Appendix A
MATLAB Programs (Batch Mode)

A- 1 Free space losses Simulation

```
% plot_PL_general.m
clear, clf
fc=1.5e9; d0=100; sigma=3; distance=[1:2:31].^2;
Gt=[1 1 0.5]; Gr=[1 0.5 0.5]; Exp=[2 3 6];
for k=1:3
y_Free(k,:)=PL_free(fc,distance,Gt(k),Gr(k));
y_logdist(k,:)=PL_logdist_or_norm(fc,distance,d0,Exp(k));
y_lognorm(k,:)=PL_logdist_or_norm(fc,distance,d0,Exp(1),sigma);
end
semilogx(distance,y_Free(1,:),'r-o', distance,y_Free(2,:),'b-^',
distance,y_Free(3,:),'k-s'), grid on, axis([1 1000 40 110]),
title(['Free Path-loss Model, f_c=',num2str(fc/1e6),'MHz'])
xlabel('Distance[m]'), ylabel('Path loss[dB]')
legend('G_t=1, G_r=1','G_t=1, G_r=0.5','G_t=0.5, G_r=0.5',2)
figure
semilogx(distance,y_logdist(1,:),'r-o', distance,y_logdist(2,:),'b-
^',distance,y_logdist(3,:),'k-s'), grid on, axis([1 1000 40 110]),
title(['Log-distance Path-loss Model, f_c=',num2str(fc/1e6),'MHz'])
xlabel('Distance[m]'), ylabel('Path loss[dB]'),
legend('n=2','n=3','n=6',2)
figure
semilogx(distance,y_lognorm(1,:),'r-o', distance,y_lognorm(2,:),'b-
^', distance,y_lognorm(3,:),'k-s')
grid on, axis([1 1000 40 110]), legend('path 1','path 2','path 3',2)
title(['Log-normal Path-loss Model, f_c=',num2str(fc/1e6),'MHz,',
'\sigma=', num2str(sigma), 'dB, n=2'])
xlabel('Distance[m]'), ylabel('Path loss[dB]')
```

A-2 Ray Ric channel fading Simulation

```
% plot_Ray_Ric_channel.m
clear, clf
N=200000; level=30; K_dB=[-40 15];
gss=['k-s'; 'b-o'; 'r-^'];
% Rayleigh model
Rayleigh_ch=Ray_model(N);
[temp,x]=hist(abs(Rayleigh_ch(1,:)),level);
plot(x,temp,gss(1,:)), hold on
% Rician model
for i=1:length(K_dB);
Rician_ch(i,:) = Ric_model(K_dB(i),N);
[temp x] = hist(abs(Rician_ch(i,:)),level);
plot(x,temp,gss(i+1,:))
end
```

```
xlabel('x'), ylabel('Occurrence')
legend('Rayleigh','Rician, K=-40dB','Rician, K=15dB')
```

A – 3 OFDM system Simulation:

```
% OFDM_basic.m
clear all
NgType=1; % NgType=1/2 for cyclic prefix/zero padding
if NgType==1, nt='CP';
elseif NgType==2, nt='ZP';
end
Ch=0; % Ch=0/1 for AWGN/multipath channel
if Ch==0, chType='AWGN';
    Target_neb=100;
else chType='CH';
    Target_neb=500;
end
figure(Ch+1), clf
PowerdB=[0 -8 -17 -21 -25]; % Channel tap power profile 'dB'
Delay=[0 3 5 6 8]; % Channel delay 'sample'
Power=10.^(PowerdB/10); % Channel tap power profile 'linear scale'
Ntap=length(PowerdB); % Chanel tap number
Lch=Delay(end)+1; % Channel length
Nbps=4; M=2^Nbps; % Modulation order=2/4/6 for QPSK/16QAM/64QAM
Nfft=64; % FFT size
Ng=Nfft/4; % GI (Guard Interval) length (Ng=0 for no GI)
Nsym=Nfft+Ng; % Symbol duration
Nvc=Nfft/4; % Nvc=0: no VC (virtual carrier)
Nused=Nfft-Nvc;
EbN0=0:5:30; % EbN0
N_iter=1e5; % Number of iterations for each EbN0
Nframe=3; % Number of symbols per frame
sigPow=0; % Signal power initialization
file_name=['OFDM_BER_' chType '_' nt '_' 'GL' num2str(Ng) '.dat'];
fid=fopen(file_name, 'w+');
norms=[1 sqrt(2) 0 sqrt(10) 0 sqrt(42)]; % BPSK 4-QAM 16-QAM
for i=0:length(EbN0)
randn('state',0);rand('state',0);
%Ber2=ber_QAM(EbN0,M,'AWGN'); % BER initialization
Neb=0; Ntb=0; % Initialize the number of error/total bits
for m=1:N_iter
% Tx_____________________________________________________________
X= randint(1,Nused*Nframe,M); % bit: integer vector
Xmod= qammod(X,M,0,'gray')/norms(Nbps);
if NgType~=2, x_GI=zeros(1,Nframe*Nsym);
elseif NgType==2, x_GI= zeros(1,Nframe*Nsym+Ng);
% Extend an OFDM symbol by Ng zeros
end
kk1=[1:Nused/2]; kk2=[Nused/2+1:Nused]; kk3=1:Nfft; kk4=1:Nsym;
for k=1:Nframe
if Nvc~=0, X_shift= [0 Xmod(kk2) zeros(1,Nvc-1) Xmod(kk1)];
```

```
else X_shift= [Xmod(kk2) Xmod(kk1)];
end
x= ifft(X_shift);
x_GI(kk4)= guard_interval(Ng,Nfft,NgType,x);
kk1=kk1+Nused; kk2= kk2+Nused; kk3=kk3+Nfft; kk4=kk4+Nsym;
end
if Ch==0, y= x_GI; % No channel
else % Multipath fading channel
channel=(randn(1,Ntap)+j*randn(1,Ntap)).*sqrt(Power/2);
h=zeros(1,Lch); h(Delay+1)=channel; % cir: channel impulse response
y = conv(x_GI,h);
end
if i==0 % Only to measure the signal power for adding AWGN noise
y1=y(1:Nframe*Nsym); sigPow = sigPow + y1*y1';
continue;
end
% Add AWGN noise_____________________________________________________
snr = EbN0(i)+10*log10(Nbps*(Nused/Nfft)); % SNR vs. Eb/N0 by Eq.
(4.28)
noise_mag = sqrt((10.^(-snr/10))*sigPow/2);
y_GI = y + noise_mag*(randn(size(y))+j*randn(size(y)));
% Rx_____________________________________________________________________
kk1=(NgType==2)*Ng+[1:Nsym]; kk2=1:Nfft;
kk3=1:Nused; kk4=Nused/2+Nvc+1:Nfft; kk5=(Nvc~=0)+[1:Nused/2];
if Ch==1
H= fft([h zeros(1,Nfft-Lch)]); % Channel frequency response
H_shift(kk3)= [H(kk4) H(kk5)];end
for k=1:Nframe
Y(kk2)= fft(remove_GI(Ng,Nsym,NgType,y_GI(kk1)));
Y_shift=[Y(kk4) Y(kk5)];
if Ch==0, Xmod_r(kk3)= Y_shift;
else Xmod_r(kk3)=Y_shift./H_shift;
 %#ok<*SAGROW> % Equalizer - channel compensation
end
kk1=kk1+Nsym; kk2=kk2+Nfft; kk3=kk3+Nused; kk4=kk4+Nfft;
kk5=kk5+Nfft;end
X_r=qamdemod(Xmod_r*norms(Nbps),M,0,'gray');
Neb=Neb+sum(sum(de2bi(X_r,Nbps)~=de2bi(X,Nbps)));
Ntb=Ntb+Nused*Nframe*Nbps; %[Ber,Neb,Ntb]=ber(bit_Rx,bit,Nbps);
if Neb>Target_neb, break; end
end
if i==0, sigPow= sigPow/Nsym/Nframe/N_iter;
else
Ber = Neb/Ntb;
fprintf('EbN0=%3d[dB], BER=%4d/%8d =%11.3e\n', EbN0(i), Neb,Ntb,Ber)
fprintf(fid, '%d\t%11.3e\n', EbN0(i), Ber);
if Ber<1e-6, break;
end
end
end
```

```
if (fid~=0), fclose(fid);
end
plot_ber(file_name,Nbps);
```

A – 4 Channel Estimation Simulation :

```
%channel_estimation.m
% for LS/DFT Channel Estimation with linear/spline interpolation
clear all; close all; clf
Nfft=32; Ng=Nfft/8; Nofdm=Nfft+Ng; Nsym=100;
Nps=4; Np=Nfft/Nps; % Pilot spacing and number of pilots per OFDM
symbol
Nbps=4; M=2^Nbps; % Number of bits per (modulated) symbol
mod_object = modem.qammod('M',M, 'SymbolOrder','gray');
demod_object = modem.qamdemod('M',M, 'SymbolOrder','gray');
Es=1; A=sqrt(3/2/(M-1)*Es); % Signal energy and QAM normalization
factor
SNR = 30; sq2=sqrt(2); MSE = zeros(1,6); nose = 0;
for nsym=1:Nsym
Xp = 2*(randn(1,Np)>0)-1; % Pilot sequence generation
msgint=randint(1,Nfft-Np,M); % bit generation
Data = A*modulate(mod_object,msgint);
ip = 0; pilot_loc = [];
for k=1:Nfft
if mod(k,Nps)==1
X(k)=Xp(floor(k/Nps)+1); pilot_loc=[pilot_loc k]; ip = ip+1;
else X(k) = Data(k-ip);
end
end
x = ifft(X,Nfft); xt = [x(Nfft-Ng+1:Nfft) x]; % IFFT and add CP
h = [(randn+j*randn) (randn+j*randn)/2]; % A (2-tap) channel
H = fft(h,Nfft); ch_length=length(h); % True channel and its length
H_power_dB = 10*log10(abs(H.*conj(H))); % True channel power in dB
y_channel = conv(xt,h); % Channel path (convolution)
yt = awgn(y_channel,SNR,'measured');
y = yt(Ng+1:Nofdm); Y = fft(y); % Remove CP and FFT
for m=1:3
if m==1, H_est = LS_CE(Y,Xp,pilot_loc,Nfft,Nps,'linear');
method='LS-linear'; % LS estimation with linear interpolation
elseif m==2, H_est = LS_CE(Y,Xp,pilot_loc,Nfft,Nps,'spline');
method='LS-spline'; % LS estimation with spline interpolation
else H_est = MMSE_CE(Y,Xp,pilot_loc,Nfft,Nps,h,SNR);
method='MMSE'; % MMSE estimation
end
H_est_power_dB = 10*log10(abs(H_est.*conj(H_est)));
h_est = ifft(H_est); h_DFT = h_est(1:ch_length);
H_DFT = fft(h_DFT,Nfft); % DFT-based channel estimation
H_DFT_power_dB = 10*log10(abs(H_DFT.*conj(H_DFT)));
if nsym==1
```

```
subplot(319+2*m), plot(H_power_dB,'b'); hold on;
plot(H_est_power_dB,'r-+'); legend('True Channel',method);
subplot(320+2*m), plot(H_power_dB,'b'); hold on;
plot(H_DFT_power_dB,'r:+');
legend('True Channel',[method ' with DFT']);
end
MSE(m) = MSE(m) + (H-H_est)*(H-H_est)';
MSE(m+3) = MSE(m+3) + (H-H_DFT)*(H-H_DFT)';
end
Y_eq = Y./H_est; ip = 0;
for k=1:Nfft
if mod(k,Nps)==1, ip=ip+1;
else Data_extracted(k-ip)=Y_eq(k);
end
end
msg_detected = demodulate(demod_object,Data_extracted/A);
nose = nose + sum(msg_detected~=msgint);
MSEs = MSE/(Nfft*Nsym);
end
```

A – 5 Ergodic_Capacity_CDF Simulation :

```
% Ergodic_Capacity_CDF.m
clear all, close all
SNR_dB=10; SNR_linear=10.^(SNR_dB/10.);
N_iter=50000; sq2=sqrt(0.5); grps = ['k:'; 'k-'];
for Icase=1:2
if Icase==1, nT=2; nR=2; % 2x2
else nT=4; nR=4; % 4x4
end
n=min(nT,nR); I = eye(n);
for iter=1:N_iter
H = sq2*(randn(nR,nT)+j*randn(nR,nT));
C(iter) = log2(real(det(I+SNR_linear/nT*H'*H)));
end
[PDF,Rate] = hist(C,50);
PDF = PDF/N_iter;
for i=1:50
CDF(Icase,i) = sum(PDF([1:i]))
end
plot(Rate,CDF(Icase,:),grps(Icase,:),'LineWidth',2); hold on
end
xlabel('Rate[bps/Hz]'); ylabel('CDF')
axis([1 18 0 1]); grid on; set(gca,'fontsize',10);
legend('{\it N_T}={\it N_R}=2','{\it N_T}={\it N_R}=4');
```

A – 6 Ergodic_Capacity vs_SNR Simulation :

```
% Ergodic_Capacity_vs_SNR.m
clear all, close all
SNR_dB=[0:5:20]; SNR_linear=10.^(SNR_dB/10);
N_iter=1000; sq2 = sqrt(0.5);
for Icase=1:5
if Icase==1, nT=1; nR=1; % 1x1
    elseif Icase==2, nT=1; nR=2; % 1x2
elseif Icase==3, nT=2; nR=1; % 2x1
elseif Icase==4, nT=2; nR=2; % 2x2
else nT=4; nR=4; % 4x4
end
n=min(nT,nR); I = eye(n);
C(Icase,:) = zeros(1,length(SNR_dB));
for iter=1:N_iter
H = sq2*(randn(nR,nT)+j*randn(nR,nT));
if nR>=nT, HH = H'*H; else HH = H*H'; end
for i=1:length(SNR_dB) % Random channel generation
C(Icase,i) = C(Icase,i)+log2(real(det(I+SNR_linear(i)/nT*HH)));
end
end
end
C = C/N_iter;
plot(SNR_dB,C(1,:),'k-o', SNR_dB,C(2,:),'k-', SNR_dB,C(3,:),'k-
s','linewidth',2);
hold on, plot(SNR_dB,C(4,:),'k->', SNR_dB,C(5,:),'k-
^','linewidth',2);
grid on
xlabel('SNR[dB]'); ylabel('bps/Hz');
```

A – 7 Multi user MIMO system Simulation :

```
% multi_user_MIMO.m
clear all; clf
mode=1; % Set 0/1 for channel inversion or regularized channel
inversion
N_frame=10; N_packet=200; % Number of frames/packet and Number of
packets
b=2; NT=4; N_user=20; N_act_user=4; I=eye(N_act_user,NT);
N_pbits = N_frame*NT*b; % Number of bits in a packet
N_tbits = N_pbits*N_packet; % Number of total bits
SNRdBs = [0:2:20]; sq2=sqrt(2);
for i_SNR=1:length(SNRdBs)
SNRdB=SNRdBs(i_SNR); N_ebits = 0; rand('seed',1); randn('seed',1);
sigma2 = NT*0.5*10^(-SNRdB/10); sigma = sqrt(sigma2);
for i_packet=1:N_packet
```

```
msg_bit = randint(1,N_pbits); % Bit generation
symbol = QPSK_mapper(msg_bit).'; x = reshape(symbol,NT,N_frame);
for i_user=1:N_user
H(i_user,:) = (randn(1,NT)+j*randn(1,NT))/sq2;
Channel_norm(i_user)=norm(H(i_user,:));
end
[Ch_norm,Index]=sort(Channel_norm,'descend');
H_used = H(Index(1:N_act_user),:);
temp_W = H_used'*inv(H_used*H_used'+(mode==1)*sigma2*I);
beta = sqrt(NT/trace(temp_W*temp_W')); % Eq.(12.17)
W = beta*temp_W; % Eq.(12.19)
Tx_signal = W*x; % Pre-equalized signal at Tx
%%%%%%%%%%%% Channel and Noise %%%%%%%%%%%%
Rx_signal = H_used*Tx_signal + ...
sigma*(randn(N_act_user,N_frame)+j*randn(N_act_user,N_frame));
%%%%%%%%%%%%% Receiver %%%%%%%%%%%%%%%%%%%%
x_hat = Rx_signal/beta; % Eq.(12.18)
symbol_hat = reshape(x_hat,NT*N_frame,1);
symbol_sliced = QPSK_slicer(symbol_hat);
demapped=QPSK_demapper(symbol_sliced);
N_ebits = N_ebits + sum(msg_bit~=demapped);
end
BER(i_SNR) = N_ebits/N_tbits;
end
semilogy(SNRdBs,BER,'b-o','linewidth',1.5), grid on
xlabel('SNR(dB)'),ylabel('BER')
```

A – 8 General MIMO OFDM System Simulation :

```
% Sim_MIMO_OFDM()
% mcc -m -B sgl Sim_MIMO_OFDM

clear all;
clc;
fprintf('Start! Please wait to inspect the results ...\n\n');

% Initial Processing:
% Define the slot structure
Nc = 512;
Ng = 32;
Ns = Nc + Ng;
Nu = Nc;
Num_Block = 1;
Fs = 20e6;
T = 1/Fs;
Tg = T * Ng;
Tu = T * Nc;
Ts = T * Ns;
DeltaF = 1/Tu;
B = DeltaF*Nu;
ModScheme = '16QAM';
M = 16;
Num_TxAnt = 2;
Num_RxAnt = 2;

Num_Bit_Frame = Num_Block * Nu * log2( M ) * Num_TxAnt;
Num_Sym_Frame = Num_Bit_Frame / log2(M);

Gen_Poly = [13 15];
Len_Constr = 4;
Len_Mem = 3;
k = 1;
n = 3;
Rate = k / n;
Trellis = poly2trellis( Len_Constr,Gen_Poly,Gen_Poly(1) );
Alg = 1;
Num_Iter_Decode = 8;
Num_InforBit = 2048;
Num_CodeBit = Num_InforBit / Rate + Len_Mem * 4;
Puncture_Pattern = [1 1; 1 0; 0 1];
[1; 1; 1];
[1 1; 1 0; 0 1];
[1 1 1 1; 1 0 0 0; 0 0 1 0];
[1 1 1 1 1 1; 1 0 0 0 0 0; 0 0 0 1 0 0];
[1 1 1 1 1 1 1 1 1 1; 1 0 0 0 1 0 0 1 0 0; 0 0 1 0 0 1 0 0 0 1];
Len_Pattern = prod( size( Puncture_Pattern ) );
Num_Reserved = sum( sum( Puncture_Pattern,1 ),2 );
Num_Punctured =  Len_Pattern - Num_Reserved;
```

```
Rate = size( Puncture_Pattern,2 ) / Num_Reserved;

Puncture_Pattern = reshape( Puncture_Pattern,1,Len_Pattern ) ;
if ( Num_Bit_Frame - fix( Num_Bit_Frame / Num_Reserved ) *
Num_Reserved ) == 0;
    Num_CodeBit_Frame = fix( Num_Bit_Frame / Num_Reserved ) *
Len_Pattern;
else
    for i = 1 : Len_Pattern
        if sum( Puncture_Pattern( 1:i ) ) == ( Num_Bit_Frame - fix(
Num_Bit_Frame / Num_Reserved ) * Num_Reserved )
            Num_CodeBit_Frame = fix( Num_Bit_Frame / Num_Reserved )
* Len_Pattern + i;
        end
    end
end
while mod( Num_CodeBit_Frame,Num_CodeBit ) ~= 0
    Num_InforBit = Num_InforBit - 1;    Num_CodeBit = Num_InforBit *
n / k + Len_Mem * 4;
end
Num_CodeBlock = Num_CodeBit_Frame / Num_CodeBit;
Num_InforBit_Frame = Num_InforBit * Num_CodeBlock;
Rate_Source = Num_InforBit_Frame / Num_Bit_Frame;
[Temp, Inner_Interlver] = sort( rand( 1,Num_InforBit ) );
Inner_Interlver = Inner_Interlver -1;
[Temp, Outer_Interlver] = sort( rand( 1,Num_Bit_Frame ) );

% Define the channel profile
% Path_Gain = [ 1 ];  Path_Delay = [0];    ChannelProfile = 'AWGN';
% Path_Gain = [0.9977    0.0680];     Path_Delay = [0 2];
ChannelProfile = 'ITU Pedestrian A';
% Path_Gain = [0.6369 0.5742 0.3623 0.2536 0.2595 0.0407];
Path_Delay = [0 1 4 6 11 18]; ChannelProfile = 'ITU Pedestrian B';
Path_Gain = [0.6964 0.6207 0.2471 0.2202 0.1238 0.0696];
Path_Delay = [0 1 2 3 4 5] + 1;
ChannelProfile = 'ITU Vehicular A';
%Path_Gain = [0.4544 0.4050 0.3610 0.3217 0.2867 0.2555 0.2277
0.2030 0.1809 0.1612 0.1437 0.1281...
%             0.1141 0.1017 0.0907 0.0808 0.0720 0.0642 0.0572
0.0510 0.0454 0.0405 0.0361 0.0322];
%Path_Delay = [0 7 14 22 29 37 45 52 59 67 75 82 90 97 104 112 119
127 135 142 150 157 164 172] + 1; ChannelProfile = 'Exponential
Decay Model';
Num_Path = length( Path_Gain );
Max_Delay = max( Path_Delay );
Fc = 3e9;
V = 3;
Fd = V * Fc / 3e8 * 1000 / 3600;
Phase = 2 * pi * rand( 1,Num_Path*Num_RxAnt*Num_TxAnt );
```

```
% Save simulation parametes
 FileName = 'Sim_MIMO_OFDM.dat';
% % % FileName = 'Sim_MIMO_OFDM.dat';
% Fid = fopen(FileName,'a+'); fprintf(Fid,'\n\n');fprintf(Fid,['%%
Created by ZZG from <' mfilename '.m> at ' datestr(now),'\n']);
% fprintf(Fid,'%% Num_Path = %d  vehicle speed = %d  carrier
frequency = %e  doppler frequency spread = %f  normalized doppler
shift = %f\n',Num_Path,V,Fc,Fd,Fd*Ts);
% fprintf(Fid,'%% system bandwidth = %e  number of subcarriers = %d
subcarrier spacing = %e\n',B,Nc,DeltaF);
% fprintf(Fid,'%% sampling duration = %e  symbol duration = %e
guard duration = %e \n',T,Ts,Tg);
% fprintf(Fid,'%% (%d, %d, %d)  Generator = %s  Num_InforBit = %d
Num_CodeBlock = %d  Num_InforBit_Frame = %d  Rate = %f
\n',n,k,Len_Constr,num2str( Gen_Poly ),Num_InforBit,Num_CodeBlock,N
um_InforBit_Frame,Rate);
% fprintf(Fid,'%% Num_Block = %d  ModScheme = %s  Num_TxAnt = %d
Num_RxAnt = %d\n',Num_Block,ModScheme,Num_TxAnt,Num_RxAnt);
% fprintf(Fid,'%% channel profile = %s\n',ChannelProfile );
% fprintf(Fid,'%% SNR  BER  FER \n\n'); fclose(Fid);

% [(0 : 1 : 3) (4 : 0.5 : 6)]
% Main loop
%SNR = [( 1:1:12 ) ];
SNR = [( 1:1:9 ) ];
MinSNR = min(SNR);
MaxSNR = max(SNR);
BER = [];
FER = [];
Num_Iter = 6;
Num_Frame = 10;
for Index = 1 : length( SNR )
    %    profile on -detail builtin
    StartPoint = 0;
    snr = SNR( Index )
    EbN0 = 10^( snr / 10 );
    Es = 1;
    N0  = Es * Num_RxAnt / ( EbN0 * Rate * Nu/Ns * log2(M) *
Num_TxAnt );
    Var = N0;
    ErrNum_Bit = zeros( 1,Num_Iter );
    ErrNum_Frame = zeros( 1,Num_Iter );
    ErrRate_Bit = zeros( 1,Num_Iter );
    ErrRate_Frame = zeros( 1,Num_Iter );

    for Frame = 1 : Num_Frame
        tic;
        % Transmitter
        Data_In = randint( 1,Num_InforBit_Frame );
        for i = 1 : Num_CodeBlock
```

```
            % Data_EnCode( (i-1)*Num_CodeBit+(1:Num_CodeBit) ) =
Enc_Conv( Data_In( (i-1)*Num_InforBit+(1:Num_InforBit)
),Trellis,InitState,Terminated );
            Data_EnCode( (i-1)*Num_CodeBit+(1:Num_CodeBit) ) =
Enc_Turbo_3gpp( Data_In( (i-1)*Num_InforBit+(1:Num_InforBit)
),Gen_Poly,Len_Constr,Inner_Interlver );
        end
        Data_EnCode = Puncture( Data_EnCode,Puncture_Pattern );
        Sym_In = reshape( Mapping( Data_EnCode( Outer_Interlver
),ModScheme ),Num_TxAnt,Nu*Num_Block ) / sqrt( Num_TxAnt );
        for TxAnt = 1 : Num_TxAnt
            Temp = reshape( Sym_In( TxAnt,: ),Nc,Num_Block );
            Temp = ifft( Temp,Nc,1 ) * sqrt( Nc );
            TransSig( TxAnt,: ) = reshape( [Temp( Nc-Ng+1:Nc,:
);Temp],1,Ns*Num_Block );
        end

        % Channel
        ChannelCoeff = MultiPathChannel( repmat(
Path_Gain,1,Num_RxAnt*Num_TxAnt ),Fd,Ts,Num_Block,StartPoint,Phase
);  StartPoint = StartPoint + Num_Block;
        % ChannelCoeff = diag( repmat(
Path_Gain,1,Num_RxAnt*Num_TxAnt ) ) * ( randn(
Num_Path*Num_RxAnt*Num_TxAnt,Num_Block ) + sqrt( -1 ) * randn(
Num_Path*Num_RxAnt*Num_TxAnt,Num_Block ) ) / sqrt( 2 );
        ChannelOut = zeros( Num_RxAnt,Ns*Num_Block+Max_Delay-1 );
        for RxAnt = 1 : Num_RxAnt
            for TxAnt = 1 : Num_TxAnt
                h( Path_Delay,1:Num_Block ) = ChannelCoeff( (RxAnt-
1)*Num_Path*Num_TxAnt + (TxAnt-1)*Num_Path + (1:Num_Path) ,:);
                H( RxAnt,TxAnt,: ) = reshape( fft( h,Nc,1
),1,Nc*Num_Block );
                for i = 1 : Num_Block
                    Temp = ChannelOut( RxAnt, (i-1)*Ns +
(1:Ns+Max_Delay-1) );
                    ChannelOut( RxAnt,(i-1)*Ns + (1:Ns+Max_Delay-1)
) = Temp + conv( h(:,i),TransSig( TxAnt,(i-1)*Ns + (1:Ns) ) );
                end
            end
        end
        RecSig = ChannelOut + sqrt( Var ) * ( randn( size(
ChannelOut ) ) + sqrt( -1 ) * randn( size( ChannelOut ) ) ) / sqrt(
2 );
        clear ChannelCoeff h ChannelOut;

        % Receiver
        RecSig = RecSig( :,1:Ns*Num_Block );
        for RxAnt = 1 : Num_RxAnt
            Temp = reshape( RecSig( RxAnt,: ),Ns,Num_Block );
            RecSig_Fre( RxAnt,: ) = reshape( fft( Temp( Ng+1:Ns,: )
```

```
) / sqrt( Nc ),1,Nc*Num_Block );
        end
        Y = RecSig_Fre;
        HH = H / sqrt( Num_TxAnt );
        clear RecSig RecSig_Fre H;

        Lu_Pri = zeros( 1,Num_InforBit_Frame );
        Lc_Pri = zeros( 1,Num_Bit_Frame );
        for Iter = 1 : Num_Iter
            Lc_Extr = MMSE_Equ(
Y,HH,Lc_Pri,Num_RxAnt,Num_TxAnt,ModScheme,Var,1);
            DeInterlv( Outer_Interlver ) = Lc_Extr;
            Lc_Pri = DePuncture(
DeInterlv,Num_CodeBit_Frame,Puncture_Pattern );
            for i = 1 : Num_CodeBlock
                % [Temp, Lc_Extr( (i-1)*Num_CodeBit+(1:Num_CodeBit)
)] = ......
                % LOG_MAP( zeros( 1,Num_InforBit + Len_Mem ),Lc_Pri(
(i-1)*Num_CodeBit+(1:Num_CodeBit) ),Trellis,1 );
                % Data_Out( (i-1)*Num_InforBit+(1:Num_InforBit) ) =
( sign( Temp( 1:Num_InforBit ) ) + 1 ) / 2;
                 [Data_Out( (i-1)*Num_InforBit+(1:Num_InforBit)
),Lu_Extr( (i-1)*Num_InforBit+(1:Num_InforBit) ),Lc_Extr( (i-
1)*Num_CodeBit+(1:Num_CodeBit) )] = ......
                     Dec_Turbo_3gpp( Lu_Pri( (i-
1)*Num_InforBit+(1:Num_InforBit) ),Lc_Pri( (i-
1)*Num_CodeBit+(1:Num_CodeBit) ),Trellis,Inner_Interlver,Alg,Num_It
er_Decode );
            end
            Lc_Extr = Puncture( Lc_Extr,Puncture_Pattern );
            Lc_Pri = Lc_Extr( Outer_Interlver );
            Error = sum( sum( sign( abs( Data_Out - Data_In ) ) ) );
            ErrNum_Bit( 1,Iter ) = ErrNum_Bit( 1,Iter ) +  Error;
            if ( Error ~= 0 )
                ErrNum_Frame( 1,Iter ) = ErrNum_Frame( 1,Iter ) +
1;
            end
            ErrRate_Bit( 1,Iter ) = ErrNum_Bit( 1,Iter ) / Frame /
Num_InforBit_Frame;
            ErrRate_Frame( 1,Iter ) = ErrNum_Frame( 1,Iter ) /
Frame;
        end
        Frame;
        ErrRate_Bit;
        ErrRate_Frame;
        toc;
    end
%     profile report
     BER = [BER; ErrRate_Bit];
     FER = [FER; ErrRate_Frame];
```

```
     Fid = fopen(FileName,'a+');
     fprintf(Fid,'%2.1f      %s      %s\n',snr,num2str(
ErrRate_Bit,'%1.10f ' ),num2str( ErrRate_Frame,'%1.10f ' ) );
     fclose(Fid);
end

figure(1)
semilogy(SNR,BER,SNR,FER,'linewidth',2)
axis([MinSNR,MaxSNR,10^-3,10^2]);
grid on;
xlabel('Eb/N0 (dB)');  ylabel('BER')
%
figure(2)
Throughput = (1 - FER) * Num_InforBit_Frame/Num_Block/Ts;
plot(SNR,Throughput/1e6,'linewidth',2)
axis([MinSNR,MaxSNR,0,110]);
grid on;
xlabel('Eb/N0 (dB)');  ylabel('Throughput (Mbps)')
figure(3)
subplot(2,1,1);
plot(SNR,BER,'linewidth',2)
xlabel('Eb/N0 (dB)');  ylabel('BER')
grid on
subplot(2,1,2);
plot(FER,BER,'linewidth',2)
xlabel('Fram Error Rate');  ylabel('BER')
grid on
```

Appendix B

User Define Functions

Special Functions used in simulation

```
function lic = actxlicense(progid)

if strcmpi(progid, 'air.airctrl.1')
lic = 'Copyright (c) 1996 ';
return;
end

function y_CFO=add_CFO(y,CFO,Nfft)
% add CFO (carrier frequency offset)
% y : Received signal
% CFO = IFO (integral CFO) + FFO (fractional CFO)
% Nfft = FFT size
nn=0:length(y)-1; y_CFO = y.*exp(j*2*pi*CFO*nn/Nfft);


function y=add_CP(x,Ng)
% Add CP (Cyclic Prefix) of length Ng
y = [x(:,end-Ng+1:end) x];

function xp=add_pilot(x,Nfft,Nps)
% CAZAC (Constant Amplitude Zero AutoCorrelation) sequence –> pilot
% Nps : Pilot spacing
if nargin <3, Nps=4;
end
Np=Nfft/Nps; % Number of pilots
xp=x; % Prepare an OFDM signal including pilot signal for
initialization
for k=1:Np
    xp((k-1)*Nps+1)=exp(j*pi*(k-1)^2/Np); % Eq.(7.17) for Pilot
boosting
end

function y_STO=add_STO(y, nSTO)
% add STO (symbol time offset)
% y : Received signal
% nSTO : Number of samples corresponding to STO
if nSTO>=0, y_STO=[y(nSTO+1:end) zeros(1,nSTO)]; % advance
else y_STO=[zeros(1,-nSTO) y(1:end+nSTO)]; % delay
end

function CFO_est=CFO_Classen(yp,Nfft,Ng,Nps)
% Frequency-domain CFO estimation using Classen method
% based on pilot tones in two consecutive OFDM symbols
if length(Nps)==1, Xp=add_pilot(zeros(1,Nfft),Nfft,Nps); % Pilot
signal
else Xp=Nps; % If Nps is an array, it must be a pilot sequence Xp
end
```

```
Nofdm=Nfft+Ng; kk=find(Xp~=0); Xp=Xp(kk); % Extract pilot tones
for i=1:2
yp_without_CP = remove_CP(yp(1+Nofdm*(i-1):Nofdm*i),Ng);
Yp(i,:) = fft(yp_without_CP,Nfft);
end
CFO_est = angle(Yp(2,kk).*Xp*(Yp(1,kk).*Xp)')/(2*pi); % Eq.(5.31)
CFO_est = CFO_est*Nfft/Nofdm;

function ber=ber_QAM(EbN0dB,M,AWGN_or_Rayleigh)
% Find analytical BER of M-ary QAM in AWGN or Rayleigh channel
% EbN0dB=EbN0dB: Energy per bit-to-noise power[dB] for AWGN channel
% =rdB : Average SNR(2*sigma Eb/N0)[dB] for Rayleigh channel
% M = Modulation order (Alphabet or Constellation size)
N=length(EbN0dB); sqM= sqrt(M);
a= 2*(1-power(sqM,-1))/log2(sqM); b= 6*log2(sqM)/(M-1);
if nargin<3, AWGN_or_Rayleigh='AWGN';
end
if lower(AWGN_or_Rayleigh(1))=='a'
ber = a*Q(sqrt(b*10.^(EbN0dB/10))); % ber=berawgn(EbN0dB,’QAM’,M)
Eq.(4.25)
else % diversity_order=1;
ber=berfading(EbN0dB,’QAM’,M,diversity_order)
rn=b*10.^(EbN0dB/10)/2; ber = 0.5*a*(1-sqrt(rn./(rn+1))); % Eq.
(4.26)
end

function hh=channel_coeff(NT,NR,N,Rtx,Rrx,type)
% correlated Rayleigh MIMO channel coefficient
% Inputs:
% NT : number of transmitters
% NR : number of receivers
% N : length of channel matrix
% Rtx : correlation vector/matrix of Tx
% e.g.) [1 0.5], [1 0.5;0.5 1]
% Rrx : correlation vector/matrix of Rx
% type : correlation type: 'complex' or 'field'
% Outputs:
% hh : NR x NT x N correlated channel
% uncorrelated Rayleigh fading channel, CN(1,0)
h=sqrt(1/2)*(randn(NT*NR,N)+j*randn(NT*NR,N));
if nargin, hh=h; return; end % Uncorrelated channel
if isvector(Rtx), Rtx=toeplitz(Rtx); end
if isvector(Rrx), Rrx=toeplitz(Rrx); end
% Narrow band correlation coefficient
if strcmp(type,'complex')
C =chol(kron(Rtx,Rrx))'; % Complex correlation
else
C =sqrtm(sqrt(kron(Rtx,Rrx))); % Power (field) correlation
end
% Apply correlation to channel matrix
```

```
hh=zeros(NR,NT,N);
for i=1:N, tmp=C*h(:,i); hh(:,:,i)=reshape(tmp,NR,NT);
end

function y = guard_interval(Ng,Nfft,NgType,ofdmSym)
if NgType==1, y=[ofdmSym(Nfft-Ng+1:Nfft) ofdmSym(1:Nfft)];
elseif NgType==2, y=[zeros(1,Ng) ofdmSym(1:Nfft)];
end

function [H_interpolated] = interpolate(H,pilot_loc,Nfft,method)
% Input: H = Channel estimate using pilot sequence
% pilot_loc = Location of pilot sequence
% Nfft = FFT size
% method = 'linear'/'spline'
% Output: H_interpolated = interpolated channel
if pilot_loc(1)>1
slope = (H(2)-H_est(1))/(pilot_loc(2)-pilot_loc(1));
H = [H(1)-slope*(pilot_loc(1)-1) H]; pilot_loc = [1 pilot_loc];
end
if pilot_loc(end) <Nfft
slope = (H(end)-H(end-1))/(pilot_loc(end)-pilot_loc(end-1));
H = [H H(end)+slope*(Nfft-pilot_loc(end))];
pilot_loc = [pilot_loc Nfft];
end
if lower(method(1))=='l', H_interpolated=interp1(pilot_loc,H,
[1:Nfft]);
else H_interpolated = interp1(pilot_loc,H,[1:Nfft],'spline');
end

function [H_LS] = LS_CE(Y,Xp,pilot_loc,Nfft,Nps,int_opt)
% LS channel estimation function
% Inputs:
% Y = Frequency-domain received signal
% Xp = Pilot signal
% pilot_loc = Pilot location
% N = FFT size
% Nps = Pilot spacing
% int_opt = 'linear' or 'spline'
% output:
% H_LS = LS Channel estimate
Np=Nfft/Nps; k=1:Np;
LS_est(k) = Y(pilot_loc(k))./Xp(k); % LS channel estimation
if lower(int_opt(1))=='l', method='linear'; else method='spline';
end
% Linear/Spline interpolation
H_LS = interpolate(LS_est,pilot_loc,Nfft,method);

function [H_MMSE] = MMSE_CE(Y,Xp,pilot_loc,Nfft,Nps,h,SNR)
% MMSE channel estimation function
% Inputs:
```

```
% Y = Frequency-domain received signal
% Xp = Pilot signal
% pilot_loc = Pilot location
% Nfft = FFT size
% Nps = Pilot spacing
% h = Channel impulse response
% SNR = Signal-to-Noise Ratio[dB]
% output:
% H_MMSE = MMSE channel estimate
snr = 10^(SNR*0.1); Np=Nfft/Nps; k=1:Np;
H_tilde = Y(1,pilot_loc(k))./Xp(k); % LS estimate Eq.(6.12) or (6.8)
k=0:length(h)-1; %k_ts = k*ts;
hh = h*h'; tmp = h.*conj(h).*k; %tmp = h.*conj(h).*k_ts;
r = sum(tmp)/hh; r2 = tmp*k.'/hh; %r2 = tmp*k_ts.’/hh;
tau_rms = sqrt(r2-r^2); % rms delay
df = 1/Nfft; %1/(ts*Nfft);
j2pi_tau_df = j*2*pi*tau_rms*df;
K1 = repmat([0:Nfft-1].',1,Np); K2 = repmat([0:Np-1],Nfft,1);
rf = 1./(1+j2pi_tau_df*Nps*(K1-K2)); % Eq.(6.17a)
K3 = repmat([0:Np-1].',1,Np); K4 = repmat([0:Np-1],Np,1);
rf2 = 1./(1+j2pi_tau_df*Nps*(K3-K4)); % Eq.(6.17a)
Rhp = rf;
Rpp = rf2 + eye(length(H_tilde),length(H_tilde))/snr; % Eq.(6.14)
H_MMSE = transpose(Rhp*inv(Rpp)*H_tilde.'); % MMSE estimat

function [mod_symbols,sym_table,M]=modulator(bitseq,b)
N_bits=length(bitseq);sq10=sqrt(10);
if b==1 % BPSK modulation
sym_table=exp(j*[0 -pi]); sym_table=sym_table([1 0]+1);
inp=bitseq; mod_symbols=sym_table(inp+1); M=2;
elseif b==2 % QPSK modulation
sym_table=exp(j*pi/4*[-3 3 1 -1]);sym_table=sym_table([0 1 3 2]+1);
inp=reshape(bitseq,b,N_bits/b);
mod_symbols=sym_table([2 1]*inp+1); M=4;
elseif b==3 % generates 8-PSK symbols
sym_table=exp(j*pi/4*[0:7]);
sym_table=sym_table([0 1 3 2 6 7 5 4]+1);
inp=reshape(bitseq,b,N_bits/b);
mod_symbols=sym_table([4 2 1]*inp+1); M=8;
elseif b==4 % 16-QAM modulation
m=0;
for k=-3:2:3 % Power normalization
for l=-3:2:3, m=m+1; sym_table(m)=(k+j*l)/sq10;
end
end
sym_table=sym_table([0 1 3 2 4 5 7 6 12 13 15 14 8 9 11 10]+1);
inp=reshape(bitseq,b,N_bits/b);
mod_symbols=sym_table([8 4 2 1]*inp+1); M=16; %16-ary symbol
sequence
else error('Unimplemented modulation');end
```

```
function PL=PL_free(fc,d,Gt,Gr)
% Free Space Path Loss Model
% Inputs: fc : Carrier frequency[Hz]
% d : Distance between base station and mobile station[m]
% Gt/Gr : Transmitter/Receiver gain
% Output: PL : Path loss[dB]
lamda = 3e8/fc; tmp = lamda./(4*pi*d);
if nargin>2, tmp = tmp*sqrt(Gt); end
if nargin>3, tmp = tmp*sqrt(Gr); end
PL = -20*log10(tmp);
function PL=PL_logdist_or_norm(fc,d,d0,n,sigma)
% Log-distance or Log-normal shadowing path loss model
% Inputs: fc : Carrier frequency[Hz]
% d : Distance between base station and mobile station[m]
% d0 : Reference distance[m]
% n : Path loss exponent
% sigma : Variance[dB]
lamda=3e8/fc; PL= -20*log10(lamda/(4*pi*d0))+10*n*log10(d/d0); %
Eq.(1.4)
if nargin>4, PL = PL + sigma*randn(size(d)); end

function plot_ber(file_name,Nbps)
EbN0dB=[0:1:30]; M=2^Nbps;
ber_AWGN = ber_QAM(EbN0dB,M,'AWGN');
ber_Rayleigh = ber_QAM(EbN0dB,M,'Rayleigh');
semilogy(EbN0dB,ber_AWGN,'r:'), hold on,
semilogy(EbN0dB,ber_Rayleigh,'r-'),hold on
a= load(file_name);
semilogy(a(:,1),a(:,2),'b*');grid on
legend('AWGN analytic','Rayleigh fading analytic', 'Simulation');
xlabel('EbN0[dB]'), ylabel('BER'); axis([a(1,1) a(end,1) 1e-5 1])

function y=Q(x)
% co-error function: 1/sqrt(2*pi) * int_x^inf exp(-t^2/2) dt. % Eq.
(4.27)
y=erfc(x/sqrt(2))/2;

function [QPSK_symbols] = QPSK_mapper(bitseq);
QPSK_table = [1 j -j -1]/sqrt(2);
for i=1:length(bitseq)/2
temp = bitseq(2*(i-1)+1)*2 +bitseq(2*(i-1)+2);
QPSK_symbols(i) =QPSK_table(temp+1);
end

function [bit_seq] = QPSK_demapper(x)
QPSK_table = [1 j -j -1]/sqrt(2);
Nx=length(x);
for i=1:Nx
```

```
x_temp(2*(i-1)+1:2*i)=dec2bin(find(QPSK_table==x(i))-1,2);
end
for i=1:Nx*2, bit_seq(i)=bin2dec(x_temp(i)); end
function H=Ray_model(L)
% Rayleigh channel model
% Input : L = Number of channel realizations
% Output: H = Channel vector
H = (randn(1,L)+j*randn(1,L))/sqrt(2);


Function y=remove_CP(x,Ng,Noff)
% Remove CP (Cyclic Prefix) of length Ng
if nargin<3, Noff=0;
end
y=x(:,Ng+1-Noff:end-Noff);
function y=remove_GI(Ng,Lsym,NgType,ofdmSym)
if Ng~=0
if NgType==1, y=ofdmSym(Ng+1:Lsym); % cyclic prefix
elseif NgType==2 % cyclic suffix
y=ofdmSym(1:Lsym-Ng)+[ofdmSym(Lsym-Ng+1:Lsym) zeros(1,Lsym-2*Ng)];
end
else y=ofdmSym;
end

function H=Ric_model(K_dB,L)
% Rician channel model
% Input : K_dB = K factor[dB]
% Output: H = Channel vector
K = 10^(K_dB/10);
H = sqrt(K/(K+1)) + sqrt(1/(K+1))*Ray_model(L);
```