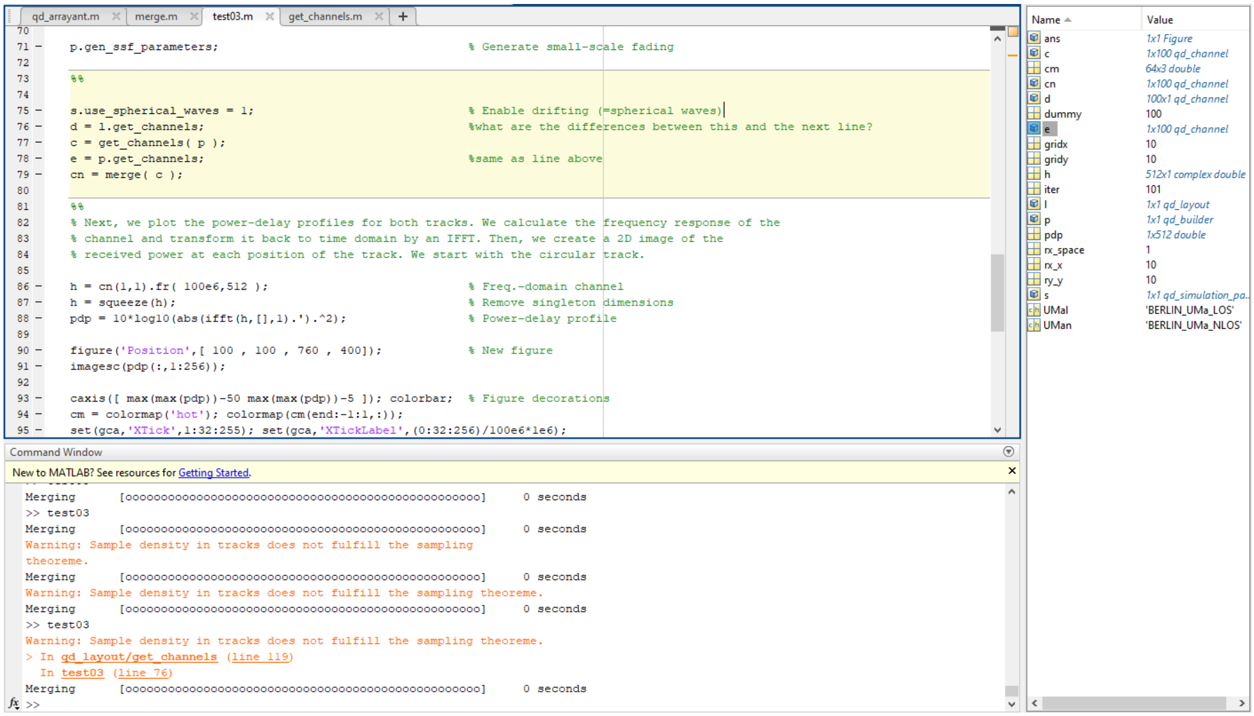
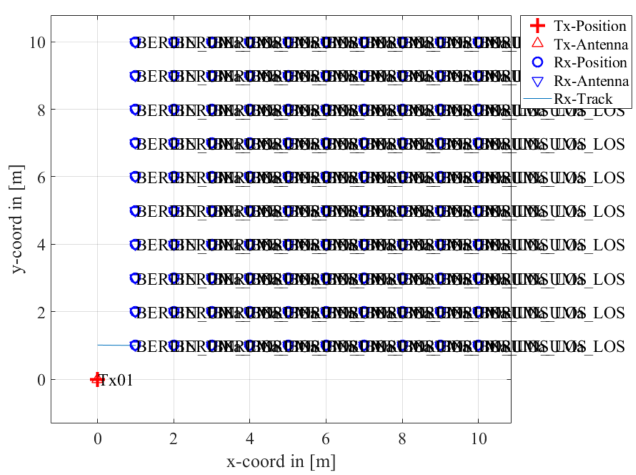
“test03” is a modified version of included tutorial “t07_time_evolution”

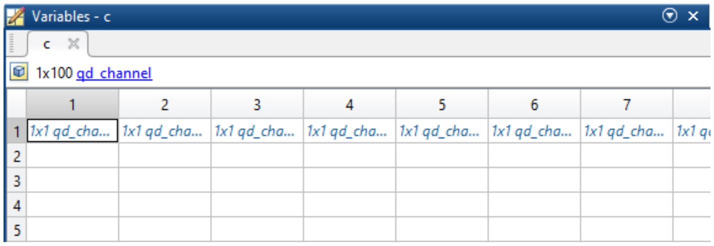


Current status of “test03” in github repo:

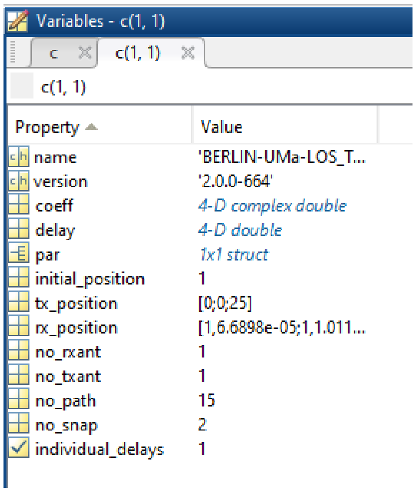
Able to manipulate file to form grids of receivers. The text seen below in the image is just confirmation that each receiver is acting on the same “scenario.”

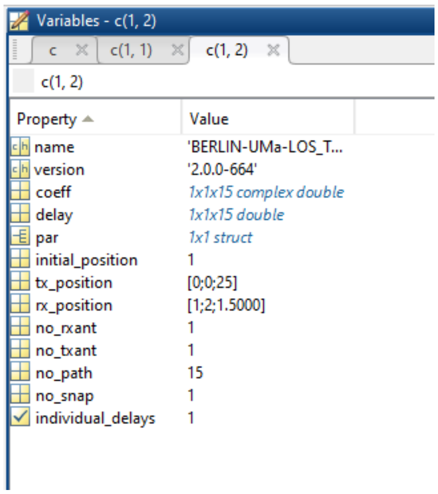


The qd_builder class is what generates the channel coefficients. Qd_channel objects hold the data for the coefficients. I saw two different ways of extracting the coefficients from the quadriga tutorials, seen above in lines 76/77. However, these different procedures produce different coefficients. Since the documentation suggests that the qd_builder objects are what generates the coefficients, I’ve been trusting the coefficients generated from **‘p’** in the above code.



Looking specifically at the qd_channel object created on line 77, **c** has 100 sub qd_channels (which corresponds to each of the 100 receivers of the test grid).



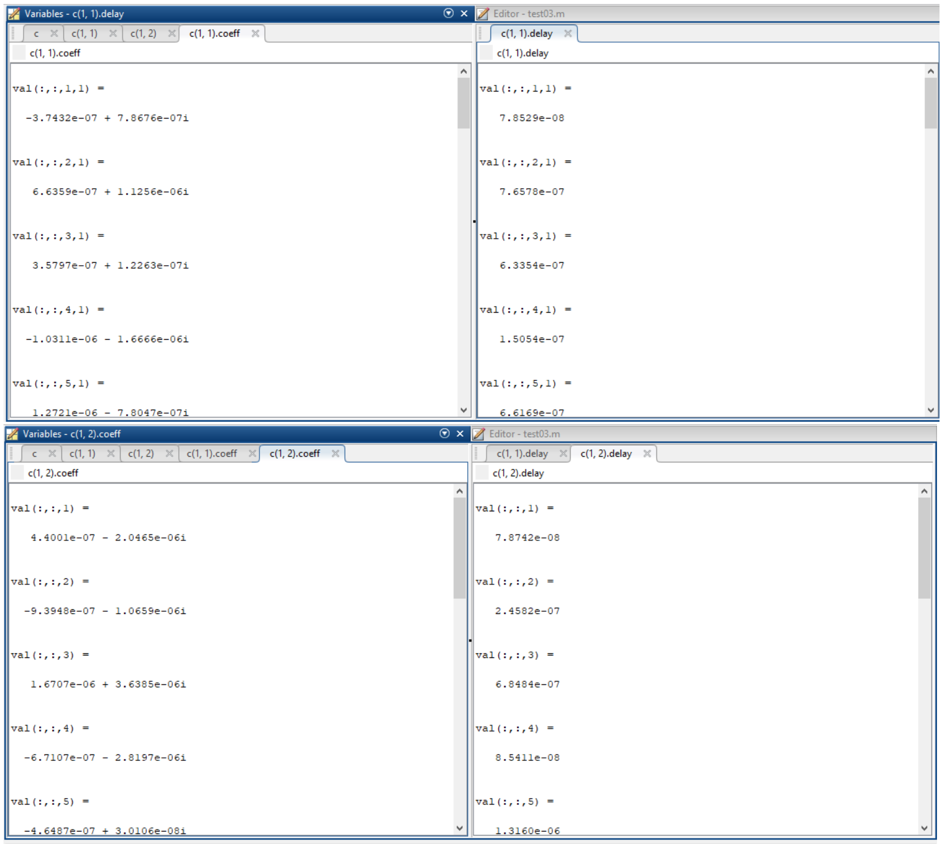


Looking at 2 different qd_channels indexed under **c**, we can see the properties of the objects. Copied from the quadriga documentation:

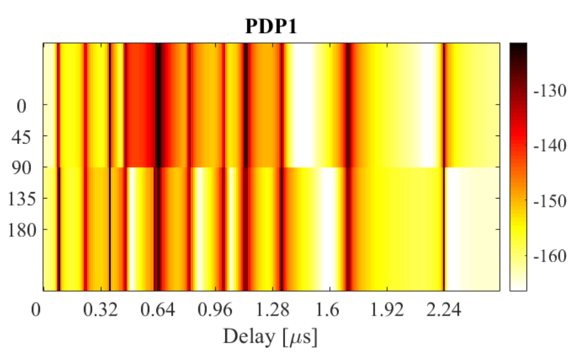
Properties

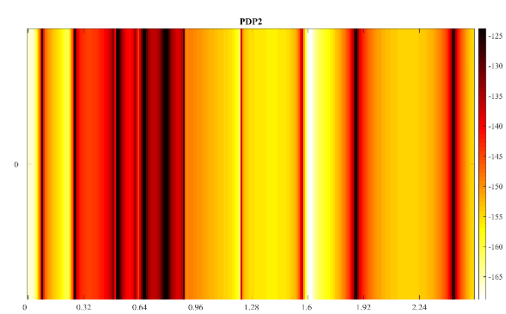
name	Name of the 'channel' object. This string is a unique identifier of the 'qd_channel' object. The 'qd_builder' creates one channel object for each MT, each Tx and each segment. They are further grouped by scenarios (propagation environments). The string consists of four parts separated by an underscore '_'. Those are: • The scenario name from 'qd_track.scenario' • The transmitter name from 'qd_layout.tx_name' • The receiver name from 'qd_layout.rx_name' • The segment number After 'channel.merge' has been called, the name string consists of: • The transmitter name from 'qd_layout.tx_name' • The receiver name from 'qd_layout.rx_name'
version	Version number of the QuaDRiGa release that was used to create the 'qd_channel' object.
coeff	The complex-valued channel coefficients for each path. The indices of the 4-D tensor are: [no_rxant , no_txant , no_path , no_snapshot]
delay	The delays for each path. There are two different options. If the delays are identical on the MIMO links, i.e. 'individual_delays = 0', then 'delay' is a 2-D matrix with dimensions [no_path , no_napshot]. If the delays are different on the MIMO links, then 'delay' is a 4-D tensor with dimensions [no_rxant , no_txant , no_path , no_snapshot].
no_rxant no_txant no_path no_snap	Number of receive elements (read only) Number of transmit elements (read only) Number of paths (read only) Number of snapshots (read only)
par initial_position	Field to store custom variables. The snapshot number for which the initial LSPs have been generated. Normally, this is the first snapshot. However, if the user trajectory consists of more than one segment, then 'initial_position' points to the snapshot number where the current segment starts. For example: If 'initial_position' is 100, then snapshots 1-99 are overlapping with the previous segment.
tx_position rx_position	Position of each Tx in global cartesian coordinates using units of [m]. The receiver position global cartesian coordinates using units of [m] for each snapshot.
individual_delays	Indicates if the path delays are identical on each MIMO link (0) or if each link has a different path delay (1).

My concern/speculation is that the coeff/delay of the different entries of the qd_channel are of different data types. Delay and coeff should 1-to-1 correspond to each other (which they do), but I do not know why they may be differing across different receivers.

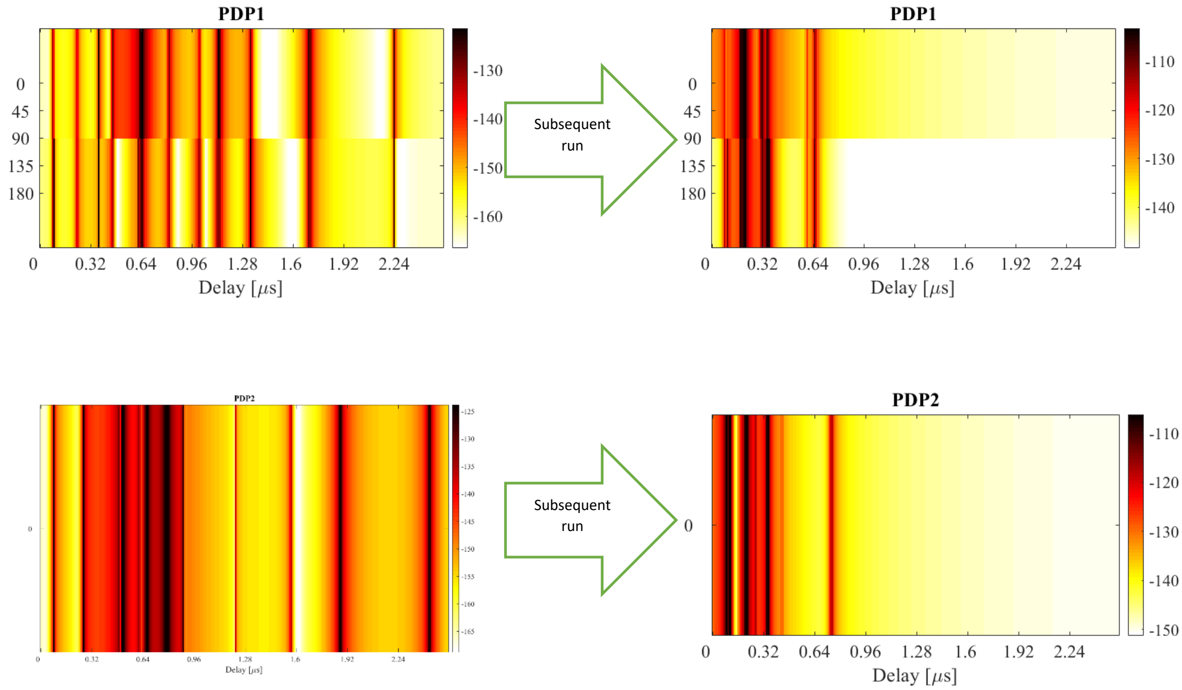


Lines 81 and on are purely from the example/tutorial, displaying “power-delay profiles.” I have the graphs displayed be the first two coeff values of **c**, which can be seen below, but all should be able to be displayed. I haven’t put much thought/effort into lines 81+, but I didn’t want to just remove if in case this could be useful for maybe displaying pdp across the grid area





Something I noticed though across multiple iterations… the pdp is changing. I don’t think there’s a randomization parameter included in the example, but I can’t think of any other reason why the graphs would be changing across subsequent runs of the code


PDP1
0
45
90
135
180
-130
-140
-150
-160
0
0.32
0.64
0.96
1.28
1.6
1.92
2.24
Delay [μs]
Subsequent run
PDP1
-110
-120
-130
-140
Delay [μs]
Subsequent run
PDP2
0
-110
-120
-130
-140
-150
Delay [μs]