- **Main**

```
% Number of users N for each BS (that could also not be assigned to any BS)
N=10;
% Cell's radius
R=3e3;
% Vector representing the number of antennas for each Base Station
n_ant_RX_vect=[1 2 4 8];
```

```
addpath('./functions')
```

➔ aggiunge la cartella "functions" al percorso del file (così posso richiamare le funzioni contenute in essa)

```
Inizializzazione_celle;
```

➔ richiama la funzione Inizializzazione_celle (descritta dopo)

```
Sensitivity=-100; %dBm
shadowing = 'uniforme'; % or set as 'non_uniforme'
std_db=6; % standard deviation of the shadowing
f_c=5e9; % frequency carrier
```

➔ setting di alcuni parametri:

- **Sensitivity**: minima richiesta in ricezione
- **Shadowing**: può essere "*uniforme*" o "*non_uniforme*"
  (definiti nella funzione shadowing nella cartella functions → /functions/**shadowing**)
- **std_db:** deviazione standard dello shadowing (es. 6dB)
- **f_c:** frequenza portante (es. 5e9 → 5 GHz)

```
Inizializzazione_Utenti;
Debug_interference;
```

➔ Richiama Inizializzazione_Utenti e Debug_interference

```
%% Run: Run_LTE_uplink


%% To plot the graph of the BER: BER_MRC_MAX_SEL
```

➔ Next steps…

- **Inizializzazione_celle**

```
%% Cells initialization

% The 3 considered cells are circular and with radius R

% Setting the intersite distance, i.e. the distance between cells
% It's required to be: d=R*sqrt(3)
% So we first set the radius "R", for example R=2e3 and then "d"
d=R*sqrt(3);

% In order to limit the area in which the "ue" (users) could be positioned,
% we draw a square with border: ds>=d+2*R (or ds=2d)
ds=d+2*R;

% Axes origin is in the middle of the square
O=[ds,ds-R/5];

% Square with border: ds
x = [ds/2 ds/2 1.5*ds 1.5*ds ds/2];
y = [ds/2 1.5*ds 1.5*ds ds/2 ds/2];
```

➔ NB. Distanze in **metri** (quindi es. R = 2e3 = 2000 m)

```
shadow={};
% Implementation of 1:4 squares inside the main square in which
% users (devices) could be positionated
for i=1:4
    switch i
        case 1
                % Squares related to Shadowing
                shadow(i).x = [ds/2 ds/2 ds ds ds/2];
                shadow(i).y = [ds 1.5*ds 1.5*ds ds ds];
        case 2
                shadow(i).x = [ds ds 1.5*ds 1.5*ds ds];
                shadow(i).y = [ds 1.5*ds 1.5*ds ds ds];
        case 3
                shadow(i).x = [ds ds 1.5*ds 1.5*ds ds];
                shadow(i).y = [ds/2 ds ds ds/2 ds/2];
        case 4
                shadow(i).x = [ds/2 ds/2 ds ds ds/2];
                shadow(i).y = [ds/2 ds ds ds/2 ds/2];
    end

end
```

➔ Definizione delle 4 aree (quadrate) separate, che serviranno in caso di *shadowing non uniforme* (per questo lo abbiamo chiamato **shadow{}**)

- In pratica abbiamo preso l'area totale (che era quadrata) e l'abbiamo divisa in 4 "sotto-quadrati", tutti con area uguale

```
% We calculate the centers of the 3 cells thanks to the apotema's formula
% of the trianle exploiting the dimension of the side L=d : a = L/(2*sqrt(3))
L=d;
a = L/(2*sqrt(3));
% Y-axis of cells related to Bs 2 and 3 ( on the left on the right)
Y2=O(2)-a;
Y3=Y2;
% X-axis of cell related to Bs 2
X2=O(1)-(L/2);
% X-axis of cell related to Bs 3
X3=O(1)+(L/2);
% X-axis of cell related to Bs 1
X1=O(1);
Y1=O(2)+(sqrt(L^2-(L/2)^2)-a);

% We draw the tringle which connects the Base Stations
tri = [X2 X1 X3 X2; Y2 Y1 Y3 Y2];
```

```
% We draw the circles corresponding to the 3 cells and the position
% of the 3 Base Stations
Bs={};

% Cella 1
Bs(1).pos = [X1 Y1];

% Left Cell, cell 2
Bs(2).pos = [X2 Y2];

% Rigth Cell, cell 3
Bs(3).pos = [X3,Y3];
```

```
figure
ax=gca;
hold (ax,'on')
%axis square
cella_superiore = circle(X1,Y1,R);
hold on
cella_sinistra = circle(X2,Y2,R);
cella_destra = circle(X3,Y3,R);
plot(x,y,'r', 'LineWidth',3);
for i=1:4
    plot(shadow(i).x,shadow(i).y);
end
plot(O(1),O(2));
plot(tri(1,:), tri(2,:));
txt=["Bs1","Bs2","Bs3"];
for i=1:3
    plot(Bs(i).pos(1),Bs(i).pos(2),'*','DisplayName',txt(1,i))
end

hold off
```

➔ Plot del triangolo

- **Inizializzazione_utenti**

```
%% Users initialization

% We must generate N UEs (User Equipment) which correspond to N IoT-devices
% associated with the 3 cells

% Nominal transmitting power Pt
Pt=10^(2.3-3); %23 dBm
% Propagation coefficient Beta (variable from 2.5 to 6)
B=2.5;

% In general, you can generate N random numbers in the interval (a,b)
% with the formula r = a + (b-a).*rand(N,1)
```

➔ **UE** che trasmette con potenza **Pt** (qui settata a 23 **dBm** scritti come $10^{\frac{23}{10}-3}$)

➔ Coefficiente di propagazione *beta* segnato come **B** e con un valore da impostare (oscillante fra 2.5 e 6 → qui settato come B=2.5)

➔ (frase alla fine : metodo di creazione di numeri casuali)

```
% Inizialize the generator of random integers
for q=1:3*N
    rng('shuffle')
    rx(q)=x(2)+(x(3)-x(2)).*rand(1,1);
    ry(q)=y(1)+(y(2)-y(1)).*rand(1,1);
end
```

➔ Inizializzazione “posizione” degli utenti (che gli verrà assegnata dopo)

```
i=1;
%"j" index "Bs", "i" index "ue"
for j=1:3
    flag=1;
    while (flag)
        ue(i).user_id=i;
        ue(i).NBULSubcarrierSpacing = '15kHz'; % 3.75kHz, 15kHz
        ue(i).NNCellID = j;                    % Narrowband cell identity
        ue(i).pos = [rx(i) ry(i)];
        ue(i).frameInterfering=[];
        ax=gca;
        figure(1)
        hold (ax,'on')
        scatter(ue(i).pos(1),ue(i).pos(2))
        i=i+1;
        if (i==N+1 || i==2*N+1 || i==3*N+1)
            flag=0;
        end
    end
end
```

➔ “j” indica la BS, è un indice che va da 1 a 3 (infatti ho 3 BSs)

➔ “i” è l’indice dell’user

➔ In questo ciclo *for* noi settiamo i diversi parametri di “ue”

➔ **“ue”** sarà una matrice che rappresenta tutti gli utenti (# utenti = # di righe)
e tutti i parametri (# parametri = # di colonne)

➔ “i” viene aggiornato ad ogni esecuzione del ciclo (i=i+1 alla fine)

➔ Flag=0 serve per uscire dall’affiliazione di 10 utenti per ogni BS → esco dal *while* → per questo faccio il controllo `if (i==N+1 || i==2*N+1 || i==3*N+1)` : ogni 10 utenti cambio BS fino a che non ho raggiunto 3*N+1 (nel nostro caso 31), al che mi fermo

```
for j=1:3
    Bs(j).Number_of_users=0;
    for i=1:numel(ue)
        ue(i).Pt=Pt;
        Bs(j).Number_of_users=0;
        % X represents the distances between the user ue(i)
        % which belongs to the first cell and to the BS
        X = [Bs(j).pos;ue(i).pos];
        Bs(j).d(i) = pdist(X,'euclidean');
        Bs(j).Pr_avg(i)=0;
        Bs(j).Pn_avg(i)=0;
        % For each "ue" we memorize the distance, the pathloss and the
        % received power
        Bs(j).PL(i)= PathLoss(Bs(j).d(i),f_c,B);
        % In the Friis formula, we need the G of ant Tx and Rx
        % We considered the antennas' gains Gtx=0dB (1) and
        % Grx=17 dB (50.12 dB) and NF(Noise Figure)=3dB
        Gtx=10^(0/10);
        Grx=10^(1.7);
        NF=10^.3;
```

```
        switch shadowing

            case 'uniforme'
                ue(i).sh=10^(shadowing_uniforme(std_db)/10);

            case 'non_uniforme'
                ue(i).sh=10^(shadowing_non_uniforme(ue(i).pos(1),ue(i).pos(2),std_db,shadow)/10);
        end

        Bs(j).Pr_nominale(i) = 10*log10((ue(i).Pt*Gtx*Grx*1e3)/(Bs(j).PL(i)*NF*ue(i).sh));
        Bs(j).Pr_avg(i)=0;
        Bs(j).Pn_avg(i)=0;
    end
end
```

- ➔ Questo è un unico ciclo *for* che viene fatto per ogni BaseStation (j che va da 1 a 3)
- ➔ Dentro, abbiamo un altro ciclo *for* che serve per considerare tutti gli user `for i=1:numel(ue)` (quindi sto considerando tutti gli user singolarmente per una BS alla volta)
- ➔ Setta la Potenza di trasmissione = Pt per l'user i-esimo (di conseguenza, tutti gli users avranno la stessa Pt) `ue(i).Pt=Pt;`
- ➔ Setta il numero di utenti iniziali alla BS j-esima = 0 per iniziare `Bs(j).Number_of_users=0;`
- ➔ Calcoliamo (e settiamo) parametri che ci serviranno per il calcolo del PathLoss, quindi distanze, guadagni (in Rx e Tx, settati da noi) e NoiseFigure (NF)
    - Consideriamo anche lo *shadowing* e distinguiamo i casi di Shadowing uniforme o non_uniforme (richiamando le apposite funzioni, scrivendo le equazioni)
- ➔ Una volta finito di calcolare/settare parametri per ogni utente, settiamo anche i valori in Rx della BS j-esima (che stiamo considerando)
- ➔ **`Bs(j).Pr_nominale(i) = 10*log10((ue(i).Pt*Gtx*Grx*1e3)/(Bs(j).PL(i)*NF*ue(i).sh));`**
  Questa è l'equazione di Friis, quindi la potenza ricevuta alla BS dall'utente i-esimo

• find_best_BS

```
%% Function which associates users to a Base Station

max=Sensitivity;
flag=0;
for n_bs=1:3 % Number of Base Stations (BS)
    if (Bs(n_bs).Pr_nominale(nue)>=max && Bs(n_bs).d(nue)<=R)
        max=Bs(n_bs).Pr_nominale(nue);
        flag=1;
        nbs_pref=n_bs;
        Bs(n_bs).Number_of_users=Bs(n_bs).Number_of_users + 1;
    end
end

if flag
    ue(nue).NNCellID=nbs_pref;
else
    ue(nue).NNCellID=0;
end

clear max nbs_pref flag
return
```

- ➔ "max" viene inizializzato come Sensitivity, infatti partiamo dalla minima potenza necessaria (definizione di Sensitivity)
- ➔ Ciclo *for* → per ogni BS
- ➔ Controllo la potenza ricevuta dalla BS dall'utente *nue*-simo; se questo utente si trova nel raggio della BS considerata & la potenza che viene ricevuta è maggiore del valore "max", allora aggiorno "max", metto un flag=1, setto come "BS preferita" dell'utente quella BS, e aggiorno il numero di utenti associati a quella BS
- ➔ Flag = 1 → serve per l'assegnamento: flag=1 vuol dire che ha trovato il massimo, quindi lo associo
- ➔ *Clear* variabili per quando viene richiamata la funzione
- ➔ Questa funzione verrà richiamata da ogni utente (in Debug_Interference)

• Debug_Interference

```
%% Debug Interference

Mod_type = 1;                % 1=QPSK , 2=16QAM , 3=64QAM
N_sim = 1;                   % Number of simulations
To=290;                      % Temperature [°Kelvin]
K = physconst('Boltzmann');  % Boltzmann's constant
TX_bw = 20e6;                % 1.4 , 3 , 5 , 10 , 15 , 20 [MHz]
SNR = linspace(0,15,6);      % Signal to Noise Ratio [dB] for the graph
SNR_comp = '3GPP' ;          % SNR computation: 'MEAS', '3GPP'
chan_type ='RAYL';           % 'AWGN','RAYL','EPA','EVA','ETU'
```

- ➔ N_sim = 1: faccio solo una simulazione per trovare le interferenze (ne faccio solo una perché poi sono sempre quelle… dopo farò un altro numero di simulazioni….ma vedremo dopo)
- ➔ Tx_bw: banda di trasmissione

➔ *Linspace:* crea un vettore che va da 0 a 15 (dB) e prende 6 punti equispaziati (sono i 6 punti che compongono il grafico del BER)

```
%Channel Estimation
CHE = 'LSf';                    % 'IDEAL','LS','MMSE'
LSi_red_factor=4;
NVLE = 'IDEAL';
Version_CHE=1;                  % 0, 1 (ideal L),
f_m = 5;                        % Max Doppler (5 Hz)
```

➔ Parametri Channel Estimation (non richiesto)

```
% Before doing the "real" simulation, it must be evaluated the BS to which
% the user will connect, i.e. the one with
% the higher received power from the "ue"

for nue=1:numel(ue)
    find_best_BS;
end
```

➔ Richiamo la funzione find_best_BS (spiegato prima)

```
% Finds all the interferences that each user could have during
% the transmission
for nue=1:numel(ue)
    ue(nue).number_users_interfering=0;
    ue(nue).users_interfering=[];
    ue(nue).flag_tx_int=0;
    k=1;
    if(ue(nue).NNCellID)
        fprintf('Utente %5.0f dovrà trasmettere alla BS %5.0f \n',nue,ue(nue).NNCellID);
        for user_int=1:numel(ue)
            if(ue(user_int).NNCellID)
                X = [ue(nue).pos;ue(user_int).pos];
                if(ue(user_int).NNCellID ~= ue(nue).NNCellID)
                    ue(nue).dist_wrto_user(user_int) = pdist(X,'euclidean');
                    fprintf("Utente %3.0f della Bs %3.0f interferirà con l'utente %3.0f\n",...
                        user_int,ue(user_int).NNCellID,nue);
                    ue(nue).number_users_interfering=ue(nue).number_users_interfering + 1;
                    ue(nue).users_interfering(k)=user_int;
                    k=k+1;
                end
            end
        end
    else
        fprintf('Utente %5.0f non può trasmettere \n',nue);
    end

end
```

➔ For per ogni utente (ue)

➔ *Ue(nue).flag_tx_int = 0* è un parametro che in teoria serve a capire se un utente stia attualmente interferendo trasmettendo (anche se poi l'abbiamo rimosso come parametro)

➔ `if(ue(nue).NNCellID)` → "se l'utente è associato a una cella (ovvero a una BS)"

➔ `ue(nue).users_interfering(k)=user_int;` → creo l'array con l'elenco degli users interferenti (dopo averne verificato prima se interferiscono)

➔ Questo pezzo di codice NON ci dice gli utenti che effettivamente stanno interferendo, ma solo quelli che *potenzialmente* potrebbero farlo (ifatti l'unico criterio qui è che l'interferente sia associato a una BS diversa da quella dell'user considerato)

```
indice=zeros(1,3);

% Random choice of interfering users
for nue=1:numel(ue)
    if ue(nue).number_users_interfering
        if round(rand())
            primo_utente_casuale=randperm(length(ue(nue).users_interfering),1);
            ue(nue).u_int(1)=ue(nue).users_interfering(primo_utente_casuale);
            if round(rand())
                indice_secondo_utente_casuale=randperm(length(ue(nue).users_interfering),1);
                secondo_utente_casuale=ue(nue).users_interfering(indice_secondo_utente_casuale);
                while ue(secondo_utente_casuale).NNCellID == ue(ue(nue).u_int(1)).NNCellID
                    indice_secondo_utente_casuale=randperm(length(ue(nue).users_interfering),1);
                    secondo_utente_casuale=ue(nue).users_interfering(indice_secondo_utente_casuale);
                end
                ue(nue).u_int(2)=secondo_utente_casuale;
                ue(nue).number_users_interfering=2;
                indice(3)=indice(3)+1;
            else
                ue(nue).number_users_interfering=1;
                indice(2)=indice(2)+1;
            end
        else
            ue(nue).number_users_interfering=0;
            indice(1)=indice(1)+1;
        end
    end
end
```

➔ `indice=zeros(1,3);` → è una matrice di zeri con 1 riga e 3 colonne (quindi un vettore)

➔ *For* ogni utente

- `if round(rand())` : lanci una moneta: se =0 vado all'else, altrimenti entro in questo if
- Se l'utente ha dei possibili utenti interferenti (controllato prima)
- `primo_utente_casuale=randperm(length(ue(nue).users_interfering),1);` → selezione dell'indice di un utente interferente casuale
- `ue(nue).u_int(1)=ue(nue).users_interfering(primo_utente_casuale);` → u_int è il vettore che ha al suo interno gli utenti interferenti: quindi qua sto facendo l'inizializzazione, prendendo casualmente l'indice di un'utente potenzialmente interferente

➔ Poi (dopo questa inizializzazione) entro in un altro "lancio di moneta":

- ```
  indice_secondo_utente_casuale=randperm(length(ue(nue).users_interfering),1);
  secondo_utente_casuale=ue(nue).users_interfering(indice_secondo_utente_casuale);
  while ue(secondo_utente_casuale).NNCellID == ue(ue(nue).u_int(1)).NNCellID
      indice_secondo_utente_casuale=randperm(length(ue(nue).users_interfering),1);
      secondo_utente_casuale=ue(nue).users_interfering(indice_secondo_utente_casuale);
  end
  ```
  → qua inizializziamo anche il secondo utente interferente (RICORDA: puoi avere al massimo 2 utenti interferenti): nel *while* controllo che non sia affiliato alla stessa BS (ovvero cambio finchè la BS è la stessa)
- Se esco dal *while*, vuol dire che la BS è diversa, quindi inizializzo anche il secondo utente interferente (e gli dico che sono 2 gli utenti interferenti)
  ```

➔ Con questo pezzo di codice, vediamo che stabiliamo se ogni utente ha 0, 1 o 2 utenti interferenti

```
fprintf("%s utenti con 0 utenti interferenti\n%s utenti con 1 utente interferente\n%s utenti con 2 utenti interferenti\n",...
    num2str(indice(1)),num2str(indice(2)),num2str(indice(3)));


n_ant_RX=8;

wtb = waitbar(0,'Attendere la creazione dei frame interferenti...');

for nue=1:numel(ue)
    for index=1:ue(nue).number_users_interfering  %For each user
       nue_int=ue(nue).u_int(index);  %In the vector of interfering users, i create an interf. frame
       if ue(nue_int).flag_tx_int %If that user has already tx interf. frame, skip interf. user
       else
           Generating_interfering_Frame;
           waitbar((nue)/(numel(ue)),wtb,['Utente interferente (', num2str(index),...
                   '/',num2str(ue(nue).number_users_interfering),'),',...
                       'Utente (',num2str(nue),'/',num2str(numel(ue)),') ',num2str(ceil((nue)/(numel(ue))*100)) '%']);
       end
    end
end

close(wtb);

% In order to see the most interfered users, could be useful a struct ordering
% wrto the number of interfering users
ue_int_order=nestedSortStruct(ue,'number_users_interfering',-1);
openvar('ue')
```

➔ Questo pezzo per ogni utente verifica la presenza di utenti interferenti
➔ (se l'utente ha già trasmesso interferendo, non fa nulla, altrimenti entra nell'*else* e richiama Generating_interfering_Frame
➔ Ultimo pezzo : ordina *ue* in base a chi ha più utenti interferenti

• Generating_interferfering_frame

➔ prima parte: commenti

➔ seconda parte:

```
%-------------------------------------------------------------------------%
%                          Simulator Parameters
%-------------------------------------------------------------------------%

% Parameters Modulator/Demodulator
```

- in questa parte vengono settati parametri (della Simulazione, Modulatore, Demodulatore….)

```
%-------------------------------------------------------------------------%
%                   Simulator configuration SCFDMA
%-------------------------------------------------------------------------%
SP = SimConfig(simP);

% Inizialization of counters e variables input/output
```

➔ inizializzazione variabili per SC-FDMA

```
%--------------------------------------------------------------------------%
%                                Simulatore
%--------------------------------------------------------------------------%

clear PRM_che_results
Data=DataGen(SP);

SP.N_frame=1;

clear PRM_che
SNRcorr=mean(SP.SNR);
Eb_tot_sim=0;
No_tot_sim=0;
```

➔ richiamo della funzione DataGen() che genera i dati
➔ inizializzazione altri dati per il simulatore

```
%--------------------------------------------------------------------------%
%                            Channel Preparation
%--------------------------------------------------------------------------%

durata=SP.N_frame*SP.T_frame;
OR=SP.N_SCFDMAsymb/SP.T_frame;

        for iant_rx=1:SP.n_ant_RX
            switch SP.chan_type
                case 'AWGN'
                    % .' -> transpose matrix
                    paths=ones(SP.N_frame*SP.N_SCFDMAsymb,1).';
                    tau=0;

                case 'RAYL'
                    [tau,paths]=ext_channel('RAYL',NaN,SP.f0,SP.f_m,durata,OR);

                % 'EPA','EVA','ETU'
                case 'EPA'
                    [tau,paths]=ext_channel('EPA',NaN,SP.f0,SP.f_m,durata,OR);

                case 'EVA'
                    [tau,paths]=ext_channel('EVA',NaN,SP.f0,SP.f_m,durata,OR);

                case 'ETU'
                    [tau,paths]=ext_channel('ETU',NaN,SP.f0,SP.f_m,durata,OR);
```

➔ In base al tipo di canale che è stato impostato nel Main, qua c'è la definizione dei diversi *channel type*

```
            tau=tau*1e-9; % [s]
            i_tau = ceil(SP.fs*tau*SP.f_upsamp)+1;

            for il=1:length(i_tau)
                h_chan(1:SP.N_frame*SP.N_SCFDMAsymb,i_tau(il),iant_rx) = paths(il,:).';
            end

            PRM_che.Lid = max(i_tau);
        end % iant_rx
```

➔ **h_chan(…) = paths(…).'** → creazione del canale (NB: h_chan è un vettore COLONNA, perché alla fine della riga c'è il " **.'** " che indica complesso coniugato, rendendo così il vettore riga come vettore colonna)

➔ **tau** serve per creare la matrice di h_chan (NB: non è il tau del MultiPath)

```
%-------------------------------------------------------------------%
%                        Modulator SCFDMA
%-------------------------------------------------------------------%

% Note
% The entire bandwidth is used for a single user

% Generation of data bits
DATA_TX(:,idx_symb) = Data(:,idx_symb);

% Mapping of bits in the costellation /
[SYMB_TX(:,idx_symb),reference_symbol,pilots_corr] = ConstMap(DATA_TX(:,idx_symb),SP,idx_frame,idx_symb);

% M-DFTreference symbols
x_TX = fft(SYMB_TX(:,idx_symb).');

if (reference_symbol)
    if (first_ref_symbol)
        SP.factor_ETX = round(sqrt(x_TX*x_TX'/length(x_TX))*100)/100;
        first_ref_symbol=0;
    end
    x_TX = SP.factor_ETX * pilots_corr.';
end

% Subcarrier mapping
x_TXmapped = [zeros(1,(zero_len)/2), x_TX, zeros(1,(zero_len)/2)];

% N-IDFT
y_TX = ifft(x_TXmapped);
```

```
% Addition of the cyclic prefix
y_TX_CP = AddCP(y_TX,idx_symb,SP);

% Upsampling to simulate continuos time
symbSCFDMA_TX = resample(y_TX_CP,SP.f_upsamp,1);
symbTX_len(idx_symb) = length(symbSCFDMA_TX);

% Vector of transmitted symbols
frame_TX(1,ifsgn+1:ifsgn+length(symbSCFDMA_TX)) = symbSCFDMA_TX;
```

➔ qua è tutto commentato dovresti capire

```
%-------------------------------------------------------------------%
%                              Channel
%-------------------------------------------------------------------%

% Channel index
ichn=ichn+1;

for iant_rx=1:SP.n_ant_RX
    symbSCFDMA_TX_chn = conv(symbSCFDMA_TX, squeeze(h_chan(ichn,:,iant_rx)));
    symbSCFDMA_TX_chn = symbSCFDMA_TX_chn(1:length(symbSCFDMA_TX));
    % Multipath creation which modify the signal both in
    % phase and amplitude
    rho=0.2+(.2-.01)*rand(1,length(symbSCFDMA_TX_chn));
    theta=0+(2*pi-0)*rand(1,length(symbSCFDMA_TX_chn));
    mpath_ext=(rho.*exp(-2*pi*1i*theta));

    if not(isempty(mpath_ext))
          symbSCFDMA_TX_chn(1:length(symbSCFDMA_TX)) = ...
            symbSCFDMA_TX_chn(1:length(symbSCFDMA_TX)) + mpath_ext(1:length(symbSCFDMA_TX));
    end

    H_che_tmp=fft(padarray(squeeze(h_chan(ichn,iant_rx)),SP.FFT_len*SP.f_upsamp));
    H_che_id(1:SP.Msymb,ichn,iant_rx)= ...
       H_che_tmp(PRM_che.zero_offset+1:PRM_che.zero_offset+SP.Msymb).';

    frame_TX_chn(iant_rx,ifsgn+1:ifsgn+length(symbSCFDMA_TX)) = symbSCFDMA_TX_chn;

end% iant_rx
```

```
    pattern_che=pattern_che+1;
    ifsgn = ifsgn + length(symbSCFDMA_TX);

end % idx_symb
```

➔ questa è la creazione dei canali

➔ **symbSCFDMA_TX_chn** deve indicare il simbolo trasmesso dopo il passaggio per il canale → per fare ciò, faccio la convoluzione [conv(…)] tra il symbSCFDMA_TX e il complesso coniugato (squeeze(…)) dell'h_chan(…)

➔ NB: definiamo qua anche il Multipath (rho e theta in quei range)

```
if not(isempty(mpath_ext))
      symbSCFDMA_TX_chn(1:length(symbSCFDMA_TX)) = ...
        symbSCFDMA_TX_chn(1:length(symbSCFDMA_TX)) + mpath_ext(1:length(symbSCFDMA_TX));
end
```

➔ → questo pezzo controlla se c'è il Multipath: se c'è il Multipath allora lo aggiunge al nostro segnale (+mpath_ext…..)

➔ Alla fine variabili che servono per la *channel estimation*

➔ (……)_che : si riferisce alla channel estimation

```
%-------------------------------------------------------------------------%
%                                  SNR
%-------------------------------------------------------------------------%

deltaIDFT = 10*log10(SP.Msymb/SP.FFT_len);
deltaCP = 10*log10(length(y_TX)/length(y_TX_CP));
snr = SNRcorr+deltaIDFT+deltaCP;

for iant_rx=1:SP.n_ant_RX

    switch SP.SNR_comp

        case 'MEAS'
            frame_RX(iant_rx,1:size(frame_TX_chn,2)) = ...
                awgn(frame_TX_chn(iant_rx,:),snr,'measured');
```

```
      case '3GPP'
          % SNR = S/N over a frame
          S=frame_TX_chn(iant_rx,:)*frame_TX_chn(iant_rx,:)'/size(frame_TX_chn,2);
          sigman=sqrt(S/(10^(snr/10)));
          frame_RX(iant_rx,1:size(frame_TX_chn,2)) = ...
              frame_TX_chn(iant_rx,1:size(frame_TX_chn,2)) + sqrt(2)/2*sigman*(randn(1,size(frame_TX_chn,2))+1i*randn(1,size(frame_TX_chn,2)));

      otherwise

          frame_RX(iant_rx,1:size(frame_TX_chn,2)) = frame_TX_chn(iant_rx,1:size(frame_TX_chn,2));
  end
end % iant_rx
```

➔ Inizializza parametri

➔ Switch: a noi interessa il 3GPP

- Adatta all'SNR il nostro frame (per il grafico: ad esempio se ho solo 6 punti considerati per fare il grafico, qua in base all'SNR riduce i campioni di rumore → più aumenti l'SNR, più devono diminuire gli errori/il BER)

```
% Analysis of interfering frames for each used
% type (number) of antennas
Frame_Interfering(nue_int).Ant1=frame_RX(1,:);
Frame_Interfering(nue_int).Ant2=frame_RX(1:2,:);
Frame_Interfering(nue_int).Ant4=frame_RX(1:4,:);
Frame_Interfering(nue_int).Ant8=frame_RX(1:8,:);

ue(nue_int).flag_tx_int=1;
```

➔ Alla fine avrò dei *frame interferenti* per ogni utente → dopo averli trasmessi per ogni utente e averli fatti passare per un canale, li salvo

➔ Qua definiamo i frame interferenti: in **LTE_uplink_v01** verranno pesati e sommati in ricezione al frame che noi vogliamo ricevere (seguendo la formula nella presentazione, slide 3)

➔ Flag alla fine: indica che abbiamo già fatto tutto (serve per un controllo di Debug_interference)

• Run_LTE_uplink

LTE_uplink_v01 : fa la simulazione insieme a Run_LTE (in serie, uno dopo l'altro)

```
%% Section for Transmissions, to change the settings see "Debug_interference.m"

N_sim = 1;      % Number of simulations
M_frame = 1;    % Number of frames

% Decomment if run only Run_LTE_uplink and viceversa
addpath('./functions')

% Simulate the real transmission, based on data of "Debug_interference.m"

BER_PER_GRAFICO={};
BER_PER_GRAFICO_CON_INTERFERENZA={};
```

➔ Definizione e inizializzazione parametri
➔ (quelli sotto ci serviranno per fare i grafici)

```
  % Initialization of struct BER_PER_GRAFICO_CON_INTERFERENZA
for i=1:3
    for j=1:numel(n_ant_RX_vect)
        n_ant_RX=n_ant_RX_vect(j);
        switch n_ant_RX
            case 1
            BER_PER_GRAFICO_CON_INTERFERENZA(i).ber1=0;
            case 2
            BER_PER_GRAFICO_CON_INTERFERENZA(i).ber2=0;
            case 4
            BER_PER_GRAFICO_CON_INTERFERENZA(i).ber4=0;
            case 8
            BER_PER_GRAFICO_CON_INTERFERENZA(i).ber8=0;
        end
    end
end
```

➔ Inizializzazione a zero per i grafici con i diversi numeri di antenne (1,2,4,8)

```
% Combiner : chosen from input
combiner = input("Choose the combiner: 1 for MRC, 0 for MS\n");

if combiner
    combiner='MRC';
else
    combiner='MAX_SEL';
end
```

➔ Vedo il combiner che viene passato in input

```
% Start of the simulation
for i=1:numel(n_ant_RX_vect)
    n_ant_RX=n_ant_RX_vect(i);
    for nue=1:numel(ue)
        if ue(nue).NNCellID
            fprintf('Utente %3.0f sta trasmettendo alla BS %5.0f \n',nue,ue(nue).NNCellID);
            EbTot=0;
            NoiseTot=0;
            indice_ebtot=1;

            % Choice of the interference: 1 exists, 0 not exist
            INTERFERENCE=1;
            LTE_uplink_v01;
```

➔ Qua inizio la vera e propria simulazione : dentro al *for* verrà fatta per ogni numero di antenne e per ogni utente

➔ Se associati a una BS, allora continuo e richiamo la funzione LTE_uplink_v01 (vediamo dopo)

```
            test=RES.BER;
            % Graph user per user of the BER
            switch n_ant_RX
                case 1
                    BER_PER_GRAFICO(nue).ber1_ant=mean(test,2);
                case 2
                    BER_PER_GRAFICO(nue).ber2_ant=mean(test,2);
                case 4
                    BER_PER_GRAFICO(nue).ber4_ant=mean(test,2);
                case 8
                    BER_PER_GRAFICO(nue).ber8_ant=mean(test,2);
            end
```

➔ Sempre dentro lo stesso *for*, quindi per ogni utente, viene salvato in "test" l'attuale valore del BER e poi in base a quale sia attualmente il numero di antenne in RX viene **mediato (**mean()**)** e salvato in **BER_PER_GRAFICO(…).ber…_ant**

```
        % Struct of graphs depending on the number of interfering users
        for users_interfering=1:3
            if users_interfering==numel(ue(nue).u_int)+1
                switch n_ant_RX
                    case 1
                    BER_PER_GRAFICO_CON_INTERFERENZA(users_interfering).ber1=BER_PER_GRAFICO_CON_INTERFERENZA(users_interfering).ber1...
                        +BER_PER_GRAFICO(nue).ber1_ant;
                    case 2
                    BER_PER_GRAFICO_CON_INTERFERENZA(users_interfering).ber2=BER_PER_GRAFICO_CON_INTERFERENZA(users_interfering).ber2...
                        +BER_PER_GRAFICO(nue).ber2_ant;
                    case 4
                    BER_PER_GRAFICO_CON_INTERFERENZA(users_interfering).ber4=BER_PER_GRAFICO_CON_INTERFERENZA(users_interfering).ber4...
                        +BER_PER_GRAFICO(nue).ber4_ant;
                    case 8
                    BER_PER_GRAFICO_CON_INTERFERENZA(users_interfering).ber8=BER_PER_GRAFICO_CON_INTERFERENZA(users_interfering).ber8...
                        +BER_PER_GRAFICO(nue).ber8_ant;
                end
            end
        end
    end
end
```

➔ Devo valutare quanti utenti interferenti ho (sempre considerando rispetto allo stesso utente di prima)

➔ In questo modo, salvo il BER raggruppando non solo per il numero di antenne in RX, ma anche per il numero di utenti che interferisco

➔ (servirà per fare i grafici)

➔ Qui finisce il *for* "lungo", quindi alla fine ha valutato praticamente finito la simulazione

```
for i=1:numel(indice)
    numero_utenti_interferenti=indice(i);
    for j=1:numel(n_ant_RX_vect)
        n_ant_RX=n_ant_RX_vect(j);
        switch n_ant_RX
            case 1
            BER_PER_GRAFICO_CON_INTERFERENZA(i).ber1=BER_PER_GRAFICO_CON_INTERFERENZA(i).ber1/numero_utenti_interferenti;
            case 2
            BER_PER_GRAFICO_CON_INTERFERENZA(i).ber2=BER_PER_GRAFICO_CON_INTERFERENZA(i).ber2/numero_utenti_interferenti;
            case 4
            BER_PER_GRAFICO_CON_INTERFERENZA(i).ber4=BER_PER_GRAFICO_CON_INTERFERENZA(i).ber4/numero_utenti_interferenti;
            case 8
            BER_PER_GRAFICO_CON_INTERFERENZA(i).ber8=BER_PER_GRAFICO_CON_INTERFERENZA(i).ber8/numero_utenti_interferenti;
        end
    end
end

openvar ('BER_PER_GRAFICO_CON_INTERFERENZA');
```

➔ Questo serve per fare la media in base a quanti utenti con lo stesso numero di interferenti ho, e divido per quel numero
→ es. prima ho sommato 10 utenti con 1 interferente, adesso divido il valore ottenuto per 10

➔ Alla fine mi apre la variabile (‘BER_PER_GRAFICO_CON_INTERFERENZA’)

• LTE_uplink_v01

➔ Fino a riga 399 è uguale a Generating_Frame_Interference

```
400 -        frame_tot_interfering=zeros(size(frame_RX,1),size(frame_RX,2));
```

➔ Inizializzazione frame_tot_interfiring

➔ Bypassa Channel Estimation part (stima del canale → utilizzo dei *pilot symbols*) (fino a riga 517)

```
%% Interference set in the Main
if(INTERFERENCE)
    if ue(nue).number_users_interfering
        for k=1:numel(ue(nue).u_int)
            idx_int=ue(nue).u_int(k);
            switch n_ant_RX
                case 1
                    frm_int=Frame_Interfering(idx_int).Ant1;
                case 2
                    frm_int=Frame_Interfering(idx_int).Ant2;
                case 4
                    frm_int=Frame_Interfering(idx_int).Ant4;
                case 8
                    frm_int=Frame_Interfering(idx_int).Ant8;
            end

            weight_interference=sqrt(dBm_to_watt(Bs(ue(nue).NNCellID).Pr_nominale(idx_int)))/...
                        sqrt(dBm_to_watt(Bs(ue(nue).NNCellID).Pr_nominale(nue)));

            frame_tot_interfering = frame_tot_interfering +...
                (frm_int).*weight_interference;
        end
    end
end
```

➔ INTERFERENCE era già settato a 1, quindi per noi entra sempre (lo puoi anche cambiare)
➔ Se esiste un utente interferente
➔ Per k che va da 1 fino al numero di utenti interferenti (quindi 1 o 2, perché se sono qua dentro non possono essere 0)
➔ Vedo qual è l'indice dell'utente che interferisce, prendo il suo frame, lo salvo in frm_int, calcolo il peso (come diciamo nella presentazione), poi lo sommo (nel caso siano 2 utenti interferenti)

```
% Addition of the interference
frame_RX=frame_RX+frame_tot_interfering;

% For MAX SELECTION COMBINER EQUALIZATION is needed, so before
% combining we consider the best channel gains, i.e. those with
% with the highest power
Channel_Gains_sum=0;
Gains=0;
```

➔ Aggiungo l'interferenza totale sa frame in ricezione
➔ Secondo pezzo: dato che nel caso MS ho bisogno di un equalizzatore, inizio a definire un paio di variabili (che verranno riconsiderate in seguito)

```
%% ------------------------------------------------------------------%
%                      Demodulator SCFDMA                           %
%%-----------------------------------------------------------------%

for idx_symb = 1:SP.N_SCFDMAsymb

    % Received symbol
    if idx_symb == 1
        start_sample = 1;
        end_sample = start_sample+symbTX_len(idx_symb)-1;
    else
        start_sample = 1 + sum(symbTX_len(1:idx_symb-1));
        end_sample = start_sample+symbTX_len(idx_symb)-1;
    end

    x_RX = zeros(1,SP.Msymb);
    x_RX_nn = zeros(1,SP.Msymb);
    N_RX = zeros(1,SP.Msymb);
    Signal_antenna={};
    power_signal=0;
```

- ➔ Index_symbol (ovvero idx_symb) va da 1 a 14 (SP.N_SCFDMAsymb)
- ➔ Poi a seconda del simbolo, identifica i pezzi da demappare (siamo in Rx)
- ➔ Poi inizializza segnale e rumore
    - x_RX: segnale
    - x_RX_nn: segnale senza rumore
    - N_RX: rumore

```
for iant_rx=1:SP.n_ant_RX
    symbSCFDMA_RX = frame_RX(iant_rx,start_sample:end_sample);
    symbSCFDMA_RX_nn = frame_TX_chn(iant_rx,start_sample:end_sample);

    % Downsampling to simulate the sampling
    y_RX_CP = symbSCFDMA_RX(1:SP.f_upsamp:end);
    y_RX_CP_nn = symbSCFDMA_RX_nn(1:SP.f_upsamp:end);

    % Removal of the cyclic prefix
    y_RX = RemCP(y_RX_CP,idx_symb,SP);
    y_RX_nn = RemCP(y_RX_CP_nn,idx_symb,SP);

    % N-DFT
    x_RXmapped = fft(y_RX);
    x_RXmapped_nn = fft(y_RX_nn);

    % Subcarrier demapping
    x_RX_ant = x_RXmapped(zero_len/2+1:end-(zero_len/2));
    x_RX_ant_nn = x_RXmapped_nn(zero_len/2+1:end-(zero_len/2));

    Signal_antenna(iant_rx).sgnl=x_RX_ant;
    Signal_antenna(iant_rx).sgnl_nn=x_RX_ant_nn;
```

(questo *for* non finisce qua, ma più sotto)

➔ Definisco il range dove campionare i campioni da cui riottengo il mio simbolo (grazie a start_sample e end_sample che ho definito prima)
➔ NB : nel caso "massimo" (quindi quando arriviamo a considerare le 8 antenne), frame_RX diventa una matrice di 8 righe, con numero di colonne che è di 14x1200 colonne → questo perché ho 1200 campioni per ognuno dei 14 simboli e in base a questi 1200 campioni io devo ricostruire il mio simbolo corrispondente (grazie a start_sample e end_sample che ho definito prima)
➔ Faccio la stessa cosa anche per il senza_rumore (……_nn)
➔ (il resto è commentato)

```
%% ------------------------------------------------------------%
%                          Combiner                           %
%%-----------------------------------------------------------%

 switch combiner

     case 'MRC'

         % Channel Estimation for each antenna
         % Hcorr_2 has to be conjugated (apex ')
         H_corr2 = squeeze(H_che(:,idx_symb,iant_rx))';

         %Hcorr_2 has been conjugated before
         x_RX = x_RX + x_RX_ant.*(H_corr2);

         % As the definition of MRC combiner, the demapped
         % signal can be combined with channel estimation,
         % each signal is by the gains of the corrensponding
         % antenna
         x_RX_nn = x_RX_nn + x_RX_ant_nn.*(H_corr2);
         N_RX = N_RX + abs(H_corr2).^2;
```

➔ Sempre dentro il ciclo *for* di prima (`for idx_symb = 1:SP.N_SCFDMAsymb`)
➔ In base al combiner scelto, facciamo cose diverse
➔ Questo è MRC
➔ Quindi faccio la channel estimation (valore complesso coniugato) (qui è H_corr2)
➔ Dato che siamo dento il ciclo *for*, l'espressione di x_RX è come se fosse una sommatoria (infatti il valore viene via via aggiornato)

```
         case 'MAX_SEL'

             H_corr2 = squeeze(H_che(:,idx_symb,iant_rx))';
             if power_signal<sqrt(sum(abs(H_corr2).^2))
                 power_signal=sqrt(sum(abs(H_corr2).^2));
                 signal=x_RX_ant;
                 signal_nn=x_RX_ant_nn;
                 gains=H_corr2;
             end
     end
 end % iant_rx
```

(qua c'è anche la fine del *for* di prima)
➔ Questo è il caso MS: qui sceglie il channel gain con la potenza maggiore (NB: nel caso MS **dopo** troveremo l'equalizzatore)

```
% Equalization needed for max_sel, but in this
% case we need to use the 'best' channel gains,
% i.e. the most powerful signal
switch combiner
    case 'MAX_SEL'
        x_RX = signal.*gains;
        x_RX_nn = signal_nn.*gains;
        N_RX = abs(gains).^2;
end
```

➔ Qui possiamo vedere l'**equalizzatore** nel caso di MS combiner (formula come nella presentazione)

```
x_RX = x_RX ./ N_RX;
x_RX_nn = x_RX_nn ./ N_RX;
```

➔ In entrambi i casi MRC e MS, devo fare la combinazione lineare (ovvero, dividere per il modulo quadro dei channel gains, ovvero N_RX) (NB: N_RX viene definito sia per MRC (vedi sopra) sia per l'MS (nell'equalizzatore))

```
% M-IDFT
SYMB_RX(:,idx_symb) = ifft(x_RX);
SYMB_RX_nn(:,idx_symb) = ifft(x_RX_nn);

% Decisore
SYMB_stima(:,idx_symb) = Decisor(SYMB_RX(:,idx_symb).',SP);

% Demapping of the symbol's costellation
[DATA_RX(:,idx_symb), TYPE_DATA, TYPE_SYM] =...
    ConstDemap(SYMB_stima(:,idx_symb).',SP,idx_frame,idx_symb);
```

➔ Commentato

```
%-------------------------------------------------------------------%
%           Measure and evaluation of wrong symbols/bit
%-------------------------------------------------------------------%
```

➔ Qua misura e valuta i simboli errati (la maggior parte era già stata fatta da chi ha programmato)

.
.
.
.

```
if (Nbit>0)
    i_sy = find(TYPE_SYM==1);
    Nsy = length(i_sy); % round(Nbit/SP.mod_bitsymb);
    [Nsy_err, S_ratio, Sy_err] = symerr(SYMB_TX(i_sy,idx_symb),SYMB_stima(i_sy,idx_symb));
    [Nbit_err, B_ratio, Bit_err] = biterr(DATA_TX(i_bit,idx_symb),DATA_RX(i_bit,idx_symb));

    % Updating the "error counters"
    RES.SEcount(isnr,isim) = RES.SEcount(isnr,isim) + Nsy_err;
    RES.BEcount(isnr,isim) = RES.BEcount(isnr,isim) + Nbit_err;
    RES.Scount(isnr,isim) = RES.Scount(isnr,isim) + Nsy;
    RES.Bcount(isnr,isim) = RES.Bcount(isnr,isim) + Nbit;
end
```

➔ Qui viene calcolato il BER (funzione matlab *biterr*)

➔ Le 4 righe finali aggiornano i bit errati e i simboli errati (così possiamo calcolare il BER)

.
.
.
.
.
.

➔ Il resto fino alla fine del file non ci interessa (l'ha fatto il prof, serve per salvare i dati)