PyWiCh User Manual

Introduction

PyWiCh software can be used as an API from another software or can be used stand alone. PyWiCh has a graphical user interface to facilitate the simulations. We try to give the all the main features using the GUI but it has less features than when PyWiCh Classes are called directly.

PyWiCh has four main objects:

- The first ones is Antenna object where the user can configure the transmit and receive antenna array.
- The second one is the Scenario object where the user can configure the size of the scenario, the Base Station position, the parameters of the system, the loss model, the shadowing model, etc..
- The third one is the FastFading object (which is close related with the Scenario) has the methods and procedures to build the wireless channel matrix and other fast fading parameters.
- The four one is the FrequencyBand object the has the information about the frequency band, the OFDM system and the noise characteristics.

There is a fifth important object that is the ChannelPerformance object. This object is used to run the simulation and to obtain the performance metrics of the system.

In order to run a simulation the five previous objects must be buils and configure. The configuration parameters of each object are explained in the software documentation and depend on the type of object. For example an Antenna can be an isotropic antenna or a 3gpp antenna model. Both types of antennas have different parameters.

The PyWiCh GUI helps to define and configure the five previous objects. This GUI also has some graphical analysis of the simulation results.

We first explains how to run the system from a Python program without using the PyWiCh GUI.
In this manual we refer as “project directory”, the directory of your disk where you download the PyWiCh code.

Run a simulation from a Python program.

The “project directory”/src/test.py is a simple example of how to configure and run the simulation. You must read this code to understand the logic of PyWiCh.

We also recommend the user to look at the “project directory”/src/qa directory (Quality Assurance) where we have test code for the different classes. This test code can help the user to understand the classes features.

The test program can be run using the following command form the “project directory”:

>> python3 ./src/test.py

This program runs the simulation and saves the results to the “project directory”/data/test directory. If you want to run another simulation, you must modify the simulation name in the test.py program

(SIM_NAME = “test”) or delete the “project directory”/data/test directory because PyWiCh does not override the data results of previous simulations.

The user can also find the source code of test.py in this user manual's Appendix.

The files and the formats of the simulation results are explained in another section of this user manual.

Run a simulation and analyze the results from the PyWiCh GUI.

The PyWiCh GUI can be invoked form the project directory (where you download the code) in the following way:

python3 ./src/gui/gui_App_graph.py

The program shows the following window (Figure 1).

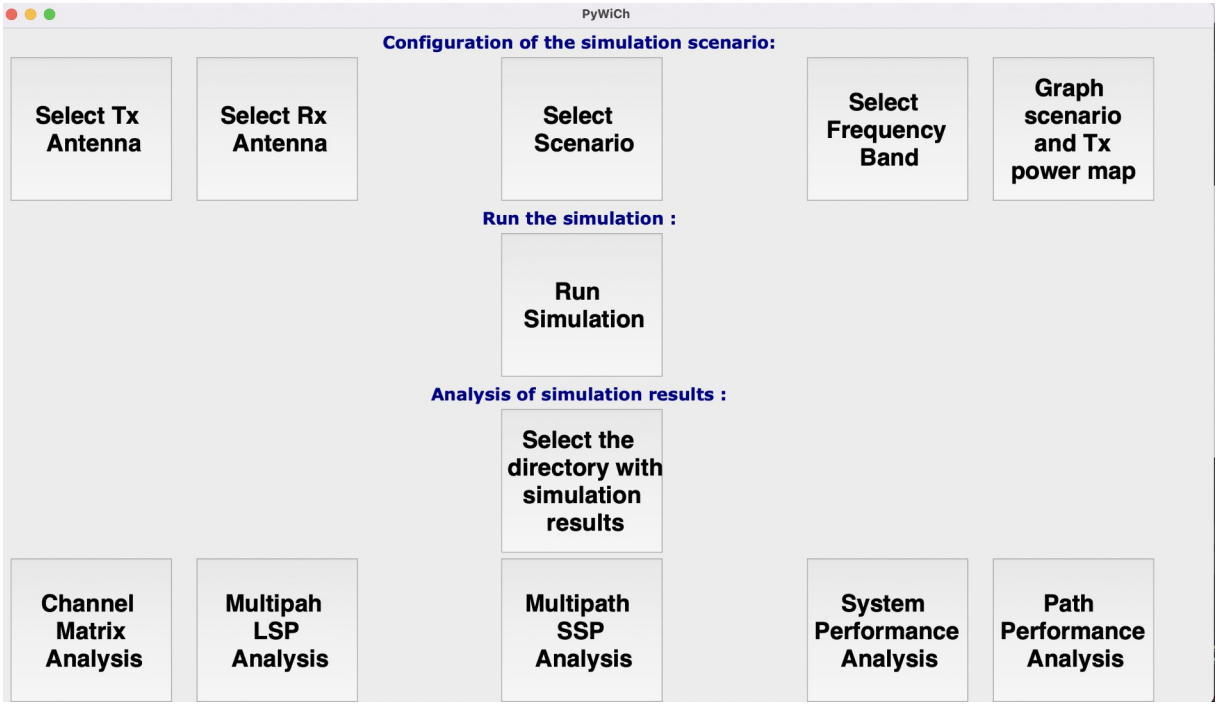

Figure 1.

In this window in the upper line of buttons, you can configure the different objects of PyWiCh. The last button: Graph scenario and Tx power map, allows you to view graphically: the scenario, the Tx power map, and the MSs routes in the scenario. It helps you to analyze before the simulation if the scenario and the antennas are well configured.

In the second row, there is the button: Run Simulation. After you finish the configuration of the PyWiCh objects, you can run the simulation. Before the simulation start, the system asks you for the name of the simulation. For example, if you give the name: “MyFirstSimulation”, the results of the simulations can be found, after it finishes, in the following directory: “Project Directory”/src/data/MyfirstSimulation/

After you run a simulation, you can analyze its results with the buttons on the last row. Previously to analyze the results, you must select with the corresponding button the directory where the results are stored.

The main graphical results are:

- Channel Matrix Analysis. In this analysis, first, PyWiCh asks for an MS, a point in its route, and a Physical Resource Block of the OFDM channel. Then, the system plots the eigenvalues and the condition number of the corresponding MIMO Channel Matrix (see Figure 2).

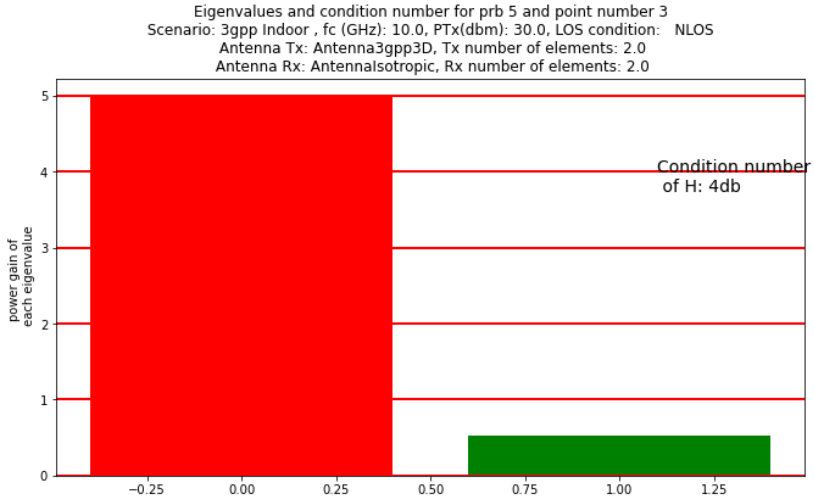

Figure 2.

- Multipath LSP Analysis. In this analysis, PyWiCh asks for selecting of one of the Large Scale Parameters (Shadowing, Ricean K Factor, Delay Spread, Angles spreads, and LOS). PyWiCh plots the parameter chosen in the scenario space (see Figure 3).

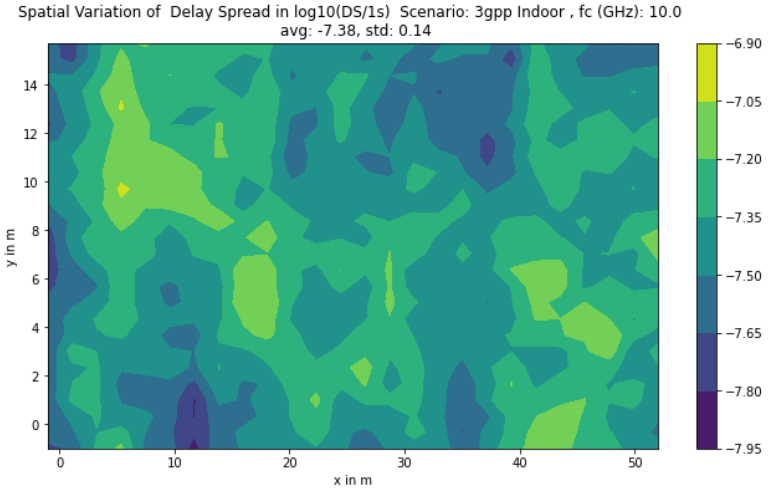

Figure 3.

- Multipath SSP Analysis. In this analysis, PyWiCh asks for selecting of one of the Short Scale Parameters (Clusters Delays, Clusters arrival and departure angles, and Power Delay Profile). PyWiCh plots the parameter chosen for one selected point in the MS route (see Figure 4).

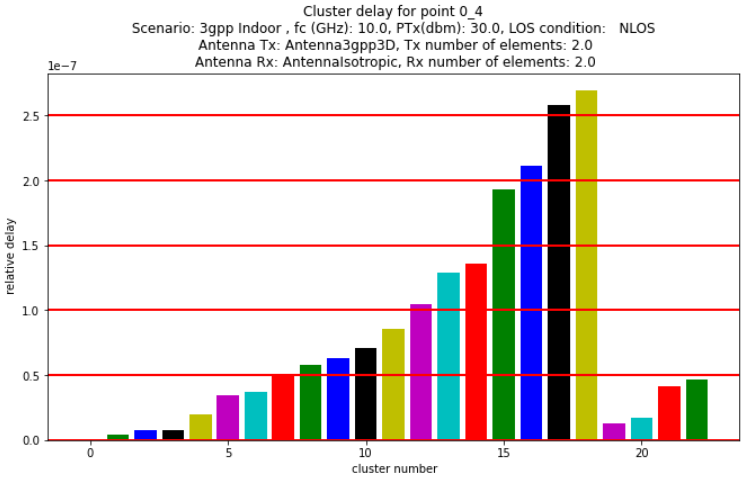

Figure 4

- System Performance Analysis. In this analysis, PyWiCh asks to select one point in the MS route. Then, PyWiCh plots for that point the average receive power and the channel matrix coefficients as a function of the prb number in the OFDM grid (see Figures 5 and 6).

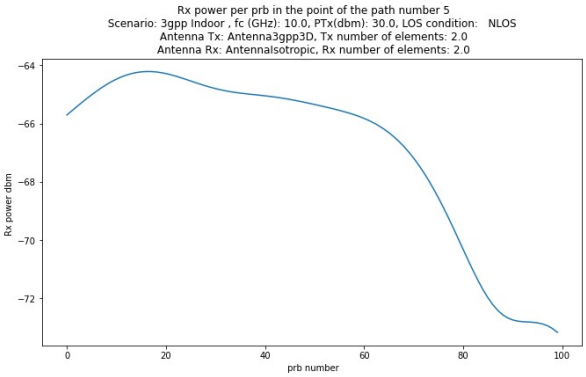

Figure 5.

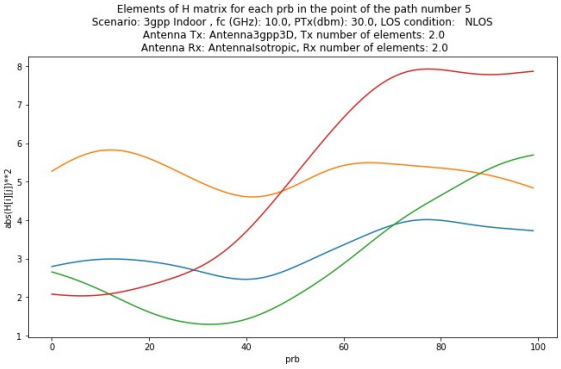

Figure 6.

- Path Performance Analysis. In this analysis, PyWiCh asks to select one physical resource block in the OFDM grid. Then, PyWiCh plots for that prb as a function of time, the SNR, the spectral efficiency, and the different components of the losses: path loss, shadowing, and fading (see Figures 7, 8 and 9).

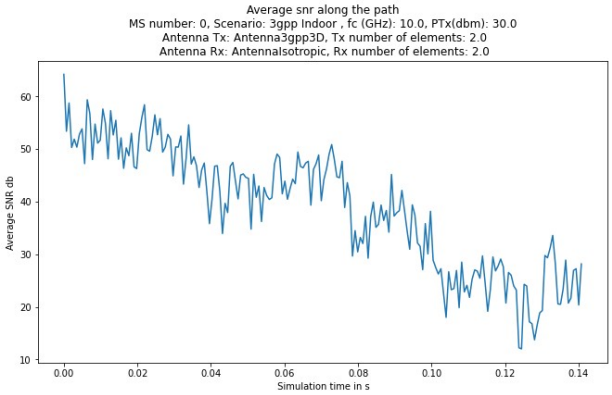

Figure 7.

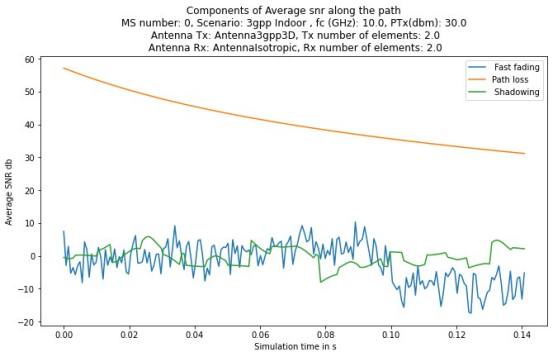

Figure 8.

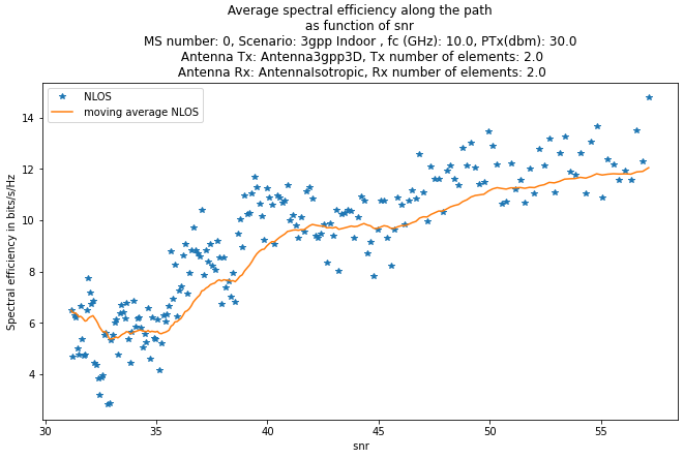

Figure 9.

Specification of the files saved with the simulation information and results.

PyWiCh saves the simulation results in the directory that the user specifies, as we have explained in the previous sections. This section describes the files stored with the simulation results and the format of these files.

The Antenna, Scenario, FastFading, FrequencyBand, and ChannelPerformance classes, and the classes that inherit from them, have their own save method. In the following paragraphs, we explain the files that each class saves and their format.

Antenna.

Antenna Element.

The information of the antenna elements of the receive and transmit antenna arrays are saved to the following two files:

1. path+'/antenna_element'+name+'_type.csv'

2. path+'/antenna_element'+name+'.csv'

where in both files names the variable name is: ‘antennaRx’ or ‘antennaTx’

In the case of an Isotropic antenna element the first file store the string ‘AntennaIsotropic’.

In the case of a 3gpp3D antenna element the file store the string ‘Antenna3gpp3D’.

The second file in the case of an Isotropic antenna stores only the gain in dbs of the antenna,

In the case of a 3gpp3D antenna element, the second file store an array with the following data:

[self.maxgaindb, self.A_max, self.SLA_v, self.beamwidth].

AntennaArray

The antenna AntennaArray3gpp class saves the following data to the following files:

1. path+'/' +self.antenna_name+'_type.csv' where antenna_name is an attribute of the class.

This file stores the name of the class of the antenna array.

2. path+'/' +self.antenna_name+'.csv'

This file stores an array with the following information: [self.get_number_of_elements(), self.d_h, self.d_v, self.n_cols, self.n_rows, self.alpha, self.beta, self.gamma, self.polarization]

Scenario

All scenarios save the following information to the following files:

1. path+'/BS_position.csv'

In this file stores the array with the 3D position of the Base Station in the scenario.

2. path+'/scenario.csv'

This file stores the following array with the information of the simulation scenario:

[self.fcGHz, self.posx_min, self.posx_max, self.posy_min, self.posy_max, self.grid_number, self.Ptx_db]

The Scenario3gpp class, and the classes that inherits from this class, store the following information in the following files:

1. path+'/scenario_gridLSP.npy'

This file with extension npy stores a multidimensional numpy array. These types of files are saved and read with the corresponding numpy save and load methods. This file stores a 3D array with the following dimensions: (7,self.grid_number+1,self.grid_number+1). Where 7 is the maximum number of LSPs and grid_number is the number of grid elements in the (x,y) space of the scenario. Each element of this array stores one LSP for one point of the scenario. The LSPs are in the following order: Shadow, Ricean K factor, delay spread, Azimuth angle of departute spread, Azimuth angle of arrival spread, Zenith angle of departure spread, Zenith angle of arrival spread.

2. path+'/scenario_gridLOS.npy'

This file with extension npy stores a multidimensional numpy array. These types of files are saved and read with the corresponding numpy save and load methods. IThis file stores a 3D array with the following dimensions: (self.grid_number+1,self.grid_number+1). Where grid_number is the number of grid elements in the (x,y) space of the scenario. This file stores for each point of the grid a Boolean True or False depending on the LOS condition for this point.

3. path+'/scenario_name.csv'

This file stores self.name, the name given to the scenario.

FrequencyBand

This class stores the following information in the following file:

1. path+'/frequency_band.csv',

This file stores an array with the following information: [self.n_prbs, self.bw_prb, self.noise_figure_db, self.thermal_noise_dbm_Hz].

FastFading3gpp

This class stores the following information in the following files for each MS_num in each point_number of its route.

1. path+'/lsp_'+str(MS_num)+ '_' +str(point_number)+'.csv'

This file stores an array with the LSPs for this MS at this point of its route:

[self.__los, self.__sigma_shadow, self.__sigma_K, self.__sigma_tau, self.__sigma_AOD_AZS, self.__sigma_AOA_AZS, self.__sigma_AOD_ELS, self.__sigma_AOA_ELS]

2. path+'/tau_'+str(MS_num)+ '_' +str(point_number)+'.csv'

This file stores an array with the delay tau for each scatter for this MS at this point of its route.

3. path+'/tauLOS_'+ str(MS_num)+ '_' +str(point_number)+'.csv'

This file stores an array with the delay tau in the LOS conddition for each scatter for this MS at this point of its route.

4. path+'/PDP_'+str(MS_num)+ '_' +str(point_number)+'.csv'

This file stores an array with the Power Delay Profile for each scatter for this MS at this point of its route.

5. path+'/PDP_LOS_' +str(MS_num)+ '_' +str(point_number)+ '.csv'

This file stores an array with the Power Delay Profile in LOS condition for each scatter for this MS at this point of its route.

6. path+'/PHI_AOA_' +str(MS_num)+ '_' +str(point_number)+ '.csv'

This file stores an array with the Azimuth Angle of Arrival for each scatter for this MS at this point of its route.

7. path+'/PHI_AOD_' +str(MS_num)+ '_' +str(point_number)+ '.csv'

This file stores an array with the Azimuth Angle of Departure for each scatter for this MS at this point of its route.

8. path+'/THETA_AOA_' +str(MS_num)+ '_' +str(point_number)+ '.csv'

This file stores an array with the Zenith Angle of Arrival for each scatter for this MS at this point of its route.

9. path+'/THETA_AOD_' +str(MS_num)+ '_' +str(point_number)+ '.csv'

This file stores an array with the Zenith Angle of Departure for each scatter for this MS at this point of its route.

10. path+'/PHI_AOA_rays_' +str(MS_num)+ '_' +str(point_number)+ '.csv'

This file stores an array with the Azimuth Angle of Arrival for each ray of each scatter for this MS at this point of its route.

11. path+'/PHI_AOD_rays_'+ str(MS_num)+ '_' +str(point_number)+ '.csv'

This file stores an array with the Azimuth Angle of Departure for each ray of each scatter for this MS at this point of its route.

12. path+'/THETA_AOA_rays_'+str(MS_num)+ '_' +str(point_number)+ '.csv'

This file stores an array with the ZenithAngle of Arrival for each ray of each scatter for this MS at this point of its route.

13. path+'/THETA_AOD_rays_' +str(MS_num)+ '_' +str(point_number)+ '.csv'

This file stores an array with the Zenith Angle of Departure for each ray of each scatter for this MS at this point of its route.

14. path+'/Xpol_'+str(MS_num)+ '_' +str(point_number)+'.csv'

This file stores an array with the Cross Polarization of each scatter for this MS at this point of its route.

15. path+'/H_usn_'+str(MS_num)+ '_' +str(point_number)+'.npy'

This file with extension npy stores a multidimensional numpy array. These types of files are saved and read with the corresponding numpy save and load methods. This file stores a 3D array with the following dimensions:

usize = self.MS_antenna.get_number_of_elements()

ssize = self.BS_antenna.get_number_of_elements()

H_usn = np.zeros((usize,ssize,n_clus+4), dtype=complex)

where n_clus is the number of cluster of the FastFading model.

Each element of this array stores a channel matrix coefficient. The channel matrix coefficients are computed for this MS in this point of its route and for each cluster, Rx antenna element and Tx antenna element.

ChannelPerformance

This class stores the following information in the following files for each MS:

1. path+'/los'+str(ms)+'.csv'

2. path+'/snr'+str(ms)+'.csv'

This file stores an array with the signal to noise ratio in each point of the MS route.

3. path+'/snr_pl'+str(ms)+'.csv'

This file stores an array with the signal-to-noise ratio in each point of the MS route. The difference with the previous one is that only the path losses is computed for the snr.

4. path+'/snr_pl_shadow'+str(ms)+'.csv'

This file stores an array with the signal-to-noise ratio at each point of the MS route. The difference with the previous one is that only the path losses and the shadowing are computed for the snr.

5. path+'/times'+str(ms)+'.csv'

This file stores an array with the times of the simulation at each point of the MS route.

6. path+'/spectral_eff' +str(ms)+ '.csv'

This file stores an array with the spectral efficiency at each point of the MS route.

7. path+'/rxpsd' +str(ms)+ '.csv'

This file stores an array with the received power spectral density at each point of the MS route.

8. path+'/linear_losses' +str(ms)+ '.csv'

This file stores an array with the path losses in linear scale at each point of the MS route.

9. path+'/positions' +str(ms)+ '.csv'

This file stores an array with the positions at each point of the MS route.

10. path+'/G' +str(ms)+ '.csv'

This file stores an array with the beamforming gain at each point of the MS route.

11. path+'/H_f' +str(ms)+ '.npy'

This file with extension npy stores a multidimensional numpy array. These types of files are saved and read with the corresponding numpy save and load methods. This file stores a 3D array with the Fourier Transform of the channel matrix that has the following dimensions:

```
usize = self.MS_antenna.get_number_of_elements()
ssize = self.BS_antenna.get_number_of_elements()
H_us_f = np.zeros((freq_band.n_prbs, usize, ssize), dtype=complex)
```

Appendix: Source code of test.py

```
import scenarios as sc

import antennas as antennas

import channel_performance as cp

import frequency_band as fb

import numpy as np

import os,sys,erno

sys.path.append('./src')

sys.path.append('./src/gui')

sys.path.append('./src/graph')

SIM_NAME = "test" #the name of the simulation. Used to give the name

#to the directory where to store the simulation results

######## Build and configure the receive and transmit antennas

nMS=2 # number of antennas in the Mobile Station antenna array

nBS = 2 # number of antennas in the Base Station antenna array

antenna_gain = 8

aeBS = antennas.Antenna3gpp3D(antenna_gain) # Build the Base Station antenna element

aBS  = antennas.AntennaArray3gpp(0.5, 0.5, 1, nBS, 0, 0, 0, aeBS, 1) # Build the Base Station antenna array

aeMS  = antennas.AntennaIsotropic(antenna_gain) #Build the Mobile Station antenna element

aMS  = antennas.AntennaArray3gpp(0.5, 0.5, 1, nMS, 0, 0, 0, aeMS, 1) # Build the MS antenna array

#############################################################

###### Build the steering vectors for beamforming according the the BS and MS positions

wBS = aBS.compute_phase_steering (0,np.pi/2,0,0)

""" BS steering vector for baemforming"""

wMS = aMS.compute_phase_steering (np.pi,np.pi/2,0,0)

""" MS steering vector for baemforming"""

#########################################################

######## Build the scenario

fcGHz = 30  # Scenario frequency in GHz

# The next variables define the maximum and minimum x and y coordinates of the system (in meters)

posx_min = -100

posx_max = 100

posy_min = -100

posy_max = 100

grid_number = 25 # The number of elements in the grid. In order to define the parameters of the scenario PyWiCh generate a grid of correlated parameters. The grid has grid_number x  grid_number points.

BS_pos = np.array([0,0,20]) # The position of the Base Station device in the scenario.

Ptx_db = 30 # The Transmit power of the BS in dbm

force_los = 2 # The Line of Sight condition. If variable is 0 Non LOS is forced in all points of the scenario. If it is 1 LOS is forced in all scenario. If its value is 2 the LOS condition come from the scenario probability model.

scf=  sc.Scenario3GPPUmi(fcGHz, posx_min,posx_max, posy_min, posy_max, grid_number, BS_pos, Ptx_db,True,force_los)

# The previous instruction creates a  3gpp Umi scenario Model."""

#########################################################

########## Build the OFDM frequency band

freq_band =  fb.FrequencyBand(fcGHz=fcGHz,number_prbs=81,bw_prb=10000000,noise_figure_db=5.0,thermal_noise_dbm_Hz=-174.0)

freq_band.compute_tx_psd(tx_power_dbm=30) # Sets the BS tranmision power and build the power spectral density of each subchannel of the OFDM systems.

#########################################################

 #### Build the channel performance object to runs the simulation  and gets the results after the simulation

performance  = cp.ChannelPerformance()

""" Channel performance object"""

#########################################################
```

```
###################### Mobile stations routes configuratiom
n_MS = 1 # number of MSs in this simulation
positions = np.empty(shape=(n_MS),dtype = object) # The positions of the route of all MSs
mspositions1= np.array(([10,10,2],[20,10,2])) # The positions of the MS 1. It has only two points in its route.
positions[0] = mspositions1
times  = np.empty(shape=(n_MS),dtype = object) # The times of each point in the MS route
timesMS1 = np.array(([0,0.01])) # Th first point of the route is in simulation time 0 s and the second in the simulation time 0.1 s.
times[0] = timesMS1
path = "./data/"+SIM_NAME  # The path to save the results of the simulation.
try:
    os.makedirs(path)
except OSError as e:
    if e.errno == errno.EEXIST:
        print('Error: ' + path+ '  Directory exists. Directory not created. Please select another name or delete the directory')
    else:
        print('Exception!',str(e)+' occurred.' )
performance.compute_path(scf, freq_band, aMS,aBS,positions,times,force_los,path,mode=2,scatters_move=False,move_probability=0,v_min_scatters=0,v_max_scatters=10)
#########################################################
```