              = zeros(c,TotNumOfSampleInABlock*b);
    for NSymb = 1:c:NSymbol-c+1
        SentSymb(:,(NSymb+(c-1))*(1/c)) =
transpose(Symb(NSymb:NSymb+c-1));
    end

end
```

8- SortCol

```
function [SortedLayer IndMI] = SortCol(A,c)
% This function returns index of layers from the weakest to the
strongest
% Usage:
%       function [SortedLayer IndMI] = Sortcol(A)
% Input:
%            A = H matrix
%            c = number of receiver antennas (same as transmitter)
% Output:
%  SortedLayer = Matrix of the index
%                1-weakest layer 2-second weakest layer ...
%                ...(c-1)-second strongest layer  c-strongest
layer
%        IndMI = Index of the layer which has maximum interference
on the
%                Strongest Layer

     % Initialization
     SortedLayer             = zeros(1,c);
     n                       = 1;

     AmpColH                 = [];
     for ii                  = 1:c
        AmpColH              = [AmpColH,norm(A(:,ii))];
     end
     while ( n < c + 1 )
        [Val Ind]            = min(AmpColH);
        SortedLayer(n)       = Ind;
```

```
        AmpColH(Ind)          = 10000;
        n                     = n + 1;
    end

    AmpH                      = [];
    for ii2                   = 1:c
        AmpH                  = [AmpH, norm(SortedLayer(c),ii2)];
    end
    AmpH(SortedLayer(c))      = 0;
    [val1 IndMI]              = max(AmpH);

end
```

9- MMSE

```
function [YMmse] = Mmse(b,c,E,F,G)
% This function returns symbol vector from the MMSE detection
process
% Usage:
%        [YMmse] = Mmse(b,c,E,F,G)
% Input:
%             b = constellation size
%             c = number of receiver antennas (same as
transmitter)
%             E = noisy transmitted vector Y
%             F = matrice of symbol from the constellation
%             G = MMSE matrix
% Output:
%         YMmse : output of the Mmse detection

                o  = ones(1, b);
                op = ones(c, 1);
                % MMSE output vector
                TempVec         = G * E;
                % slicing
                [temp IndMmse]  = min(abs(TempVec * o - op *
F.')');
                YMmse           = F(IndMmse);
end
```

10- V-BLAST

```
function [YVBlast] = VBlast(A,b,c,E,F,K,L,M,n)
% This function returns symbol vector from the V-BLAST detection
process
% Usage:
%        [YVBlast] = VBlast(A,b,c,E,F,K,L,M,n)
% Input:
%             A = H matrix
%             b = constellation size
%             c = number of receiver antennas (same as
transmitter)
%             E = noisy transmitted vector Y
%             F = matrix of symbol from the constellation
%             K = indice(s) of the symbol(s) to be detected
%             L = YVBlast, initial value
%             M = SNR
%             n = snr
% Output:
%       YVBlast : output of the VBLAST detection

        YVBlast   = L;
        H_        = A;
        Y_        = E;
        for IndSymb = 1:c
            % MMSE matrix
            G_ = inv(H_' * H_ + (1 / M(n)) * eye(c - (IndSymb -
1))) * H_';

            % ordering: detection goes from strongest to weakest
layer
            % detection starts with the row of G_ which has
minimum norm

            % norm of G_'s row
            AmpG_               = [];
            for ii              = 1:c-(IndSymb-1)
                AmpG_           = [AmpG_, norm(G_(ii, :))^2];
            end
            [val1 GRowInd]      = min(AmpG_);

            % nulling: MMSE is used to detect symbol from each
layer
            CurDtdSymb          = G_(GRowInd, :) * Y_;

            % slicing
            Dist                = abs(F - ones(b,1) *
CurDtdSymb).^2;
            [val2, IndVBlast]   = min(Dist);
            YVBlast(K(GRowInd)) = F(IndVBlast);

            % cancellation : produces deflation in columns of H_
            %                removes interference from detected
symbol
            Y_     = Y_ - H_(:, GRowInd) * YVBlast(K(GRowInd));
```

```
            HTemp  = [];
            Temp   = [];
            for ii = 1:c-(IndSymb-1),
                if (ii  ~= GRowInd)
                   HTemp = [HTemp, H_(:, ii)];
                   Temp  = [Temp, K(ii)];
                end
            end
            H_     = HTemp;
            K      = Temp;
        end

end
```

11- FR-BLAST

```
function [YFRBlast] = FRBlast(A,b,c,E,F,G,i,M,n,k,S)
% This function returns symbol vector from the FR-BLAST detection
process
% Usage:
%        [YFRBlast] = FRBlast(A,b,c,E,F,G,M,n,k,S)
% Input:
%             A   = H matrix
%             b   = constellation size
%             c   = number of receiver antennas (same as
transmitter)
%             E   = noisy transmitted vector Y
%             F   = matrix of symbol from the constellation
%             G   = MMSE matrix
%             i   = Index of the first layer to be detected
%                   1-weakest layer 2-second weakest layer ...
%                   ...(c-1)-second strongest layer  c-strongest
layer
%             M   = SNR
%             n   = snr
%             k   = size of the fixed search space
%             S   = Matrix of statistic to build the subset
% Output:
%      YFRBlast : output of the FRBLAST detection

     YFRBlastTest  = zeros(c,k);
     Epsilon       = zeros(c,k);
     % a search inside a search space is performed on the first
layer to be
     % detected, i.e. detection starts with the 'i'-th layer
     % conventional VBLAST for the N-1 remaining layers
     % norm of G's row

     % original MMSE is run to estimate the position of the first
symbol
     [YMmse]       = Mmse(b,c,E,F,G);

     % now we build a set of k closest symbol including this one
     [Subset]      = FixedSearchSet(S,YMmse(i),k,b);
```

```
    % search inside the subset for the first symbol,
    for ii = 1:k
        H_                                      = A;
        UnDtdSymb                               = 1:c;
        Y_                                      = E;
        YFRBlastTest(UnDtdSymb(i),ii)           = Subset(ii);

        % cancellation : produces deflation in columns of H_
        %                removes interference from temptative
symbol
        Y_        = Y_ - H_(:,i) * YFRBlastTest(UnDtdSymb(i),ii);
        HTemp1    = [];
        Temp1     = [];
        for ii1   = 1:c
            if (ii1 ~= i)
               HTemp1   = [HTemp1,H_(:,ii1)];
               Temp1    = [Temp1, UnDtdSymb(ii1)];
            end
        end
        H_        = HTemp1;
        UnDtdSymb = Temp1;

        % Original V-BLAST detection over the N-1 remaining
layers
       [YVBlast] = VBlast(H_,b,c-
1,Y_,F,UnDtdSymb,YFRBlastTest(:,ii),M,n);
        YFRBlastTest(:,ii) = YVBlast;

        % Compute noise for all candidate
        Epsilon(:,ii) = A * YFRBlastTest(:,ii) - E;
    end

    % Find the best candidate
    AmpEps  = [];
    for ii1 = 1:k
        AmpEps = [AmpEps, norm(Epsilon(:,ii1))];
    end
    [val4 MinEpsInd] = min(AmpEps);
    YFRBlast         = YFRBlastTest(:,MinEpsInd);
end
```

12- F-BLAST

```
function [YFBlast] = FBlast(A,b,c,E,F,i,M,n)
% This function returns symbol vector from the F-BLAST detection
process
% Usage:
%        [YFBlast] = FBlast(A,b,c,E,F,i,M,n)
% Input:
%             A   = H matrix
%             b   = constellation size
%             c   = number of receiver antennas (same as
transmitter)
%             E   = noisy transmitted vector Y
%             F   = matrix of symbol from the constellation
%             i   = Index of the first layer to be detected
%                   1-weakest layer 2-second weakest layer ...
%                   ...(c-1)-second strongest layer  c-strongest
layer
%             M   = SNR
%             n   = snr
% Output:
%       YFBlast : output of the FBLAST detection

     YFBlastTest = zeros(c,b);
     Epsilon     = zeros(c,b);
     % an exhaustive search is performed on the first layer to be
detected
     % i.e detection starts with the 'i'_th layer
     % conventional VBLAST for the N-1 remaining symbols

     % Exhaustive search on the first layer
     for ii = 1:b
         H_                                  = A;
         UnDtdSymb                           = 1:c;
         Y_                                  = E;
         YFBlastTest(UnDtdSymb(i),ii)        = F(ii);
         % cancellation : produces deflation in columns of H_
         %                removes interference from temptative
symbol
         Y_              = Y_ - H_(:,i) *
YFBlastTest(UnDtdSymb(i),ii);
         HTemp1          = [];
         Temp1           = [];
         for ii1         = 1:c
             if (ii1    ~= i)
                HTemp1   = [HTemp1,H_(:,ii1)];
                Temp1    = [Temp1, UnDtdSymb(ii1)];
             end
         end
         H_              = HTemp1;
         UnDtdSymb       = Temp1;

         % Original V-BLAST detection over the N-1 remaining
layers
```

```
       [YVBlast]   = VBlast(H_,b,c-
1,Y_,F,UnDtdSymb,YFBlastTest(:,ii),M,n);
         YFBlastTest(:,ii) = YVBlast;
         % Compute noise for all candidate
         Epsilon(:,ii)     = A * YFBlastTest(:,ii) - E;
     end
     % Find the best candidate
     AmpEps           =[];
     for ii1          = 1:b
         AmpEps       = [AmpEps, norm(Epsilon(:,ii1))];
     end
     [val4 MinEpsInd] = min(AmpEps);
     YFBlast          = YFBlastTest(:,MinEpsInd);
end
```

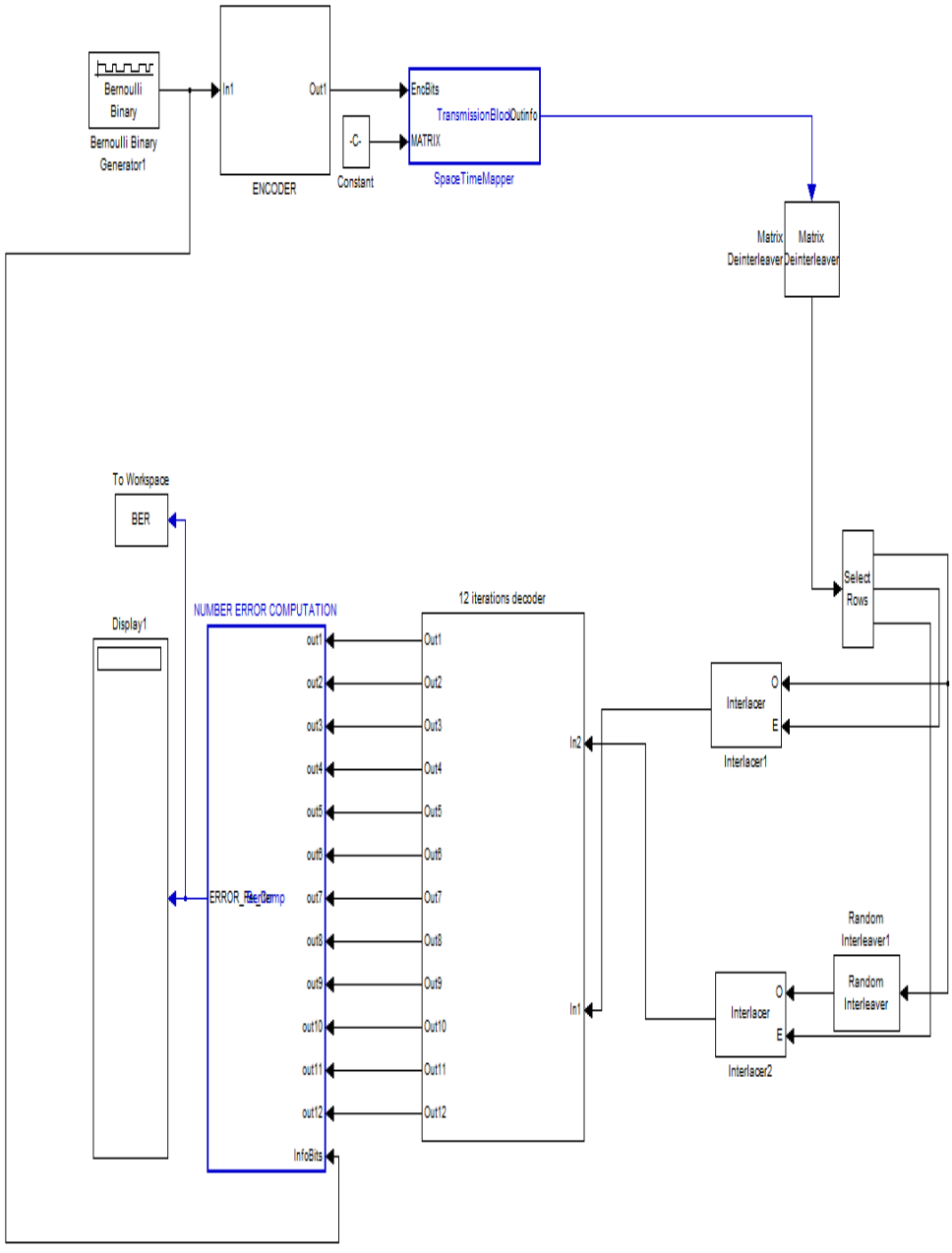
13- ML

```
function [YMl] = Ml(A,b,c,E,F)
% This function returns symbol vector from the ML detection
process
% Usage:
%        [YMl] = Ml(A,b,c,E,F)
% Input:
%             A = H matrix
%             b = constellation size
%             c = number of receiver antennas (same as
transmitter)
%             E = noisy received vector Y
%             F = matrix of symbols from the constellation
% Output:
%           YMl : output of the Ml detection

    % Initialization
             ErrMl    = zeros(1,b^c);
             TempVec  = zeros(c,b^c);
             n        = 1;

             if ( c == 2 ) % 2 antennas
                 for ii1 = 1:b
                     for ii2 = 1:b
                         TempVec(:,n) = [F(ii1);F(ii2)];
                         ErrMl(n)     = (norm(E - A *
TempVec(:,n)))^2;
                         n            = n + 1;
                     end
                 end
             elseif ( c == 3 ) % 3 antennas
                  for ii1 = 1:b
                     for ii2 = 1:b
                         for ii3 = 1:b
                             TempVec(:,n) =
[F(ii1);F(ii2);F(ii3)];
```

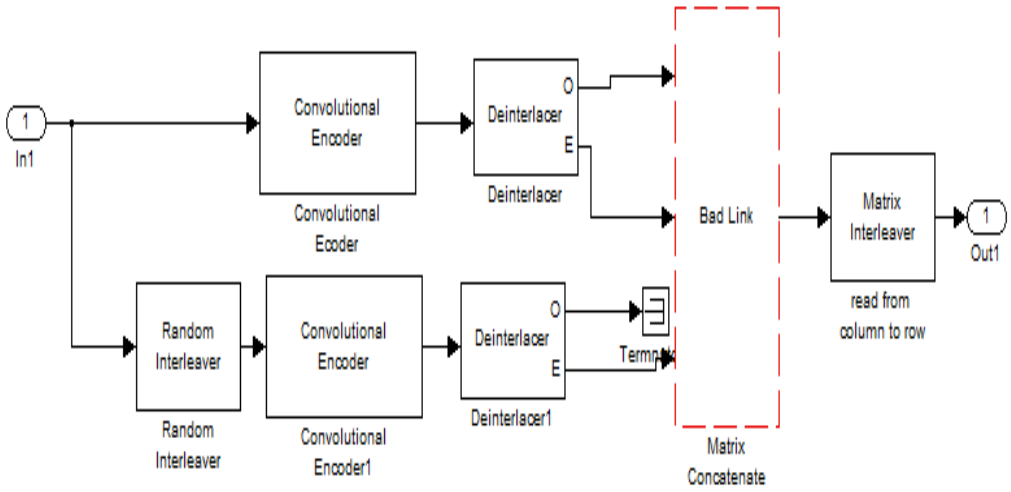
```
                            ErrMl(n)     = (norm(E - A *
TempVec(:,n)))^2;
                            n            = n + 1;
                        end
                    end
                end
            elseif ( c == 4 ) % 4 antennas
                for ii1 = 1:b
                   for ii2 = 1:b
                       for ii3 = 1:b
                           for ii4 = 1:b
                       TempVec(:,n) =
[F(ii1);F(ii2);F(ii3);F(ii4)];
                       ErrMl(n)     = (norm(E - A *
TempVec(:,n)))^2;
                       n            = n + 1;
                           end
                       end
                    end
                end
            elseif ( c == 5 ) % 5 antennas
                   for ii1 = 1:b
                       for ii2 = 1:b
                           for ii3 = 1:b
                               for ii4 = 1:b
                                   for ii5 = 1:b
        TempVec(:,n) = [F(ii1);F(ii2);F(ii3);F(ii4);F(ii5)];
        ErrMl(n)     = (norm(E - A * TempVec(:,n)))^2;
        n            = n + 1;
                                   end
                               end
                           end
                       end
                   end
            elseif ( c == 6 ) % 6 antennas
                   for ii1 = 1:b
                       for ii2 = 1:b
                           for ii3 = 1:b
                               for ii4 = 1:b
                                   for ii5 = 1:b
                                       for ii6 = 1:b
        TempVec(:,n) =
[F(ii1);F(ii2);F(ii3);F(ii4);F(ii5);F(ii6)];
        ErrMl(n)     = (norm(E - A * TempVec(:,n)))^2;
        n            = n + 1;
                                       end
                                   end
                               end
                           end
                       end
                   end
            end

           [C,IndMl] = min(ErrMl);
           YMl       = TempVec(:,IndMl);
end
```

5. Matlab Scripts and Simulink Models for the Coded System

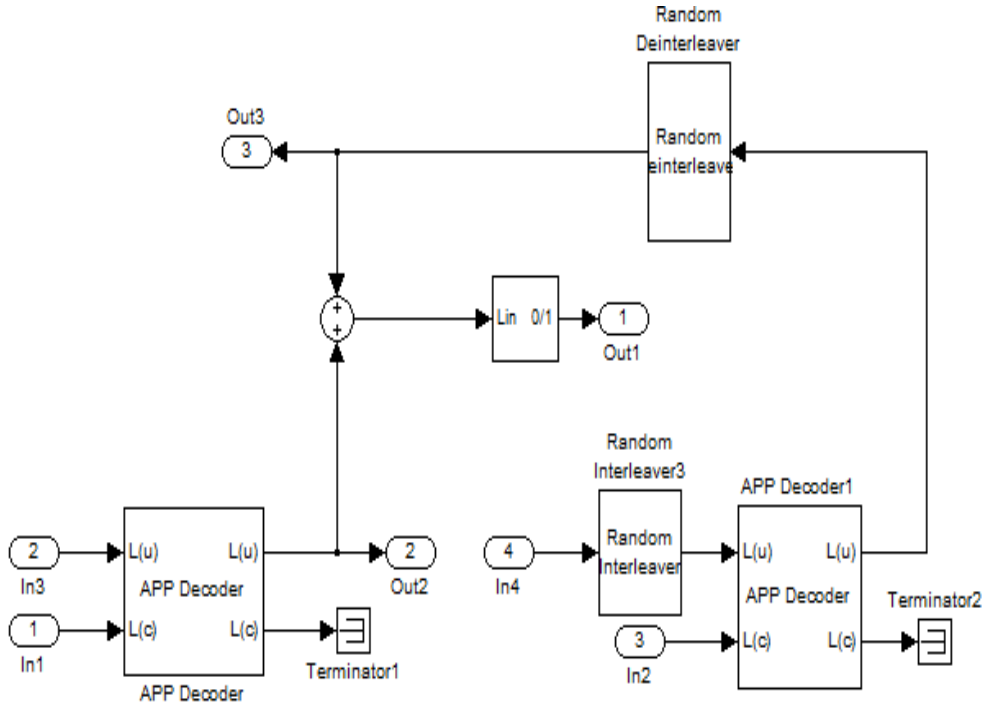
1- Simulink Model



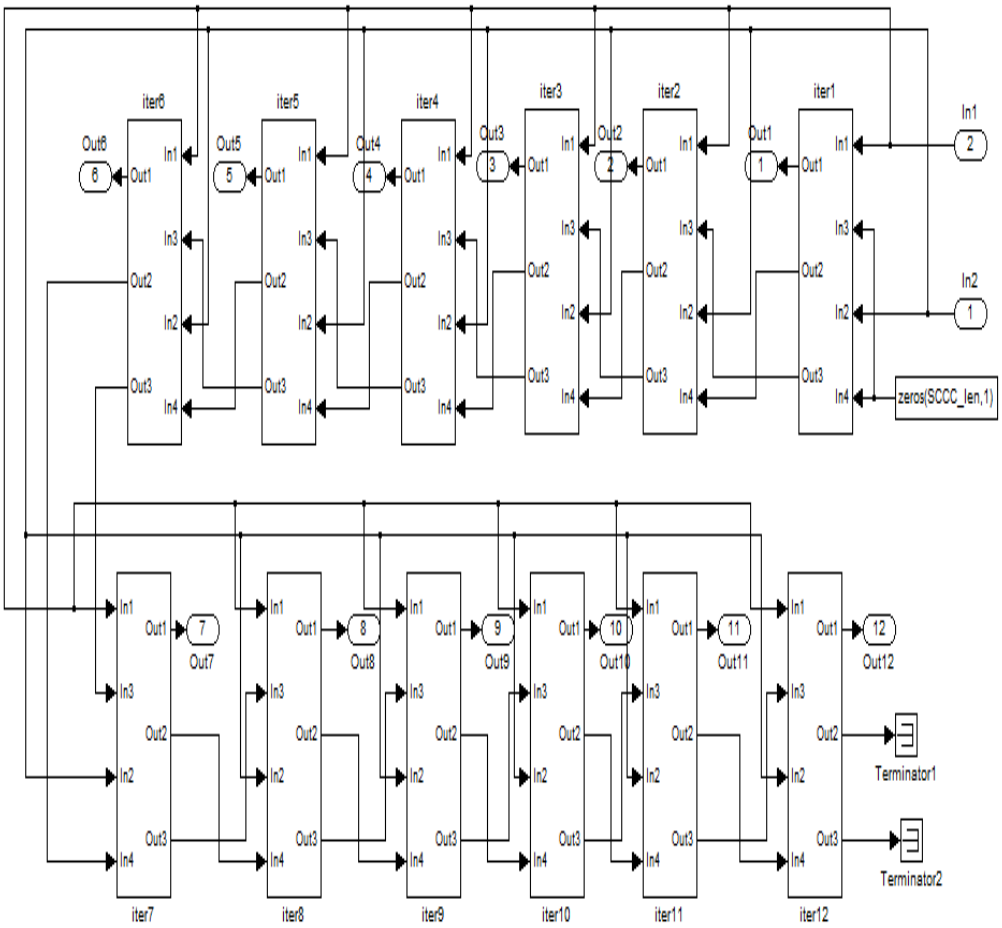
2- Simulink Turbo Encoder



3- Simulink Turbo Decoder



4- Simulink Turbo Decoder with 12 Iterations



5- Space-Time Mapper Matlab - Embedded Function

```
function Outinfo = TransmissionBlock(EncBits, MATRIX)

% NoiseVar=(2*CodeRate * #bit/pt * EbNo linear)^-1 =
N*SignalVariance/SNR

% parameters
SCCC_R = MATRIX(1); SCCC_len = MATRIX(2); N = MATRIX(3); ModSc =
MATRIX(4);
DoNoDetector = MATRIX(5); DoMmseDetect = MATRIX(6);
DoVBlastDetect = MATRIX(7); DoFBlastDetect = MATRIX(8);
DoFRBlastDetect = MATRIX(9); DoRealMmseDetect = MATRIX(10);
DoRealVBlastDetect = MATRIX(11); DoRealFBlastDetect = MATRIX(12);
SizeOfSphere = MATRIX(13); FBStartLayer = MATRIX(14);
FRBStartLayer = MATRIX(15); SNR =MATRIX(16);

% initialization of the call of external function
eml.extrinsic('Modulation','SampleNChannel','Detection');

% initialization of complex computation
i = sqrt(complex(-1));
j = sqrt(complex(-1));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Symbol Matrix, Constant and sample generation %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[SymbMat SignalVar ConstSize NumbBitPerPt StatMat Motif] =
Modulation(ModSc);

FrameSize              = 2 * ConstSize; % number of samples in a
frame

NumFrameInABlock       = SCCC_len / (SCCC_R * NumbBitPerPt * N *
FrameSize); % number of frames in a block

TotNumOfSampleInABlock = FrameSize * NumFrameInABlock; % number of
samples in a block

% 1- sample block generation
NSymbol     = length(EncBits) / NumbBitPerPt;
NoiseVar    = (N * SignalVar) / SNR;

[SampleBlock MIMOchResponse] =
SampleNChannel(NSymbol,NumbBitPerPt,EncBits,SymbMat,N,TotNumOfSamp
leInABlock,NumFrameInABlock,FrameSize);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Computation of output sequence of soft bits for various detector
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Outinfo   = zeros(size(EncBits));
```

```
[Outinfo] = Detection(NumFrameInABlock, N, FrameSize, NoiseVar,
SNR, SampleBlock, MIMOchResponse, DoNoDetector,ModSc,
DoMmseDetect, DoVBlastDetect, DoFRBlastDetect, DoFBlastDetect,
DoRealMmseDetect, DoRealVBlastDetect, DoRealFBlastDetect,
ConstSize, SymbMat, StatMat, SizeOfSphere, Motif, FBStartLayer,
FRBStartLayer);

end
```

6- Number Error Computation - Embedded Function

```
function ERROR_Per_Iter = BerComp(out1, out2, out3, out4, out5,
out6, out7, out8, out9, out10, out11, out12, InfoBits)

% initialization of the call of external function
eml.extrinsic('nnz');
ERROR_Per_Iter = zeros(double(12),1);

ERROR_Per_Iter(1)  = nnz(out1 - InfoBits);
ERROR_Per_Iter(2)  = nnz(out2 - InfoBits);
ERROR_Per_Iter(3)  = nnz(out3 - InfoBits);
ERROR_Per_Iter(4)  = nnz(out4 - InfoBits);
ERROR_Per_Iter(5)  = nnz(out5 - InfoBits);
ERROR_Per_Iter(6)  = nnz(out6 - InfoBits);
ERROR_Per_Iter(7)  = nnz(out7 - InfoBits);
ERROR_Per_Iter(8)  = nnz(out8 - InfoBits);
ERROR_Per_Iter(9)  = nnz(out9 - InfoBits);
ERROR_Per_Iter(10) = nnz(out10 - InfoBits);
ERROR_Per_Iter(11) = nnz(out11 - InfoBits);
ERROR_Per_Iter(12) = nnz(out12 - InfoBits);

end
```

7- MainTurbo

```
%================================================================
========
% This program was designed with the help of the "iterative
decoding of a
% serially concatenated convolutional code" which can be found in
the help
% menu of matlab and some tips on signal processing from mathworks
%
% The convolutional encoder reset every frame
% No puncturation ; code rate = 1/3
% the model works for a fixed number of 12 iterations
%
% note      : design based on my course project on convolutional
encoding
%
% student   : Arsene Pankeu Yomi
% University of Alberta, dept. of electrical and Computer
Engineering
%================================================================
========
```

```
clear;clc;close all;
% plotting preference
k1 = 1; k2 = 2; k3 = 3; k4 = 6; k5 = 8; k6 = 12; % 'k_' iterations
BER will be plot

% main parameters
NumBlock          = 1;
SCCC_R            = 1/3; % code rate
SCCC_len          = 65536; % # information bits
SCCC_trellis1     = poly2trellis(5, [37 21],37); % Recursive
Sytematic Convolutional Encoder
SCCC_trellis2     = SCCC_trellis1; % Recursive Sytematic
Convolutional Encoder
SCCC_seed         = 54123; % to have identical de/interleaver
SCCC_SNR_Max      = 30;
SCCC_SNR_Min      = 10;
SCCC_SNR_Step     = 2;
SCCC_SNRdB        = SCCC_SNR_Min:SCCC_SNR_Step:SCCC_SNR_Max; %
EbNo in dB
SCCC_SNR          = 10.^(SCCC_SNRdB/10);
SCCC_SNR_Lgth     = length(SCCC_SNR);

% detection parameters
% !!! FR-BLAST will work only for 64Q and 256Q !!!
N                 = 4; % # receiver antennas = # transmitter
antennas
ModSc             = 1; % 1 ->16qam    2 ->64qam   4 ->256qam
% Launch !!! ONE DETECTION AT THE TIME !!!
% '1' In order to launch detection scheme otherwise '0'
% LAUNCH ONE DETECTOR AT THE TIME
DoNoDetector      = 0;   % No detector; simple BPSK transmission
DoMmseDetect      = 0;   % MMSE detector
DoVBlastDetect    = 0;   % V-BLAST detector
DoFBlastDetect    = 0;   % F-BLAST detector; starting layer
indicated below
DoFRBlastDetect   = 0;   % FR-BLAST; starting layer indicated
below
DoRealMmseDetect  = 0;   % real-valued MMSE detector
DoRealVBlastDetect= 0;   % real-valued V-BLAST detector; starting
layer indicated below
DoRealFBlastDetect= 1;   % real-valued F-BLAST detector; starting
layer indicated below
SizeOfSphere      = 9;   % number of symbol in the search space
% starting layer 1 for the weakest 2 for second weakest ....N for
strongest
FBStartLayer      = 1;   % starting layer of the F-BLAST scheme
FRBStartLayer     = 2;   % starting layer of the FR-BLAST scheme

% initialisation,

% on the i-th row and j-th column, BER from the i-th iteration and
j-th EbNo value
ber               = zeros(12,SCCC_SNR_Lgth);
% on the i-th row simulation time for the i-th EbNo value
Time              = zeros(SCCC_SNR_Lgth,1);
```

```
for snr = 1:SCCC_SNR_Lgth % multiple EbNo run
   BlockCounter   = 0;
   BlockError     = zeros(12,NumBlock); % # in the decoded block
   % matrix of parameters !!! ORDER !!!
   MATRIX = [SCCC_R SCCC_len N ModSc DoNoDetector DoMmseDetect
DoVBlastDetect DoFBlastDetect DoFRBlastDetect DoRealMmseDetect
DoRealVBlastDetect DoRealFBlastDetect SizeOfSphere FBStartLayer
FRBStartLayer SCCC_SNR(snr)];
   tic;
   while (BlockCounter < NumBlock)
       BernouilliSeed = 12343 * (BlockCounter + 1);
       sim('turbo') % launch simulink model
       % bit-error-rate from SIMULINK
       % BER of the given SNR
       BlockCounter   = BlockCounter + 1;
       BlockError(:,BlockCounter) = BER.signals.values / SCCC_len;
   end
   for kk = 1:NumBlock
    ber(:,snr)    = ber(:,snr) + BlockError(:,kk);
   end
   ber(:,snr) = ber(:,snr) / NumBlock;
   toc;
   Time(snr)     = toc;

   % save data (.MAT file) in directory folder
   % rename it ACCORDINGLY, in order to save data from multiple
simulations
   save('TurboData_16Qtestreal','ber','SCCC_SNRdB','Time')
end

%%%%%%%%%%%%%%%%%%%%
% BER semilogy Plot %
%%%%%%%%%%%%%%%%%%%%
 figure(1)
 semilogy(SCCC_SNRdB,ber(k1,:),'+-',SCCC_SNRdB,ber(k2,:),'v-
',SCCC_SNRdB,ber(k3,:),'*-',SCCC_SNRdB,ber(k4,:),'o-
',SCCC_SNRdB,ber(k5,:),'.-',SCCC_SNRdB,ber(k6,:),'--');
 title('BER For Various Iterations of Turbo Code');
 legend('1 iteration','2 iteration','3 iteration','6 iteration','8
iteration','12 iteration'); % according to k1 k2 k3 k4 k5 k6
 xlabel('SNR, dB');
 ylabel('BER');
```

8- Detection

```
function [SoftBit] = Detection(NumFrameInABlock, N, FrameSize,
NoiseVar, SNR, SampleBlock, MIMOchResponse, DoNoDetector,ModSc,
DoMmseDetect, DoVBlastDetect, DoFRBlastDetect, DoFBlastDetect,
DoRealMmseDetect, DoRealVBlastDetect, DoRealFBlastDetect,
ConstSize, SymbMat, StatMat, SizeOfSphere, Motif, FBStartLayer,
FRBStartLayer)

SoftBit = [];
i = sqrt(complex(-1));
j = sqrt(complex(-1));

for frameCntr = 1:NumFrameInABlock
              %%%%%%%%%%
              % Channel %
              %%%%%%%%%%
              H     = MIMOchResponse(:, :, 1+(frameCntr-
1)*FrameSize);
              N_    =
sqrt(0.5)*(randn(N,FrameSize)+i*randn(N,FrameSize));
              W     = N_*sqrt(NoiseVar);
              % S is the current frame
              S     = SampleBlock(:,1+FrameSize*(frameCntr-
1):frameCntr*FrameSize);
              Y     = zeros(N, FrameSize);
              for ii = 1:FrameSize
                  % Y=H*S+W
Y(:,ii)=MIMOchResponse(:,:,ii+(frameCntr-1)*FrameSize) * S(:,ii) +
W(:,ii);
              end

              %%%%%%%%%%%%%%%%%%%%
              % Detection process %
              %%%%%%%%%%%%%%%%%%%%

              %%%% No detector used %%%%
              if (DoNoDetector == 1)
                  SoftOutputs = [];
                  for vectorCntr = 1:FrameSize
                    for kk = 1:N
                        SoftOutputs = [SoftOutputs,
llr(Y(kk),ModSc)];
                    end
                  end
              end

              %%%% MMSE detection %%%%
              if (DoMmseDetect == 1)
                  G = inv(H' * H + (1 / SNR) * eye(N)) * H';
                  SoftOutputs = [];
                  for vectorCntr = 1:FrameSize
                        [Soft] = SoftMmse(N,Y(:,vectorCntr),G);
                        SoftOut         = [];
                        for ii = 1:N
                        SoftOut  = [SoftOut, llr(Soft(ii),ModSc)];
```

```
                    end
                    SoftOutputs = [SoftOutputs, SoftOut];
                end
            end

            %%%% V-BLAST detection %%%%
            if (DoVBlastDetect == 1)
                SoftOutputs = [];
                for vectorCntr = 1:FrameSize
                    % initialize position of the undetected
symbols
                    Pos = 1:N;
                    % initialize detected symbol vector
                    YVB = zeros(N,1);
                    SVB = zeros(N,1);
                  [Soft YVBlast] =
SoftVBlast(H,ConstSize,N,Y(:,vectorCntr),SymbMat,Pos,YVB,SVB,SNR);
                  for kk = 1:N
                      SoftOutputs = [SoftOutputs,
llr(Soft(kk),ModSc)];
                  end
                end
            end

            %%%% F-BLAST detection %%%%
            if (DoFBlastDetect == 1)
                SoftOutputs = [];
                [SortedLayer IndMI] = SortCol(H,N);
                for vectorCntr = 1:FrameSize
                  [Soft] =
SoftFBlast(H,ConstSize,N,Y(:,vectorCntr),SymbMat,SortedLayer(FBSta
rtLayer),SNR);
                    SoftOut         = [];
                    for ii = 1:N
                    SoftOut  = [SoftOut, llr(Soft(ii),ModSc)];
                    end
                    SoftOutputs = [SoftOutputs, SoftOut];
                end
            end

            %%%% FR-BLAST detection %%%%
            if (DoFRBlastDetect == 1)
                SoftOutputs = [];

            G = inv(H' * H + (1 / SNR) * eye(N)) * H';
            [SortedLayer IndMI] = SortCol(H,N);
                for vectorCntr = 1:FrameSize
                  [Soft] =
SoftFRBlast(H,ConstSize,N,Y(:,vectorCntr),SymbMat,G,SortedLayer(FR
BStartLayer),SNR,SizeOfSphere,StatMat,ModSc);
                  SoftOutputs = [SoftOutputs, Soft];
                end
            end

            %%% setup for real-valued detection %%%
```

```
if ((DoRealMmseDetect + DoRealVBlastDetect + DoRealFBlastDetect) >
0)
                  RealH          = [real(H) -imag(H);imag(H)
real(H)];

                  RealY          = zeros(2*N, FrameSize);
                  RealS          = zeros(2*N, FrameSize);
                  RealW          = zeros(2*N, FrameSize);
                  for ii = 1:FrameSize
                      TempRealY  = zeros(N, FrameSize);
                      TempImagY  = zeros(N, FrameSize);
                      TempRealS  = zeros(N, FrameSize);
                      TempImagS  = zeros(N, FrameSize);
                      TempRealW  = zeros(N, FrameSize);
                      TempImagW  = zeros(N, FrameSize);
                      for hh = 1:N
                          TempRealY(hh,ii) = real(Y(hh,ii));
                          TempImagY(hh,ii) = imag(Y(hh,ii));
                          TempRealS(hh,ii) = real(S(hh,ii));
                          TempImagS(hh,ii) = imag(S(hh,ii));
                          TempRealW(hh,ii) = real(W(hh,ii));
                          TempImagW(hh,ii) = imag(W(hh,ii));
                      end



                      RealY(:,ii) =
[TempRealY(:,ii);TempImagY(:,ii)];
                      RealS(:,ii) =
[TempRealS(:,ii);TempImagS(:,ii)];
                      RealW(:,ii) =
[TempRealW(:,ii);TempImagW(:,ii)];
                  end
end

               %%%% Real-valued MMSE detection %%%%
               if (DoRealMmseDetect == 1)
     RealG = inv(RealH' * RealH + (1 / SNR) * eye(2*N)) * RealH';
                  SoftOutputs = [];
                  for vectorCntr = 1:FrameSize
                        [SoftR] =
SoftMmse(2*N,RealY(:,vectorCntr),RealG);

                       % complex transformation
                       YReal = zeros(N,1);
                       for ii = 1:N
                           YReal(ii) = SoftR(ii) + i *
SoftR(N+ii);
                       end

                       SoftOut         = [];
                       for ii = 1:N
                           SoftOut  = [SoftOut,
llr(YReal(ii),ModSc)];
                       end
```

```
                    SoftOutputs = [SoftOutputs, SoftOut];
                end

            end
            %%%% Real-valued VBlast detection %%%%
            if (DoRealVBlastDetect == 1)
                RealConstSize = (ConstSize)^.5;
                SoftOutputs = [];
                for vectorCntr = 1:FrameSize
                    % initialize position of the undetected
symbols
                    Pos = 1:2*N;
                    % initialize detected symbol vector
                    YVB = zeros(2*N,1);
                    SVB = zeros(2*N,1);
                  [Soft, YVBlast] =
SoftVBlast(RealH,RealConstSize,2*N,RealY(:,vectorCntr),Motif,Pos,Y
VB,SVB,SNR);

                    % complex transformation
                    YReal = zeros(N,1);
                    for ii = 1:N
                          YReal(ii) = Soft(ii) + i * Soft(N+ii);
                    end
                    for kk = 1:N
                       SoftOutputs = [SoftOutputs,
llr(YReal(kk),ModSc)];
                    end
                end
            end

            %%%% Real-valued FBlast detection %%%% %%%%
            if (DoRealFBlastDetect == 1)
                SoftOutputs                 = [];
                [RealSortedLayer RealIndMI] =
SortCol(RealH,2*N);
                RealConstSize               = (ConstSize)^.5;
                for vectorCntr = 1:FrameSize
                  [Soft] =
SoftFBlast(RealH,RealConstSize,2*N,RealY(:,vectorCntr),Motif,RealS
ortedLayer(FBStartLayer),SNR);
                    SoftOut         = [];

                    % complex transformation
                    YReal = zeros(N,1);
                    for ii = 1:N
                          YReal(ii) = Soft(ii) + i * Soft(N+ii);
                    end

                    SoftOut         = [];
                    for ii = 1:N
                      SoftOut  = [SoftOut,
llr(YReal(ii),ModSc)];
                    end
                    SoftOutputs = [SoftOutputs, SoftOut];
```

```
            end
        end

        SoftBit = [SoftBit, SoftOutputs];

end % end frameCntr
```

9- FixedSearchSet

```
function [Subset] = FixedSearchSet(A,b,c,d)
% This function returns searchset for the noisiest layer, 64QAM
% Usage:
%       [Subset] = FixedSearchSet(a,b,c,d)
% Input:
%            A = Matrix of statistic
%            b = Mmse noisiest estimate
%            c = size of the fixed search set
%            d = ConstSize
%
%
% Output:
%    Subset          : set of candidate symbol

    [symb indx]             = min(abs(b * ones(1, d) - A(1,:)));
    A(indx+1,indx)          = 0;
    Subset                  = [b];
    for kk = 1:c-1
        [value position]    = max(A(indx+1,:));
        A(indx+1,position)  = 0;
        Subset              = [Subset,A(1,position)];
    end
end
```

10- llr

```
function [LLR] = llr(Pt,d)
% This function returns LLR
% Usage:
%       [LLR] = llr(Pt,d)
% Input:
%
% Pt = Point in the constellation / unsliced symbol
% d  = Modulation scheme : 1 ->16qam    2 ->64qam    4 ->256qam
%
% Output:
%
% LLR: Log-Likelihood-Ratio
%      [LLR(b0) LLR(b1) LLR(b2) ... LLR(bn)] for n-bit
constellation signal

        if     (d == 1)
               LLR    = zeros(1,4);
               LLR(1) = - abs(imag(Pt)) + 2;
               LLR(2) = - imag(Pt);
               LLR(3) = - abs(real(Pt)) + 2;
```

```
            LLR(4) = real(Pt);

        elseif (d == 2)
            LLR    = zeros(1,6);
            LLR(1) = - abs(abs(imag(Pt)) - 4) + 2;
            LLR(2) = - abs(imag(Pt)) + 4;
            LLR(3) = - imag(Pt);
            LLR(4) = - abs(abs(real(Pt)) - 4) + 2;
            LLR(5) = - abs(real(Pt)) + 4;
            LLR(6) = real(Pt);

        elseif (d == 4)
            LLR    = zeros(1,8);
            if (abs(imag(Pt)) < 8)
                LLR(1) = - abs(abs(imag(Pt)) - 4) + 2;
            else
                LLR(1) = - abs(abs(imag(Pt)) - 12) + 2;
            end
            LLR(2) = - abs(abs(imag(Pt)) - 8) + 4;
            LLR(3) = - abs(imag(Pt)) + 8;
            LLR(4) = - imag(Pt);
            if (abs(real(Pt)) < 8)
                LLR(5) = - abs(abs(real(Pt)) - 4) + 2;
            else
                LLR(5) = - abs(abs(real(Pt)) - 12) + 2;
            end
            LLR(6) = - abs(abs(real(Pt)) - 8) + 4;
            LLR(7) = - abs(real(Pt)) + 8;
            LLR(8) = real(Pt);

        end
end
```

11- Modulation

```
function [SymbMat SignalVar ConstSize NumbBitPerPt StatMat Motif]
= Modulation(d)
% This function returns uncorellated Modulated symbols.
% Usage:
%        [SymbMat SignalVar ConstSize NumbBitPerPt StatMat Motif]
= Modulation(d)
% Input:
%
%             d = Modulation scheme : 1 ->16qam    2 ->64qam   4 -
>256qam
% Output:
%
% SymbMat                : Constellation matrice
% NumbBitPerPt           : Number of bits per point
% StatMat                : Subset for the restricted search
% SignalVar              : Variance of signal from the given
constellation
% ConstSize              : number of point in the constellation


         if     (d==1)
                M              = 16;
                NumbBitPerPt   = 4 ;
                Motif          = [-3 -1 1 3]';
                load SymetricalSearchSpace16Q.mat;

         elseif (d==2)
                M              = 64;
                NumbBitPerPt   = 6 ;
                Motif          = [-7 -5 -3 -1 1 3 5 7]';
                load SymetricalSearchSpace64Q.mat;

         elseif (d==4)
                M              = 256;
                NumbBitPerPt   = 8 ;
                Motif          = [-15 -13 -11 -9 -7 -5 -3 -1 1 3 5
7 9 11 13 15]';
                load SymetricalSearchSpace256Q.mat;

         end

         x          = [0:M-1];
         %matlab function for gray mapping with QAM constellation
         SymbMat    =
modulate(modem.qammod('M',M,'SymbolOrder','Gray'),x);
         SymbMat    = SymbMat.';

         ConstSize  = M;
         SignalVar  = SymbMat' * SymbMat / M;
end
```

12- SampleNChannel

```
function [SampleBlock MIMOchResponse] =
SampleNChannel(NSymbol,NumbBitPerPt,EncBits,SymbMat,N,TotNumOfSamp
leInABlock,NumFrameInABlock,FrameSize)

     % 1- generate symbols from encoded bits
     SymbolBlock = zeros(1,NSymbol);
     for pp = 1:NSymbol
         tt = 0;
         for zz = 1:NumbBitPerPt
             tt = tt + 2^(zz - 1) * EncBits(zz + (pp - 1) *
NumbBitPerPt);
         end
         SymbolBlock(pp) = SymbMat(tt + 1);
     end

     % 2- generate sample from symbols
     SampleBlock = zeros(N,NSymbol / N);
     for NSymb = 1:N:NSymbol - N + 1
         SampleBlock(:,(NSymb + (N - 1)) / N) =
transpose(SymbolBlock(NSymb:NSymb + N - 1));
     end

     % channel generation
     MIMOchResponse    = zeros(N,N,TotNumOfSampleInABlock);
     for ii = 1:NumFrameInABlock
         % generating CHANNEL
         H  = (randn(N) + i*randn(N)) * sqrt(0.5);
          for ii1 = 1:N
             for ii2 = 1:N
                 MIMOchResponse(ii1, ii2, (1+(ii-
1)*FrameSize:ii*FrameSize)) = H(ii1, ii2); % one channel response
per frame
             end
          end
     end
```

13- SortCol

```
function [SortedLayer IndMI] = SortCol(A,c)
% This function returns index of layers from the weakest to the
strongest
% Usage:
%       function [SortedLayer IndMI] = Sortcol(A)
% Input:
%            A = H matrix
%            c = number of receiver antennas (same as transmitter)
% Output:
%  SortedLayer = Matrix of the index
%                1-weakest layer 2-second weakest layer ...
%                ...(c-1)-second strongest layer  c-strongest
layer
%        IndMI = Index of the layer which has maximum interference
on the
%                Strongest Layer

     % Initialization
     SortedLayer             = zeros(1,c);
     n                       = 1;

     AmpColH                 = [];
     for ii                  = 1:c
        AmpColH              = [AmpColH,norm(A(:,ii))];
     end
     while ( n < c + 1 )
        [Val Ind]            = min(AmpColH);
        SortedLayer(n)       = Ind;
        AmpColH(Ind)         = 10000;
        n                    = n + 1;
     end

     AmpH                    = [];
     for ii2                 = 1:c
         AmpH                = [AmpH, norm(SortedLayer(c),ii2)];
     end
     AmpH(SortedLayer(c))    = 0;
     [val1 IndMI]            = max(AmpH);

end
```

14- Mmse

```
function [YMmse] = Mmse(b,c,E,F,G)
% This function returns symbol vector from the MMSE detection
process
% Usage:
%        [YMmse] = Mmse(b,c,E,F,G)
% Input:
%             b = constellation size
%             c = number of receiver antennas (same as
transmitter)
%             E = noisy transmitted vector Y
%             F = matrice of symbol from the constellation
%             G = MMSE matrix
% Output:
%         YMmse : output of the Mmse detection

                o  = ones(1, b);
                op = ones(c, 1);
                % MMSE output vector
                TempVec         = G * E;
                % slicing
                [temp IndMmse]  = min(abs(TempVec * o - op *
F.')');
                YMmse           = F(IndMmse);
end
```

15- SoftVB

```
function [SoftOut YVBlast] = SoftVBlast(A,b,c,E,F,K,L,L_,n)
% This function returns soft output from the V-BLAST detection
process
% Usage:
%        [SoftOut YVBlast] = SoftVBlast(A,b,c,E,F,K,L,L_,n)
% Input:
%             A = H matrix
%             b = constellation size
%             c = number of receiver antennas (same as
transmitter)
%             E = noisy transmitted vector Y
%             F = matrix of symbol from the constellation
%             K = indice(s) of the symbol(s) to be detected
%             L = YVBlast, initial value
%            L_ = soft info initial value
%             n = SNR
% Output:
%       SoftOut : unsliced symbol vector from the VBLAST detection
%       YVBlast : hard output of the VBLAST detection

        SoftOut   = L_;
        YVBlast   = L;
        H_        = A;
        Y_        = E;
        for IndSymb = 1:c
            % MMSE matrix
```

```
            G_ = inv(H_' * H_ + (1 / n) * eye(c - (IndSymb - 1)))
* H_';

            % ordering: detection goes from strongest to weakest
layer
            % detection starts with the row of G_ which has
minimum norm

            % norm of G_'s row
            AmpG_               = [];
            for ii              = 1:c-(IndSymb-1)
                AmpG_           = [AmpG_, norm(G_(ii, :))^2];
            end
            [val1 GRowInd]      = min(AmpG_);

            % nulling: MMSE is used to detect symbol from each
layer
            CurDtdSymb          = G_(GRowInd, :) * Y_;

            % slicing
            Dist                = abs(F - ones(b,1) *
CurDtdSymb).^2;
            [val2, IndVBlast]   = min(Dist);
            YVBlast(K(GRowInd)) = F(IndVBlast);

            % soft decision
            SoftOut(K(GRowInd)) = CurDtdSymb;

            % cancellation : produces deflation in columns of H_
            %                removes interference from detected
symbol
            Y_     = Y_ - H_(:, GRowInd) * YVBlast(K(GRowInd));
            HTemp  = [];
            Temp   = [];
            for ii = 1:c-(IndSymb-1),
                if (ii  ~= GRowInd)
                   HTemp = [HTemp, H_(:, ii)];
                   Temp  = [Temp, K(ii)];
                end
            end
            H_     = HTemp;
            K      = Temp;
        end
end
```

16- SoftMmse

```
function [Out] = SoftMmse(c,E,G)
% This function returns soft output from the MMSE detection
process
% Usage:
%        [Out] = SoftMmse(c,E,G)
% Input:
%             c = number of receiver antennas (same as
transmitter)
%             E = noisy transmitted vector Y
%             G = MMSE matrix
% Output:
%         Out : soft output of the Mmse detection


                % MMSE output vector
                Out             = G * E;
end
```

17- SoftFBlast

```
function [SoftOut] = SoftFBlast(A,b,c,E,F,i,n)
% This function returns soft output from the F-BLAST detection
process
% Usage:
%        [SoftOut] = SoftFBlast(A,b,c,E,F,i,n)
% Input:
%             A   = H matrix
%             b   = constellation size
%             c   = number of receiver antennas (same as
transmitter)
%             E   = noisy transmitted vector Y
%             F   = matrix of symbol from the constellation
%             i   = Index of the first layer to be detected
%                   1-weakest layer 2-second weakest layer ...
%                   ...(c-1)-second strongest layer  c-strongest
layer
%             n   = SNR
% Output:
%       SoftOut : soft output of the FBLAST detection

     SoftTest    = zeros(c,b);
     YFBlastTest = zeros(c,b);
     Epsilon     = zeros(c,b);
     % an exhaustive search is performed on the first layer to be
detected
     % i.e detection starts with the 'i'_th layer
     % conventional VBLAST for the N-1 remaining symbols

     % Exhaustive search on the first layer
     for ii = 1:b
         H_                                  = A;
         UnDtdSymb                           = 1:c;
         Y_                                  = E;
         YFBlastTest(UnDtdSymb(i),ii)        = F(ii);
         SoftTest(UnDtdSymb(i),ii)           = F(ii);
```

```
        % cancellation : produces deflation in columns of H_
        %                removes interference from temptative
symbol
        Y_              = Y_ - H_(:,i) *
YFBlastTest(UnDtdSymb(i),ii);
        HTemp1          = [];
        Temp1           = [];
        for ii1         = 1:c
            if (ii1    ~= i)
               HTemp1   = [HTemp1,H_(:,ii1)];
               Temp1    = [Temp1, UnDtdSymb(ii1)];
            end
        end
        H_              = HTemp1;
        UnDtdSymb       = Temp1;

        % Original V-BLAST detection over the N-1 remaining
layers
[Soft YVBlast]   = SoftVBlast(H_,b,c-
1,Y_,F,UnDtdSymb,YFBlastTest(:,ii),SoftTest(:,ii),n);
        YFBlastTest(:,ii) = YVBlast;
        SoftTest(:,ii)    = Soft;
        % Compute noise for all candidate
        Epsilon(:,ii)     = A * YFBlastTest(:,ii) - E;
    end
    % Find the best candidate
    AmpEps           =[];
    for ii1          = 1:b
        AmpEps       = [AmpEps, norm(Epsilon(:,ii1))];
    end
    [val4 MinEpsInd] = min(AmpEps);

       SoftOut = SoftTest(:,MinEpsInd);

end
```

18- SoftFRBlast

```
function [SoftOut] = SoftFRBlast(A,b,c,E,F,G,i,n,k,S,d)
% This function returns symbol vector from the FR-BLAST detection
process
% Usage:
%        [SoftOut] = SoftFRBlast(A,b,c,E,F,G,n,k,S,d)
% Input:
%             A   = H matrix
%             b   = constellation size
%             c   = number of receiver antennas (same as
transmitter)
%             d   = mode scheme
%             E   = noisy transmitted vector Y
%             F   = matrix of symbol from the constellation
%             G   = MMSE matrix
%             i   = Index of the first layer to be detected
%                   1-weakest layer 2-second weakest layer ...
```

```
%                    ...(c-1)-second strongest layer  c-strongest
layer
%             n   = SNR
%             k   = size of the fixed search space
%             S   = Matrix of statistic to build the subset
% Output:
%      SoftOut : output of the FRBLAST detection

     SoftOut       = [];
     SoftTest      = zeros(c,k);
     YFRBlastTest  = zeros(c,k);
     Epsilon       = zeros(c,k);
     % a search inside a search space is performed on the first
layer to be
     % detected, i.e. detection starts with the 'i'-th layer
     % conventional VBLAST for the N-1 remaining layers
     % norm of G's row

     % original MMSE is run to estimate the position of the first
symbol
     [YMmse]       = Mmse(b,c,E,F,G);

     % now we build a set of k closest symbol including this one
     [Subset]      = FixedSearchSet(S,YMmse(i),k,b);

     % search inside the subset for the first symbol,
     for ii = 1:k
         H_                                      = A;
         UnDtdSymb                               = 1:c;
         Y_                                      = E;
         YFRBlastTest(UnDtdSymb(i),ii)           = Subset(ii);
         SoftTest(UnDtdSymb(i),ii)               = Subset(ii);
         % cancellation : produces deflation in columns of H_
         %                removes interference from temptative
symbol
         Y_        = Y_ - H_(:,i) * YFRBlastTest(UnDtdSymb(i),ii);
         HTemp1    = [];
         Temp1     = [];
         for ii1   = 1:c
             if (ii1 ~= i)
                HTemp1   = [HTemp1,H_(:,ii1)];
                Temp1    = [Temp1, UnDtdSymb(ii1)];
             end
         end
         H_        = HTemp1;
         UnDtdSymb = Temp1;

         % Original V-BLAST detection over the N-1 remaining
layers
[Soft YVBlast] = SoftVBlast(H_,b,c-
1,Y_,F,UnDtdSymb,YFRBlastTest(:,ii),SoftTest(:,ii),n);
         YFRBlastTest(:,ii) = YVBlast;
         SoftTest(:,ii)     = Soft;
         % Compute noise for all candidate
         Epsilon(:,ii) = A * YFRBlastTest(:,ii) - E;
```

```
    end

    % Find the best candidate
    AmpEps  = [];
    for ii1 = 1:k
        AmpEps = [AmpEps, norm(Epsilon(:,ii1))];
    end
    [val4 MinEpsInd] = min(AmpEps);
    for kk = 1:c
       SoftOut = [SoftOut, llr(SoftTest(kk,MinEpsInd),d)];
    end
end
```