中山大學
SUN YAT-SEN UNIVERSITY

数值计算
期中大作业

专业：通信工程

学号：22309080

姓名：梁倍铭

时间：2023.12.21

目录

1 MIMO 系统简介

MIMO 全称是 multiple-in multiple-out，多输入多输出。示意图如图 1, 具有 N_T 根发射天线和 N_R 根接收天线，且 $N_R > N_T$，数据在 N_T 个等长的数据子流（称为层）中进行解复用。这些子流被映射成 M-PSK 或 M-QAM 符号, 或者，可以使用前向纠错（FEC）码在映射之前对数据子流进行编码。

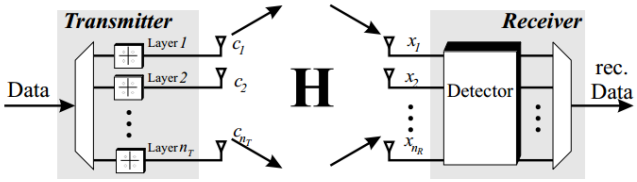

图 1: MIMO 系统示意图

用 $c = \begin{pmatrix} c_1 & c_2 & \dots & c_{nt} \end{pmatrix}^\top$ 来表示发射的信号，用 $x = \begin{pmatrix} x_1 & x_2 & \dots & x_{nr} \end{pmatrix}$ 来表示接收到的信号。信道 H 的大小为 $N_T * N_R$，信道 H 可表示为

$$H = \begin{pmatrix} h_{1,1} & \dots & h_{1,nT} \\ \dots & \dots & \dots \\ h_{nR,1} & \dots & h_{nR,nT} \end{pmatrix}$$

由于两篇论文所使用的符号不尽相同，这里采用第一篇论文的符号进行统一描述，用 v 来表示接收天线中的高斯白噪声，$v = \begin{pmatrix} v_1 & v_2 & \dots & v_{NR} \end{pmatrix}^\top$，因此，整个系统可以用 $x = Hc + v$ 来描述。

其中我们假设假设所有天线每维的方差 N0=2 的不相关高斯白噪声。传输的符号被归一化，使得每比特的平均接收能量为一。我们假设静态平坦衰落环境，即信道矩阵 H 在帧内保持不变，并且在帧与帧之间独立变化。假定不同的衰落增益是不相关的并且接收机完全了解这些增益。

2 《Efficient Algorithm for Detecting Layered Space-Time Codes》中算法的复现

2.1 简介

分层空时码的设计是为了在瑞利衰落环境中利用多天线系统的容量优势。在《Efficient Algorithm for Detecting Layered Space-Time Codes》一文中，提出了一种基于排序 QR 分解的新高效检测算法。与需要多次计算信道矩阵伪

逆的标准检测算法相比，它只需要一小部分计算量。导出的算法不限于分层时空架构，而是通常可用于矢量信道系统中的检测。

本部分将用 matlab 软件对 V-BLAST、QR 分解及 QR 分解的排序算法进行仿真复现，并对其进行性能分析。

2.2 各算法的分析及仿真代码

2.2.1 V-BLAST 算法分析及仿真

从上文的描述可以看出，接收信号是 nT 个发射信号的线性组合。在接收器处恢复 N_T 信号的最佳方法是最大似然检测，但由于其巨大的复杂性，这是不可行的。

Foschini 等人提出了一种连续干扰消除技术，该技术通过使用迫零（ZF）归零向量对接收信号向量进行线性加权来消除干扰源。在每个检测步骤中，除了一个信号之外的所有信号都被视为干扰源。通过将调零矢量应用于干扰消除，这些信号的影响被消除，目标信号被检测到并随后从接收到的信号矢量中减去（干扰消除）。算法伪代码如图 2,matlab 代码如图 3

(1) for $i = 1, \ldots, n_T$
(2) $\quad \mathbf{G}_i = \mathbf{H}_i^+$
(3) $\quad k_i = \arg\min_j \left\| \mathbf{g}_i^{(j)} \right\|^2$
(4) $\quad \mathbf{w}_{k_i}^T = \mathbf{g}_i^{(k_i)}$
(5) $\quad y_{k_i} = \mathbf{w}_{k_i}^T \cdot \mathbf{x}_i$
(6) $\quad \hat{c}_{k_i} = \mathcal{Q}[y_{k_i}]$
(7) $\quad \mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{h}_{k_i} \cdot \hat{c}_{k_i}$
(8) $\quad \mathbf{H}_{i+1} = \mathbf{H}_i^{\overline{k_i}}$
(9) end

图 2: V-blast 伪代码

```
function c = vblast_fun(H,x)
%VBLAST_FUN 此函数是对v-blast算法的仿真实现
%    输入参数H：信道矩阵
%    输入参数x：接收到的信号
%    输出参数c：解调出来的信号矩阵
[~,col]=size(H);%获取H矩阵的大小
c_hat=zeros(col,1);
index=1:col;%用于记录解调顺序的索引
c=zeros(col,1);
for i=1:col%进行迭代
    [~,col]=size(H);
    G=pinv(H);%求出矩阵H的伪逆
    k=1;
    v=norm(G(1,:));
    for j=1:col%找出伪逆G中二范数最小的行
        temp=norm(G(j,:));
        if temp<v
            v=temp;
            k=j;
        end
    end
    c_hat(index(k),:)=G(k,:)*x;%得到近似解
        if c_hat(index(k),:)>0.5%对近似解进行判决
            c(index(k),:)=1;
        else
            c(index(k),:)=0;
        end
    x=x-H(:,k)*c(index(k),:);%将改成判决得到的信号从x中减去
    index(k)=[];%更新索引
    H(:,k)=[];%删去H对应的列
end
```

图 3: V-blast matlab 代码

2.2.2 Unsorted QRD 算法分析及仿真

由于 V-BLAST 算法的复杂度较高，我们可以通过变换信道矩阵 H 避免矩阵的求逆，获得较低的计算复杂度。根据论文，该算法首先对信道矩阵 H 进行了 QR 分解，即

$$H = Q \cdot R$$

其中 Q 为酉矩阵，R 为上三角阵。将接收到的信号左乘 Q^H 得到 $n_T \times 1$ 修正的接收信号向量

$$y = Q^H \cdot x = R \cdot c + \eta$$

由于 Q 是正交归一化的，所以 η 的统计特性保持不变，y 的第 k 个元素为

$$y_k = r_{k,k} \cdot c_k + \eta + d_k$$

其中 d_k 为干扰项

$$d_k = \sum_{i=k+1}^{n_T} r_{k,i} \cdot c_i$$

从第 n_T 层开始检测，即可逐层获取发射信号。伪代码如图 4，matlab 代码如图 5。

$R = 0, Q = H, S = (1, \cdots, n_T)$
for $i = 1, \cdots, n_T$ **do**
  $r_{i,i} = \|q_i\|$
  $q_i = q_i / r_{i,i}$
  for $l = i + 1, \cdots, n_T$ **do**
    $r_{i,l} = q_i^H \cdot q_l$
    $q_l = q_l - r_{i,l} \cdot q_i$

图 4: Unsorted QRD 伪代码

```
function c = qr_fun(H,x)
%QR_FUN 此函数是对未排序QR算法的仿真实现
%    输入参数H：信道矩阵
%    输入参数x：接收到的信号
%    输出参数c：解调出来的信号矩阵
[~,col]=size(H);%求信道矩阵H的列数
c=zeros(col,1);
[Q,R]=qr(H);%对信道矩阵进行QR分解
y=Q'*x;%对接收到的信号进行处理
d=zeros(col,1);
z=zeros(col,1);
for i=col:-1:1%从最后一层开始检测
    for j=i+1:col%通过迭代获得每一层的干扰项
        d(i,1) = d(i,1) + R(i, j) * c(j,1);
    end
    z(i,1) = y(i,1) - d(i,1);%减去干扰项
    tmp = z(i,1) / R(i, i);%获得原始解调出来的信号
        if tmp < 0.5%对信号进行判决
            c(i,1) = 0;
        else
            c(i,1) = 1;
        end
end
```

图 5: Unsorted QRD matlab 代码

2.2.3 Sorted QRD 算法分析及仿真

由于 QR 算法不能决定最先检验层，而最先检验层会影响后续检验的精确性。而 Sorted QR 则是非常有效的新方法，其错误性能接近 V-BLAST。它基本上是改进的 Gram-Schmidt 算法的扩展，即在每个正交化步骤中对 H 的列进行排序。

根据论文，上三角阵 R 的对角元素 $|r_{k,k}|$ 与信号的估计误差成反比，所以在每次检测中先选取最大的 $r_{k,k}|$ 来检测信号，来做到误差最小化。伪代码如图 6，matlab 代码如图 7。

SQRD Algorithm

(1) $\mathbf{R} = \mathbf{0}$, $\mathbf{Q} = \mathbf{H}$, $\mathcal{S} = (1, \ldots, n_T)$
(2) for $i = 1, \ldots, n_T$
(3) $\quad k_i = \arg \min_{l=i,\ldots,n_T} \|\mathbf{q}_l\|^2$
(4) $\quad$ exchange col. i and k_i in $\mathbf{Q}, \mathbf{R}, \mathcal{S}$
(5) $\quad r_{i,i} = \|\mathbf{q}_i\|$
(6) $\quad \mathbf{q}_i = \mathbf{q}_i / r_{i,i}$
(7) $\quad$ for $l = i + 1, \ldots, n_T$
(8) $\quad\quad r_{i,l} = \mathbf{q}_i^H \cdot \mathbf{q}_l$
(9) $\quad\quad \mathbf{q}_l = \mathbf{q}_l - r_{i,l} \cdot \mathbf{q}_i$
(10) $\quad$ end
(11) end

Signal Detection

(12) $\mathbf{y} = \mathbf{Q}^H \cdot \mathbf{x}$
(13) for $k = n_T, \ldots, 1$
(14) $\quad \hat{d}_k = \sum_{i=k+1}^{n_T} r_{k,i} \cdot \hat{c}_i$
(15) $\quad z_k = y_k - \hat{d}_k$
(16) $\quad \hat{c}_k = \mathcal{Q}[z_k / r_{k,k}]$
(17) end
(18) Permutate $\hat{c}$ according to $\mathcal{S}$

图 6: Sorted QRD 伪代码

```matlab
function C = sqrd_fun(H, x)
%SQRD_FUN 此函数是对未排序QR算法的仿真实现
%   输入参数H：信道矩阵
%   输入参数x：接收到的信号
%   输出参数C：解调出来的信号矩阵
[row, col] = size(H);%计算信道矩阵H的大小
c = zeros(col, 1);
C = zeros(col, 1);
q = H;
r = zeros(col, col);
s = 1:col; % 用于记录排序的索引

% 进行QR 分解
for i2 = 1:col
    min_norm = norm(q(:, i2));  % 初始化最小范数
    k = i2;
    % 寻找最小范数列
    for j2 = i2:col
        if min_norm > norm(q(:, j2))
            min_norm = norm(q(:, j2));
            k = j2;
        end
    end
    % 交换列，更新 QR 分解矩阵
    q(:, [i2, k]) = q(:, [k, i2]);
    r(:, [i2, k]) = r(:, [k, i2]);
    s([i2, k]) = s([k, i2]);

    r(i2, i2) = norm(q(:, i2));
    q(:, i2) = q(:, i2) / r(i2, i2);

    % 更新 R 矩阵的剩余部分
    for l = i2+1:col
        r(i2, l) = q(:, i2)' * q(:, l);
        q(:, l) = q(:, l) - r(i2, l) * q(:, i2);
    end
end
y = q' * x;  % 计算中间变量 y
d = zeros(row, 1);
z = zeros(row, 1);

% 反向解调
for i = col:-1:1
    for j = i+1:col
        d(i) = d(i) + r(i, j) * c(j);  % 计算干扰项
    end

    z(i) = y(i) - d(i);  % 消除干扰项
    k = z(i) / r(i, i);%得到解调出来的原始信号
    if k < 0.5%对原始信号进行判决
        c(i) = 0;
    else
        c(i) = 1;
    end
end
% 根据排序结果将解调后的结果排序并返回
for i4 = 1:length(s)
    C(s(i4)) = c(i4);
end
end
```

图 7: Sorted QRD matlab 代码

2.3 仿真测试

2.3.1 误码率测试

测试思路 通过假设在满足 $N_T > N_R$ 条件的天线数，通过设定不同的信噪比，计算不同算法的误码率，来对各个算法进行分析，进行误码率计算的 matlab 代码如图 8。由于计算量巨大，使用了 matlab 的并行计算工具包 Parallel Computing Toolbox 减少代码的运行时间。

```
function ber = ber_test_fun(Eb_No_dB,N_R,N_T,N_frame,func)
%VBLAST_TEST_FUN 该函数用来测试算法的误码率
%   输入参数Eb_No_dB：信噪比
%   输入参数N_R：接收天线数
%   输入参数N_R：发射天线数
%   输入参数N_frame：测试信号的帧数
%   输入参数func：函数句柄，使用解调算法的函数
%   输出参数ber:计算得到的误码率
total_bit=N_T*N_frame;%总的比特数
error_bit=0;%初始化错误的比特数
for i=1:N_frame
    bit_stream_tx=randi([0,1],N_T,1);%发送的比特流
    c=bit_stream_tx;
    H=randn(N_R,N_T);
    Eb_No=10^(Eb_No_dB/10);    % 转换为绝对值
    bits_per_symbol=1;  % 假设采用二元调制映射，  每个符号含有1bit信息
    SNR=Eb_No*bits_per_symbol;        % 根据符号所含比特数 增大信噪比
    sigma=1/SNR;     % 假设发送符号能量平均为1，SNR定义为  符号能量/噪声能量，  可求出噪声能量（方差）
    v=sqrt(sigma)*randn(N_R,1);     % 生成满足当前信噪比条件的 高斯随机噪声样本
    x=H*c+v;%接收到的比特流
    bit_stream_rx=func(H,x);%进行信号解调
    error_bit_vector=bit_stream_rx-bit_stream_tx;%累加错误的比特数
    error_bit=error_bit+length(find(error_bit_vector));
end
ber=error_bit/total_bit;%计算误码率
end
```

图 8: 误码率测试代码

测试结果 对不同情况的误码率仿真测试结果如下：

1. $n_T = 8, n_R = 12$, 信噪比为 0-20dB, 测试帧数为 1000000，结果如图 9
2. $n_T = 4, n_R = 6$, 信噪比为 0-20dB, 测试帧数为 1000000，结果如图 10

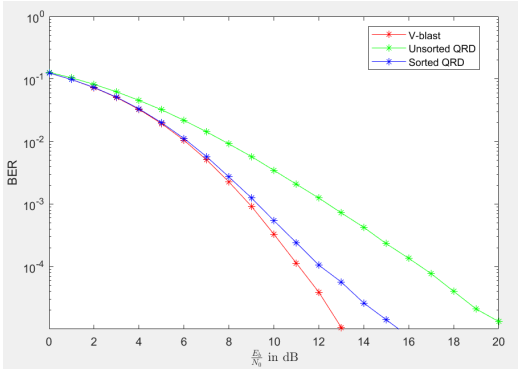

图 9: $n_T = 8, n_R = 12$, 信噪比为 0-20dB, 测试帧数为 1000000 的误码率测试结果图

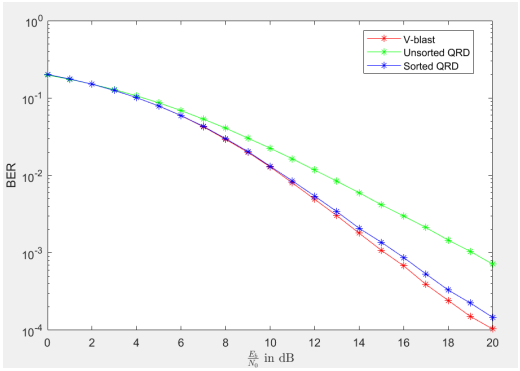

图 10: $n_T = 4, n_R = 6$, 信噪比为 0-20dB, 测试帧数为 1000000 的误码率测试结果图

2.3.2 误帧率测试

测试思路 通过假设在满足 $N_T > N_R$ 条件的天线数，通过设定不同的信噪比，假定一定的帧数，采用类似误码率的统计方法，来对各个算法进行分析，进行误码率计算的 matlab 代码如图 11。由于计算量巨大，使用了 matlab 的并行计算工具包 Parallel Computing Toolbox 减少代码运行时间。

```
function ber = fer_test_fun(Eb_No_dB,N_R,N_T,N_frame,func)
%VBLAST_TEST_FUN 该函数用来测试算法的误码率
%   输入参数Eb_No_dB：信噪比
%   输入参数N_R：接收天线数
%   输入参数N_R：发射天线数
%   输入参数N_frame：测试信号的帧数
%   输入参数func：函数句柄，使用解调算法的函数
%   输出参数fer:计算得到的误帧率
total_frame=N_frame;%总的比特数
error_frame=0;%初始化错误的比特数
for i=1:N_frame
    bit_stream_tx=randi([0,1],N_T,1);%发送的比特流
    c=bit_stream_tx;
    H=randn(N_R,N_T);
    Eb_No=10^(Eb_No_dB/10);    % 转换为绝对值
    bits_per_symbol=1;  % 假设采用二元调制映射，  每个符号含有1bit信息
    SNR=Eb_No*bits_per_symbol;       % 根据符号所含比特数 增大信噪比
    sigma=1/SNR;     % 假设发送符号能量平均为1，SNR定义为  符号能量/噪声能量，  可求出噪声能量（方差）
    v=sqrt(sigma)*randn(N_R,1);     % 生成满足当前信噪比条件的 高斯随机噪声样本
    x=H*c+v;%接收到的比特流
    bit_stream_rx=func(H,x);%进行信号解调
if any(bit_stream_rx-bit_stream_tx)%累加错误的帧数
    error_frame=error_frame+1;
end
end
ber=error_frame/total_frame;%计算误码率
end
```

图 11: 误帧率测试代码

测试结果 对不同情况的误帧率仿真测试结果如下：

1. $n_T = 8, n_R = 12$, 信噪比为 0-20dB, 测试帧数为 1000000，如图 12
2. $n_T = 4, n_R = 6$, 信噪比为 0-20dB, 测试帧数为 1000000，如图 13

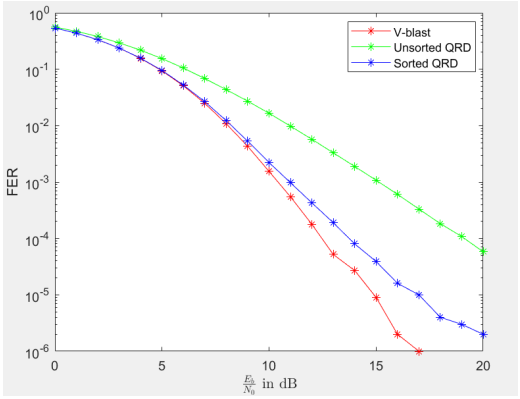

图 12: $n_T = 8, n_R = 12$, 信噪比为 0-20dB, 测试帧数为 1000000 的误帧率测试结果图

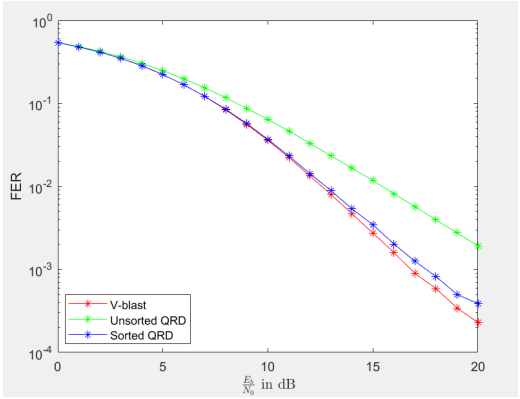

图 13: $n_T = 4, n_R = 6$, 信噪比为 0-20dB, 测试帧数为 1000000 的误帧率测试结果图

2.3.3 结果分析

从仿真结果可以看出，在相同的信噪比下，发射天线和接收天线的数目增加时可以降低误码率，因此我们可以在 MIMO 系统中增加发射天线和接收天线的数量，来有效降低误码率。但是在实际应用中，收发天线的数目往往是受到限制的，特别是在移动终端中，而且随着天线数目的增加，计算量随之增多，会加大系统的功耗，因此有必要探寻高效的算法来降低误码率。

根据仿真结果图，三种算法中 V-BLAST 算法的误码率和误帧率在不同的信噪比下一直保持在较小水平，QR 分解算法的误码率误帧率显著高于其他两种算法，说明 QR 分解的检测性能略差，但对 QR 分解进行排序后，误码率和误帧率有了一定降低，性能得到显著提升，结合算法的复杂度，V-BLAST 算法的复杂度最高，换来的是精确的结果，而 QR 算法精度的损失，获得了最快的运行速度, 而 SQRD 在保持较高的精度情况下，运算复杂度只比 QR 算法有微弱的增加，综上，SQRD 算法的综合性能最好。

3 《MMSE Extension of V-BLAST based on Sorted QR Decomposition》中算法的复现

3.1 简介

V-BLAST 架构利用了多个天线系统的容量优势，使用垂直分层编码结构，其中独立的代码块（称为层）与特定的发射天线相关联。在接收器处，这些层通过连续干扰消除技术进行检测，该技术通过使用迫零归零向量 (ZF-BLAST) 对接收信号向量进行线性加权来消除干扰源。这种连续检测需要多次计算伪逆，这是一项计算成本高昂的任务。

在《MMSE Extension of V-BLAST based on Sorted QR Decomposition》一文中，作者提出了基于 QR 分解的 MMSE 算法，能够有效降低算法复杂度并明显提高算法复杂度，本部分将用 matlab 软件实现对 ZF、MMSE、MMSE-QRD 和 MMSE-SQRD 的仿真，并进行性能分析。

3.2 各算法分析及仿真代码

3.2.1 Zero-Forcing Detector (ZF) 算法分析及仿真

对于线性信号，ZF 算法是指先对对信道矩阵求伪逆得到 G，如何将接收到的信号 x 与 G 相乘，即

$$G = H^{+} = (H^{H}H)^{-1}H^{H}$$

$$\hat{c} = Gx$$

然后对所有层进行并行决策。当 $H^{H}H$ 的特征值较小时，会因噪声放大而导致较大的误差，这种效应在具有相同数量的发射和接收天线的系统中尤其明显。matlab 代码如图 14。

```
function c=zf_fun(H,x)
%ZF_FUN 此函数是对ZF的仿真实现
%    输入参数H：信道矩阵
%    输入参数x：接收到的信号
%    输出参数c：解调出来的信号矩阵
    G=pinv(H);%求信道矩阵H的伪逆
    c=G*x;%得到初始信号
    [row,col]=size(c);
    for i=1:row%遍历每个元素进行判决
        for j=1:col
            if c(i,j)<0.5
                c(i,j)=0;
            else
                c(i,j)=1;
            end

        end
    end
end
```

图 14: ZF matlab 代码

3.2.2 MMSE 算法分析及实现

如文中所述，ZF 算法没有考虑到噪声的影响，当 $H^H H$ 的特征值较小时，会因噪声放大而导致较大的误差，为了提高检测性能，MMSE 算法能够最小化实际传输符号与线性检测器输出之间的均方误差 (MSE)，并得出滤波器矩阵：

$$G_{MMSE} = (H^H H + \sigma^2 I_{nT})^{-1} H^H$$

进而得到检测信号：

$$\hat{c}_{MMSE} = H^{+} x$$

matlab 代码如图 15。

```
function c=mmse_fun(H,x,n)
%MMSE_FUN 此函数是对MMSE的仿真实现
%    输入参数H：信道矩阵
%    输入参数x：接收到的信号
%    输入参数n：噪声的标准差
%    输出参数c：解调出来的信号矩阵
[~,col]=size(H);%计算H的列数
    I=n*eye(col);
    G=inv(H'*H+I)*H';%得到G矩阵
    c=G*x;
    [row2,col2]=size(c);
    for i=1:row2%遍历元素进行判决
        for j=1:col2
            if c(i,j)<0.5
                c(i,j)=0;
            else
                c(i,j)=1;
            end
        end
    end
end
```

图 15: MMSE matlab 代码

3.2.3 MMSE-QR 算法分析及实现

将 MMSE 算法和 QR 算法进行结合，考虑噪声的影响，同时对信道矩阵进行 QR 分解，避免求伪逆。算法过程如下：

假设噪声方差为 σ, 可以将信道矩阵 H 扩展为 $\underline{H} = \begin{pmatrix} H \\ \sigma_n I_{nT}, \end{pmatrix}$, 而后对其 QR 分解，采用上面提到的 QR 分解算法进行计算即可。matlab 代码如图 16。

```
function c= mmse_qr_fun(H,x,n)
%MMSE_QR_FUN 此函数对MMSE-QRD算法的实现
%    输入参数H：信道矩阵
%    输入参数x：接收的信号
%    输入参数n：噪声的方差
[~,col]=size(H);
I=n*eye(col);
H=[H;I];%对信道矩阵进行扩展
[q,r]=qr(H);
x=[x;zeros(col,1)];%对接收的信号进行处理
y=q'*x;%对接收到的信号进行处理
d=zeros(col,1);
z=zeros(col,1);
for i=col:-1:1%从最后一层开始检测
    for j=i+1:col%通过迭代获得每一层的干扰项
        d(i,1) = d(i,1) + r(i, j) * c(j,1);
    end
    z(i,1) = y(i,1) - d(i,1);%减去干扰项
    tmp = z(i,1) / r(i, i);%获得原始解调出来的信号
    if tmp < 0.5%对信号进行判决
        c(i,1) = 0;
    else
        c(i,1) = 1;
    end
end
end
```

图 16: MMSE-QRD matlab 代码

3.2.4 MMSE-SQRD 算法的分析及实现

同样地，由于 QR 分解的误码率较高，因此可以考虑每次能够选取最大的 $|k_{r,r}|$ 进行检测，这样可以获得更低的误码率。同时，我们与上述算法一样，将 H 矩阵扩展，并用 SQRD 来计算，可以获得较为精确的发射信号 c。matlab 代码如图 17。

```matlab
function C = mmse_sqrd_fun(H,x,n)
%MMSE_SQRD_FUN 该函数实现MMSE-QRD算法
%   输入参数H：信道矩阵
%   输入参数x：接收到的信号
%   输入参数n：噪声的标准差
%   输出参数c：解调出来的信号矩阵
[~,col]=size(H);
I=n*eye(col);
H=[H;I];%对信道矩阵进行扩展
x=[x;zeros(col,1)];%对接收的信号进行处理
[row,col]=size(H);
c = zeros(col, 1);
C = zeros(col, 1);
q = H;
r = zeros(col, col);
s = 1:col; % 用于记录排序的索引

% 进行QR 分解
for i2 = 1:col
    min_norm = norm(q(:, i2));  % 初始化最小范数
    k = i2;
    % 寻找最小范数列
    for j2 = i2:col
        if min_norm > norm(q(:, j2))
            min_norm = norm(q(:, j2));
            k = j2;
        end
    end
    % 交换列，更新 QR 分解矩阵
    q(:, [i2, k]) = q(:, [k, i2]);
    r(:, [i2, k]) = r(:, [k, i2]);
    s([i2, k]) = s([k, i2]);

    r(i2, i2) = norm(q(:, i2));
    q(:, i2) = q(:, i2) / r(i2, i2);

    % 更新 R 矩阵的剩余部分
    for l = i2+1:col
        r(i2, l) = q(:, i2)' * q(:, l);
        q(:, l) = q(:, l) - r(i2, l) * q(:, i2);
    end
end
y = q' * x;  % 计算中间变量 y
d = zeros(row, 1);
z = zeros(row, 1);

% 反向解调
for i = col:-1:1
    for j = i+1:col
        d(i) = d(i) + r(i, j) * c(j);  % 计算干扰项
    end

    z(i) = y(i) - d(i);  % 消除干扰项
    k = z(i) / r(i, i);%得到解调出来的原始信号
    if k < 0.5%对原始信号进行判决
        c(i) = 0;
    else
        c(i) = 1;
    end
end
% 根据排序结果将解调后的结果排序并返回
for i4 = 1:length(s)
    C(s(i4)) = c(i4);
end
end
```

图 17: MMSE-SQRD matlab 代码

3.3 仿真测试

采用与上一部分相同的方式，分别进行四种算法的误码率和误帧率测试。然后比较这几种算法在相同信噪比下的误码率来进行性能分析。

3.3.1 误码率测试

测试思路 通过假设在满足 $N_T > N_R$ 条件的天线数，通过设定不同的信噪比，在该信噪比下，用错误的比特数除以总比特数，得到不同算法的误码率，来对各个算法进行分析。matlab 代码如图 18。由于计算量巨大，使用了 matlab 的并行计算工具包 Parallel Computing。Toolbox

```
function ber = ber_test_fun2(Eb_No_dB,N_R,N_T,N_frame,func)
%VBLAST_TEST_FUN 该函数用来测试算法的误码率
%    输入参数Eb_No_dB：信噪比
%    输入参数N_R：接收天线数
%    输入参数N_R：发射天线数
%    输入参数N_frame：测试信号的帧数
%    输入参数func：函数句柄，使用解调算法的函数
%    输出参数ber:计算得到的误码率
total_bit=N_T*N_frame;%总的比特数
error_bit=0;%初始化错误的比特数
for i=1:N_frame
    bit_stream_tx=randi([0,1],N_T,1);%发送的比特流
    c=bit_stream_tx;
    H=randn(N_R,N_T);
    Eb_No=10^(Eb_No_dB/10);    % 转换为绝对值
    bits_per_symbol=1;  % 假设采用二元调制映射，  每个符号含有1bit信息
    SNR=Eb_No*bits_per_symbol;       % 根据符号所含比特数 增大信噪比
    sigma=1/SNR;     % 假设发送符号能量平均为1，SNR定义为  符号能量/噪声能量，  可求出噪声能量（方差）
    v=sqrt(sigma)*randn(N_R,1);     % 生成满足当前信噪比条件的 高斯随机噪声样本
    x=H*c+v;%接收到的比特流
    info = functions(func);%对信号进行解调
    if strcmp(info.function, 'zf_fun')
        bit_stream_rx=func(H,x);
    else
        bit_stream_rx=func(H,x,sigma);
    end
    error_bit_vector=bit_stream_rx-bit_stream_tx;%累加错误的比特数
    error_bit=error_bit+length(find(error_bit_vector));
end
ber=error_bit/total_bit;%计算误码率
end
```

图 18: 误码率测试代码

测试结果　对不同情况的误码率仿真测试结果如下：

1. $n_T = 8, n_R = 12$, 信噪比为 0-20dB, 测试帧数为 1000000，结果如图 19
2. $n_T = 4, n_R = 6$, 信噪比为 0-20dB, 测试帧数为 1000000，结果如图 20

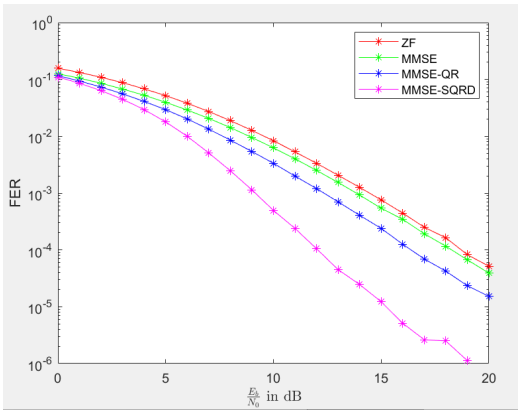

图 19: $n_T = 8, n_R = 12$, 信噪比为 0-20dB, 测试帧数为 1000000 的误码率测试结果图

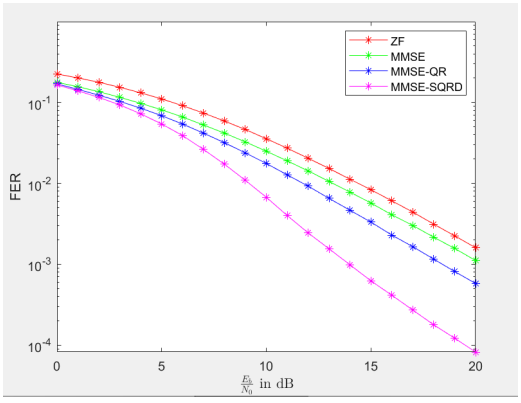

图 20: $n_T = 4, n_R = 6$, 信噪比为 0-20dB, 测试帧数为 1000000 的误码率测试结果图

3.3.2 误帧率测试

测试思路 通过假设在满足 $N_T > N_R$ 条件的天线数，通过设定不同的信噪比，假定一定的帧数，采用类似误码率的统计方法，计算得到误帧率。然后通过比较误帧率和计算复杂度来对各个算法进行分析，进行误帧率计算的 matlab 代码如图 21。由于计算量巨大，使用了 matlab 的并行计算工具包 Parallel Computing Toolbox。

```
function ber = fer_test_fun2(Eb_No_dB,N_R,N_T,N_frame,func)
%VBLAST_TEST_FUN 该函数用来测试算法的误码率
%   输入参数Eb_No_dB：信噪比
%   输入参数N_R：接收天线数
%   输入参数N_R：发射天线数
%   输入参数N_frame：测试信号的帧数
%   输入参数func：函数句柄，使用解调算法的函数
%   输出参数fer:计算得到的误帧率
total_frame=N_frame;%总的比特数
error_frame=0;%初始化错误的比特数
for i=1:N_frame
    bit_stream_tx=randi([0,1],N_T,1);%发送的比特流
    c=bit_stream_tx;
    H=randn(N_R,N_T);
    Eb_No=10^(Eb_No_dB/10);    % 转换为绝对值
    bits_per_symbol=1;  % 假设采用二元调制映射，  每个符号含有1bit信息
    SNR=Eb_No*bits_per_symbol;        % 根据符号所含比特数 增大信噪比
    sigma=1/SNR;     % 假设发送符号能量平均为1，SNR定义为  符号能量/噪声能量，  可求出噪声能量（方差）
    v=sqrt(sigma)*randn(N_R,1);     % 生成满足当前信噪比条件的 高斯随机噪声样本
    x=H*c+v;%接收到的比特流
    info = functions(func);%对信号进行解调
    if strcmp(info.function, 'zf_fun')
        bit_stream_rx=func(H,x);
    else
        bit_stream_rx=func(H,x,sigma);
    end
if any(bit_stream_rx-bit_stream_tx)%累加错误的帧数
    error_frame=error_frame+1;
end
end
ber=error_frame/total_frame;%计算误码率
end
```

图 21: 误帧率测试代码

测试结果 对不同情况的误帧率仿真测试结果如下：

1. $n_T = 8, n_R = 12$, 信噪比为 0-20dB, 测试帧数为 1000000，如图 22
2. $n_T = 4, n_R = 6$, 信噪比为 0-20dB, 测试帧数为 1000000，如图 23

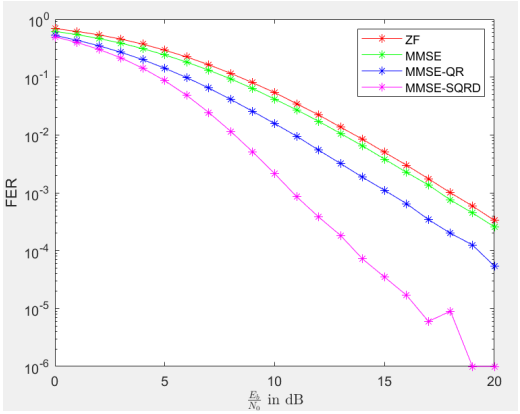

图 22: $n_T = 8, n_R = 12$, 信噪比为 0-20dB, 测试帧数为 1000000 的误帧率测试结果图

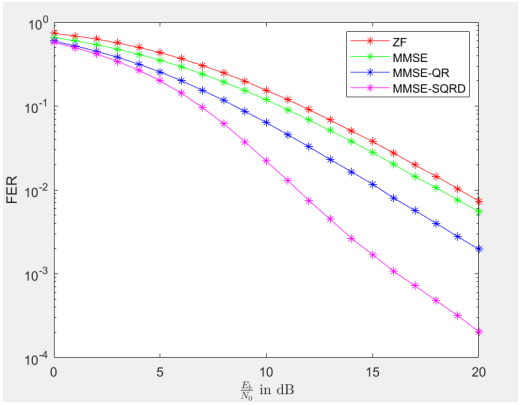

图 23: $n_T = 4, n_R = 6$, 信噪比为 0-20dB, 测试帧数为 1000000 的误帧率测试结果图

3.3.3 结果分析

从得到的图像可以看出，ZF 算法是这四个算法中误码率最高的，因为其没有考虑到噪声的影响，而 MMSE 算法考虑到了噪声的影响，所以误码率有所下降，但是其需要对 H 求逆运算，所以复杂度较高。通过考虑对 H 进行 QR 分解，可以降低复杂度，对 QR 分解进行排序，可以进一步降低误码率。因此，采用 MMSE-SQRD 算法可以获得较高的性能。

4 最大似然算法的实现

4.1 简介

通过阅读资料，该算法的基本思想是通过比较接收到的信号与各种可能发送信号序列之间的似然性，来确定最可能的发送序列。它假设了接收端知道发送信号的统计特性以及噪声的统计特性，并基于这些假设进行数据检测。最似然检测算法能够以很高的准确度来检测出发射信号，因为最似然检测算法计算出信号所有可能的集合，并通过最小距离判决来寻找出最接近的信号，高精度的代价是计算的复杂度更高。

4.2 算法的实现

用 matlab 实现的代码如图 24

```
function c = ml_fun(H,x)
%ML_FUN 该函数实现最大似然检测（ML）算法
%   输入参数H：信道矩阵
%   输入参数x：接收到的信号
%   输出参数c：解调出来的信号矩阵
[row,col]=size(H);%求信道矩阵H的大小
total_num=2^col;
total_probability_t=2*ones(col,total_num);
i=1;
while 1%生成发射信号的所有可能
    randan_arr=randi([0,1],col,1);
    for j=1:total_num
        if randan_arr==total_probability_t(:,j)
            break;
        end
        if j==total_num
            total_probability_t(:,i)=randan_arr;
            i=i+1;
        end
    end
    if i>total_num
        break
    end
end
total_probability_r=H*total_probability_t;%得到接收信号的所有可能
distance=zeros(1,total_num);
for i=1:total_num%计算实际接收到的信号和所有可能的的距离
    distance(1,i)=0;
    for j=1:row
        distance(1,i)=distance(1,i)+(total_probability_r(j,i)-x(j,1))^2;
    end
end
min_distance_index=1;
min_distance=distance(1,1);
for i=1:total_num%找出最小距离所对应的发射信号
    if distance(1,i)<min_distance
        min_distance=distance(1,i);
        min_distance_index=i;
    end
end
c=total_probability_t(:,min_distance_index);%返回得到的发射信号
```

图 24: ML matlab 代码

4.3 仿真测试

利用与上述相同的方法，对 ML 算法和 ZF 算法进行误码率和误帧率测试，然后将两者进行比较分析。由于 ML 算法的计算量随着天线数目的增加呈现指数级增长，所以本部分只对这两者算法进行 $n_T = 4, n_R = 6$, 信噪比为 0-20dB 的仿真测试。

4.3.1 误码率测试

$n_T = 4, n_R = 6$, 信噪比为 0-20dB, 测试帧数为 1000000，结果如图 25

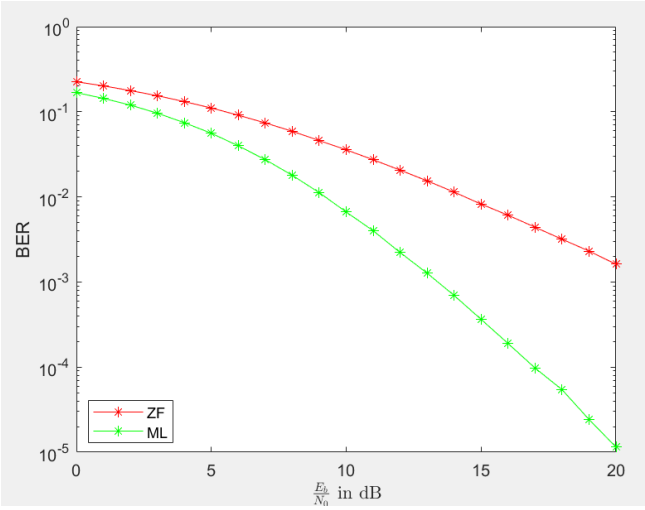

图 25: $n_T = 4, n_R = 6$, 信噪比为 0-20dB, 测试帧数为 1000000 的误码率测试结果图

4.3.2 误帧率测试

$n_T = 4, n_R = 6$, 信噪比为 0-20dB, 测试帧数为 1000000，如图 26

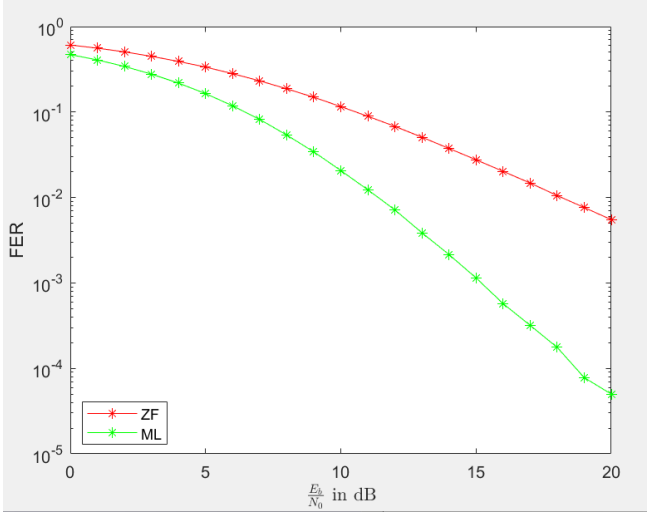

图 26: $n_T = 4, n_R = 6$, 信噪比为 0-20dB, 测试帧数为 1000000 的误帧率测试结果图

4.3.3 结果分析

从图中可以看出，在相同的信噪比下，ML 的误码率和误帧率对于 ZF 算法来说显著降低，主要原因是 ZF 算法计算了发送信号所有可能的集合，在理论上可以以最高的准确度找到发送信号，但是随着通道数的增加，信号的可能性指数增长，运算量急剧增大，所有在实际应用中，ML 算法的应用范围受到了很大的限制，实用性不强。

5 对算法实现过程一些内容的补充

5.1 自编写 QR 算法

在文中提到的许多算法中，多处需要用到 QR 分解，在上面的仿真实现中，都是直接调用了系统中的函数，下面尝试自己编写 QR 分解算法进行补充。

实现代码如图 27，测试结果如图 28，Q，R 是使用自编写 QR 分解得到的结果；q,r 是使用系统 QR 分解得到的结果。

```
function [q,r] = my_qr_fun(H)
%MY_QR_FUN 该函数实现qr分解
q=H;
[~,col]=size(q);%计算H的列数
for i=1:col
    r(i,i)=norm(q(:,i));
    q(:,i)=q(:,i)/r(i,i);
    for j=i+1:col
        r(i,j)=q(:,i)'*q(:,j);
        q(:,j)=q(:,j)-r(i,j)*q(:,i);
    end
end
end
```

图 27: 自编写 QR 分解代码

```
Q =

    0.1690    0.8971
    0.5071    0.2760
    0.8452   -0.3450


R =

    5.9161    7.4374
         0    0.8281


q =

   -0.1690    0.8971    0.4082
   -0.5071    0.2760   -0.8165
   -0.8452   -0.3450    0.4082


r =

   -5.9161   -7.4374
         0    0.8281
         0         0
```

图 28: 自编写 QR 分解测试结果

5.2 将实数信号拓展到复数信号

5.2.1 原理

在上述的讨论中，我们假定一帧信号所携带的信息只有 1 比特，即信号的可能性只有 0 和 1。这里考虑发射天线发射的信号为复数，即 $c = ai + bj$，这样，一帧信号所携带的信息则变为 2 比特。信号在接收端经过信道矩阵 H 处理后，所得到的 x 仍为复数。所得到的信号经过解调后，对其进行复数平面的欧式最小距离判决，即可的到发射信号。

在 matlab 中，复数的运算已经有了定义，所以将实数信号拓展为复数信号十分方便，只需要更改判决方式即可。

5.2.2 实现

这里将 ZF 算法重写，使其能够对复数信号进行解调，其与解调实数信号主要的不同在于对信号的判决方式不同。重写的 matlab 代码如图 29。

```
function C=zf_fun_img(H,x)
%ZF_FUN_IMG 此函数是对ZF复数信号的仿真实现
%   输入参数H：信道矩阵
%   输入参数x：接收到的信号
%   输出参数C：解调出来的信号矩阵
    G=pinv(H);%求信道矩阵H的伪逆
    c=G*x;%得到初始信号
    [row,col]=size(c);
    distance=zeros(1,4);
    C=zeros(row,col);
    for i=1:row%遍历每个元素进行判决
        for j=1:col
            distance(1,1)=real(c(i,j))^2+imag(c(i,j))^2;
            distance(1,2)=real(c(i,j))^2+(imag(c(i,j))-1)^2;
            distance(1,3)=(real(c(i,j))-1)^2+imag(c(i,j))^2;
            distance(1,4)=(real(c(i,j))-1)^2+(imag(c(i,j))-1)^2;
            min_index=find(distance==min(distance));
            if min_index==1
            C(i,j)=0+0i;
            elseif min_index==2
                C(i,j)=0+1i;
            elseif min_index==3
                C(i,j)=1+0i;
            else
                C(i,j)=1+1i;
            end
        end
    end
end
```

图 29: 适用于复数信号的 ML matlab 代码

因此我们可以看出，在 matlab 中，要实现从实数到复数的转变，只需要更改算法的判决方式即可。其他算法同理，因此不一一重写。

5.2.3 测试

我们采用与前文相同的测试方法，测试适用于实数的 ZF 算法和适用于复数的 ZF 算法的误码率，比较分析得到复数信号的优点。测试代码如图 30。

```
function ber = ber_test_fun_img(Eb_No_dB,N_R,N_T,N_frame,func)
%VBLAST_TEST_FUN_BER 该函数用来测试算法的误码率
%   输入参数Eb_No_dB：信噪比
%   输入参数N_R：接收天线数
%   输入参数N_R：发射天线数
%   输入参数N_frame：测试信号的帧数
%   输入参数func：函数句柄，使用解调算法的函数
%   输出参数ber:计算得到的误码率
total_bit=N_T*N_frame*2;%总的比特数
error_bit=0;%初始化错误的比特数
for i=1:N_frame
    bit_stream_tx1=randi([0,1],N_T,1);%发送的比特流
    bit_stream_tx2=randi([0,1],N_T,1);
    bit_stream_tx=complex(bit_stream_tx1,bit_stream_tx2);
    c=bit_stream_tx;
    H1=randn(N_R,N_T);
    H2=randn(N_R,N_T);
    H=complex(H1,H2);
    Eb_No=10^(Eb_No_dB/10);    % 转换为绝对值
    bits_per_symbol=1;  % 假设采用二元调制映射，  每个符号含有1bit信息
    SNR=Eb_No*bits_per_symbol;       % 根据符号所含比特数 增大信噪比
    sigma=1/SNR;     % 假设发送符号能量平均为1，SNR定义为  符号能量/噪声能量，  可求出噪声能量（方差）
    v1=sqrt(sigma)*randn(N_R,1);     % 生成满足当前信噪比条件的 高斯随机噪声样本
    v2=sqrt(sigma)*randn(N_R,1);
    v=complex(v1,v2);
    x=H*c+v;%接收到的比特流
    bit_stream_rx=func(H,x);
    error_bit_vector_real=real(bit_stream_rx)-real(bit_stream_tx);%累加错误的比特数
    error_bit_vector_imag=imag(bit_stream_rx)-imag(bit_stream_tx);
    error_bit=error_bit+length(find(error_bit_vector_real))+length(find(error_bit_vector_imag));
end
ber=error_bit/total_bit;%计算误码率
end
```

图 30: 测试复数信号代码

通过测试适用于实数的 ZF 算法和适用于复数的 ZF 算法在 1000000 帧时，信噪比在 0-20dB 之间的误码率，得到如图 31 和图 32 所示结果。通过观

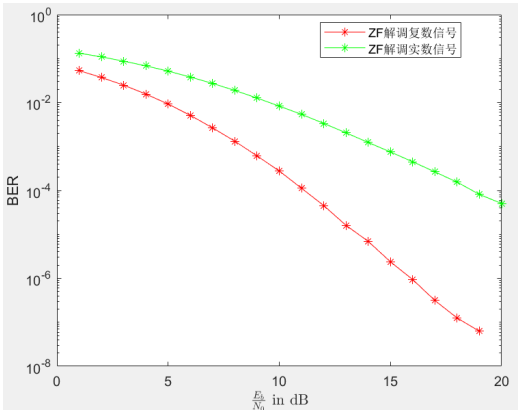

图 31: $n_T = 8, n_R = 12$, 信噪比为 0-20dB, 测试帧数为 1000000 的误帧率测试结果图

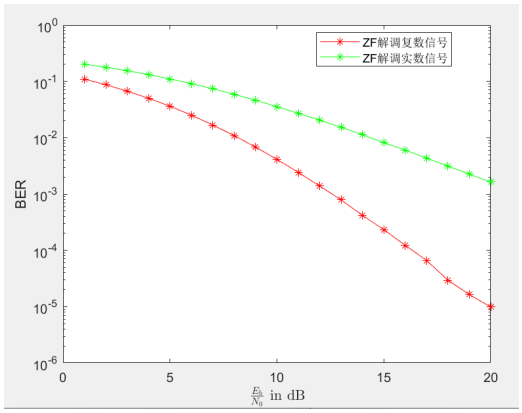

图 32: $n_T = 4, n_R = 6$, 信噪比为 0-20dB, 测试帧数为 1000000 的误帧率测试结果图

察实验结果，发现复数信号比实数信号的误码率更低，因为复数信号相比实数信号携带的信息更多，因此使用复数信号可以提供更大的灵活性和描述能力，但有时也会增加计算复杂度。

6 对算法复杂度的测试

6.1 说明

在上面的算法实现过程中，着重考虑了各个算法的误码率，对各个算法的复杂度只是进行了理论的分析，并没有进行测试比较。这里对算法复杂度进行重新测试，作为补充。

6.2 对论文 1 中各个算法复杂度测试

假设发射天线数为 8，接收天线数为 12 分别用 V-BLAST、QRD、SQRD 算法来处理 10000 帧信号，计算他们所用的时间，即可比较他们的计算复杂度，测试结果如图 33

算法	V-BLAST	QRD	SQRD
时间（秒）	27.210518	2.409250	6.079538

图 33: 对论文 1 各个算法复杂度测试

6.3 对论文 2 中各个算法复杂度测试

假设发射天线数为 8，接收天线数为 12 分别用 ZF、MMSE、MMSE-QRD、MMSE-SQRD 算法来处理 10000 帧信号，计算他们所用的时间，即可比较他们的计算复杂度，测试结果如图 34

算法	ZF	MMSE	MMSE-QRD	MMSE-SQRD
时间（秒）	6.789264	1.594085	4.168950	6.960158

图 34: 对论文 2 各个算法复杂度测试

6.4 总结

在 matlab 中对这几个代码的测试，基本符合理论分析，但是在对 MMSE 算法的测试中，发现 MMSE 算法所用的时间较短，不符合理论分析，可能的原因是 matlab 对矩阵求逆运算的优化。另一方面，在模拟的过程中，很难准确计算出算法运算的时间，因为包含了一些其他的函数处理，导致了结果不太精准。

7 总结与思考

在这次大作业中，最大的收获是如何进行通信仿真。从一开始的对论文无处下手，再到慢慢看懂里面的算法，最后用 matlab 代码将其实现出来。一步步地掌握了仿真的要领。

以本次作业的 MIMO 系统为例，要将这样的一个系统，用 matlab 表征出来。首先是对系统的理解，即对四个系统量 c, x, v, H 的理解，如何用 0 和 1 代表信号，用矩阵的线性运算模拟解调过程，最后用不同的算法由接收信号得到发射信号。在算法实现过程中，最难的是看懂论文中对算法的描述，以及给出的伪代码图。不同的论文。可能对同一变量的描述不一样，因此要学会用统一的符号进行转述。

本文对论文中所有提到的算法都进行了复现并都在不同的 n_R, n_T 下测试了误码率和误帧率，并且还实现了最大似然检测算法。在算法的实现过程中，遇到了许多困难，例如流程图中的许多符号看不懂，以及进行排序时如何记录排列顺序等等。但最后都在同学的帮助下解决了。虽然都对算法进行了复现，但是在进行编程的过程中，有些地方处理地比较笨拙，导致代码繁琐，因此还需改进。

本文用 matlab 进行仿真，实现起来可能比较简单，因为 matlab 丰富的函数库、矩阵运算以及能够实时可见的工作区变量，为算法的复现带来了很大便利。通过使用 matlab 中的串行计算工具包，极大地减少了仿真测试的时间，使得能够在短时间内获得大量的数据。

通过本次对 MIMO 系统中许多算法的仿真测试，我对 MIMO 通信系统有了较深的理解，同时也深刻感受到了在计算机科学计算中，数值计算方法是一门很大的学问。课堂上学习到的知识，在这里有了更深的体会。