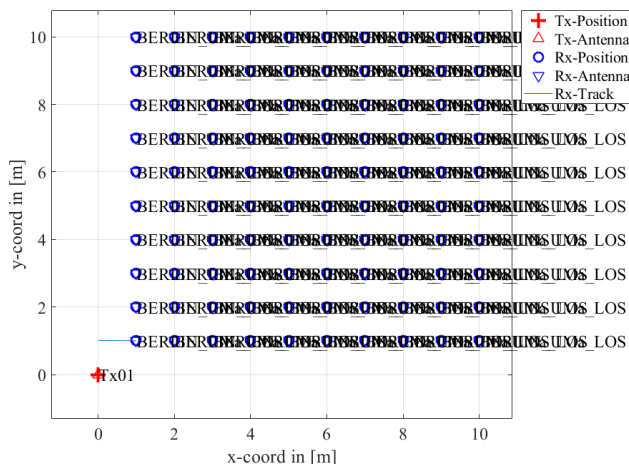


“test03” is a modified version of included tutorial “t07_time_evolution”

The screenshot shows the MATLAB R2020a interface. The main window displays the `test03.m` script. The script includes comments and code for generating small-scale fading, enabling drifting, and plotting power-delay profiles. The Command Window shows the execution of `test03`, with warnings about sample density in tracks not fulfilling the sampling theorem. The Variable Editor on the right shows the values of variables defined in the script, such as `ans`, `c`, `cm`, `cn`, `d`, `dummy`, `e`, `gridx`, `gridy`, `h`, `iter`, `i`, `p`, `pdp`, `rx_space`, `rx_x`, `rx_y`, `s`, `UMa1`, and `UMa2`.

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.

Variables - c							
c							
1x100 qd_channel							
	1	2	3	4	5	6	7
1	1x1 qd_cha...	1x1 qd_cha...	1x1 qd_cha...	1x1 qd_cha...	1x1 qd_cha...	1x1 qd_cha...	1x1 qd_cha...
2							
3							
4							
5							

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).

Variables - c(1, 1)		Variables - c(1, 2)	
c(1, 1)		c(1, 2)	
Property	Value	Property	Value
name	'BERLIN-UMa-LOS_T...	name	'BERLIN-UMa-LOS_T...
version	'2.0.0-664'	version	'2.0.0-664'
coeff	4-D complex double	coeff	1x1x15 complex double
delay	4-D double	delay	1x1x15 double
par	1x1 struct	par	1x1 struct
initial_position	1	initial_position	1
tx_position	[0;0;25]	tx_position	[0;0;25]
rx_position	[1,6.6898e-05;1,1.011...	rx_position	[1;2;1.5000]
no_rxant	1	no_rxant	1
no_txant	1	no_txant	1
no_path	15	no_path	15
no_snap	2	no_snap	1
individual_delays	1	individual_delays	1

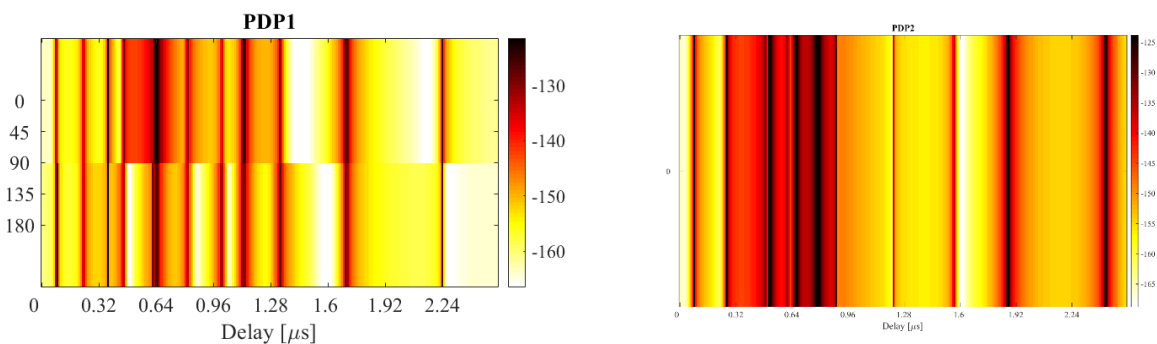
Looking at 2 different qd_channels indexed under **c**, we can see the properties of the objects. Copied from the quadriga documentation:

Properties	
name	<p>Name of the 'channel' object.</p> <p>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:</p> <ul style="list-style-type: none"> • 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 <p>After 'channel.merge' has been called, the name string consists of:</p> <ul style="list-style-type: none"> • 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	<p>The delays for each path.</p> <p>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_snapshot]. 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].</p>
no_rxant no_txant no_path no_snap	<p>Number of receive elements (read only)</p> <p>Number of transmit elements (read only)</p> <p>Number of paths (read only)</p> <p>Number of snapshots (read only)</p>
par initial_position	<p>Field to store custom variables.</p> <p>The snapshot number for which the initial LSPs have been generated.</p> <p>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.</p>
tx_position rx_position	<p>Position of each Tx in global cartesian coordinates using units of [m].</p> <p>The receiver position global cartesian coordinates using units of [m] for each snapshot.</p>
individual_delays	Indicates if the path delays are identical on each MIMO link (0) or if each link has a different path delay (1).

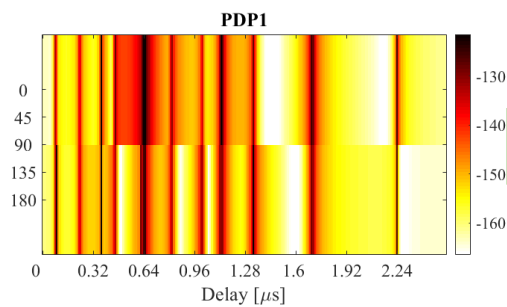
My concern/speculation is that the coeff/delay of the different entries of the qd_channel are of different data types. Delay and coeff should 1-to-1 correspond to each other (which they do), but I do not know why they may be differing across different receivers.

The image shows two MATLAB variable editor windows. The top window displays the variable `c(1,1).coeff` with a 5x1 column vector of values: `-3.7432e-07 + 7.8676e-07i`, `6.6359e-07 + 1.1256e-06i`, `3.5797e-07 + 1.2263e-07i`, `-1.0311e-06 - 1.6666e-06i`, and `1.2721e-06 - 7.8047e-07i`. The bottom window displays the variable `c(1,2).coeff` with a 5x1 column vector of values: `4.4001e-07 - 2.0465e-06i`, `-9.3948e-07 - 1.0659e-06i`, `1.6707e-06 + 3.6385e-06i`, `-6.7107e-07 - 2.8197e-06i`, and `-4.6487e-07 + 3.0106e-08i`. Both windows also show the corresponding `.delay` arrays.

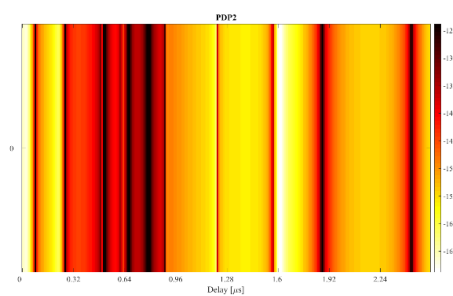
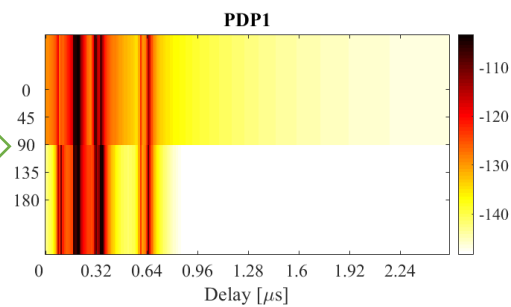
Lines 81 and on are purely from the example/tutorial, displaying “power-delay profiles.” I have the graphs displayed be the first two coeff values of `c`, which can be seen below, but all should be able to be displayed. I haven’t put much thought/effort into lines 81+, but I didn’t want to just remove 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



Subsequent
run



Subsequent
run

