12/6/2020

Television Broadcasting Systems

Practice

Darío Pérez-Calderón Rodríguez
BONCH-BRUEVICH SAINT - PETERSBURG STATE UNIVERSITY OF TELECOMMUNICATIONS

СПб ГУТ)))

Contents

1. Target

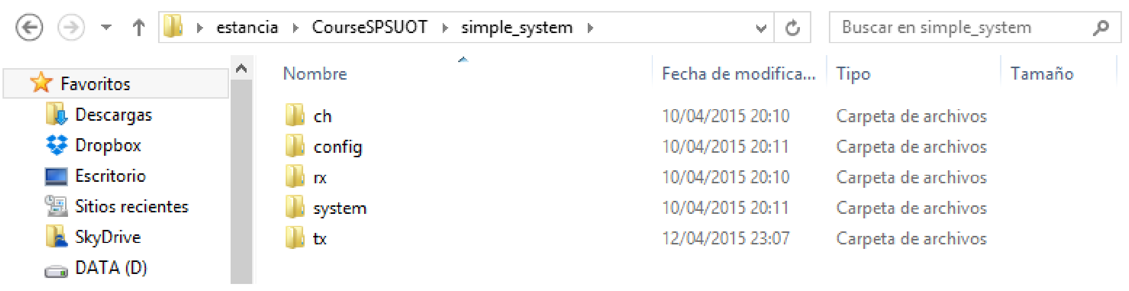
In this guide it will be carefully explained all the steps to follow in order to complete a very simple model of an OFDM digital broadcasting system by using MATLAB.

2. Introduction

The teacher has provided the studentswith a set of M-files with a MATLAB code that emulates the main parts of a basic OFDM transmitter (random data generation, bit to constellation mapping, reverse Fourier transform). With the knowledge obtained during the course the students must end the transmission chain (add the Cyclic Prefix to have a complete OFDM signal in the base band) and perform the reception process (remove the Cyclic Prefix, apply the Fourier transform, and perform the demapping process -QAM symbols to bits) and finally add AWGN noise to see how it affects the system.

2.1. File organization

As a general way to work is important to have a well-organized file structure. That is why the code provided by the teacher is divided in different folders, corresponding to the different conceptual and physical parts in a system. This is, the transmitter (tx folder), the receiver (rx folder), system configuration (config folder), propagation scenario (ch folder), and finally it is necessary a wrapper for all the system (system folder). Inside of these folder it will be allocated the files with code related to the topic its name makes reference to. In Fig. 1 the folder structure can be observed.



Nombre	Fecha de modifica...	Tipo	Tamaño
ch	10/04/2015 20:10	Carpeta de archivos	
config	10/04/2015 20:11	Carpeta de archivos	
rx	10/04/2015 20:10	Carpeta de archivos	
system	10/04/2015 20:11	Carpeta de archivos	
tx	12/04/2015 23:07	Carpeta de archivos	

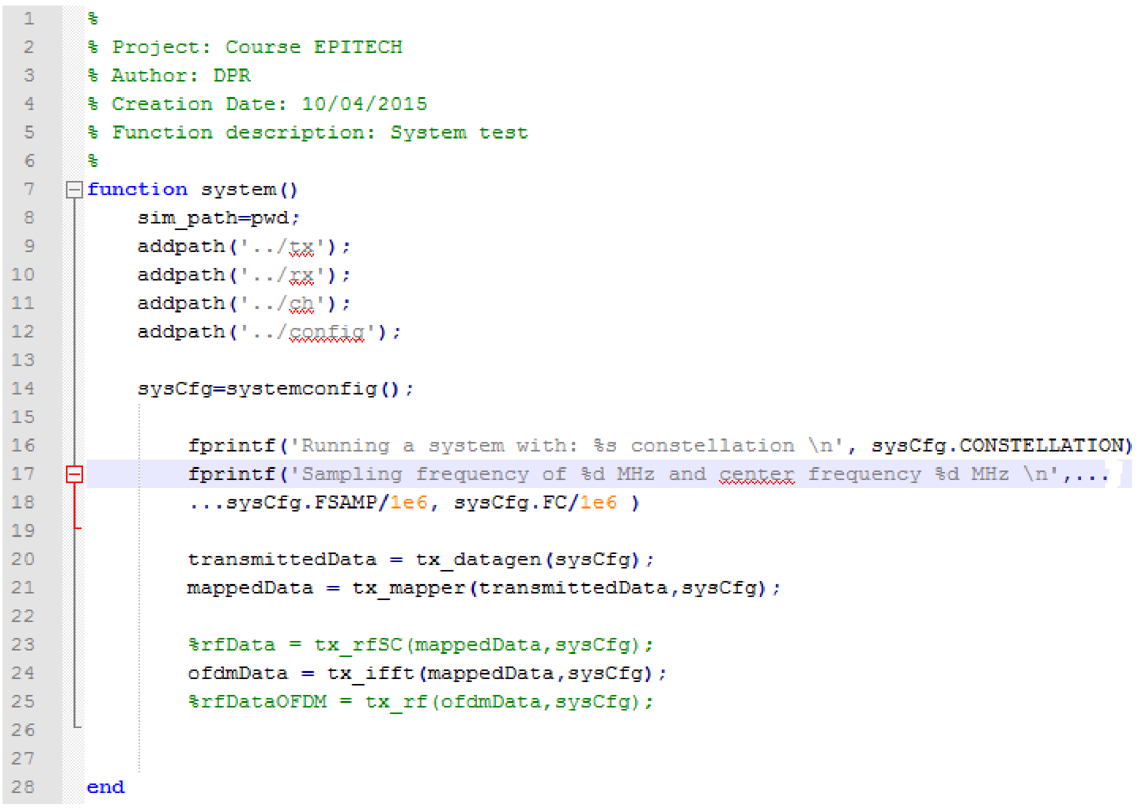
Fig. 1. Sub-folders in simple_system

As a starting point it will be found in the tx folder the m files tx_datagen.m, tx_mapper.m, and tx_ifft.m. The config folder will contain systemconfig.m. The system folder will contain the system_run.m file. The rest of the folders will be empty.

2.2. Base code explanation

To better understand how everything works and to be able to add the remaining blocks to the system a brief overview of all the provided code will be carried out, as it was during the course.

The first point to review is the file that works as a wrapper for the rest of the code, this is system_run.m.

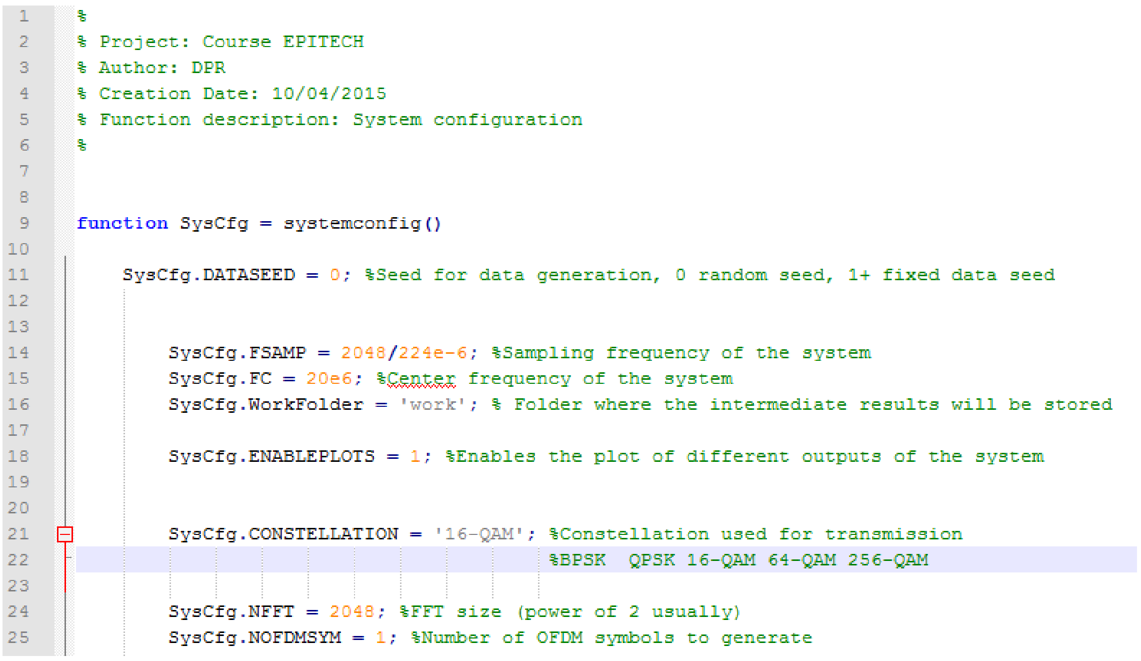


```
1   %
2   % Project: Course EPITECH
3   % Author: DPR
4   % Creation Date: 10/04/2015
5   % Function description: System test
6   %
7   function system()
8       sim_path=pwd;
9       addpath('../tx');
10      addpath('../rx');
11      addpath('../ch');
12      addpath('../config');
13
14      sysCfg=systemconfig();
15
16          fprintf('Running a system with: %s constellation \n', sysCfg.CONSTELLATION)
17          fprintf('Sampling frequency of %d MHz and center frequency %d MHz \n',...
18          ...sysCfg.FSAMP/1e6, sysCfg.FC/1e6 )
19
20          transmittedData = tx_datagen(sysCfg);
21          mappedData = tx_mapper(transmittedData,sysCfg);
22
23          %rfData = tx_rfSC(mappedData,sysCfg);
24          ofdmData = tx_ifft(mappedData,sysCfg);
25          %rfDataOFDM = tx_rf(ofdmData,sysCfg);
26
27
28  end
```

This file contains the code of a function that receives no input arguments and returns nothing. Lines 9 to 12 act as import in Java language, they tell the MATLAB interpreter where to look for the functions that are going to be used along the code.

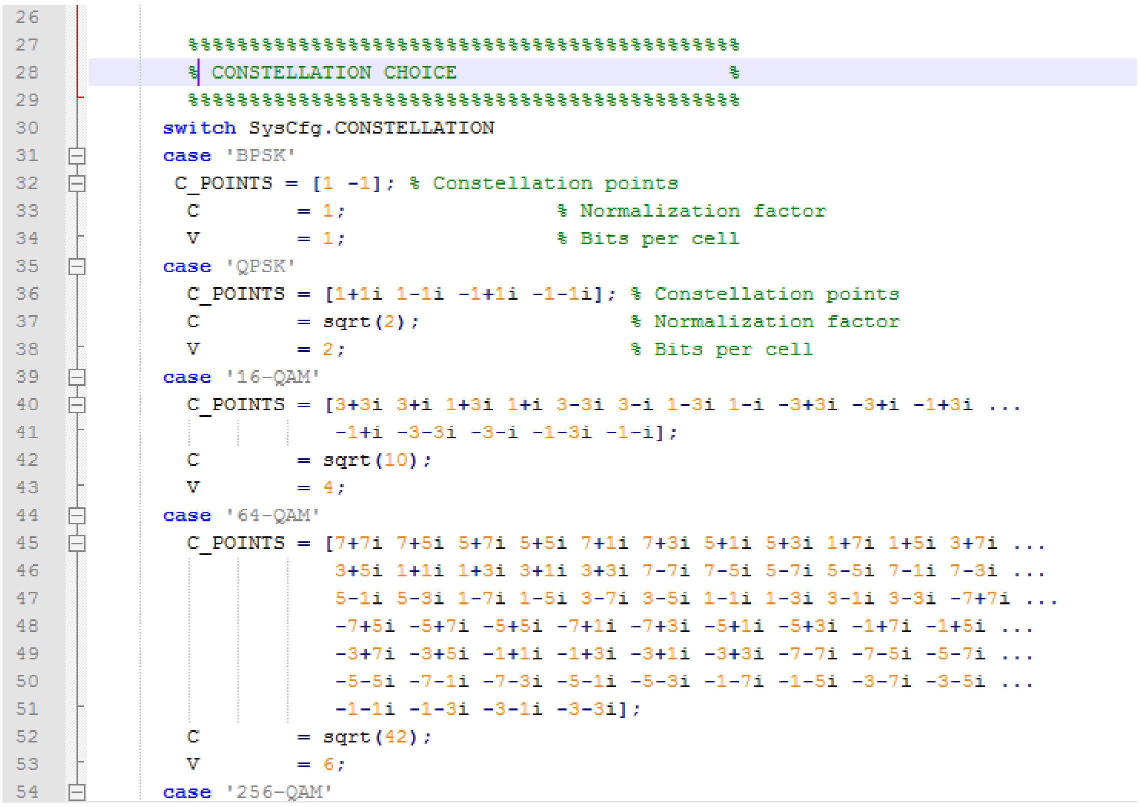
After these lines the code only makes calls to the functions in the corresponding order to emulate the transmission chain and some plots. First loads the system configuration in sysCfg variable (line 14). Lines 16 and 17 print in the shell some data from the configuration, to check which one we are using. The rest are function calls to the functions implementing the different parts of the system, in this case the random data generation (line 20), mapping bits to QAM symbols (line 21) and applying the reverse Fourier transform to get the temporal signal (line 24). Note that lines 23 and 25 must be commented (they were used in the classes to show how the spectrum of the OFDM signal and a normal single carrier system looked like), if they are not in your code you should do it.

The following file/function would be in order of appearance systemconfig.m



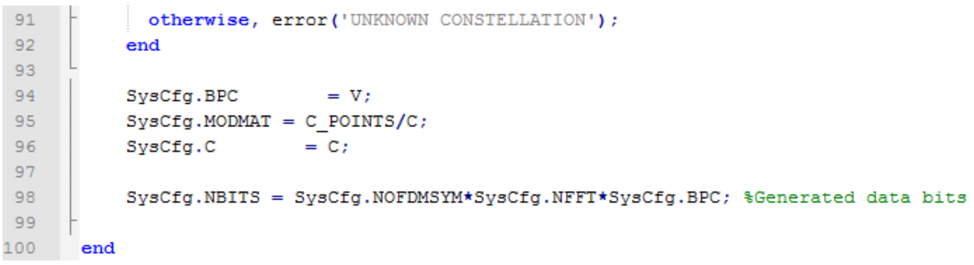
```
%
% Project: Course EPITECH
% Author: DPR
% Creation Date: 10/04/2015
% Function description: System configuration
%


function SysCfg = systemconfig()

    SysCfg.DATASEED = 0; %Seed for data generation, 0 random seed, 1+ fixed data seed


        SysCfg.FSAMP = 2048/224e-6; %Sampling frequency of the system
        SysCfg.FC = 20e6; %Center frequency of the system
        SysCfg.WorkFolder = 'work'; % Folder where the intermediate results will be stored

        SysCfg.ENABLEPLOTS = 1; %Enables the plot of different outputs of the system


        SysCfg.CONSTELLATION = '16-QAM'; %Constellation used for transmission
                                         %BPSK  QPSK 16-QAM 64-QAM 256-QAM

        SysCfg.NFFT = 2048; %FFT size (power of 2 usually)
        SysCfg.NOFDMSYM = 1; %Number of OFDM symbols to generate
```

This function only sets the values for the different fields of the structure SysCfg. The fields correspond with the parameters needed for other functions of the system to work. The second part just makes the shape of the constellation to be used.



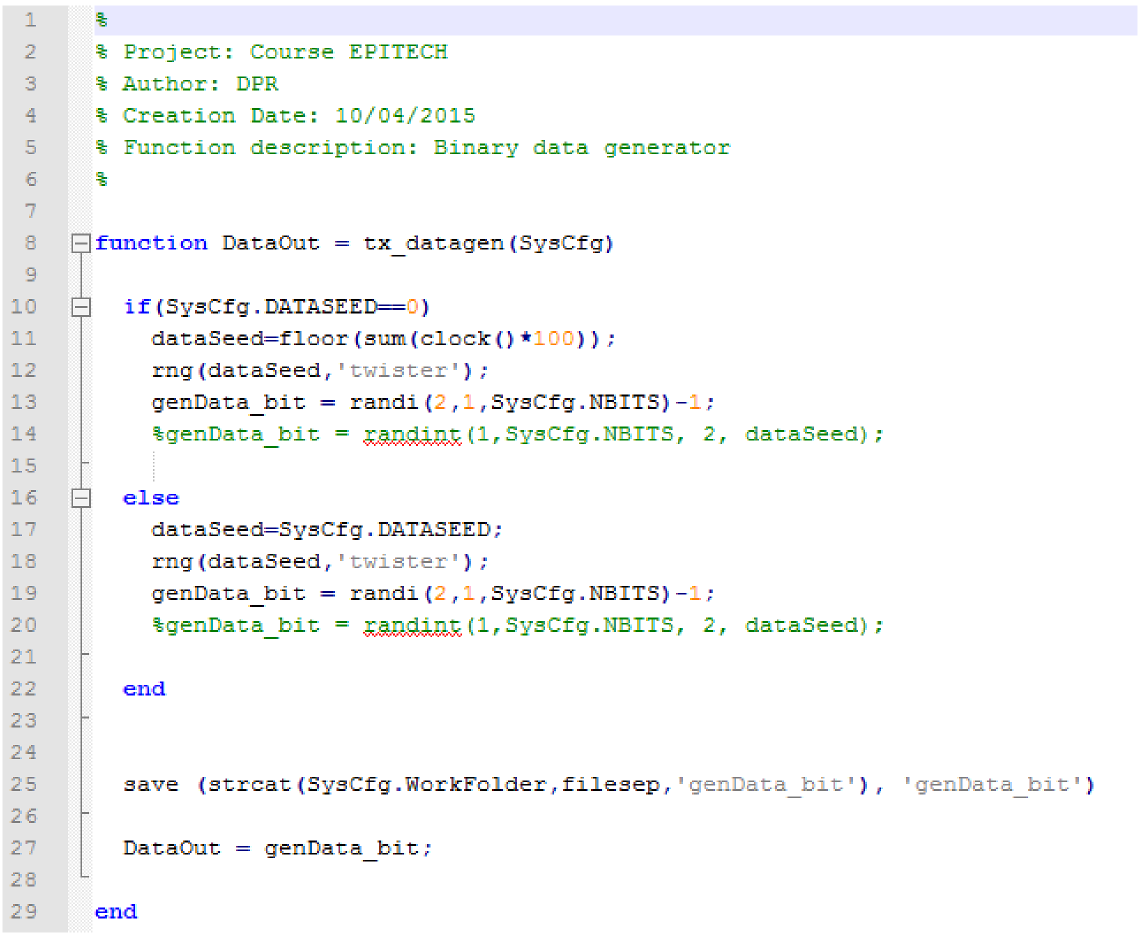
```

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % CONSTELLATION CHOICE                      %
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      switch SysCfg.CONSTELLATION
      case 'BPSK'
       C_POINTS = [1 -1]; % Constellation points
        C        = 1;                % Normalization factor
        V        = 1;                % Bits per cell
      case 'QPSK'
        C_POINTS = [1+1i 1-1i -1+1i -1-1i]; % Constellation points
        C        = sqrt(2);                 % Normalization factor
        V        = 2;                       % Bits per cell
      case '16-QAM'
        C_POINTS = [3+3i 3+i 1+3i 1+i 3-3i 3-i 1-3i 1-i -3+3i -3+i -1+3i ...
                    -1+i -3-3i -3-i -1-3i -1-i];
        C        = sqrt(10);
        V        = 4;
      case '64-QAM'
        C_POINTS = [7+7i 7+5i 5+7i 5+5i 7+1i 7+3i 5+1i 5+3i 1+7i 1+5i 3+7i ...
                    3+5i 1+1i 1+3i 3+1i 3+3i 7-7i 7-5i 5-7i 5-5i 7-1i 7-3i ...
                    5-1i 5-3i 1-7i 1-5i 3-7i 3-5i 1-1i 1-3i 3-1i 3-3i -7+7i ...
                    -7+5i -5+7i -5+5i -7+1i -7+3i -5+1i -5+3i -1+7i -1+5i ...
                    -3+7i -3+5i -1+1i -1+3i -3+1i -3+3i -7-7i -7-5i -5-7i ...
                    -5-5i -7-1i -7-3i -5-1i -5-3i -1-7i -1-5i -3-7i -3-5i ...
                    -1-1i -1-3i -3-1i -3-3i];
        C        = sqrt(42);
        V        = 6;
      case '256-QAM'
```

To end with the para parameters related to the different configurations and the calculus of the bits to be generated.



```
            otherwise, error('UNKNOWN CONSTELLATION');
        end

        SysCfg.BPC          = V;
        SysCfg.MODMAT = C_POINTS/C;
        SysCfg.C          = C;

        SysCfg.NBITS = SysCfg.NOFDMSYM*SysCfg.NFFT*SysCfg.BPC; %Generated data bits

    end
```

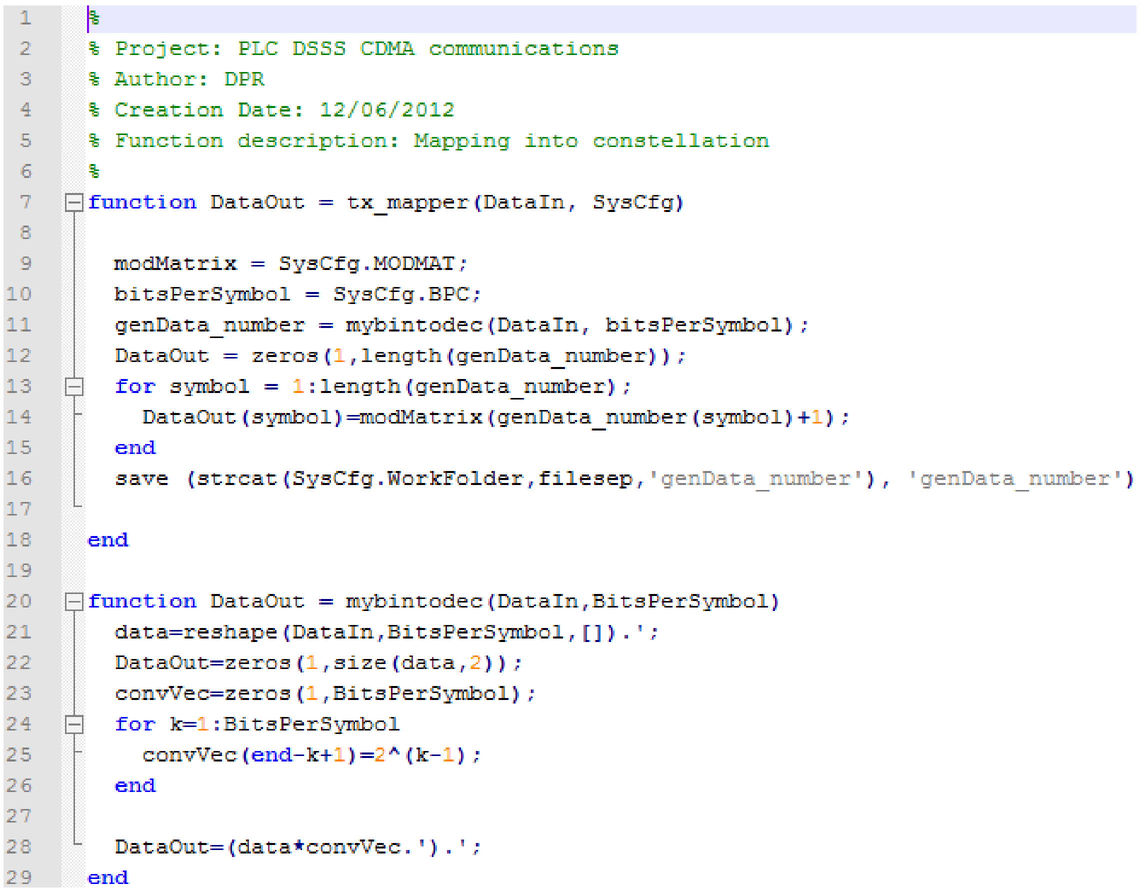
The next function to be called is the one in tx_datagen.m, which target is to generate a number of random data bits determined by SysCfg.NBITS.



```
%
% Project: Course EPITECH
% Author: DPR
% Creation Date: 10/04/2015
% Function description: Binary data generator
%

function DataOut = tx_datagen(SysCfg)

  if(SysCfg.DATASEED==0)
    dataSeed=floor(sum(clock()*100));
    rng(dataSeed,'twister');
    genData_bit = randi(2,1,SysCfg.NBITS)-1;
    %genData_bit = randint(1,SysCfg.NBITS, 2, dataSeed);

  else
    dataSeed=SysCfg.DATASEED;
    rng(dataSeed,'twister');
    genData_bit = randi(2,1,SysCfg.NBITS)-1;
    %genData_bit = randint(1,SysCfg.NBITS, 2, dataSeed);

  end


  save (strcat(SysCfg.WorkFolder,filesep,'genData_bit'), 'genData_bit')

  DataOut = genData_bit;

end
```

This code uses the function randi (you can type “help randi” in MATLAB shell to get a full description of the function) to generate a 1xSysCfg.NBITS vector and assign it to genData_bit. This vector will take only 2 values (‘0’ and ‘1’). After generating this data bits the variable genData_bit is saved in a folder in the path indicated in SysCfg.WorkFolder. This will be stored as genData_bit.mat.

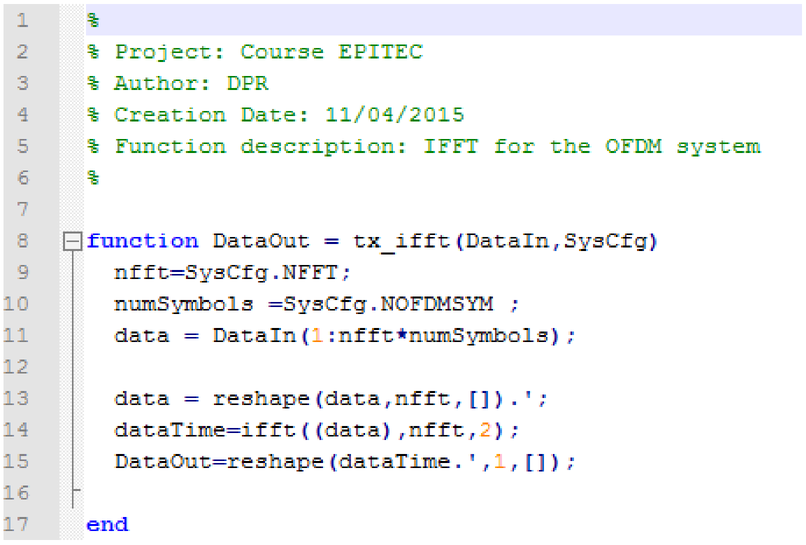
The next function is the one in tx_mapper.m that performs the mapping of the bits into the selected QAM constellation.



```
1  %
2  % Project: PLC DSSS CDMA communications
3  % Author: DPR
4  % Creation Date: 12/06/2012
5  % Function description: Mapping into constellation
6  %
7  function DataOut = tx_mapper(DataIn, SysCfg)
8
9    modMatrix = SysCfg.MODMAT;
10   bitsPerSymbol = SysCfg.BPC;
11   genData_number = mybintodec(DataIn, bitsPerSymbol);
12   DataOut = zeros(1,length(genData_number));
13   for symbol = 1:length(genData_number);
14     DataOut(symbol)=modMatrix(genData_number(symbol)+1);
15   end
16   save (strcat(SysCfg.WorkFolder,filesep,'genData_number'), 'genData_number')
17
18 end
19
20 function DataOut = mybintodec(DataIn,BitsPerSymbol)
21   data=reshape(DataIn,BitsPerSymbol,[]).';
22   DataOut=zeros(1,size(data,2));
23   convVec=zeros(1,BitsPerSymbol);
24   for k=1:BitsPerSymbol
25     convVec(end-k+1)=2^(k-1);
26   end
27
28   DataOut=(data*convVec.').';
29 end
```

The first function in the file takes the vector SysCfg.MODMAT and groups the bits coming as input in sets of the number of bits of each symbol (following the configuration). In line 11 the string of input bits is transformed into a string of decimal numbers that go from 0 to $2^{\text{bitsPerSymbol}}$. These numbers will be used as index to get a value from modMatrix to assign to the output symbol. These index will be stored (line 16) to check at the receiver if there were mistakes during the transmission.

The last function in the base code provided is the one in tx_ifft.m, this function applies the reverse Fourier transform to the input data.



```
%
% Project: Course EPITEC
% Author: DPR
% Creation Date: 11/04/2015
% Function description: IFFT for the OFDM system
%

function DataOut = tx_ifft(DataIn,SysCfg)
  nfft=SysCfg.NFFT;
  numSymbols =SysCfg.NOFDMSYM ;
  data = DataIn(1:nfft*numSymbols);

  data = reshape(data,nfft,[]).';
  dataTime=ifft((data),nfft,2);
  DataOut=reshape(dataTime.',1,[]);

end
```

This function uses reshape (type “help reshape” in a matlab shell for a full description) to change the dimensions of the input stream that will be a row vector with a length of NFFT*NOFDMSYM (the length of the FFT used for each OFDM symbol and the number of OFDM symbols generated respectively), see Fig. 2. To use in a more easy way the ifft function it is recommendable to have every OFDM symbol to which it will be applied the IFFT in a separate row or column.

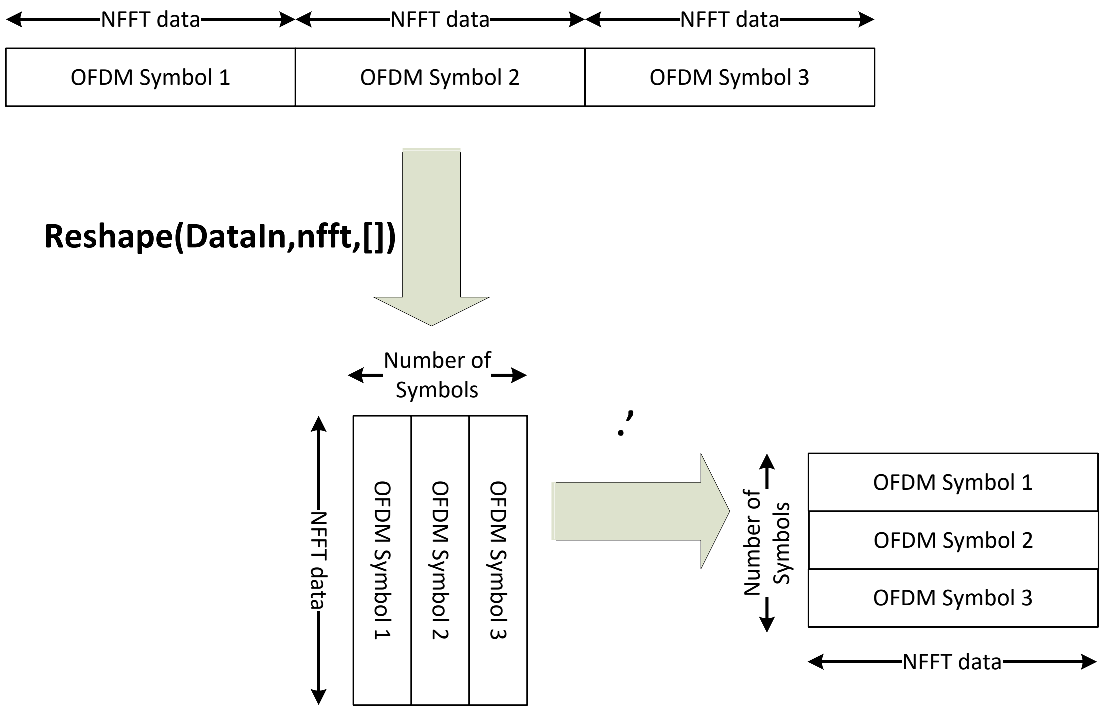


Fig. 2. Reshape explanation

After applying the IFFT to the data (by rows), again we use reshape to have a row vector at the ouput, dataTime needs to be transposed because reshape always reads and writes by columns.

3. Tasks to perform

In this section it will be described every function of the ones to be accomplished. In general it will be given a template and some of the lines to be coded will be deliberately left in blank, so the student should be able to fill it with the explanations given.

3.1. Cyclic prefix insertion: tx_cp.m

As shown during the course (more concretely in the part related to the introduction to OFDM, and also in DVB-T standard) the last part in an OFDM modulation is the cyclic prefix insertion. This consists in copying the last data at the beginning of the OFDM symbol (see Fig. 3).

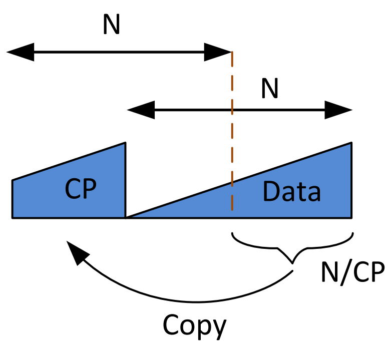
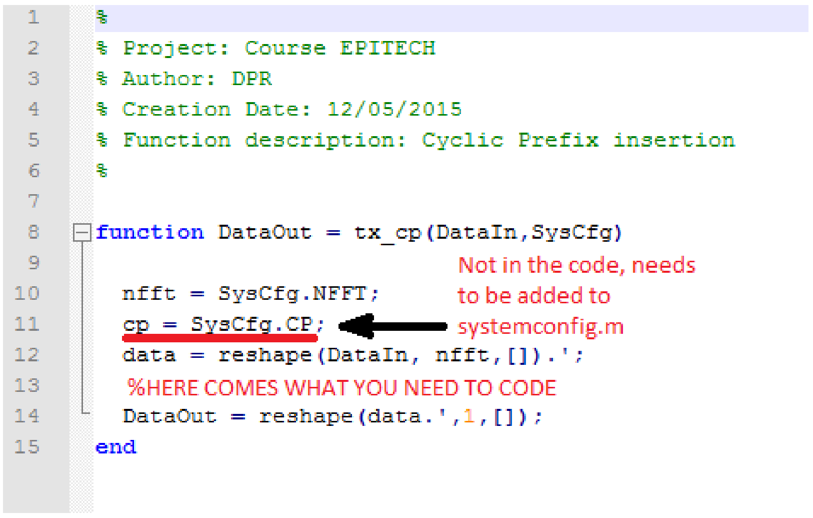


Fig. 3. Cyclic prefix insertion

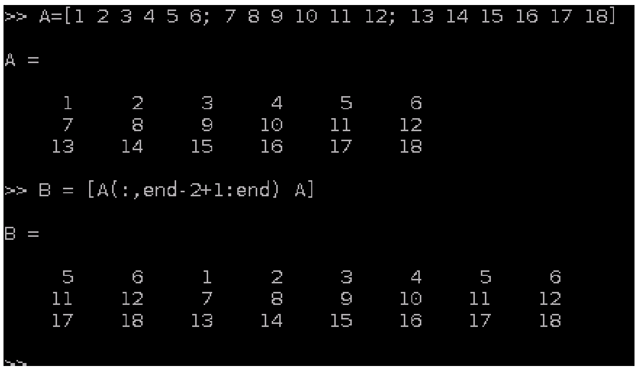
Taking all this in mind our function must look like this:



```
%
% Project: Course EPITECH
% Author: DPR
% Creation Date: 12/05/2015
% Function description: Cyclic Prefix insertion
%

function DataOut = tx_cp(DataIn,SysCfg)

  nfft = SysCfg.NFFT;
  cp = SysCfg.CP;
  data = reshape(DataIn, nfft,[]).';
  %HERE COMES WHAT YOU NEED TO CODE
  DataOut = reshape(data.',1,[]);
end
```

Not in the code, needs to be added to systemconfig.m

This function will just use reshape to put all the symbols divided by rows, SysCfg.CP needs to be added to systemconfig.m, and given a value according to the ones available in DVB-T (4, 8, 16, 32). The functionality to be coded in this function is to take the corresponding last columns (nfft/cp) of the resulting matrix and copy it in the beginning. As an example you can try the following in a MATLAB shell:



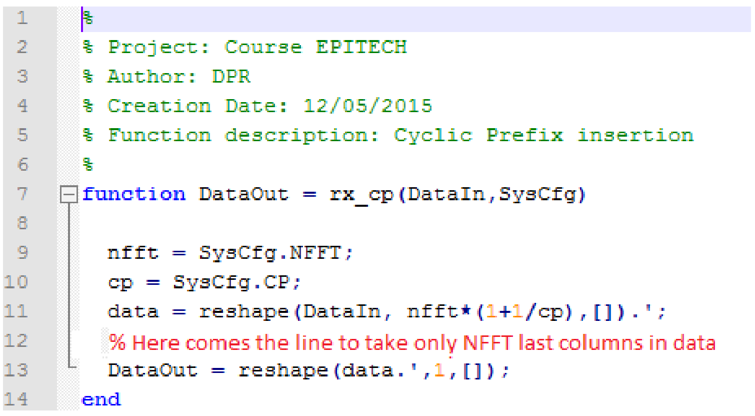
```
>> A=[1 2 3 4 5 6; 7 8 9 10 11 12; 13 14 15 16 17 18]

A =

     1     2     3     4     5     6
     7     8     9    10    11    12
    13    14    15    16    17    18

>> B = [A(:,end-2+1:end) A]

B =

     5     6     1     2     3     4     5     6
    11    12     7     8     9    10    11    12
    17    18    13    14    15    16    17    18

>>
```

If you look carefully first we have a matrix with 3 rows and 6 columns. Using this matrix we create a second one, B matrix is the same as A but with copying the last 2 columns in the beginning, that is exactly what we want to do when we add the cyclic prefix. The only thing to change to this example in the creation of the B matrix is the number of columns to copy in the beginning. With a very similar line in the line 13 of the given code the function will work.

In order to test if you succeed you can subtract to the first NFFT/CP data the last NFFT/CP. To make a “breakpoint” in MATLAB to debug write in the line of your code where you want to stop the word “keyboard”, the program will stop and you will have the control of the shell again to make any operation or to check any variable in the workspace. To return to the execution of the program type “return” in the shell and press enter.

3.2. Cyclic prefix removal: rx_cp.m

This is the first block in the receiver (supposing a perfect detection and time and frequency synchronization), the receiver performs the reverse operations of the ones in the transmitter in reverse order, so the first part in the receiver will be the reverse of the last in the transmitter, the cyclic prefix removal. This function will be very similar to the tx_cp, but in this case instead of adding a copy of the last NFFT/CP samples in every OFDM symbol we have to remove this number of sample from the beginning of each symbol, or looking it from another point of view we need to take the last NFFT samples of every one of them. Said so the function will look like this:

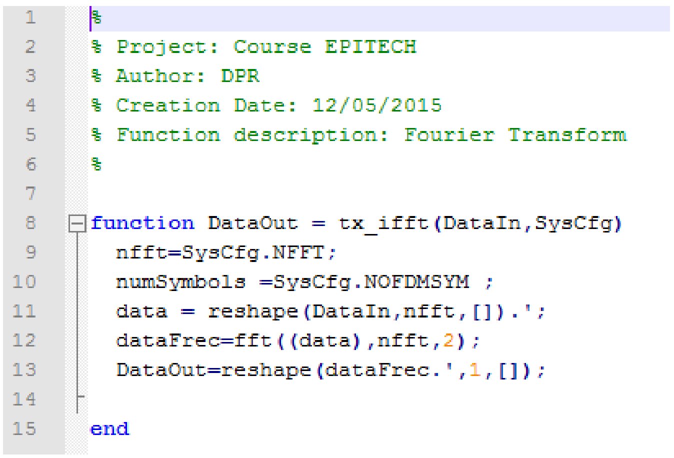


```
%
% Project: Course EPITECH
% Author: DPR
% Creation Date: 12/05/2015
% Function description: Cyclic Prefix insertion
%
function DataOut = rx_cp(DataIn,SysCfg)

  nfft = SysCfg.NFFT;
  cp = SysCfg.CP;
  data = reshape(DataIn, nfft*(1+1/cp),[]).';
  % Here comes the line to take only NFFT last columns in data
  DataOut = reshape(data.',1,[]);
end
```

There is only one line to be added to this code proposed, the one in with we take the last NFFT samples of every row (remember as it was shown in the previous section that for that we will need to use something similar to: B=A(:,startingValue:end)).

3.3. Fourier Transform: rx_fft.m

This function will take the data from the time domain to the frequency domain so it has again the shape of a QAM constellation. This function is exactly the same as the tx_ifft but changing the ifft line for fft line, so I will provide the whole code for this file because there is not a lot of difficulty in it.



```
%
% Project: Course EPITECH
% Author: DPR
% Creation Date: 12/05/2015
% Function description: Fourier Transform
%

function DataOut = tx_ifft(DataIn,SysCfg)
  nfft=SysCfg.NFFT;
  numSymbols =SysCfg.NOFDMSYM ;
  data = reshape(DataIn,nfft,[]).';
  dataFrec=fft((data),nfft,2);
  DataOut=reshape(dataFrec.',1,[]);

end
```

3.4. Symbol demapping: rx_demap.m

This is the more complex part to code, but I will try to make it easy. This part of the receiver is the opposite to translate the bits into constellation points, this process is not exactly instantaneous because usually there is noise in the channel or attenuation and what we receive is far from being what we transmitted. So this is more an statistical problem, which symbol of the ones we know that have been possibly transmitted looks more a like to what we received? The answer to this question is the one that is closer in distance.

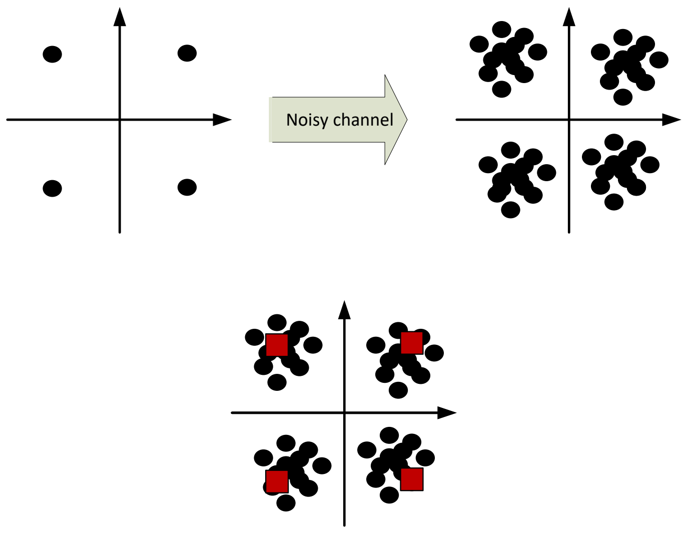
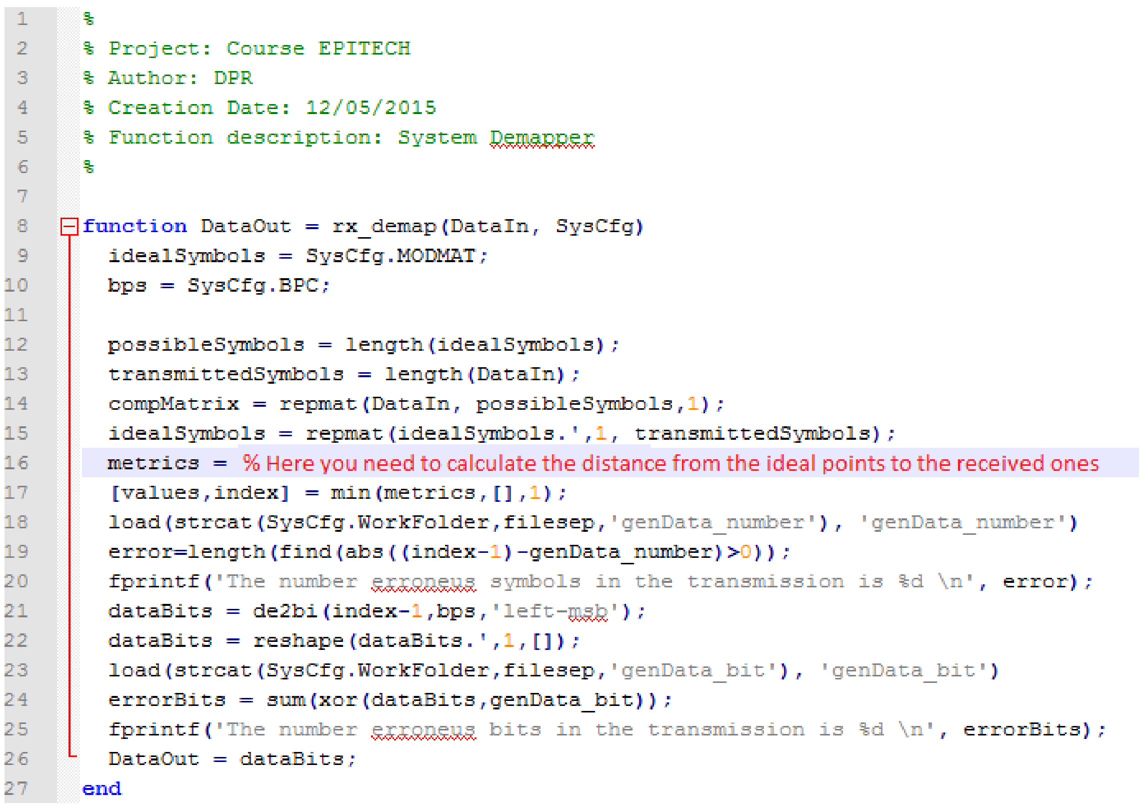


Fig. 4. Demapping a received point

In Fig. 4 we can see how a transmitted point will suffer a distortion because of the noise and instead or receiving the points in their ideal position we receive them as a cloud of points, that will be more or less similar to the ideal transmitted constellation depending on the noise. In red squares it is marked the ideal constellation points, every point received (black circles) would had been more probably transmitted as the ideal point that is closer. This is, we need to measure the distance of every point received to the ideal ones and the one that is smaller we will take as the transmitted point.

With all this theory in mind the code could look like this:



```
1  %
2  % Project: Course EPITECH
3  % Author: DPR
4  % Creation Date: 12/05/2015
5  % Function description: System Demapper
6  %
7
8  function DataOut = rx_demap(DataIn, SysCfg)
9    idealSymbols = SysCfg.MODMAT;
10   bps = SysCfg.BPC;
11
12   possibleSymbols = length(idealSymbols);
13   transmittedSymbols = length(DataIn);
14   compMatrix = repmat(DataIn, possibleSymbols,1);
15   idealSymbols = repmat(idealSymbols.',1, transmittedSymbols);
16   metrics =  % Here you need to calculate the distance from the ideal points to the received ones
17   [values,index] = min(metrics,[],1);
18   load(strcat(SysCfg.WorkFolder,filesep,'genData_number'), 'genData_number')
19   error=length(find(abs((index-1)-genData_number)>0));
20   fprintf('The number erroneus symbols in the transmission is %d \n', error);
21   dataBits = de2bi(index-1,bps,'left-msb');
22   dataBits = reshape(dataBits.',1,[]);
23   load(strcat(SysCfg.WorkFolder,filesep,'genData_bit'), 'genData_bit')
24   errorBits = sum(xor(dataBits,genData_bit));
25   fprintf('The number erroneus bits in the transmission is %d \n', errorBits);
26   DataOut = dataBits;
27 end
```

There is only one line to be filled (line 16), in this line you need to calculate the distance from the ideal symbols (idealSymbols) to the received ones (compMatrix). The two matrix have the right dimensions and the absolute value in matlab is the function abs(), when applied to a matrix it gives the absolute value of all its elements.

This code is optimized to not to use for loops so I will explain it a bit so you don't get very lost. First of all we get the vector of the points in our constellation (in the case of a QPSK 1/sqrt(1)*[1+j,-1+j,-1-j,1-j]) and store it in idealSymbols and the bits per symbol (bps, in the case of a QPSK 2 bits per symbol) and also the number of points in the constellation (possibleSymbols, in a case of QPSK 4). In line 14 we create a matrix that has as many columns as received points, this would make a row vector with all the received points, and we repeat this vector as many times as number of points are in the constellation by using repmat function (see Fig. 5).

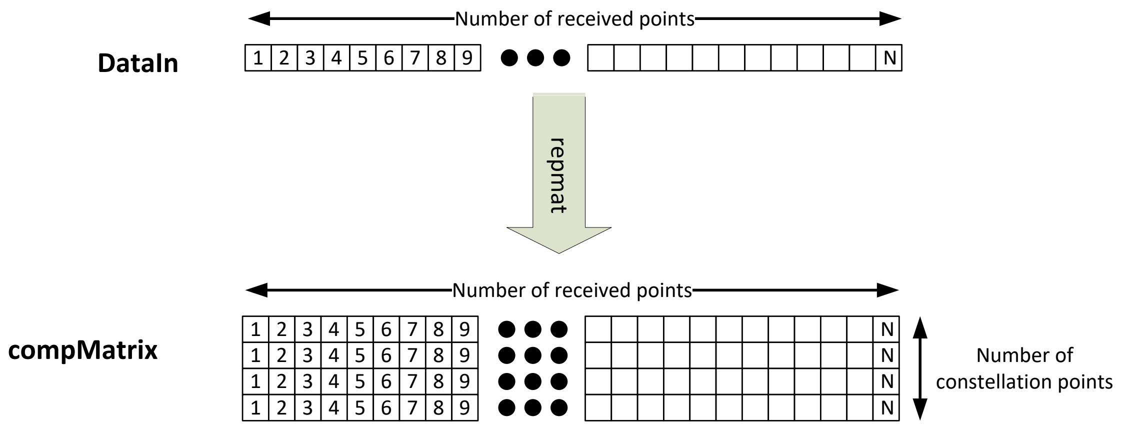
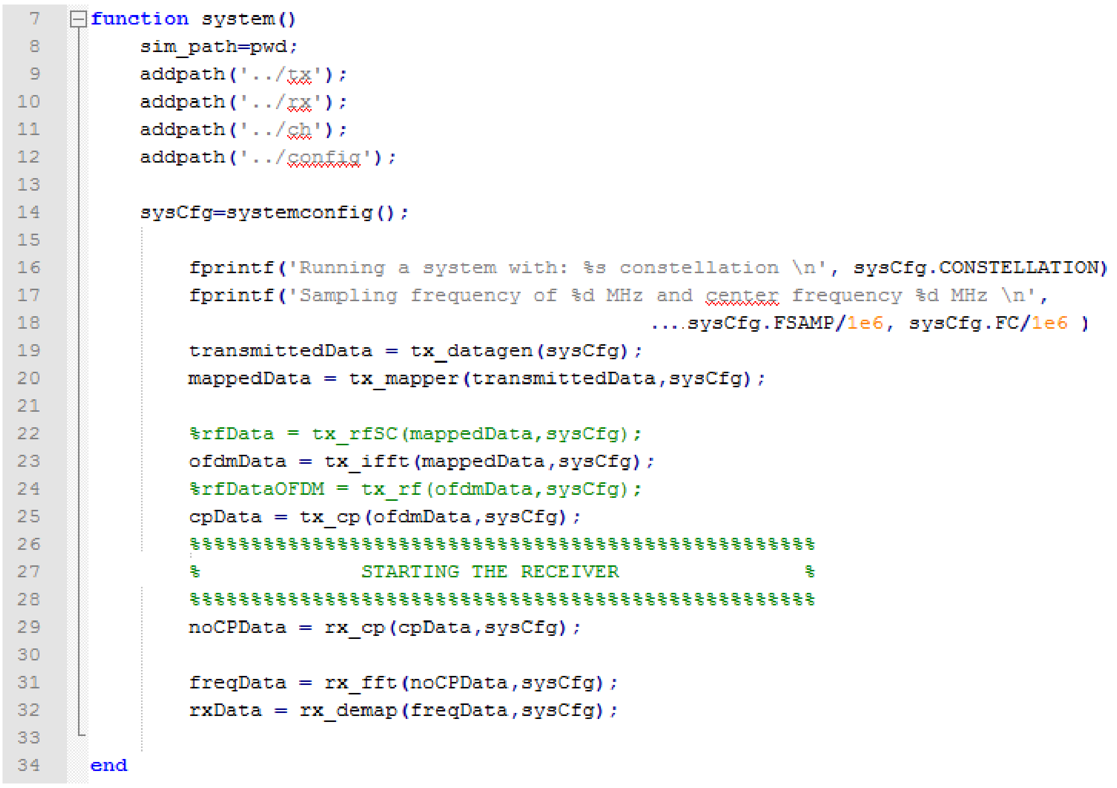

Number of received points
DataIn
1 2 3 4 5 6 7 8 9 ● ● ● N
repmat
Number of received points
compMatrix
1 2 3 4 5 6 7 8 9 ● ● ● N
Number of constellation points

Fig. 5. Repmat usage

We do something similar in 15 with the ideal constellation points (idealSymbols), we repeat them to have in the rows the different points and this is repeated N times to have a column for each received symbol.

After this, in line we look for the minimum in every column and it will tell us the index of the transmitted data. The rest of the code is to have a graphical way to verify if the process was correct, we calculate the errors we had and plot them with the symbols and later with the bits.

Take into account that the system_run.m file needs to be modified to call all the functions created, it should look like this:

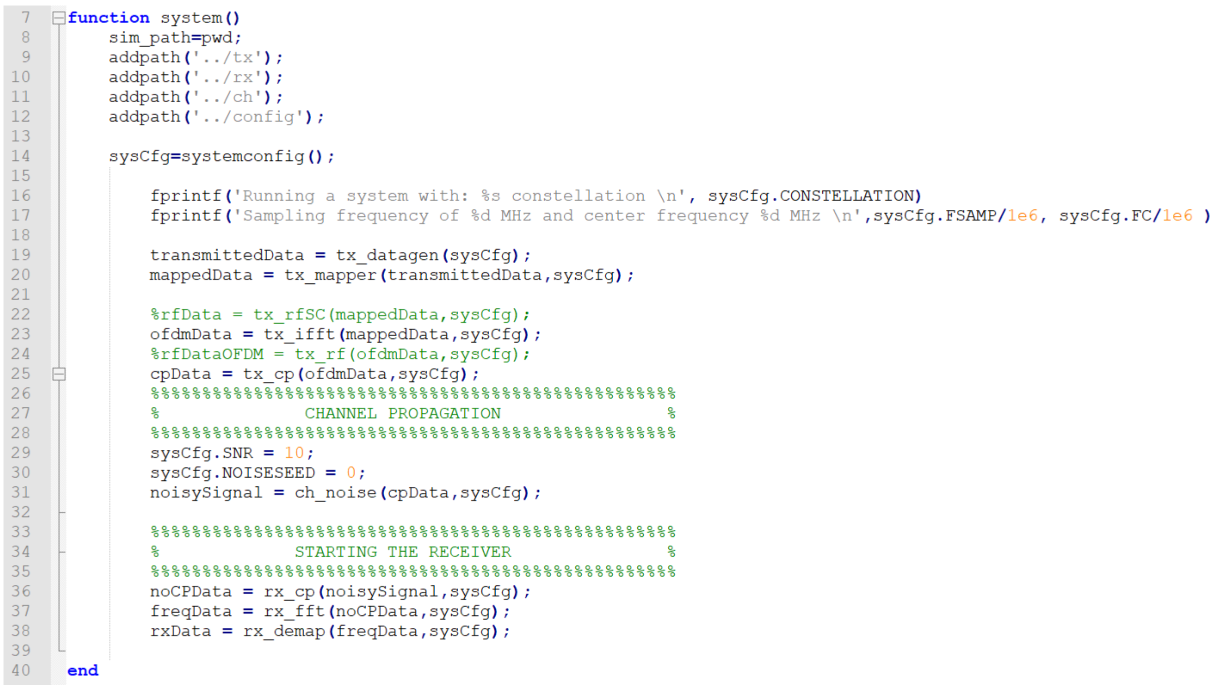


```
function system()
    sim_path=pwd;
    addpath('../tx');
    addpath('../rx');
    addpath('../ch');
    addpath('../config');

    sysCfg=systemconfig();

        fprintf('Running a system with: %s constellation \n', sysCfg.CONSTELLATION)
        fprintf('Sampling frequency of %d MHz and center frequency %d MHz \n',
                                      ...sysCfg.FSAMP/1e6, sysCfg.FC/1e6 )
        transmittedData = tx_datagen(sysCfg);
        mappedData = tx_mapper(transmittedData,sysCfg);

        %rfData = tx_rfSC(mappedData,sysCfg);
        ofdmData = tx_ifft(mappedData,sysCfg);
        %rfDataOFDM = tx_rf(ofdmData,sysCfg);
        cpData = tx_cp(ofdmData,sysCfg);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %              STARTING THE RECEIVER              %
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        noCPData = rx_cp(cpData,sysCfg);

        freqData = rx_fft(noCPData,sysCfg);
        rxData = rx_demap(freqData,sysCfg);

end
```

After coding all the files and running “system_run” in the Matlab shell it should be obtained that 0 errors were produced in the transmission.

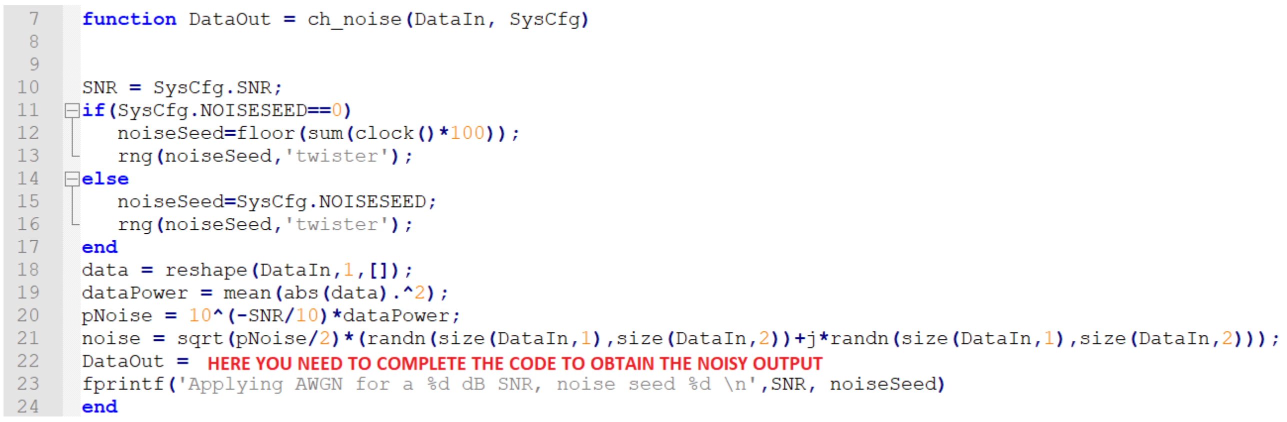
3.5. Adding AWGN

The last step of the practice will be to add some noise to our signal so the transmission has several errors. In order to do so, depending on the SNR (Signal to Noise Ratio) that we have, we will modify “system_run.m” and create another file, “ch_noise.m”, inside the “ch” folder.



```
function system()
    sim_path=pwd;
    addpath('../tx');
    addpath('../rx');
    addpath('../ch');
    addpath('../config');

    sysCfg=systemconfig();

        fprintf('Running a system with: %s constellation \n', sysCfg.CONSTELLATION)
        fprintf('Sampling frequency of %d MHz and center frequency %d MHz \n',sysCfg.FSAMP/1e6, sysCfg.FC/1e6 )

        transmittedData = tx_datagen(sysCfg);
        mappedData = tx_mapper(transmittedData,sysCfg);

        %rfData = tx_rfSC(mappedData,sysCfg);
        ofdmData = tx_ifft(mappedData,sysCfg);
        %rfDataOFDM = tx_rf(ofdmData,sysCfg);
        cpData = tx_cp(ofdmData,sysCfg);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %               CHANNEL PROPAGATION               %
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        sysCfg.SNR = 10;
        sysCfg.NOISESEED = 0;
        noisySignal = ch_noise(cpData,sysCfg);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %              STARTING THE RECEIVER              %
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        noCPData = rx_cp(noisySignal,sysCfg);
        freqData = rx_fft(noCPData,sysCfg);
        rxData = rx_demap(freqData,sysCfg);

end
```

As it can be seen, now we have a propagation part in the “system_run” which takes as an input the output of the transmitter and whose output will be the input of the receiver. Also we are adding to the configuration structure the SNR that we want for the simulation and the noise seed for the random noise generation.



```
function DataOut = ch_noise(DataIn, SysCfg)


SNR = SysCfg.SNR;
if(SysCfg.NOISESEED==0)
    noiseSeed=floor(sum(clock()*100));
    rng(noiseSeed,'twister');
else
    noiseSeed=SysCfg.NOISESEED;
    rng(noiseSeed,'twister');
end
data = reshape(DataIn,1,[]);
dataPower = mean(abs(data).^2);
pNoise = 10^(-SNR/10)*dataPower;
noise = sqrt(pNoise/2)*(randn(size(DataIn,1),size(DataIn,2))+j*randn(size(DataIn,1),size(DataIn,2)));
DataOut =  HERE YOU NEED TO COMPLETE THE CODE TO OBTAIN THE NOISY OUTPUT
fprintf('Applying AWGN for a %d dB SNR, noise seed %d \n',SNR, noiseSeed)
end
```