LAB2

吴非 519021910924

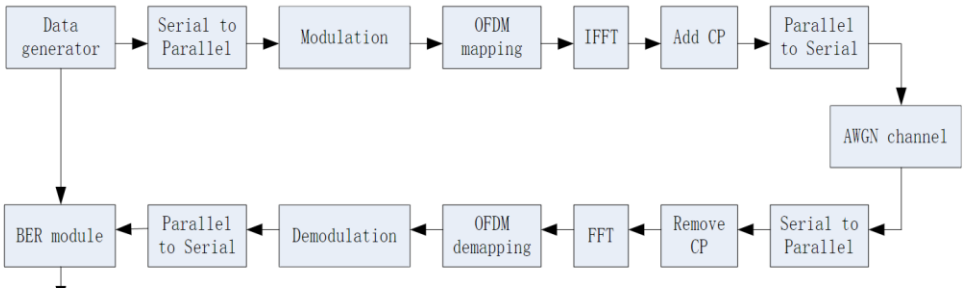
功能模块描述和实现：

函数介绍：

1. ofdm.m: 主函数，计算 OFDM 系统 BER 性能；
2. ofdmmod.m: 调制函数；
3. ofdmdemod.m: 解调函数；
4. comb.m: AWGN 噪声与信道输出生成函数；
5. ofdmmap.m: OFDM 符号映射；
6. addcp.m: 加入循环前缀；
7. removecp.m: 去除循环前缀；
8. ofdmdemap.m : OFDM 符号解映射

具体功能可参考系统框图和实验课助教的讲解，不过多赘述。

系统框图 — OFDM



主要说一下自己补充的代码：

编码：

```
        case 4 % 16-QAM
            I=[-3,-1,3,1];
            Q=[-3,-1,3,1];
            [x, y] = meshgrid(I, Q);
            OFDM.Constellations = reshape([complex(x(:), y(:))],1,[]); % you should fix it;
        case 6 % 64-QAM
            I=[-7,-5,-1,-3,7,5,1,3];
            Q=[-7,-5,-1,-3,7,5,1,3];
            [x, y] = meshgrid(I, Q);
            OFDM.Constellations = reshape([complex(x(:), y(:))],1,[]); % you should fix it;
%             I=[1,2,3]
%             Q=[4,5,6]
%             [x, y] = meshgrid(I, Q);
%             [x(:), y(:)]
```

就是实现了笛卡尔积并将其转化为复数，并 reshape 拉平矩阵。

解码：

有两个版本，第一次我使用嵌套的 for 循环，也是助教不建议的，(比较慢而且代码冗长)：

```
% %                    y(i,j)=result(2);
            data_in_i = real(data_in);
            data_in_q = imag(data_in);
% %              qam_16 =containers.Map({-3,-1,1,3},{[0,0],[0,1],[1,1],[1,0]});
%              tmp_i_1=ones(size(data_in_i,1),size(data_in_i,2));
%              tmp_i_2=ones(size(data_in_i,1),size(data_in_i,2));
%              tmp_q_1=ones(size(data_in_q,1),size(data_in_q,2));
%              tmp_q_2=ones(size(data_in_q,1),size(data_in_q,2));
% %              disp(size(x,1))
%              for i = 1:size(data_in_i,1)
%                  for j = 1:size(data_in_i,2)
%                      tmp_i=data_in_i(i,j);
%                      tmp_q=data_in_q(i,j);
%                      result_i=qam_16(tmp_i);
%                      result_q=qam_16(tmp_q);
%                      tmp_i_1(i,j)=result_i(1);
%                      tmp_i_2(i,j)=result_i(2);
%                      tmp_q_1(i,j)=result_q(1);
%                      tmp_q_2(i,j)=result_q(2);
%                  end
%              end
%              data_out(:, 1 : 4 : end - 3) = tmp_i_1;
%              data_out(:, 2 : 4 : end - 2) = tmp_i_2;
%              data_out(:, 3 : 4 : end - 1) = tmp_q_1;
%              data_out(:, 4 : 4 : end) = tmp_q_2;
```

主要思想就是挨个处理矩阵的每个元素。

简单的版本(使用向量操作)如下：

```
%              data_out(:, 4 : 4 : end) = tmp_q_2;
            data_out(:, 1 : 4 : end - 3) = (data_in_i>=0&data_in_i<=4);
            data_out(:, 2 : 4 : end - 2) = (data_in_i>=-2&data_in_i<=2);
            data_out(:, 3 : 4 : end - 1) = (data_in_q>=0&data_in_q<=4);
            data_out(:, 4 : 4 : end) = (data_in_q>=-2&data_in_q<=2);
        case 6 % 64-QAM
            % Please determine data_out~
            data_in_i = real(data_in);
            data_in_q = imag(data_in);
            data_out(:, 1 : 6 : end - 5) = (data_in_i>0&data_in_i<=8);
            data_out(:, 2 : 6 : end - 4) = (data_in_i>=-4&data_in_i<=4);
            data_out(:, 3 : 6 : end - 3) = (data_in_i>=-6&data_in_i<-2)|(data_in_i>=2&data_in_i<6);
            data_out(:, 4 : 6 : end - 2) =  (data_in_q>0&data_in_q<=8);
            data_out(:, 5 : 6 : end - 1) = (data_in_q>=-4&data_in_q<=4);
            data_out(:, 6 : 6 : end) = (data_in_q>=-6&data_in_q<-2)|(data_in_q>=2&data_in_q<6);
        otherwise
```

即根据编码表设置上下阈值处理即可：

16-QAM encoding table

Input bit (b_0b_1)	I-out
00	-3
01	-1
11	1
10	3

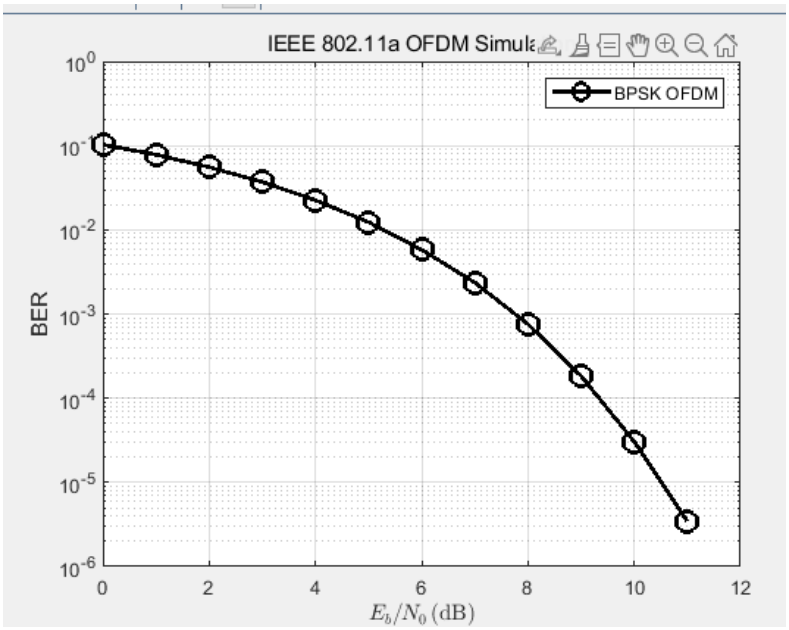
Input bit (b_2b_3)	Q-out
00	-3
01	-1
11	1
10	3

经实验两种方式都可以得到正确图像。

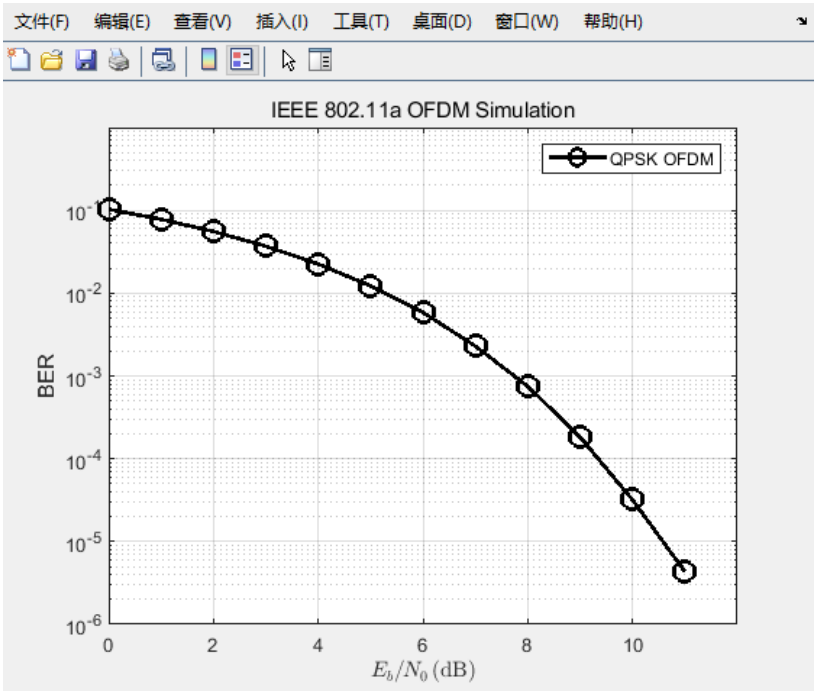
最后是 test.m，是一些关于 matlab 基础语法的测试（已经用%注释掉了），没有注释的部分为最后展示多个线条在一张图像中的代码。

Ber 曲线模拟和理论结果：

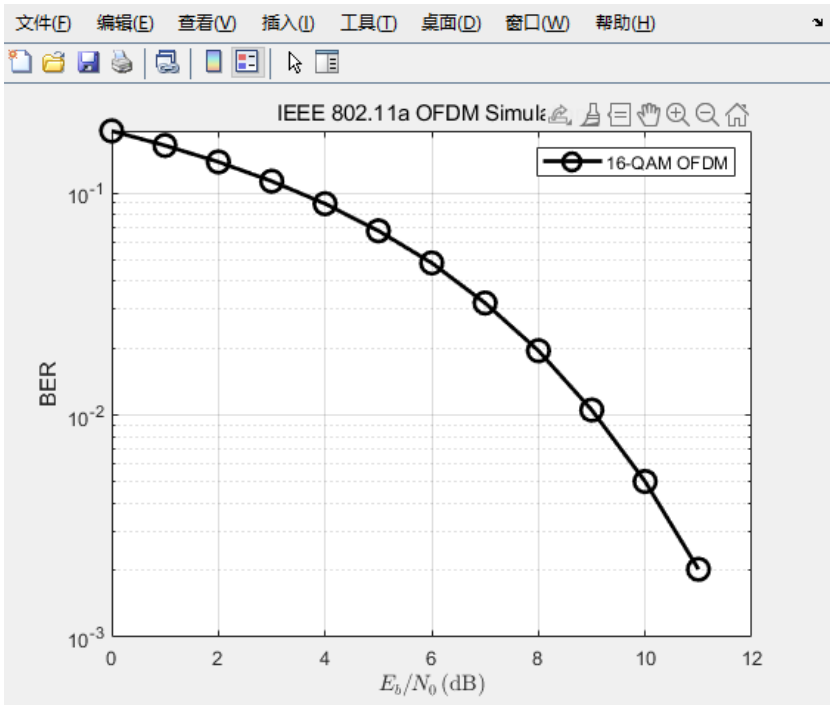
模拟结果：



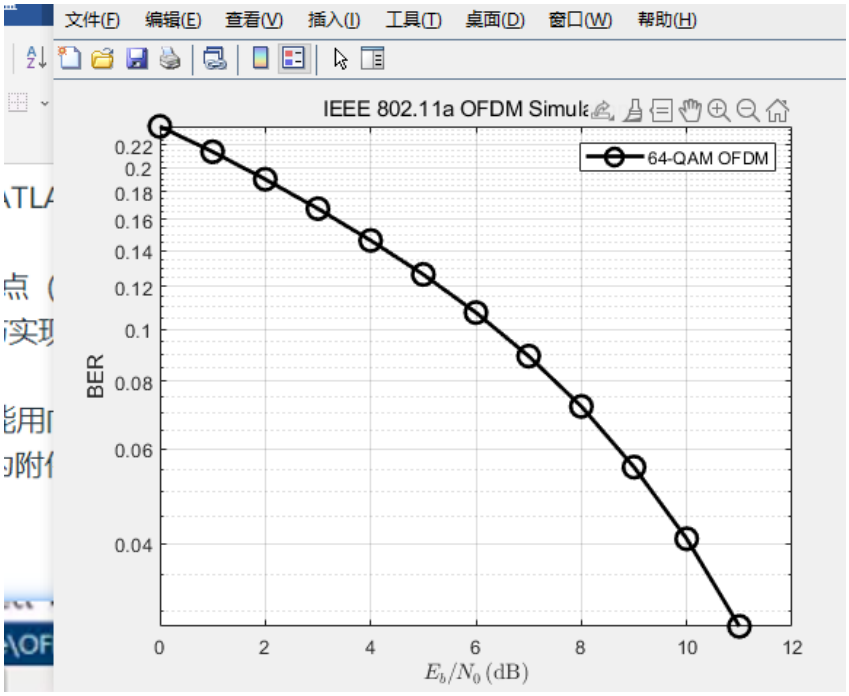

```
Begin the simulation of Eb/N0 = 11 dB;
Simulation end. The BER of the system is:
    0.1027    0.0781    0.0555    0.0370    0.0225    0.0123    0.0058    0.0024    0.0008    0.0002    0.0000    0.0000
```




```
Begin the simulation of Eb/N0 = 11 dB;
Simulation end. The BER of the system is:
    0.1029    0.0779    0.0555    0.0369    0.0225    0.0122    0.0059    0.0023    0.0008    0.0002    0.0000    0.0000
```



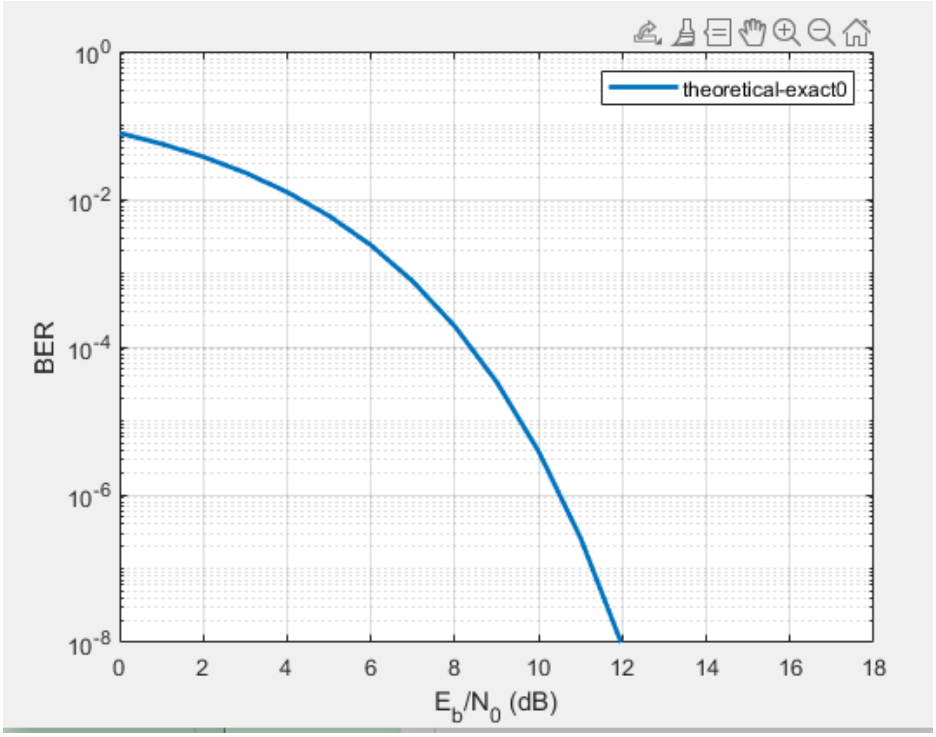

```
Simulation end. The BER of the system is:
    0.1905    0.1637    0.1381    0.1131    0.0895    0.0676    0.0484    0.0320    0.0195    0.0105    0.0050    0.0020
```




```
Simulation end. The BER of the system is:
    0.2383    0.2138    0.1901    0.1676    0.1463    0.1265    0.1074    0.0892    0.0719    0.0554    0.0408    0.0280
```

理论结果:

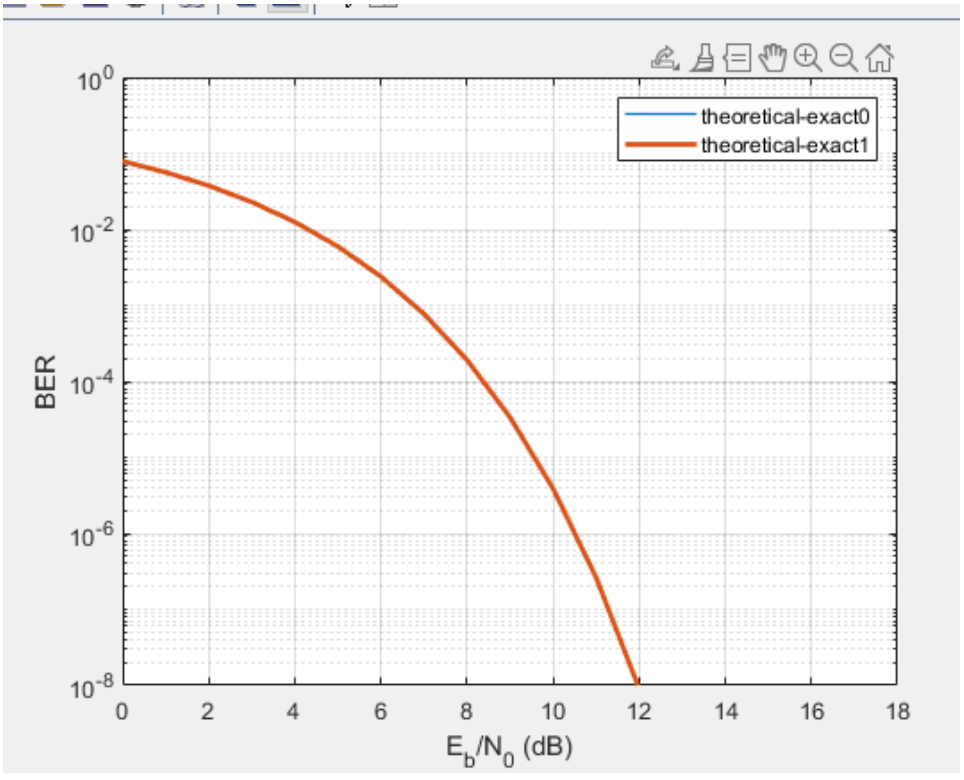
BPSK:



	1	2	3	4	5	6	7	8	9
1	0.0787	0.0563	0.0375	0.0229	0.0125	0.0060	0.0024	7.7267e-04	1.9091e-04

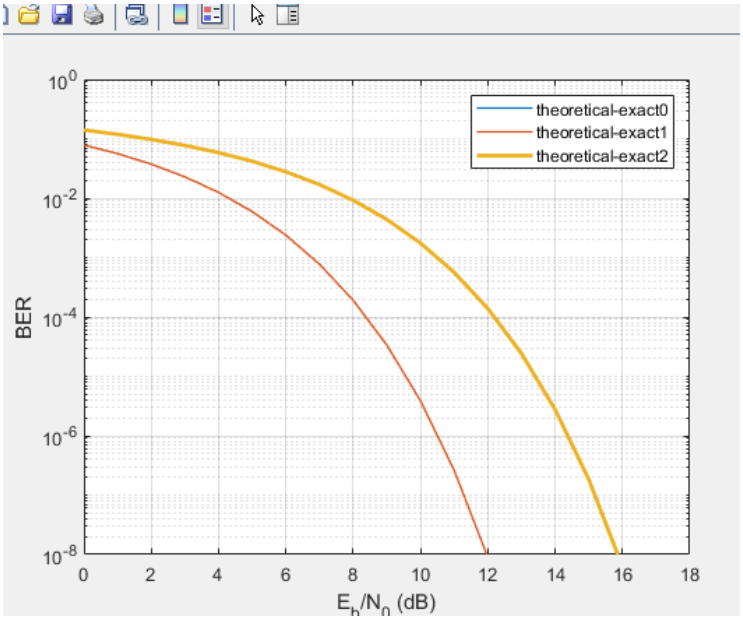
	10	11	12	13	14	15	16	17	18	19
4	3.3627e-05	3.8721e-06	2.6131e-07	9.0060e-09	1.3329e-10	6.8102e-13	9.1240e-16	2.2674e-19	6.7590e-24	1.3960e-29

QPSK:



可见曲线和 BPSK 重合，所以具体数据值就不展示了。

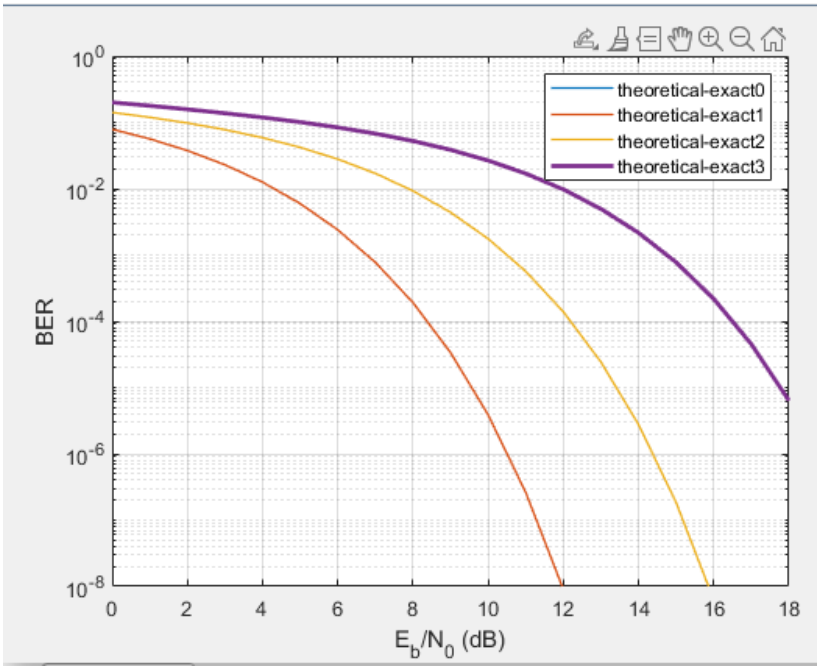
16-QAM:



1	2	3	4	5	6	7	8	9	10
0.1410	0.1190	0.0977	0.0775	0.0586	0.0419	0.0279	0.0170	0.0092	0.004

11	12	13	14	15	16	17	18	19
0.0018	5.6471e-04	1.3866e-04	2.4234e-05	2.7632e-06	1.8419e-07	6.2502e-09	9.0716e-11	4.5223e-13

64-QAM:



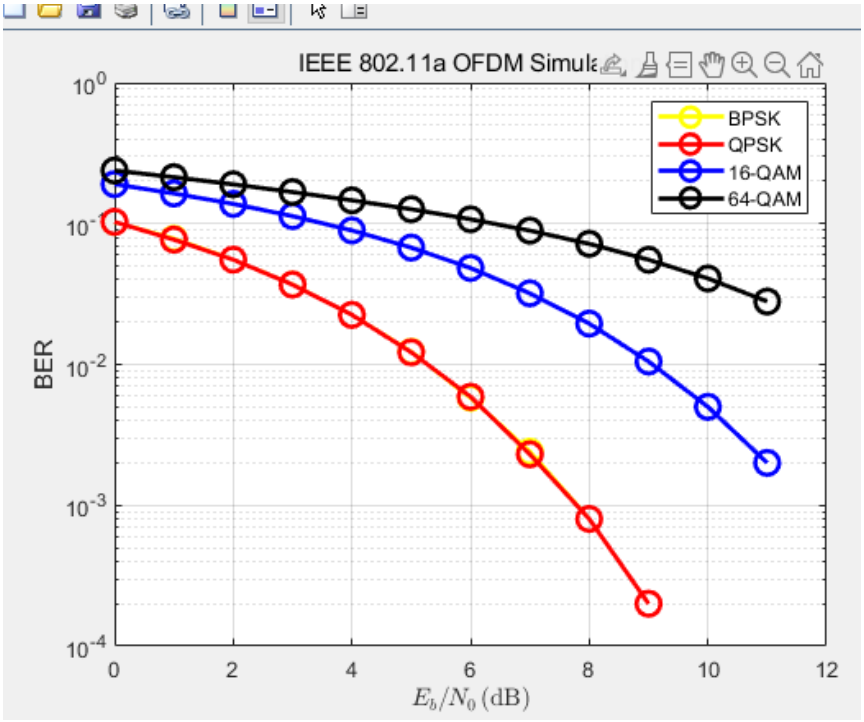
1	2	3	4	5	6	7	8	9	10
0.1998	0.1779	0.1570	0.1372	0.1185	0.1008	0.0838	0.0676	0.0523	0.0385

11	12	13	14	15	16	17	18	19
0.0265	0.0169	0.0097	0.0049	0.0022	7.7247e-04	2.1717e-04	4.4989e-05	6.3511e-06

性能对比和分析（数据后处理）：

本部分根据之前得到的结果值，重新编写代码绘制，来将其展示在一张图中。

模拟曲线间的对比：



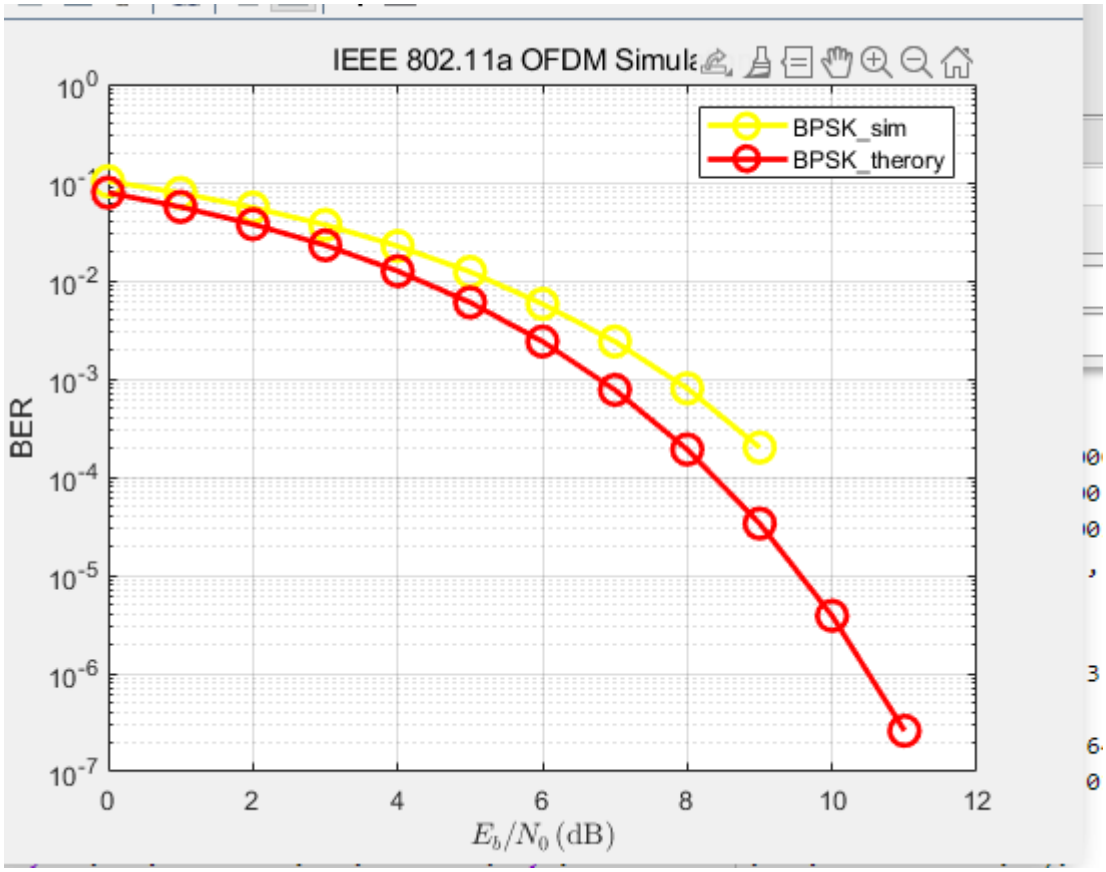
可以得到的结论是，BER 性能曲线 64-QAM 大于 16-QAM 大于 BPSK=QPSK。

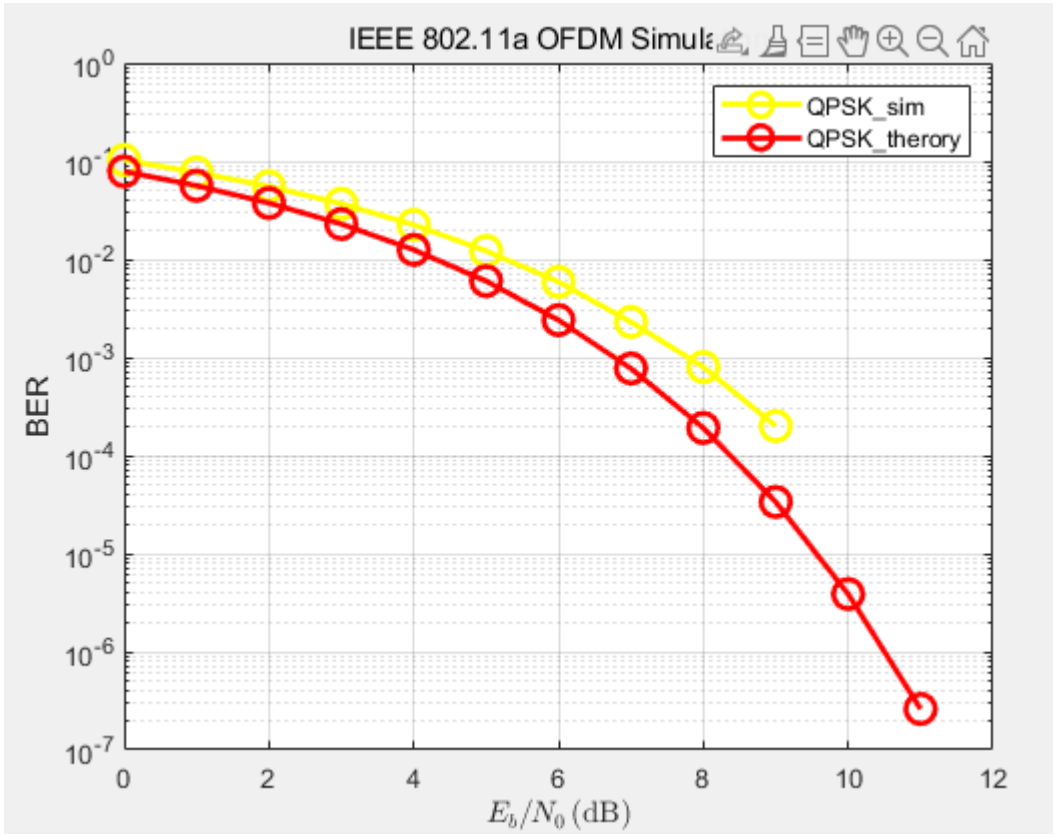
（而 BER 为误码率曲线，因此这也代表着各种调制方式间误码率之间的关系）

这和理论符合，因为 M 越大（64>16）频带利用率越大，但是也意味着编码后信号之间的距离越近（直观上可从星座图上观察得到），加入白噪声后误码率显然会越大，这是个 trade-off。

模拟和理论曲线对比：

由于模拟只模拟了 0-11 之前的值，对于理论图像也取对应的值即可。

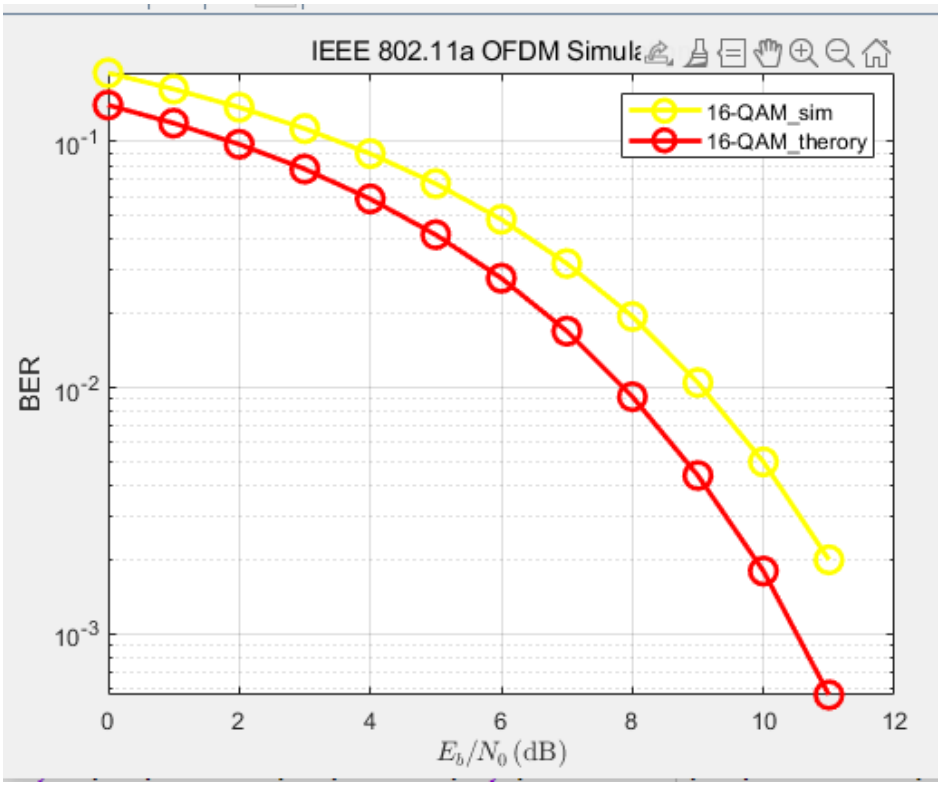


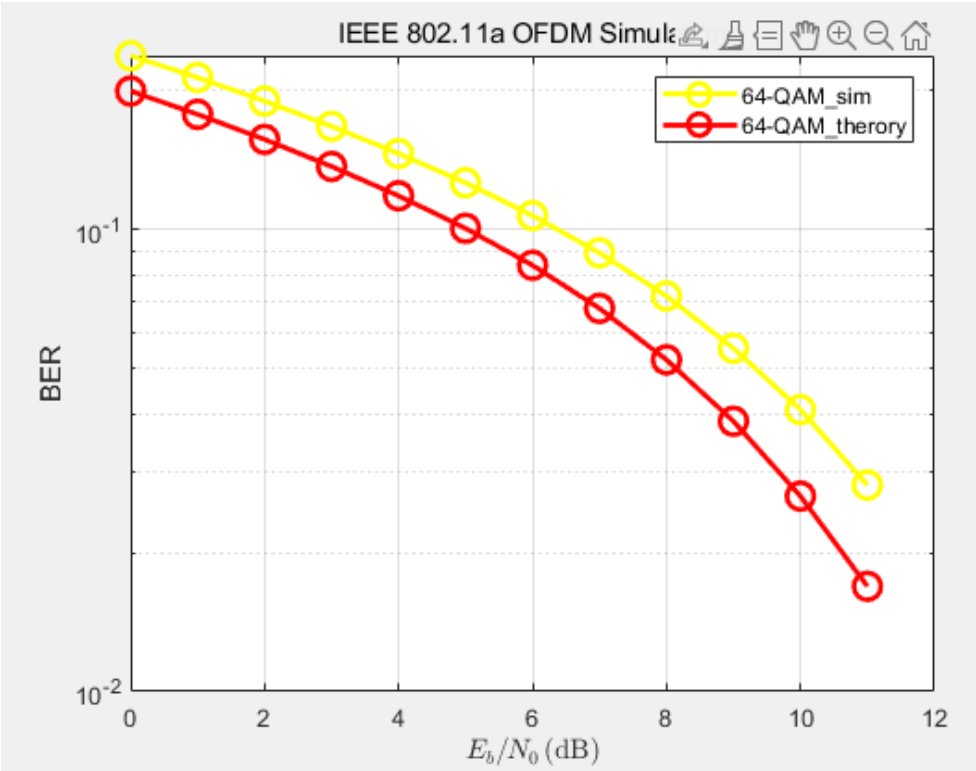


可以注意到这两张图的模拟后两个值是零，但是展示在原图的时候却不是 0，原因应该是

```
disp(BER_res);
% plot 16e
semilogy(EbN0_dB_list, BER_res, 'LineStyle','-', 'Marker','o', 'Color', [0 0 0],'MarkerSize', 11, 'LineWidth', 2);
legend_label = [OFDM.mod_type, ' OFDM'];
legend(legend_label)
xlabel('$E_b / N_0\, \mathrm{(dB)}$','interpreter','latex', 'FontSize', 11);
ylabel('BER');
title('IEEE 802.11a OFDM Simulation');
grid on
```

disp 函数打印的值精度不够。





可以看到模拟的值都大于理论值，这也是**思考题三**的问题：
原因如下：

```
%% Guard interval insertion
ch3 = addcp(ch2, OFDM);
```

```
%% Noise power calculation by the definition of Eb/N0
Es = sum(abs(ch3).^2) / (OFDM.Nd * OFDM.para); % Es is the average symbol power
N0 = (Es / OFDM.mod_lev) / EbN0_linear_list(cnt);  % use Eb/N0 to calculate N0, Eb = Es / mod_lev
```

其原因主要在于 addcp 添加了循环前缀以便减轻信号间的干扰，但是这也让信号的能量也即 Es 变大，而分母 N0 的值没变，这也就意味着 y 轴的值不变，x 轴的值向右移动，也就是模拟的值都大于理论值这种现象出现的原因。

具体偏差值为 $10\lg\frac{64}{48}$=1.3dB.

思考题：

1:
IEEE 802.11a 中使用了 52 个子载波（实际上应为 53 个，其中 k=0 处的直流子载波上不传输符号），由于 IFFT 算法基于 2 点，故采用 64 点的 IFFT。53 个子载波在频率分配时分别在编号低端和高端留有 6 个和 5 个空符号，即 k=— 32…，— 27，27，…，31，这样就可以保证系统的子载波频谱集中，从而使得系统占用的频谱带宽尽可能窄，以节约频谱资源，减少信道间干扰。所以，52 个非零子信道映射到 64 点输入的 IFFT 当中应按照图中所指定的方式，把子信道 1~26 映射到相同标号的 IFFT 输入端口；而子信道—26~-1 被映射到标记为 38~63 的 IFFT 输入端口；其余的 IFFT 输入端口，即 27~37 输入空值。
2:
可以从 BPSK 和 QPSK 的编码表得到：

BPSK encoding table

Input bit (b_0)	I-out	Q-out
0	-1	0
1	1	0

可见 BPSK 的 Q 路没有携带信息，都是 0.

QPSK encoding table

Input bit (b_0)	I-out
0	-1
1	1

Input bit (b_1)	Q-out
0	-1
1	1

可见 QPSK 的 I 路和 Q 路的值一样，这也就意味着 QPSK 相对于 BPSK 仅仅是多传了一路信号 多花了一条路的能量，QPSK 通道的符号差错率为 BPSK 通道误码率的两倍；而 QPSK 通道的比特差错率为其符号差错率的一半,所以误码率当然一致。

教材也有提到：每个正交的 BPSK 通道和混合而成的 QPSK 都具有相同的 Eb/N0。

3:
见之前描述。

分工:

吴非同学自己做的。

总结:

通过 matlab 仿真 ofdm 各种调制方法，更深刻地了解了各种调制方法的具体每个步骤的代码实现，也通过 matlab 的仿真进行了理论与仿真性能曲线的比较，验证了理论的正确性。