Final-year Project:

Deadlines:

- Oral presentation (**10%**) before Christmas break: describe your background, progress and future plans for the project
- Interim report (**10%**): to be submitted before the first day of the Spring semester, and it should describe the progress that has been made
- Conference research paper (**15%**): to be done and submitted by week 8 of Spring semester (mainly assesses your ability to present and communicate ideas and results)
- Final project assessment (**65%**): two members of staff (one of them being your academic advisor) read your report, and assess your presentation… then they agree on a grade…
- **Deadlines**: project presentation **Week 11** (November **22nd** to **26th**)
  Interim report is due by the **16th** of January…

22/09/2021:
Final-year Project:

- Worth 25 credits
- Requires research and you will almost certainly need to make *references* (learn how to asap)
- Research: "new" knowledge or perspective acquired;
- Types:
  Blue sky: no clear application, and driven by curiosity… there's a risk that no tangible results will be had…
  Fundamental research (basic or *pure*): increase or diversify the understanding of already-understood ideas or concepts
  Directed fundamental: (has a medium risk)
  Applied research: definite applications in mind; decent idea of *how* to go about doing the research and seeking answers to the questions you have in mind… (has a low risk)
- Read up, or research **first!** Understand the **current solutions**; maybe **propose and design** a **better solution**; built or simulate or analyse new proposed solution;
- OR you *could* analyse a current design, and try to gain new insights or perspectives on the current solutions…

Good research:

- **Novelty**: new things (maybe in a technical sense); they should hopefully improve things *and* should not be too obvious…
- **Technically solid**
- **Mathematically sound**
- **Assumptions CLEARLY stated…**
- **No "holes" in your arguments**
- **No contradictions**
- **Convincing results:** present them **well**, and in **many ways**…
- **EXPLAIN** *incomplete* or *unexpected* results

Some extra (and *not so obvious*) aspects of **good research**:

- Objectivity (intellectual honesty)
  Be unbiased; offer ***all*** results, even if they paint your approach or method unfavourably…
- Imagine you are writing to your **peers** - be precise! And don't oversimplify…

Journals and Conferences:
- Some are good and some are bad
- IEEE conferences are usually good and trustworthy…
- IEEE journals are decent
- We have to write a conference paper (talk to your supervisor)
  Which conference? Ask supervisor

You have 4 bases for making a statement:
- You are given a reference
- You are making a solid logical argument that supports your statement (experimental data, or mathematical implication)
- You are stating well-known or obvious ideas or concepts: Ohm's Law; aspects of the multivariable calculus
- When you are stating your opinion (don't do this so often), and **make it clear** when you are doing this… "In the Author's opinion… "

30/09/2021:
- I've got a feel for what OFDM aims at doing, the principle on which it's based
- I'll review *where* and *when* FDM and OFDM are used - this will give meaning or weight to the further study I do, and avoid the scenario in which I feel I'm studying or working with arbitrary equations...
- I'll need to review FDM, and the maths underlying it…
- Some questions:
  What key topics must be reviewed if there are any doubts about my understanding of them?
  Are there any very useful softwares or applications which will help to visualise the ideas or concepts?
  Are there clear milestones which I should be working towards?

03/10/2021:
Questions that need answering before the day is out:
What is radar, and how does it work?
What are the different forms of radar?
What are the key performance measures used for radar systems?
- Q1: answered, in the basic sense at least
- Q2: answered, again, in a basic sense
- By using information *about* the EM wave that is echoed back, the distance of the object *from* the radar can be deduced. If the object is moving, then its speed can be deduced…
- For long-range radar, low frequencies are preferable - this is because as the frequency of an EM wave decreases, the *wavelength* increases… This increase in wavelength leads to a decrease in the *path loss*… which is desirable…

- The power of the echo depends on the cross section of the radar's target (the object in the path of the radio wave)
- Radar stands for: RAdio Detection And Ranging
- The round-trip time is the time it takes an EM wave to be emitted, reflect off an object, and return to the radar receiver

  $$\Delta T = \frac{2R}{c}$$

  In the above, *R* is the range of the target (assume it's at rest here).
  c is the speed of light…
  ΔT is the round-trip time

- One of the metric for radar performance would be the *maximum distance* at which an object can be detected
- Another would be the *resolution* or *accuracy* with which an object can be identified… how *exact* can the radar be…
- Intensity of waves is often represented with the letter *Q*…

05/10/2021:

- *Baud rate* and the *symbol rate* seem to be the same. Now, each symbol represents a certain *number* of bits… so the *bit rate* can be calculated (or at least approximated) by:

  Bit Rate (BR) = (No. of bits per symbol)*(symbol rate)*(No. of channels)

  The unit of the symbol rate is the *baud* (Bd)
- The *symbol time*, T_s, is equal to:

  $T_s = (1/f_s)$, where f_s is the *symbol rate*

  Note, the *symbol time* is also sometimes called the **unit interval**…

- Some questions that we need to answer:
  Baseband signal?
  Double-sideband signal?
  What does an N-point DFT or IDFT actually mean?
  What are channel coefficients, *h*?
  How does convolution in one domain relate to multiplication in another?

- Basic Plan for following:

  (1) By 12/10/2021 have answered the above questions

12/10/2021:

- Channel encoding: the channel encodes the data, typically by adding **redundancy bits**, so as to protect against bits being corrupted due to poor channel conditions
- There are two main protocols: FEC (Forward Error Control), and ARQ (Automatic Repeat Request)
- In an (FEC) protocol, redundancy bits are added so that the receiver (Rx-er) can detect and correct the error - examples include:

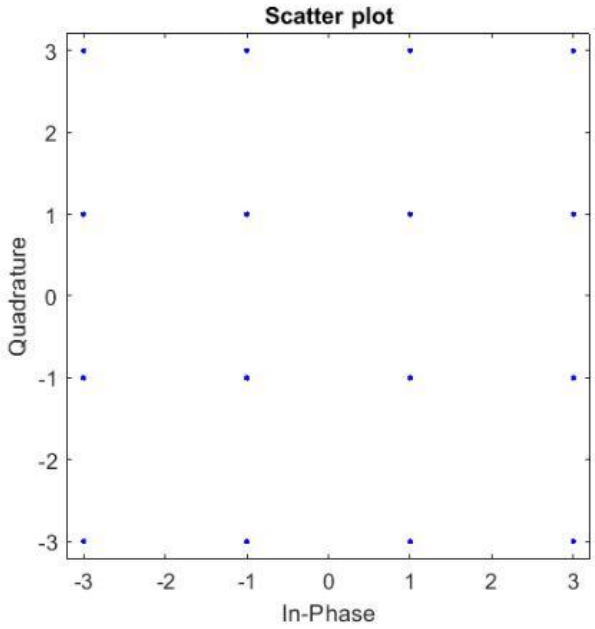
Repetition codes (where each information bit is transmitted 3 times - the Rx-er uses majority voting to decide whether a received bit is more likely a 1 or 0)
Hamming codes (where 3 redundancy bits are added to each block of 4 information bits, forming a *codeword* - Hamming codes allow for the detection and correction of single-bit errors per 7-bit block. Also, since Hamming code works with blocks of bits, it's referred to as a *block code*)

- In an (ARQ) protocol, errors are simply detected, and the Rx-er requests a packet of bits to be resent - with the hope being that it's unlikely the bits will be indefinitely corrupted. ARQ protocols typically use much less redundancy bits, as the redundancy bits added are merely to allow the Rx-er to identify that there has been an error… not to *also* help the Rx-er figure out *where* the error is… less redundancy bits leads to higher data rates and better bandwidth utilisation…
- An example of a non-block encoding scheme is *convolutional encoding* (read up on this one)
- Frequency modulation: each user's information signal is modulated onto an assigned carrier of a specific frequency. These are then combined to form a single frequency-division multiplexed signal
- Sometimes the channel *itself* acts as the *combine*: think of different radio stations being allowed to transmit into the open air at the same time - provided each station is assigned a *different* frequency band, then there are no huge issues…
- I've plotted the output of a QAM modulator (simulated QAM modulator) in MatLab:

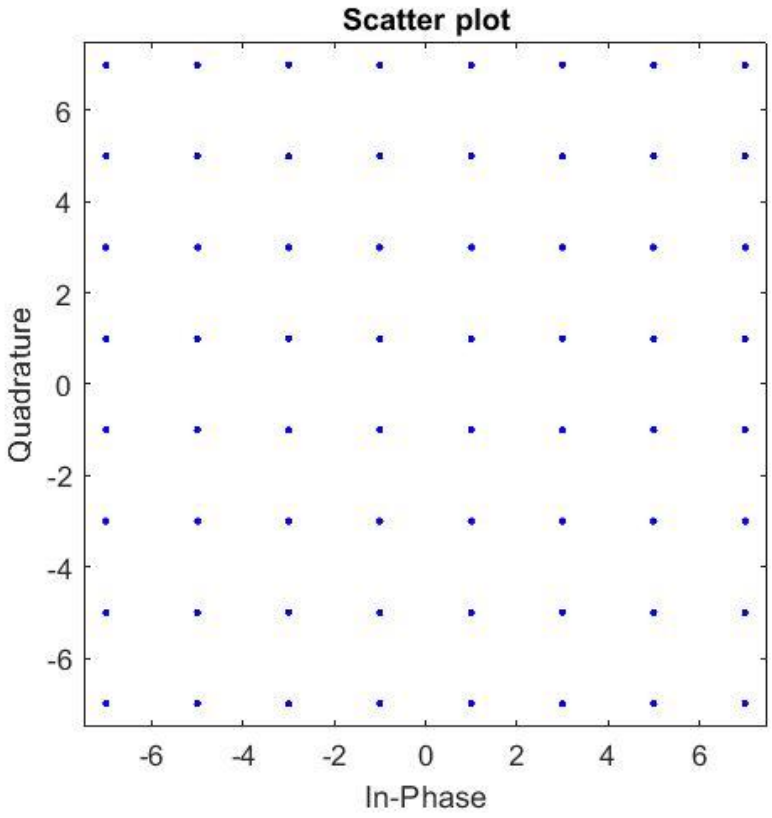
I was using order 16 QAM - meaning there were **16 symbols*

```
Symbol 1 is: -3.000000  3.000000i
Symbol 2 is: -3.000000  1.000000i
Symbol 3 is: -3.000000  -3.000000i
Symbol 4 is: -3.000000  -1.000000i
Symbol 5 is: -1.000000  3.000000i
Symbol 6 is: -1.000000  1.000000i
Symbol 7 is: -1.000000  -3.000000i
Symbol 8 is: -1.000000  -1.000000i
Symbol 9 is: 3.000000  3.000000i
Symbol 10 is: 3.000000  1.000000i
Symbol 11 is: 3.000000  -3.000000i
Symbol 12 is: 3.000000  -1.000000i
Symbol 13 is: 1.000000  3.000000i
Symbol 14 is: 1.000000  1.000000i
Symbol 15 is: 1.000000  -3.000000i
Symbol 16 is: 1.000000  -1.000000i
```

And the *Constellation graph* is:



If we make the order, M, equal to 64, we get:



Cool stuff!

- AWGN: stands for *Additive White Gaussian Noise*… it provides one way to model a communication channel…

- If we've a set of, say, 64 symbols, then we can easily generate 1000 such symbols (randomly), using the randi( ) function in MatLab:

```
M = 64; % Modulation order of the QAM modulation scheme
%x = (0:M-1)'; % Our input signal...
x = randi([0 M-1], 1000, 1)
disp(x) % Let's display the signal x...
```

x = 1000×1

	1
1	6
2	17
3	35
4	61
5	61
6	10
7	62
8	61

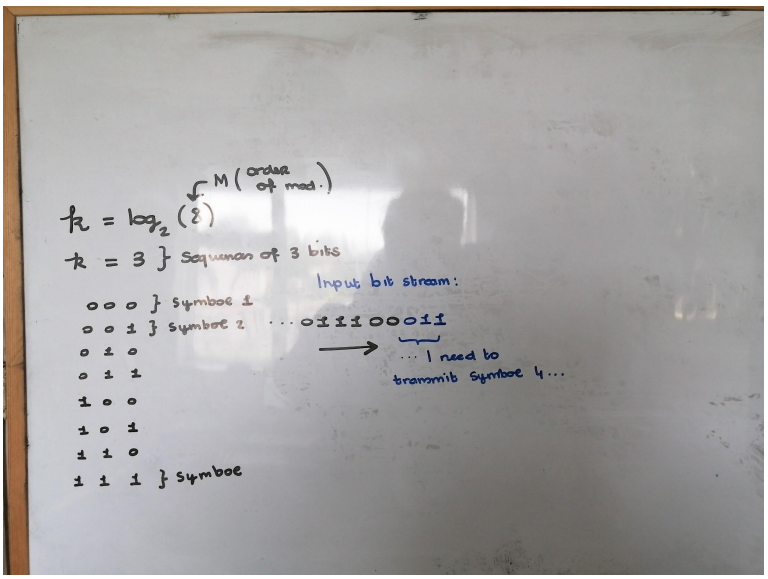
```
    61
    31
    51
     9
    26
    58
    50
    61
    41
     2
    54
    59
    43
    48
```

- I can create a input data bit stream nicely now in MatLab:

```
k = 6
data = 1×3000
     1     1     0     0     1     0     1     0     1     0     1     0     1     0     0     1     0     0     1     0 ...
```

The *k = 6* signifies that since 64-QAM is being employed, each *symbol* will represent 6 bits ($\log_2(64)$):

```
M = 64; % 64 symbols...
k = log2(M) % Each symbol is representing 6 bits!
data = randi([0 1], 500*k, 1)' % Our bit stream
```



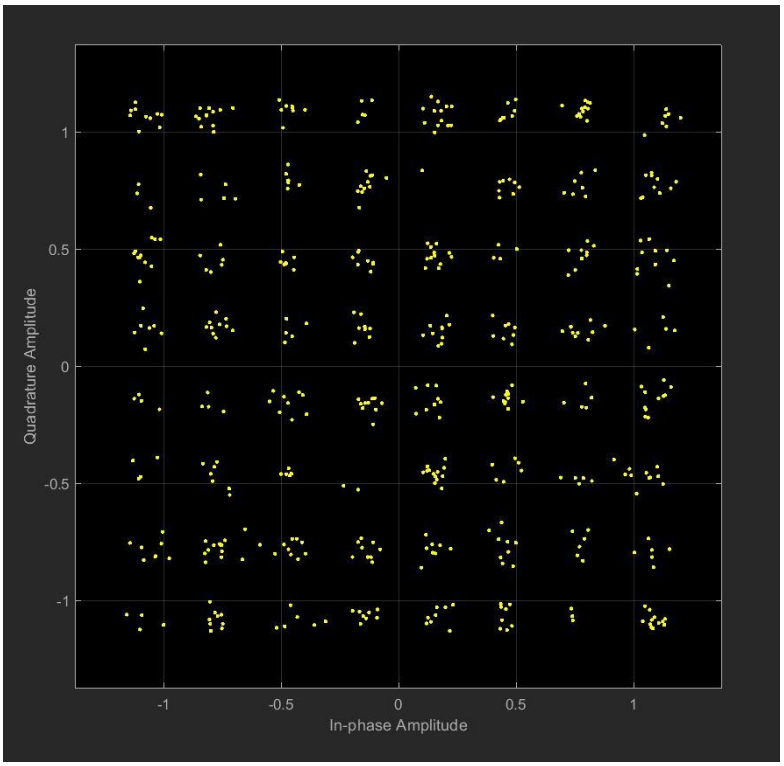
- We can module this input data bit stream
  My guess is that since M = 64, and the 'InputType' is bit, the 'qammod' function knows that 6 quantities of the input type, i.e. bits, will go towards each symbol…

```
txSig = qammod(data, M, 'InputType', 'bit')
```

qammod(x,M,Name1,Value1,Name2,Value2)

Enter a value for **Value**

- Below is shown the Constellation plot at the Rx, after the signal passes through a AWGN channel (64-QAM scheme being used):



13/10/2021:

- Scrambler: rearranges the data message (at the sender or Tx side) so that any Rx without an appropriately-set descrambler cannot make sense of the message - in this way, the scrambler helps protect against unauthorised data access…
- Scrambler circuits seem to often use *shift* registers…
- Scrambling can also help with improving the *timing extraction* for the Rx side - suppose we needed to transmit:

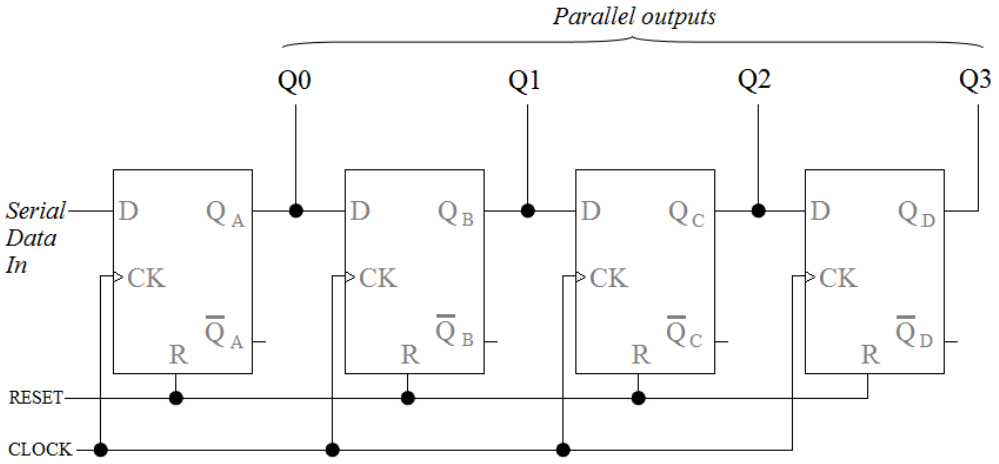
  00001111 (It might be hard for the Rx to extract the timing info. from this)

  But, if we "scramble" the bits up and send:

  01010101

  The Rx side will find it easier to extract the timing information from the signal…

- (SIPO) standing for (Serial-In-Parallel-Out); this block are often realised using a series of flip flops or gated latches:



serial-to-parallel converter

- For the time being, and merely to move ahead, I'm going to try implement a serial-to-parallel converter by using the reshape( ) function in MATLAB:

```
M = 64; % 64 symbols...
k = log2(M) % Each symbol is representing 6 bits!
data = randi([0 1], 500*k, 1)' % Our bit stream
parallel_data_block = reshape(data, [4, 750])'

A = [1 4 2 5 6 6 7 4]
B = reshape(A, [2, 4])
```

```
k = 6
data = 1×3000
       1     1     1     0     1     0     0     0 ...

parallel_data_block = 750×4
       1     1     1     0
       1     0     0     0
       0     0     1     1
       1     0     1     1
       1     0     1     1
       1     0     1     1
       1     0     1     0
       1     1     0     1
       1     0     1     0
       1     0     0     0
        .
```

- Since I'm using 64-QAM, I've decided to parse the 3000 serial bit stream into 500 packets of 6 bits, where each pack of 6 bits will map to one QAM symbol:

```
[r, c] = size(parallel_data_block)
modulated_data = zeros(500, 1);

for row = 1:r
    modulated_data(row, :) = qammod(parallel_data_block(row, :)', M, "InputType", "bit", "UnitAveragePower", true);
end

modulated_data
txSig = qammod(data, M, 'InputType', 'bit', 'UnitAveragePower', true)
```

```
modulated_data = 500×1 complex
    -0.7715 + 1.0801i
     0.1543 + 0.1543i
     1.0801 - 0.7715i
    -0.4629 - 0.4629i
    -0.4629 + 0.1543i
     0.7715 + 0.4629i
     1.0801 + 0.1543i
    -0.7715 + 0.1543i
    -0.7715 + 0.1543i
     0.1543 + 0.7715i
        ⋮

txSig = 500×1 complex
    -0.7715 + 1.0801i
     0.1543 + 0.1543i
     1.0801 - 0.7715i
    -0.4629 - 0.4629i
    -0.4629 + 0.1543i
     0.7715 + 0.4629i
     1.0801 + 0.1543i
    -0.7715 + 0.1543i
    -0.7715 + 0.1543i
     0.1543 + 0.7715i
        ⋮
```

In the example above, *modulated_data* is a matrix, containing the output from the 64 QAM modulator - with the input *to* the modulator being a row of the *parallel_data_block*…

- According to most block diagrams that I've seen, each of these complex values now goes into an IFFT block:

14/10/2021:

- *passive radar* systems: they use *broadcast* signals to map the locality… these are probably the closest to what has been proposed in my current project…

- Saves hardware by performing both at once (communication *and* radar)
- The frequency spectrum is used more effectively, as both radar and communications use the same band of frequencies (or *at least* much of their bands overlap)
- Research project “RadCom”...
- V2V (Vehicle-to-Vehicle) communication…
- ITS (Intelligent Transportation Systems - e.g. cars, trucks, boats, and more)
- Radars at the front of the car for adaptive cruise control (ACC), and collision avoidance...
- Radars on the side for “blind spot” detection (someone swerving into you from another lane on your right or left)
- To analyse something ***empirically*** means to draw conclusions or analyse it based mainly on observation and experience, as opposed to theory or pure logic…
- Big decision: to aim to *avoid* interference, **or** to aim to *deal* with it…

> In general, current research focuses on interference avoidance and mitigation techniques, which is exemplified by the results from the MOSARIM project, e.g. [61]. Here, OFDM is in fact considered as a method to cope with mutual interference, but further analysis is not given. One suggestion as to how to handle interference instead of avoiding it is given in [20], which is also the only publication which directly researches interference in OFDM radar networks. The paper suggests an interference mitigation technique, but only for the very specific scenario of one single interferer.

- SINR: Signal-to-Interference-and-Noise-Ration
  This is potentially a very useful quantity to measure performance of OFDM for different car-to-car configurations…
- NASA: electronic components on “Orbiter” that switch between comms. and radar functionality

2.3 Combined Communication and Radar systems

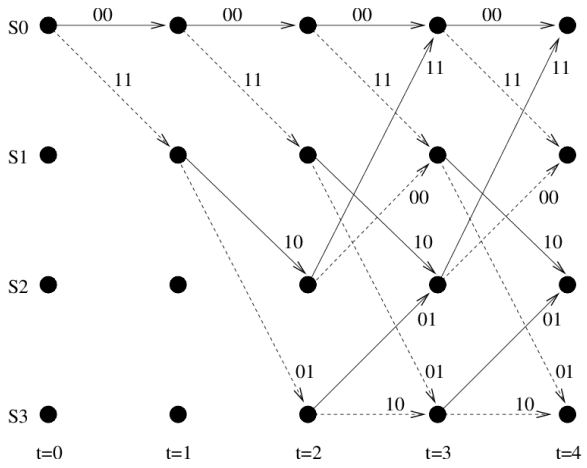
> The idea of combining communication and radar first came up during the developments triggered by the Second World War [76]. However, active development on combined systems is rare. One of the few examples of deployed combined systems was implemented in the NASA Space Shuttle “Orbiter” [77]. This system could switch between radar and communications functionality, but not perform both at the same time.

both results for the quality of the radar imaging as well as the data link [39, 40, 41, 42, 44]. However, they did fail to describe the signal processing aspect correctly and identify potential optimizations. Sturm and Wiesbeck were the first to suggest a simple signal processing algorithm for OFDM radar, which is identified as a two-dimensional periodogram in Chapter 3, and present some simulations and measurement results to prove the applicability [19, 18].

- Periodogram: gives you a very good idea of the *spectral density* of a signal
  It can be used to detect or hone in on the **dominant frequencies** of a signal - recall that for a time-varying signal, it's power (P) is proportional to the square of the amplitude…
- In a lot of the papers, questions of optimization or questions rooted in the theory of OFDM's application are usually not pondered… a lot of *empirical* analysis…
- **Mutual Interference** between radars operating in the same band, at the same time, and in the same network

19/10/2021:

- Convolutional coding uses shift registers, in tandem with *binary operations* to create *code bits*. Then the decoder on the Rx side uses the code bits, and knowledge about the state diagram on the Tx side to decode the code bits received. The state diagram on the Tx side can be represented also by a ***Trellis*** diagram:



- The **Hamming** distance is the **number of bits** in error between the received and transmitted message
- The **code rate** is the number of inputs bits divided by the number of **code** bits (output one):
  For a standard convolutional encoder, there are two code bits, say C_0 and C_1, and at each time step there is 1 input bit, so the code rate (*r*) is:

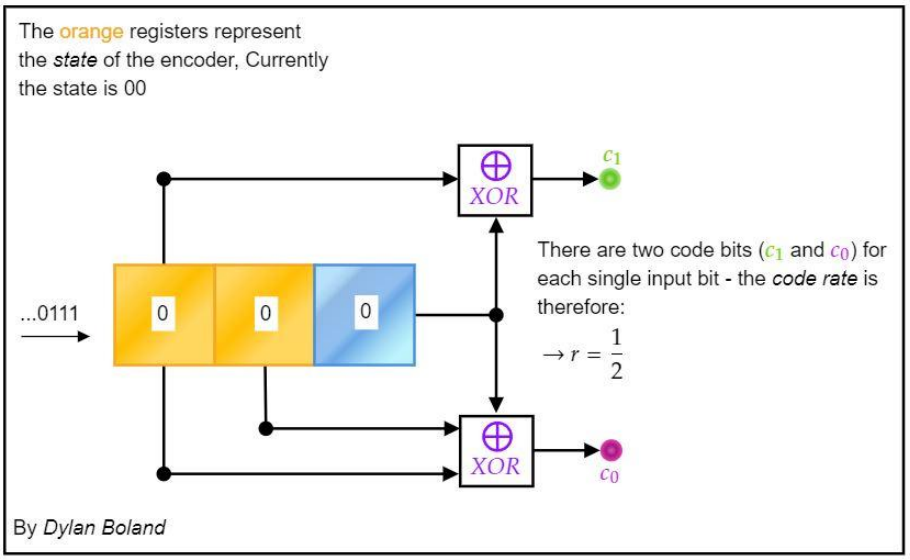
  ***r* = ½**

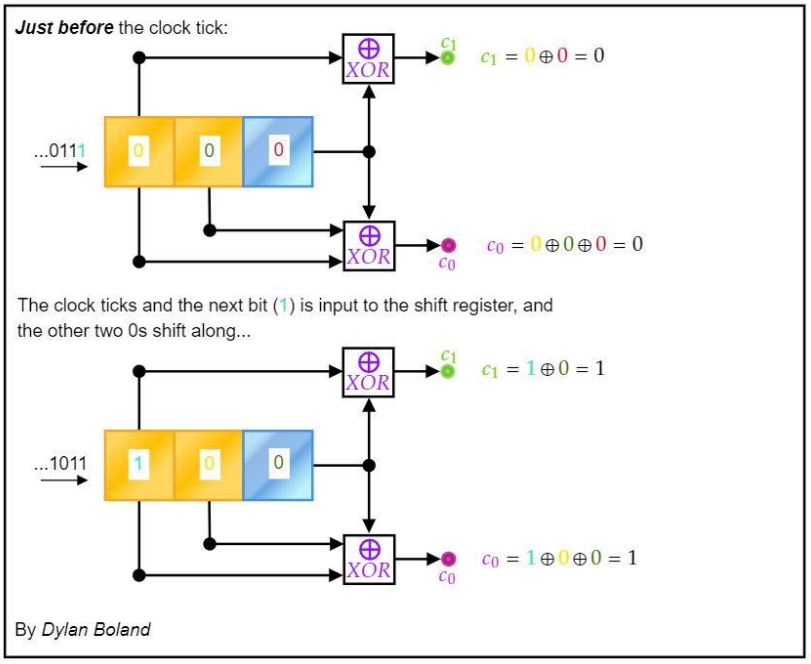
- On the Rx side, the ***Viterbi decoder*** finds the most likely path through the *Trellis* diagram, thereby allowing it to find the message bits that were most likely sent…

- Convolutional encoding is good for bit streams of **arbitrary length**!
- Input bits *convolve* to output ones, based on the logic of the encoder (modulo 2 division, or perhaps some other scheme!)
- Output bits depend not only on the current input, but also past input bits that are held or stored in memory - cool stuff!
- Three important parameters of a convolutional encoder: *k, n, r*

  k: the number of bits shifted into the encoder at each time step
  n: the number of code bits generated at the output for the input *k*
  r: the code-rate or code-bit rate - this equals: *k*/*n*

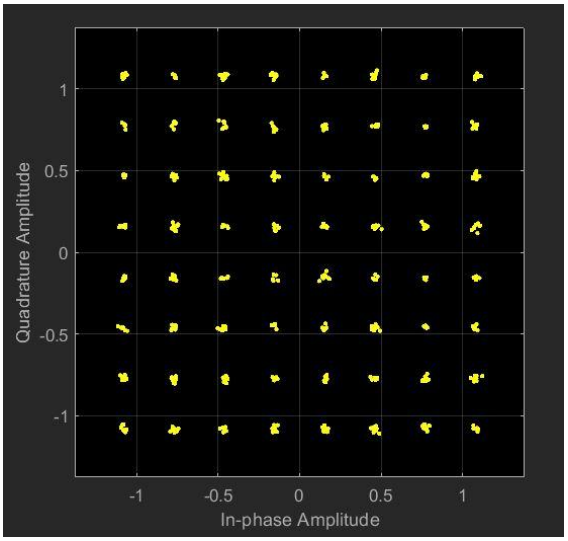
- Have understood a little better *how* the convolutional encoders work, and done up some diagrams to support my reading, and so that I can reference them later on:



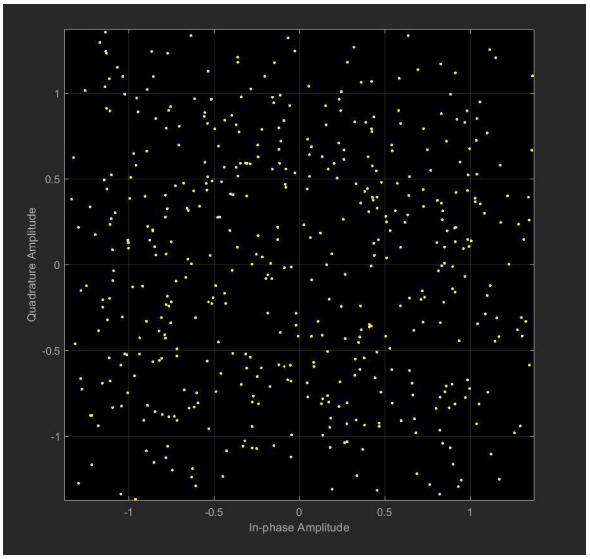


- Re-running the code that I had last time, I can verify again that SNR matters in the transmission of symbols (which represent bit configurations)

  High SNR (Signal is much stronger than noise):



  Low SNR (Signal is overpowered by the noise):



- I think I've been able to encode my 3072 bit data stream (variable being "*data*") with a *code rate* of ½, meaning the encoded data is 6144 bits:

```
% Let's create our encoder:
convEnc = comm.ConvolutionalEncoder;
encodedData = convEnc(data)
```

```
data = 3072×1
       1
       0
       1
       1
       1
       1
       1
       0
       0
       0
       ⋮

encodedData = 6144×1
       1
       1
       1
       0
       0
       0
       1
       0
       1
       0
       ⋮
```

If we look at the first bit in our data bit stream, it is a **1**. When this is loaded into the first register, it would lead to the following values for the code bits:

c_1 = 1 XOR 0 = 1

c_0 = 1 XOR 0 XOR 0 = 1

So the two-bit output sequence is {c_0, c_1} = {1, 1}

When the next 0 goes in, we get new values for c_0 and c_1:

c_1 = 0 XOR 0 = 0
c_0 = 0 XOR 1 XOR 0 = 1

So that two-bit output sequence is {c_0, c_1} = {1, 0}
That's what the first four bits of the encoded data are: 1110…

- **Pilot Symbols:** these can be used for *symbol* synchronization at the Rx side. A special *pilot symbol*, which is a complete OFDM symbol, is sent to the Rx side, which knows to look out for it. The pilot symbol is repeated at a certain rate. The Rx signal is then compared or correlated with the pilot symbol to find the *start* of the symbol transmission. This seems similar to how **start flags** and **end flags** are used in line codes - for example, 8-bit flags are used in data-packets to signal when the *payload* data is coming up…

  111000111 // This might be some start flag sequence…

- **The Cyclic Prefix:** this can also be used for symbol synchronization, as it's a known repetition of some part of the Rx signal which can be detected using the **autocorrelation**…
- A ***frequency bin*****:** small intervals on the frequency axis created when we go from continuous-time frequency to discrete-time frequency:

*Let's not forget: the *autocorrelation* function helps you find patterns in your data; it gives you a feeling about the similarity between data points at two different time flags (instances) in your data sequence or stream…

20/10/2021:

- Small time guard interval, T_G, is inserted before **every OFDM symbol**. This guard-time interval is a fraction of the symbol duration - say ¼ or ⅛:

> **Cyclic Prefix** In a final step, a *guard interval* is inserted before every OFDM symbol. It is of duration T_G, which is chosen as an integer fraction of the symbol duration. Typical values are $T_G/T = 1/4$ or $T_G/T = 1/8$; choosing a valid guard interval duration is discussed in Section 3.6. Of course, this increases the total OFDM symbol duration to $T_O = T + T_G$.
>
> Without a guard interval, a time-dispersive (i.e. frequency-selective) channel would leak energy from one OFDM symbol into the next, causing *inter-symbol interference* (ISI). A guard interval therefore prevents the loss of orthogonality between adjacent OFDM symbols. The most common way to create a guard time is to prepend a cyclic prefix (CP), which is a copy of the last T_G/T-th part of the OFDM symbol (see [94] for more details on cyclic prefixes and why they are used). For the rest of this work, the usage of a CP is assumed.

  If no guard-time interval were used, then a time-dispersive or frequency-selective channel might cause one symbol to interfere with the next, leading to **Inter-Symbol Interference**.
- The guard intervals help to maintain the orthogonality between the subcarriers.
- The *length* of the IFFT corresponds to the number of subcarriers you have… and we should, as recommended, make the length of the IFFT a power of two.
- The PAPR (Peak-to-Average-Power Ratio): this appears to be an important metric for OFDM

> ### 3.1.1 PAPR
>
> For OFDM systems in particular, the peak-to-average power ratio (PAPR) is an important metric. It is defined as
>
> $$\text{PAPR} = \frac{\max[|s(t)|^2]}{\text{mean}[|s(t)|^2]}, \qquad (3.11)$$
>
> i.e. as the peak power divided by the mean power of the signal.

- Notice the use of sections: 3.1.1, for example. As for diagrams, 1 decimal place is probably enough, as you'll have at most 8-10 diagrams (figures) per section:

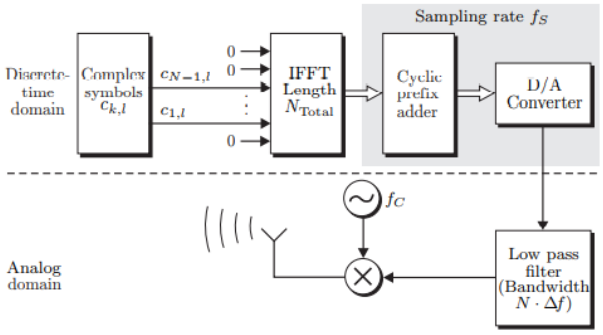


Figure 3.2: Block diagram of an OFDM transmitter

- We are modelling the channel, just to begin with, as a 3-tap FIR filter:

  h = {1, 0.4, 1} // These might not be realistic filter coefficients, but they're just for an example

  x = {1, 1, 0, 1, 0} // Imagine this is a signal that we are sending…

  For a 3-tap filter, the output will be:

  $y[n] = h_0x[n] + h_1x[n-1] + h_2x[n-2]$

  When the first bit is received, there *won't be* "past values" (*x*[n-1] or *x*[n-2]) for our signal *x*, as it's the first value or bit received - as a result, we are going to append to 0s to account for this:

  x = {0, 0, 1, 1, 0, 1, 0}

24/10/2021:

- There are a few **subcarrier types**: *data subcarriers* are used for data transmission; *pilot* or *reference-signal subcarriers* are used for channel estimation and coherent detection; *null* subcarriers are used for **guard bands**;
- I've got a basic method for imposing so-called *Hermitian symmetry* on the input to the IFFT, meaning the output of the IFFT is **real valued**…

```
ifft_op = 2050×1
   -0.0202
   -0.0066
    0.0189
   -0.0141
    0.0078
   -0.0000
   -0.0004
    0.0082
   -0.0357
   -0.0077
      ⋮
```

```
modulated_data = [0; modulated_data(1:end); 0; conj(modulated_data(end:-1:1))]
```

26/10/2021:

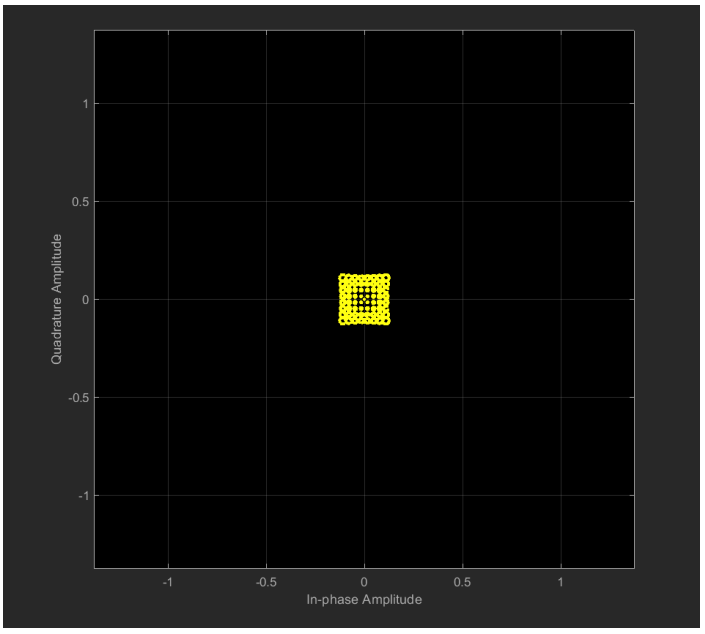
- The IFFT algorithm is applied to the frequency-domain subcarriers in order to produce an OFDM symbol in the time domain
- Guard intervals are added between OFDM symbols in the time domain…
- Having **orthogonal** subcarriers allows for more subcarriers per bandwidth, thereby leading to better or higher spectral **efficiency**…
- Each complex symbol going into the IFFT block describes both the **amplitude** and **phase** of the sinusoid of the corresponding subcarrier…
- Block of *N* IFFT outputs constitutes a single OFDM symbol…

27/10/2021:

- Path loss in the signal strength is **proportional to the square of the frequency** of the carrier wave…
- But higher frequencies allow for smaller antennas - as antenna size is related to the wavelength of the E.M. waves being transmitted… so the higher the frequency, the lower the wavelength, and the smaller the antenna needed... (in general!)
- The cyclic prefix makes the linear convolution of the channel appear as circular convolution the the DTF process block at the Rx side
- Watched the first lecture in the MIT OCW DSP course; I'll watch and study two of these per week from here on in…

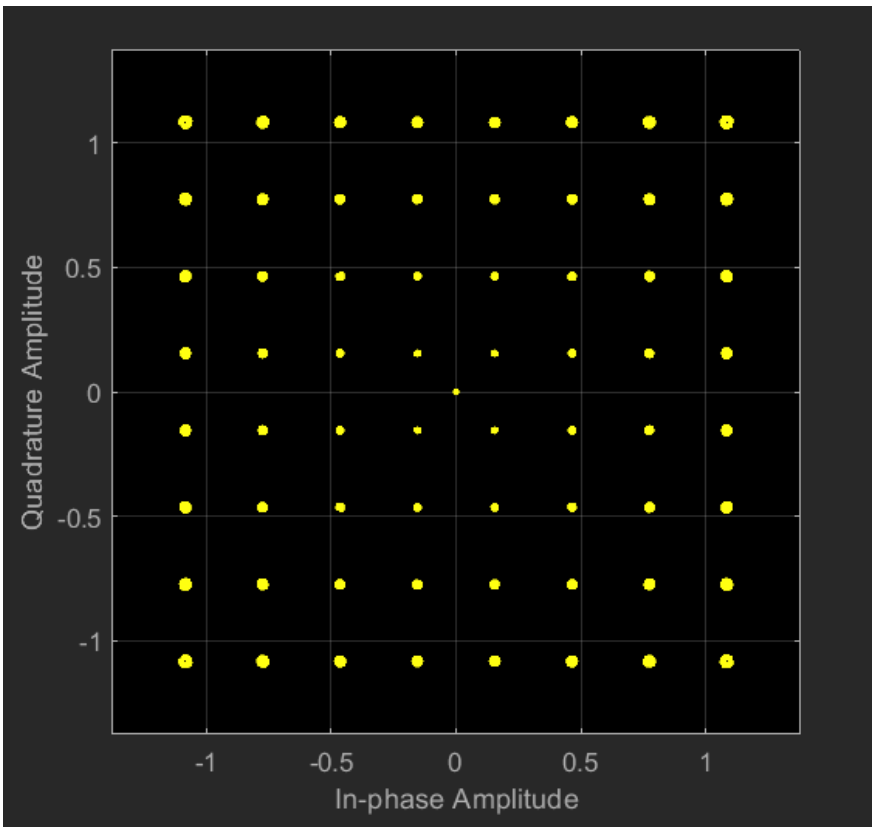
28/10/2021:

- I'm currently prefixing the last 500 values of the ifft output (real values) to the ifft output - so it's now 2550 elements in length. This means that the cyclic prefix is 500 in length… almost certainly a suboptimal value, but I'm just using it to move ahead and see what the effect looks like…
- It seems that on the Rx side, the cyclic prefix is removed, and *then* the values are passed into the FFT block…
- When I demodulated the data I saw the following:



:(

Changing the channel coefficients, especially h_0 from 0.1 to 1, gave what *appears to be* much better results! 😃



- It turns that my function, although it works, wasn't *completely* necessary - this is because the output *y* from the channel can be computed quickly using the *conv*( ) function in MatLab: the output from a channel is the convolution between the input and the channel (with the channel being described by a vector of coefficients)
- I should have 2048 samples going into the IFFT block: 2 will be used for 0 Hz or DC; the other 2046 are for the data. Here comes an important point, as I figured out last week: in order for the output of the IFFT block to be real, the input vector must be conjugate symmetric. This means that *of* the 2046 data inputs, 1023 will be my actual data, and the other 1023 will be the *complex conjugates* of my data - in essence, I only use 1023 pins on the IFFT chip, and the other 1023 are to needed to ensure that the output of the IFFT block is real.
- The cyclic prefix (cp) need not be *too long*: it depends on the length of the channel. In my case, seeing as though I'm using a three-tap FIR filter, the cyclic prefix does not need to be 500 samples in length - a value closer to 10 should do (in *theory* a cyclic prefix of length 2 or 3 would be sufficient; 2 being 1 less than the length of the filter...)
- I need to alter *how* I'm generating my data: I'll start at the IFFT block and work backwards…

31/10/2021:

- I've changed *how* I generate my data now:

```
data = 3069×1
       0
       0
       0
       0
       0
       0
       0
       0
       1
       1
       ⋮
```

```
N = 2048; % Size of the IFFT block available at Tx
M = 64; % Order of QAM modulation being used at the Tx...
k = log2(M); % The number of bits each symbol will represent...
codeRate = 0.5; % The code-rate being used by the convolutional encoder at the Tx...
lenData = (N-2)*(1/2)*(k)*(codeRate); % The amount (or length) of the data we should generate...

data = randi([0 1], lenData, 1) % Our data bit stream
```

01/11/2021:

- I seem to be removing the cyclic prefix correctly (it's of length 10 at the moment):

```
ans = 2058
y = 2048×1
   -0.0021
    0.0298
    0.0440
    0.0262
    0.0157
   -0.0026
    0.0023
   -0.0358
```

- I'm also generating a signal, y_2, which is the **linear convolution** of the ofdm symbol with the channel coefficients:

```
y2 = 2060×1
   -0.0243
    0.0207
   -0.0160
    0.0191
   -0.0017
    0.0099
    0.0225
   -0.0061
   -0.0054
    0.0123
       ⋮
```

Notice that it's **2** longer than the signal *y*, which has a length of 2058: this value of 2 is **1 less than the channel length (number of channel coefficients)**

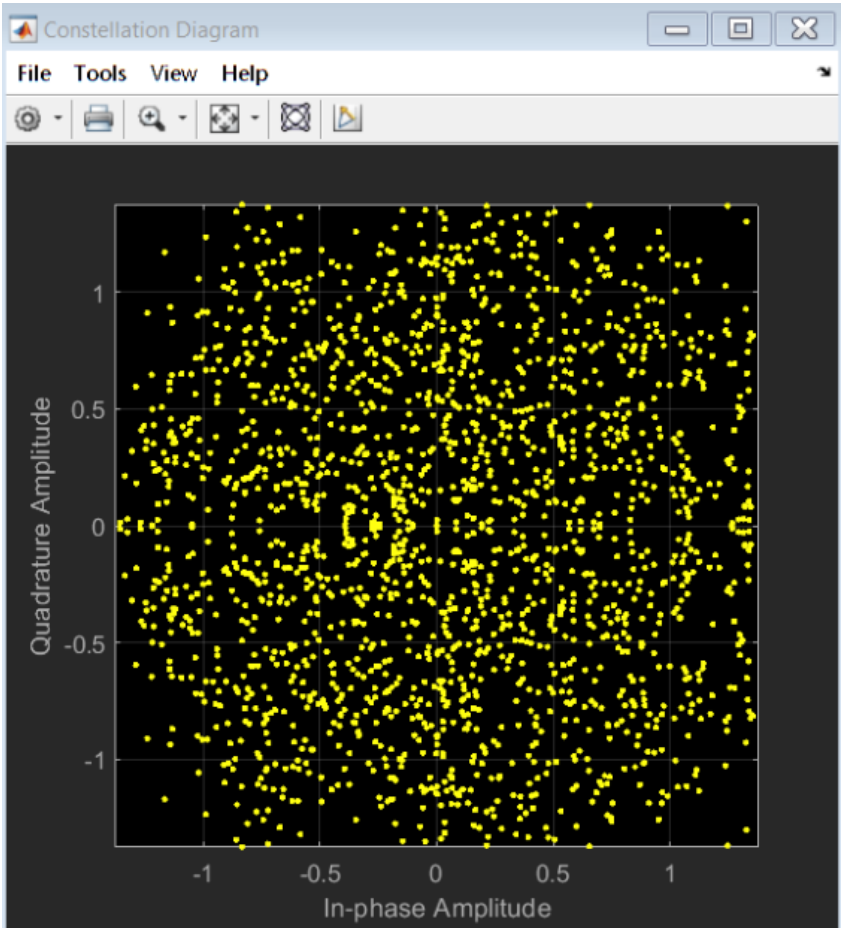
For example, if I add another 2 coefficients to the channel:

```
h = [1 0.01 0.002 0.0001 0.00002]; % Channel coefficients...
```

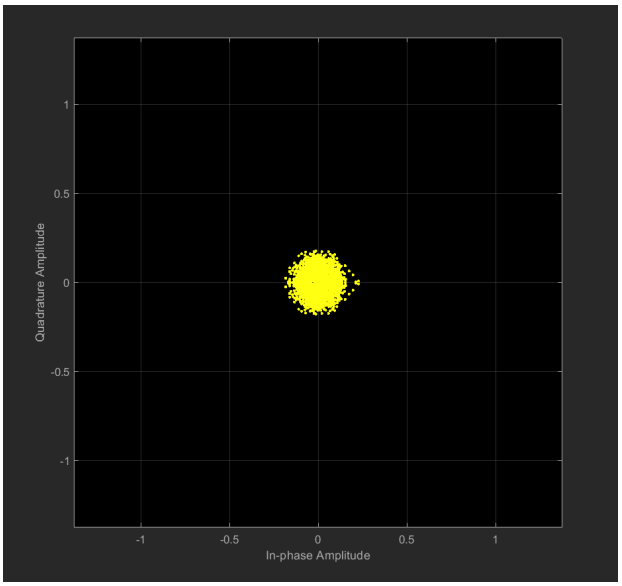
We get:

```
ans = 2064
y = 2048×1
       0.0162
      -0.0425
       0.0000
      -0.0241
      -0.0232
      -0.0118
       0.0085
       0.0129
      -0.0400
       0.0075
          :
          :
y2 = 2068×1
       0.0175
       0.0048
      -0.0019
       0.0254
       0.0385
       0.0174
      -0.0197
      -0.0122
      -0.0368
       0.0602
          :
          :
```

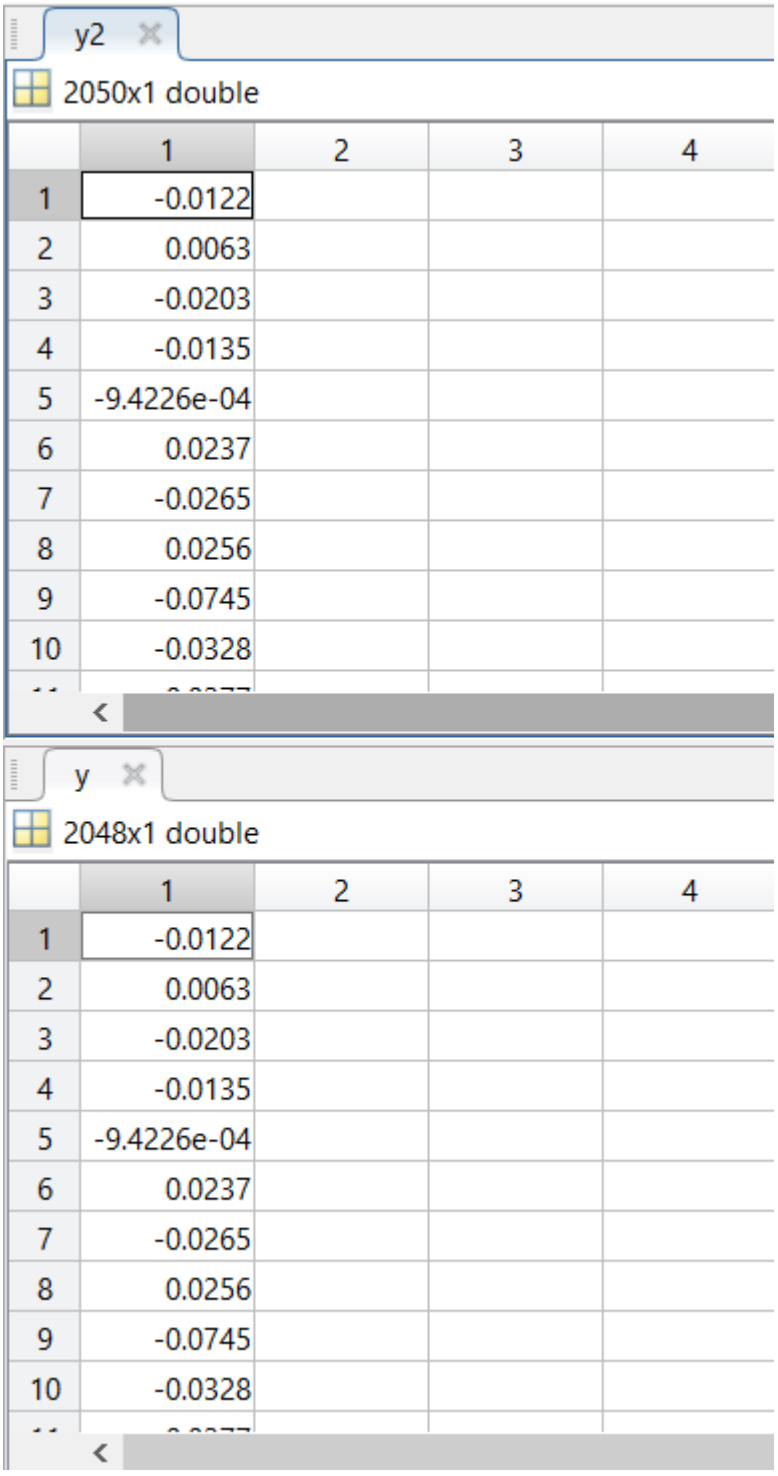
Notice that y_2 has an extra 4 values: 4 = 5 - 1 (Cool stuff!)



And now let's see what happens if the channel conditions get even worse…



- I've also verified that the *conv*( ) function in MatLab is giving the same result as my function *channel*( ) which was representing the channel:



y2

2050x1 double

	1	2	3	4
1	-0.0122			
2	0.0063			
3	-0.0203			
4	-0.0135			
5	-9.4226e-04			
6	0.0237			
7	-0.0265			
8	0.0256			
9	-0.0745			
10	-0.0328			

y

2048x1 double

	1	2	3	4
1	-0.0122			
2	0.0063			
3	-0.0203			
4	-0.0135			
5	-9.4226e-04			
6	0.0237			
7	-0.0265			
8	0.0256			
9	-0.0745			
10	-0.0328			

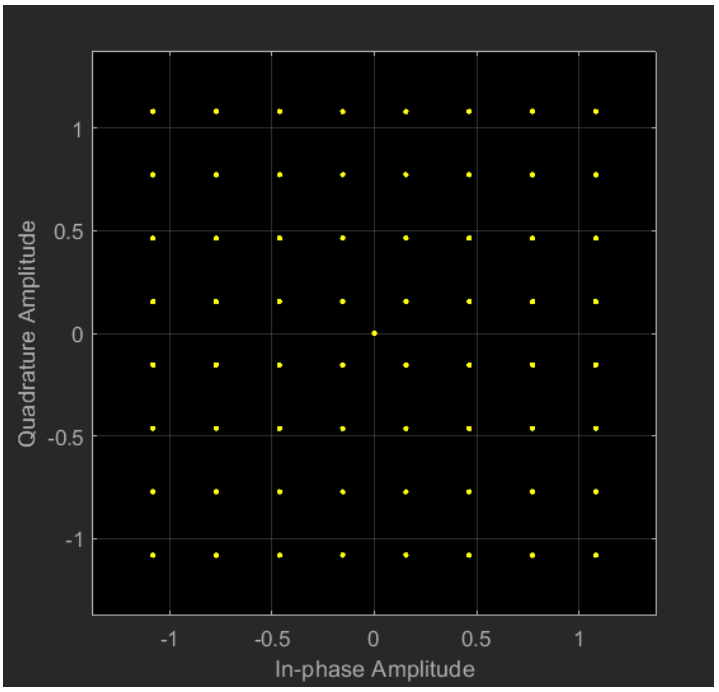
The key difference is that y_2 is **two samples** longer than y - this is because the convolution sum returns a vector equal in length to:

length(v_1) + length(v_2) - 1

Whereas my function (*channel*( )) simply returns a vector equal in length to the input vector…

- A logical question to ask is what happens if **we remove** the extra two samples at the end of the signal y_2?

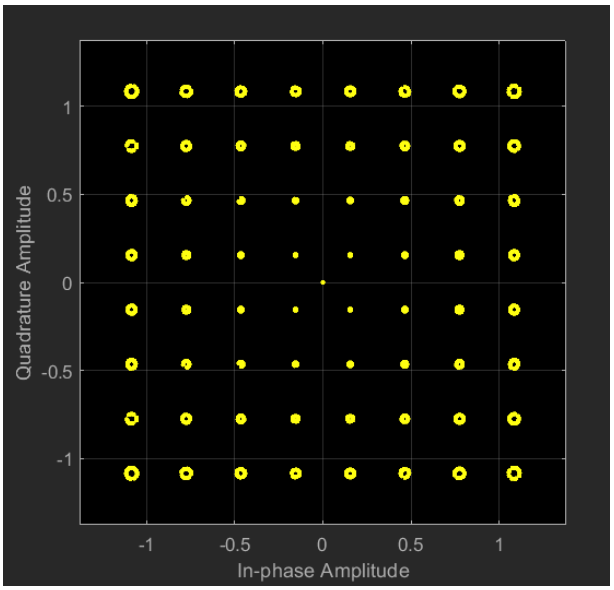
```
y2 = y2(cpLen+1:end)
length(y2)
y2 = y2(1:end-(length(h) - 1))
length(y2)
```



😃 better results it appears!

```
h = [1 0.02 0.003]; % Channel coefficients...
```

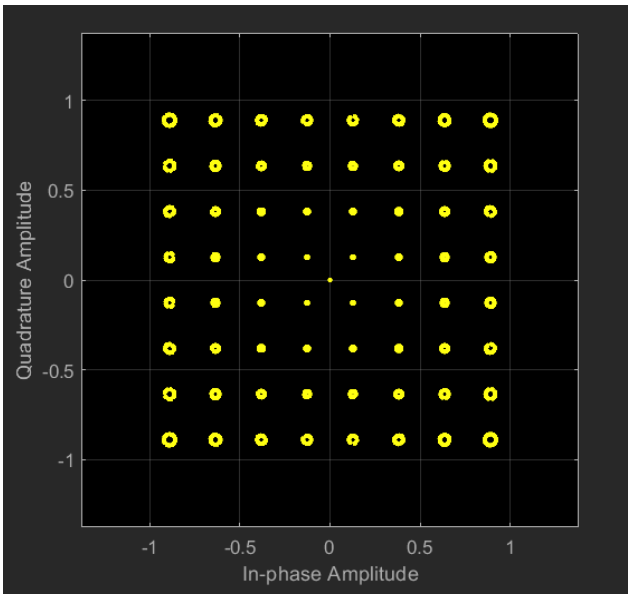
...Altering the filter coefficients



The results still seem *pretty good*…

What about changing the first coefficient to 0.82:

```
h = [0.82 0.02 0.003]; % Channel coefficients...
```



Definitely more of a degradation at the Rx side…

- From reading MIMO-OFDM Wireless Communications (with MatLab), it seems that the *cyclic prefix* is one way to realise or implement the much-needed **guard interval**. Another option is zero padding. The requisite is that the length of the guard interval (cyclic prefix) should be **greater than or equal** to the **maximum delay** of the **multipath channel**… This greatly reduces the chance of **intersymbol interference**, and maintains the orthogonality of the subcarriers.
- ISI and ICI can *still occur* **if the timing of the FFT Window start point is off**… This is important to keep in mind…
- This book mentions another key point again:

in Figure 4.16. Since $Y_l[k] = H_l[k]X_l[k]$ under no noise condition, the transmitted symbol can be detected by one-tap equalization, which simply divides the received symbol by the channel (i.e., $X_l[k] = Y_l[k]/H_l[k]$). Note that $Y_l[k] \neq H_l[k]X_l[k]$ without CP, since $\text{DFT}\{y_l[n]\} \neq \text{DFT}\{x_l[n]\} \cdot \text{DFT}\{h_l[n]\}$ when $\{y_l[n]\} = \{x_l[n]\} * \{h_l[n]\}$ for the convolution operation $*$. In fact, $Y_l[k] = H_l[k]X_l[k]$ when $\{y_l[n]\} = \{x_l[n]\} \otimes \{h_l[n]\}$ where $\otimes$ denotes the operation for circular convolution. In other words, insertion of CP in the transmitter makes the transmit samples circularly-convolved with the channel samples, which yields $Y_l[k] = H_l[k]X_l[k]$ as desired in the receiver.

The addition of the **cp** makes the **linear convolution** between the **signal and the channel** *appear* as **circular convolution** to the FFT block at the Rx side…

- STO: Symbol Timing Offset
  Symbol timing synchronization is **also known as**: *timing recovery*, *clock synchronization*, and *clock recovery*

02/11/2021:

- I've come to discover that there is quite some debate about whether there should be a space between a number and a percent sign (%). For other quantities, the unit should be specified **after** a space - but it appears that percentages are exceptions:

> Frequent Errors to Avoid | IEEE Council on Superconductivity
>
> No **space** inserted **between a number** and its unit. For example, "4.2K" is not acceptable; "4.2 K" is correct. An exception is a **percentage** (e.g. 25%).

03/11/2021:

- I figured out yesterday that if I make the channel condition perfect, the output of the FFT block on the Rx side is the same as the input to IFFT block on the Tx side:

modulatedData

1023x1 complex double

	1	2	3	4	5	6
1	-0.7715 - 0.1543i					
2	-1.0801 - 1.0801i					
3	-1.0801 + 0.7715i					
4	1.0801 + 0.4629i					
5	0.4629 + 1.0801i					
6	0.1543 - 1.0801i					
7	0.1543 - 0.4629i					
8	-0.7715 - 1.0801i					
9	0.4629 + 1.0801i					
10	-0.7715 + 0.7715i					
11	0.7715 + 1.0801i					
12	1.0801 + 0.1543i					

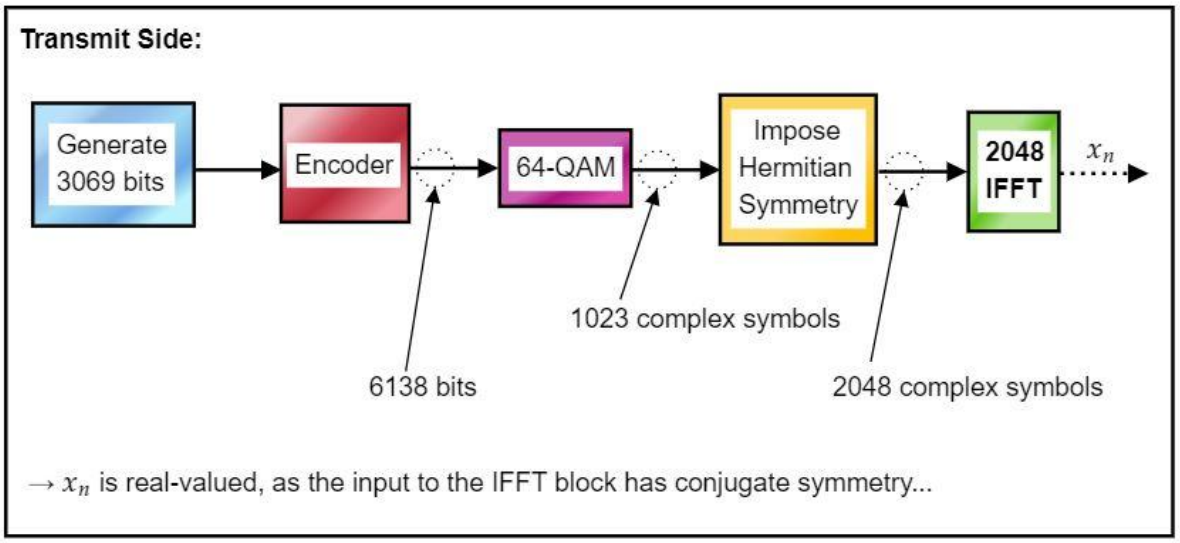
fftOp

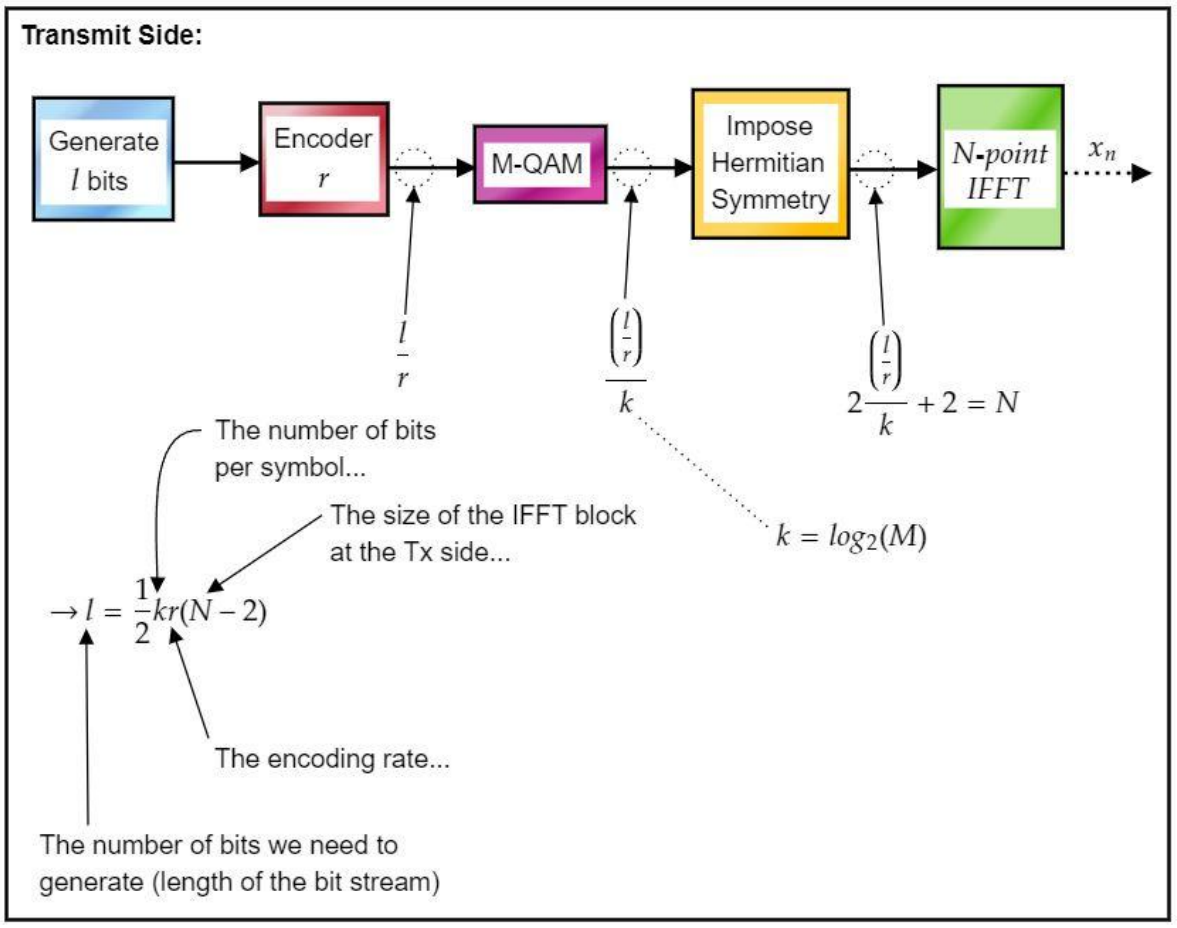
2048x1 complex double

	1
1	-2.7756e-17 + 0.0000e+00i
2	-0.7715 - 0.1543i
3	-1.0801 - 1.0801i
4	-1.0801 + 0.7715i
5	1.0801 + 0.4629i
6	0.4629 + 1.0801i
7	0.1543 - 1.0801i
8	0.1543 - 0.4629i
9	-0.7715 - 1.0801i
10	0.4629 + 1.0801i
11	-0.7715 + 0.7715i
12	0.7715 + 1.0801i

For the signal vector on the right, we can ignore the first value - it is the 0 + 0j that went into the first frequency bin of the IFFT at the transmit side… (it's not shown in modulated data as it's added after that stage)

- I've done up some diagrams to hopefully clearly depict how I'm generating the data bits on the transmit side:





- I learned about hard- Vs soft-decision decoding, and the differences between them. It seems that Viterbi decoders use soft-decision algorithms to decode convolutional code…
- I'm going to try and generate some graphs showing how the bit-error rate is affected as the channel conditions change (which for now I'll do by merely changing the channel coefficients)
- The blocks of code below (got from some MatLab documentation) seem pretty useful:

```
conEnc = comm.ConvolutionalEncoder;
modDPSK = comm.DPSKModulator('BitInput',true);
chan = comm.AWGNChannel('NoiseMethod','Signal to noise ratio (SNR)','SNR',10);
demodDPSK = comm.DPSKDemodulator('BitOutput',true);
vDec = comm.ViterbiDecoder('InputFormat','Hard');
error = comm.ErrorRate('ComputationDelay',3,'ReceiveDelay',34);
```

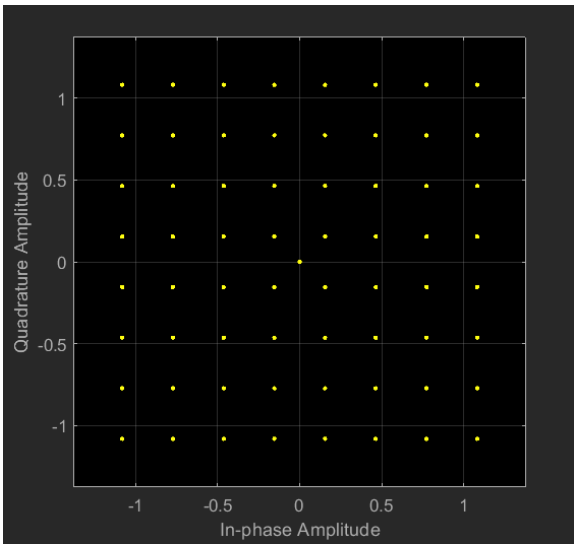
```
for counter = 1:20
    data = randi([0 1],30,1);
    encodedData = conEnc(data);
    modSignal = modDPSK(encodedData);
    receivedSignal = chan(modSignal);
    demodSignal = demodDPSK(receivedSignal);
    receivedBits = vDec(demodSignal);
    errors = error(data,receivedBits);
end
```

- The *reshape*( ) function offers a nice way to reshape the rows of 6 bits that comes from the demodulator into 1 output bit stream (a row vector, essentially)

Convert Demodulator Output to Binary

```
bitsOut = de2bi(demodOp, k)
bitStreamOut = reshape(bitsOut, 1, []) % Our output bit stream...
```

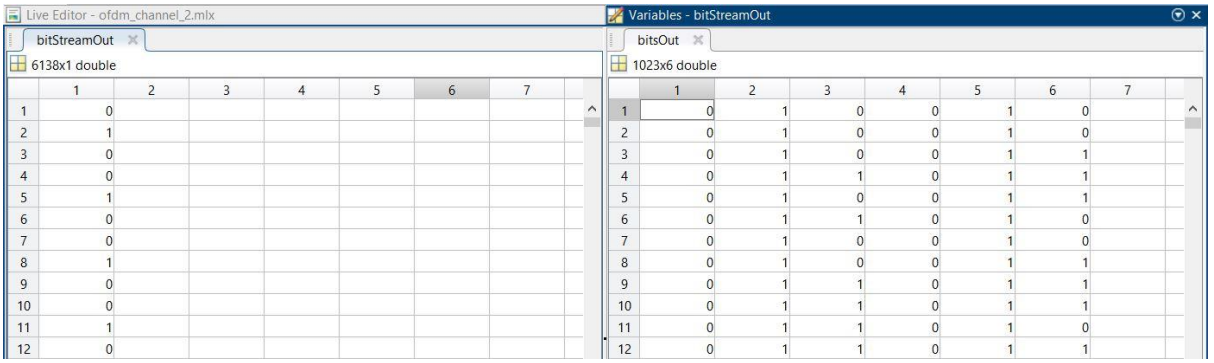
- When I think about it, 'cd( )' might stand for '**C**onstellation **D**iagram', as the function produces the constellation chart or diagram:



- I'm actually going to generate the single output bit stream slightly differently (as the method mentioned just above does not work):

Convert Demodulator Output to Binary

```
bitsOut = de2bi(demodOp, k)
bitStreamOut = bitsOut' % Our output bit stream...
bitStreamOut = bitStreamOut(:)
```



From the right side of the above snapshot, if you take the first two rows and write them one after the other (i.e. concatenate the second to the first one), you get the 12 elements shown on the left side of the above diagram! Cool stuff!

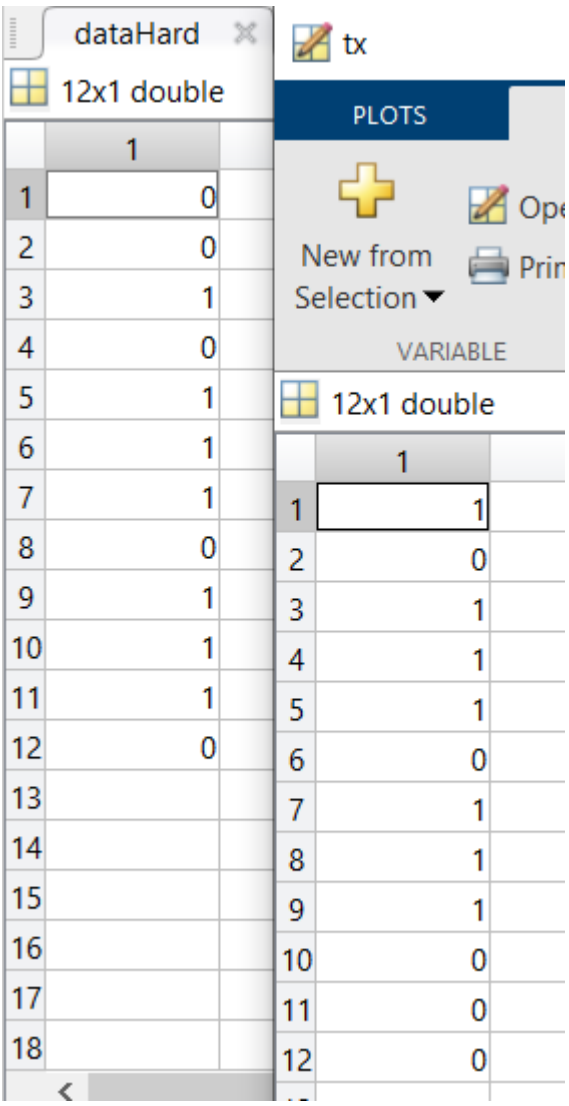
I've finally got to the stage where I can convolutionally encode a bit sequence, and then use Viterbi's algorithm to decode it - giving me back the original sequence…

Command Window

```
>> tx = [1 0 1 1 1 0]';
>> trellis = poly2trellis(7,[171 133]);
>> tbl = 32;
>> rate = 1/2;
>> dataEnc = convenc(tx, trellis);
>> dataHard = vitdec(dataEnc, trellis, tbl, 'cont', 'hard')

dataHard =
```

```
dataHard =

     0
     1
     0
     1
     1
     1
```

- I made the traceback 2:



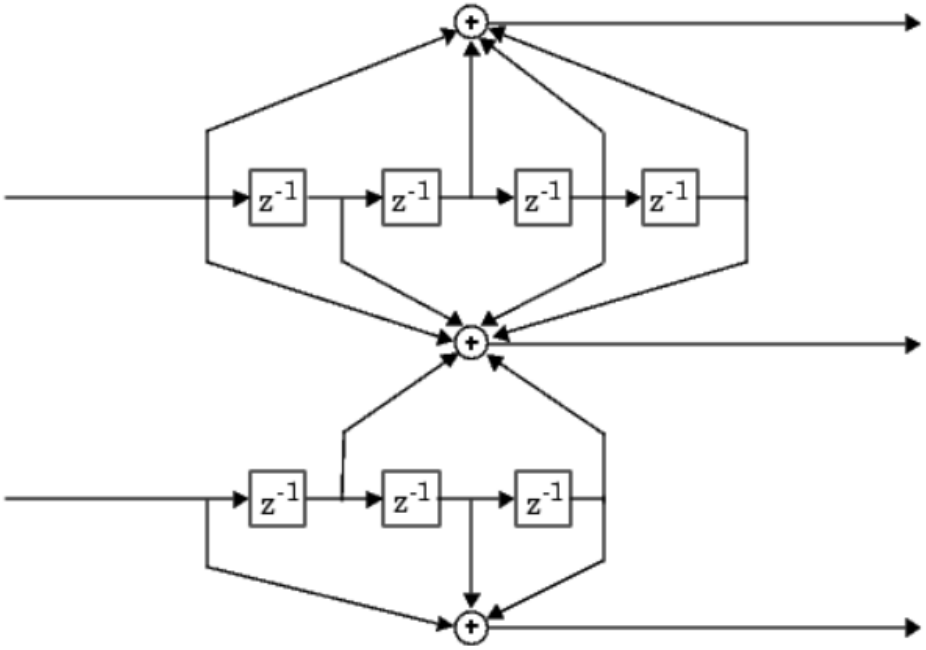
It looks like the output from the Viterbi decoder is shifted forward by 2 extra 0 bits… maybe I can remove the front of the output and append it to the end in order to end up with the original transmit signal?

04/11/2021:

- Below is shown the diagram for a ⅔ encoder:

Example: A Rate-2/3 Feedforward Encoder

The example below uses the rate 2/3 feedforward encoder depicted in the sch
perform coding using this encoder.



The **constraint length** of this encoder will be a vector of **two elements**, as the encoder has **two inputs**. The values of this vector are determined by the number of bits stored in each **shift register** (there is one up top, and one below):

Top Register: this stores 4 bits, and takes in 1 bit at the input, giving a total of **5**
Lower Register: this one stores 3 bits. Again, it takes 1 bit at the input, giving a total of **4**

The **constraint length** of the encoder is **[5 4]**

Now, as this encoder takes in **2** bits and produces **3** code bits, it can be described by a 2 X 3 matrix:

Look at the third output bit. The two rightmost bits, and the leftmost bit, go into the **sum** operator block. These bits **contribute** to the output - and this information can be captured by:

1011, which in octal is 13

Take the second output bit, and ask how the second **input** bit (and the bits in its register) contribute to *this* output bit - doing so will lead to:

0101 (5 in octal)

The leftmost bit does not go into the adder; the second bit does; the third does not; the fourth does;

Doing the same for the first input bit gives us the following generator matrix:

[27 33 0; 0 5 13]

```
trel = poly2trellis([5 4],[27 33 0;0 5 13]); % Define trellis.
```

We can use these parameters to quickly form the trellis structure

With all the above in mind, I now better understand lines such as:

```
% txTrellis = poly2trellis(7, [171 133]);
```

Since the constraint length is a vector of 1 element, this means the corresponding encoder takes in 1 bit and produces 2 code bits… (hence why each generator matrix has to **2 columns** and **1 row**)

The first 68 elements of decoded are zeros, while subsequent elements represent the decoded

while subsequent elements represent the decoded

messages.

Sure enough: the first 34 samples of the decoded message are zeros…

10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0
29	0
30	0
31	0
32	0
33	0
34	0

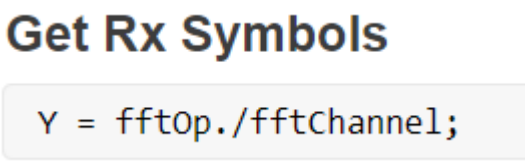
This diagram and text might be useful

Finally, the example computes the bit error rate and the number of bit errors.

```
len = 3000; msg = randint(len,1,2,94384); % Random data
t = poly2trellis(7,[171 133]); % Define trellis.
code = convenc(msg,t); % Length is 2*len.
tcode = -2*code+1; % Transmit -1s and 1s.

% Puncture by removing every third value.
punctcode = tcode;
punctcode(3:3:end)=[]; % Length is (2*len)*2/3.

ncode = awgn(punctcode,8,'measured',1234); % Add noise.

% Insert zeros.
nicode = zeros(2*len,1); % Zeros represent inserted data.
nicode(1:3:end) = ncode(1:2:end); % Write actual data.
nicode(2:3:end) = ncode(2:2:end); % Write actual data.

decoded = vitdec(nicode,t,96,'trunc','unquant'); % Decode.
[number,ratio]=biterr(decoded,msg); % Bit error rate
```

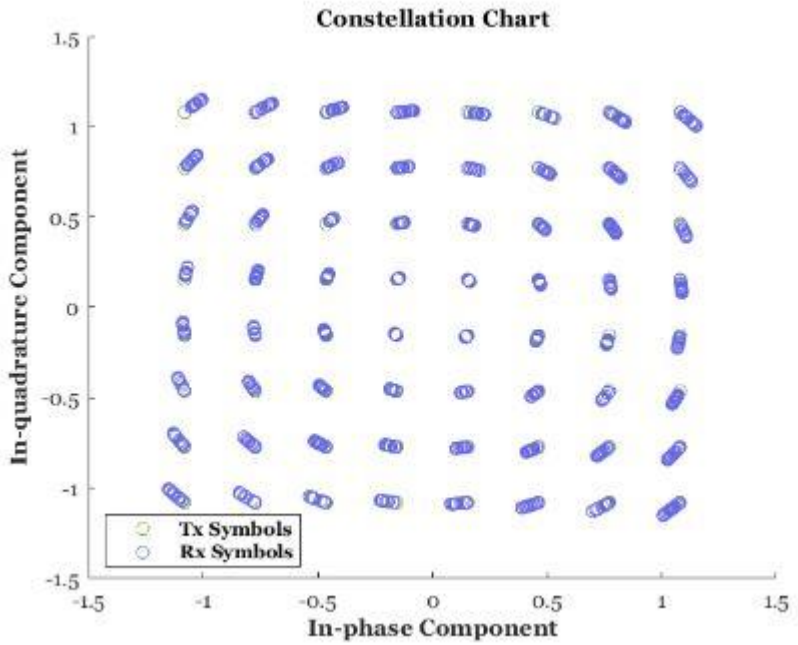
- I need to add an **equaliser** on the Rx side; the fft of a perfect channel appears to be made up of all 1s...

-
- I think I've performed channel equalisation:

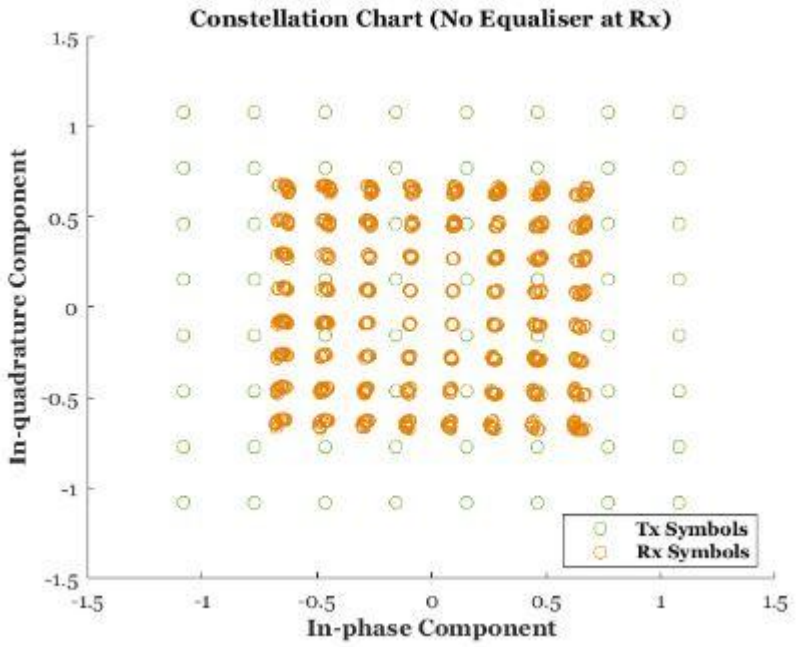


Element-wise division is very useful here!

With a perfect channel, this signal *Y* should be equal to the output from the QAM modulator. We can use the *scatter*( ) function in MatLab to plot the Constellation chart:



And we can also plot the Tx symbols Vs the Rx symbols **before** they go into the **equaliser**:



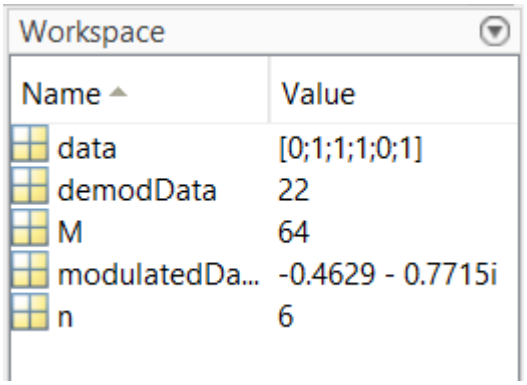
As can be seen from the above, the **equaliser** seems to be playing a **critical role**…

- The demod. function does not seem to be acting as I thought it would:

```
n = 6;
data = randi([0 1], n, 1) % Some 'random' data
M = 64;
modulatedData = qammod(data, M, "InputType", "bit", "UnitAveragePower", true);
demodData = qamdemod(modulatedData, M, 'OutputType', 'bit');
```

```
data = 6×1
     0
     1
     0
     1
     0
     0
```

data	[0;1;0;1;0;0]
demodData	[0;1;0;1;1;0]
M	64
modulatedDa...	-0.1543 - 1.0801i
n	6



When I don't specify *binary*, I get the output is 22 (decimal system presumably) - converting this to binary *outside* of the function gives:

10110

From the above, it seems that if I flip the "bitsOut" vector, then I'll retrieve the data that I initially sent….

05/11/2021:

- It seems that removing the "Unit Average Power" is "True" parameter from my *qammod*( ) function might have fixed my issue with the demodulated output not being the same as the data bits that were modulated:

```
data = 6×1
     1
     1
     1
     0
     0
     0

modulatedData = 3.0000 + 7.0000i
demodData = 6×1
     1
     1
     1
     0
     0
     0
```

```
data = 6×1
       1
       0
       1
       1
       1
       0

modulatedData = 5.0000 - 1.0000i
demodData = 6×1
       1
       0
       1
       1
       1
       0
```

```
M = 64;
modulatedData = qammod(data, M, "InputType", "bit", "UnitAveragePower", true)
demodData = qamdemod(modulatedData, M, "OutputType", "bit")
```

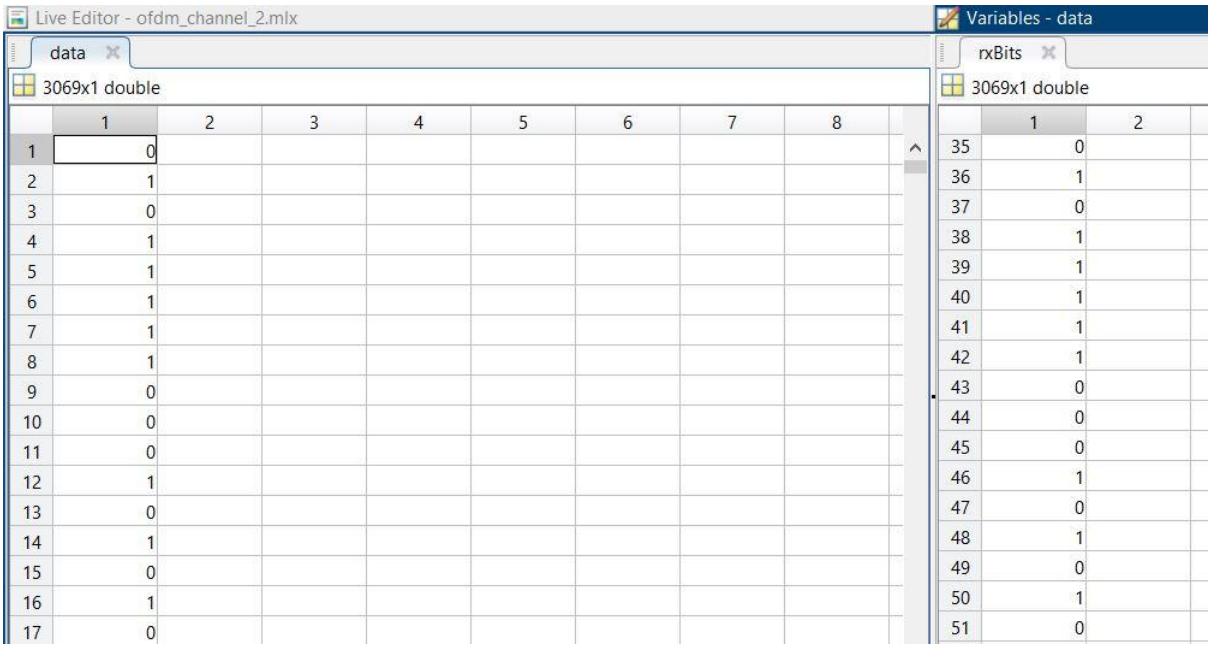
I'll see now if I can keep "UnitAveragePower" set during modulation on the Tx side, and specify it during the demodulation stage on the Rx side:
👀

```
data = 6×1
       0
       0
       1
       0
       1
       1

modulatedData = -0.7715 + 0.4629i
demodData = 6×1
       0
       0
       1
       0
       1
       1
```

😃👍

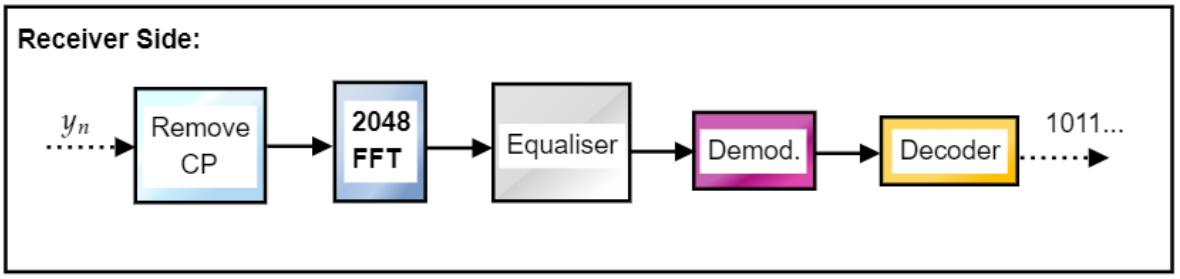
```
n = 6;
data = randi([0 1], n, 1) % Some 'random' data
M = 64;
modulatedData = qammod(data, M, "InputType", "bit", "UnitAveragePower", true)
demodData = qamdemod(modulatedData, M, "OutputType", "bit", "UnitAveragePower", true)
```

I've made the above changes to my main script and the results are much better!

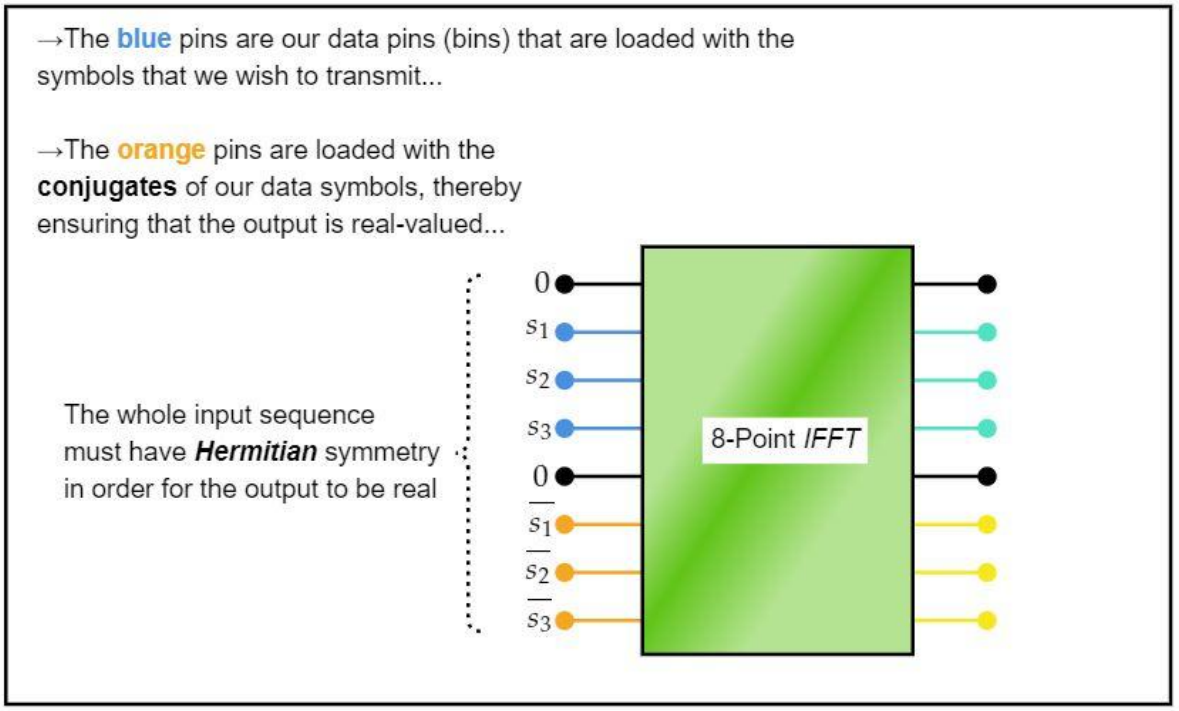


number	0	// And the number of errors is 0 too!
ratio	0	// The bit-error rate is 0…

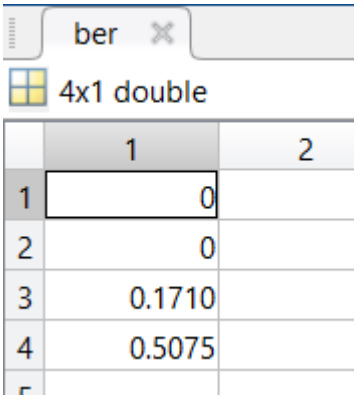
So my Rx side functionality is now something like:



And below is shown another diagram just to recall how data is loaded into the *IFFT* chip on the Tx side:



- I've done up a litte program which computes the bit-error rate for various vectors of channel coefficients:



We don't need to generate 3069 bits on each iteration of the for loop. Instead I set *N* to 512, which means I only generate 765 bits…

I must remember that the vector *h* is the channel's ***impulse response***…

H(ω) = FFT(*h*)

09/11/2021:

- I've generated a vector of 15 elements corresponding to the BERs for 15 different channel conditions:

ber

15x1 double

	1
1	0
2	0
3	0.1614
4	0.0027
5	0.1259
6	0.1765
7	0.0383
8	0.1286
9	0.0109
10	0.0027
11	0.1778
12	0.0027
13	0
14	0.1026
15	0
16	

The project will involve the steps of

- Review of literature to understand the fundamentals of radar systems and OFDM-based communication systems, including the key performance metrics for both systems.
- Simulation of an OFDM-based communication link following 4G/5G standards to study how OFDM signals are affected by the radar operation
- Formulation of the OFDM radar estimation problem
- Implementation of estimation algorithms (such as those based on periodogram) to solve the radar estimation problem and evaluate their performance

- Next, I will investigate channel estimation methods…

10/11/2021:

- Dynamic channel estimation is often needed…
- There are two main ways of doing so using **pilot tones**:
  (1) Block-type
  (2) Comb-type
- Block-type pilot tone channel estimation: in this method, all the subcarriers of the OFDM symbol are loaded with pilot tones **periodically** in order to sense or estimate the channel. This method or approach makes two big assumptions:
  (1) That we are working with a slow fading channel…
  (2) The transfer function of the channel is not changing that rapidly…
- In the block-type approach, estimation is often done using **least square (LS)** or **Minimum-Mean-Square-Error (MMSE)**...
- Comb-type pilot tone channel estimation: in this approach, pilot tones are inserted uniformly into the OFDM symbol (say at every *k*th subcarrier) in order to estimate the

channel. Interpolation (linear, second-order, or low-pass, etc.) is used in order to get a more full view of how the channel is behaving. This approach assumes a few things such as:

(1) The channel conditions are typically changing from one OFDM symbol to another…
(2) The transfer function of the channel is typically changing quickly with time…

- I'm reading what seems to be a very good paper on channel estimation, and when they refer to MatLab they use MATLAB… I will keep this in mind when doing formal reports or the likes…
- A channel estimation block on the Rx side is needed:
  $X_e = Y(k)/H_e(k)$

  And $H_e = Y_p/X_p$ // In other words, we use the pilot tones to estimate the channel…

- The OFDM Wireless Communications with MATLAB has some nice diagrams on block-type, comb-type, and lattice-type pilot tone channel estimation:

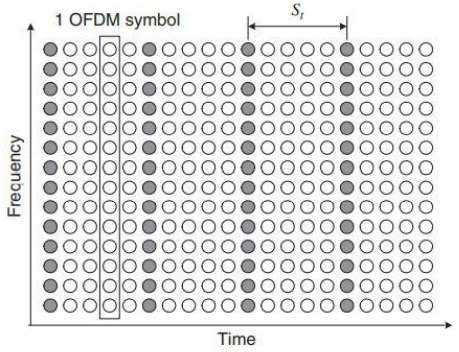


Figure 6.1 Block-type pilot arrangement.

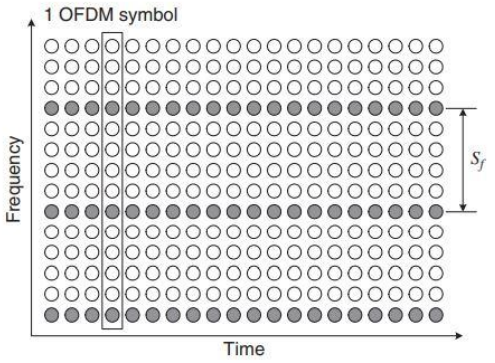


Figure 6.2 Comb-type pilot arrangement.

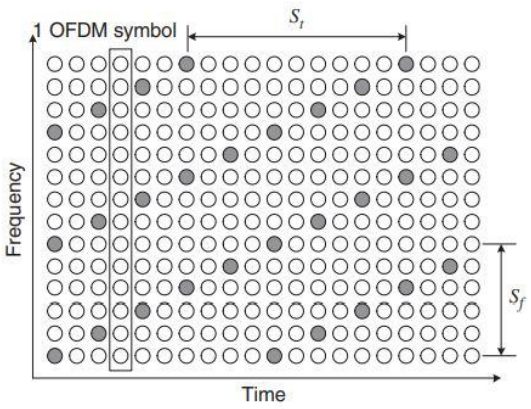
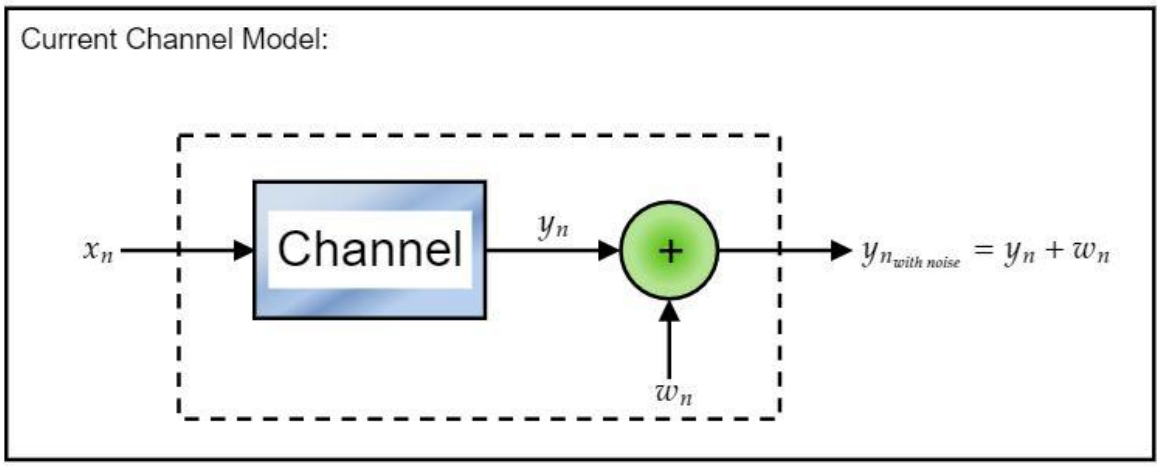


Figure 6.3 Lattice-type pilot arrangement.

- The **coherence bandwidth** is the portion of signal **bandwidth** over which **fading does not typically occur**…
  It can be thought of as the range of frequencies over which the channel can be considered **flat**… a range of frequencies which, statistically speaking, behave similarly…
  The **coherence bandwidth** helps RF engineers figure out what frequency width can be used as an OFDM subchannel…
- Below is the basic channel model that I'm currently using:



With good channel conditions, and a reasonable SNR, we get:

number	3
ofdmSymbol	*2058x1 double*
ratio	9.8847e-04

A tiny number of bits errors, and the bit error rate is really, really small…

11/11/2021:

- The transmitted EM waves when reflected off moving objects cause doppler shifts in frequency…
- i.i.d stands for "Independent and Identically Distributed"...
- A transmitted OFDM **frame** (a block of OFDM symbols) can be represented in matrix form as:

$$\mathbf{F}_{\mathrm{Tx}} = \begin{pmatrix} c_{0,0} & \cdots & c_{0,M-1} \\ c_{1,0} & \cdots & c_{1,M-1} \\ \vdots & \ddots & \vdots \\ c_{N-1,0} & \cdots & c_{N-1,M-1} \end{pmatrix} \in \mathcal{A}^{N \times M}.$$

Each column represents an OFDM symbol, and each row represents a particular subcarrier…

- The two major aspects to the estimation problem with OFDM communications are:
  (1) The round-trip delay
  (2) The doppler shift…

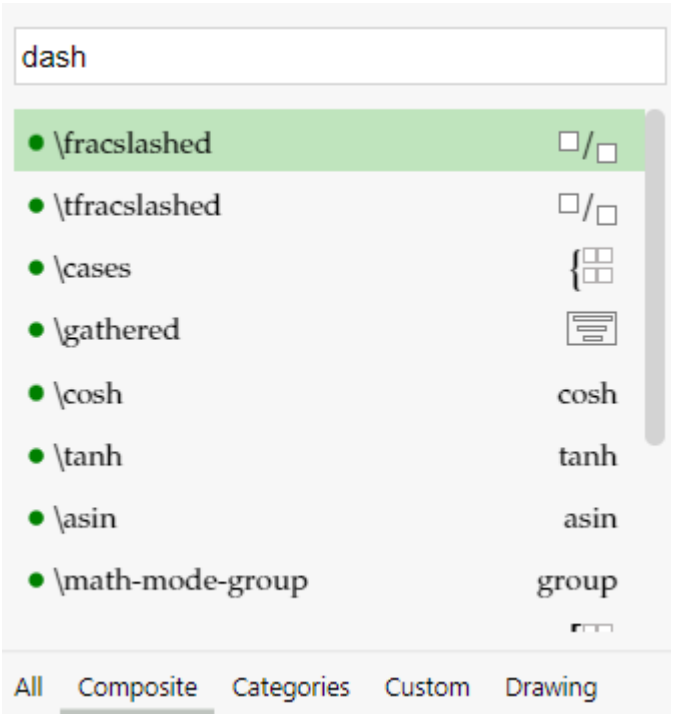
- Some assumptions need to be made too:
  (1) The CP length is greater than the longest round-trip time associated with the further target…
  (2) The subcarrier **distance** (width in **frequency**?) is at least one order of magnitude larger than the largest occurring Doppler shift…
  (3) The doppler shift is the same on every subcarrier…
  (4) The target's distance remains constant during the transmission of one frame…
- It seems that the above assumptions are easily met when appropriate signal parameters are chosen…
- The two **assumptions (1 and 2, shown in blue above)** ensure that the **orthogonality** of the subcarriers is maintained, which is crucial to the functioning of the OFDM scheme…
- Periodogram for estimation:

3.3 Periodogram-based estimation algorithms

- My supervisor has helped to resolve a small issue that was causing my constellation charts to be different than expected… it was to do with how when I took the transpose of the channel coefficient vector I was, in fact, changing the channel response… causing the Tx symbols to be rotated…

20/11/2021:

- "In situ" roughly means "In its original position or place"...
- A lot the signal processing that is needing to be done in these OFDM radar endeavours is not too complex for FPGA platforms it seems (at least from reading the paper)
- At the end of the paper is given a list of symbols, variables, and abbreviations used throughout: this might be nice for you to include in your write up…
- I've found out how to do cases (those braces that are used when defining piecewise functions and the likes):



As can be seen above, they're located in the *Composite* section…

- ≃ stands for "Asymptotic equality": when two functions eventually become equivalent (Lim (x→∞) f(x)/g(x) = 1)
- ≈ is used to mean approximately equal…
- Below I'm listing the sections and key points for the presentation that is coming up next week (week 11):

12 mins means approx. 10-11 slides, with 1 minute spent on each slide (Of course, some slides might be short and only require 30 seconds, but this affords you more time to slowly discuss the more intricate slides

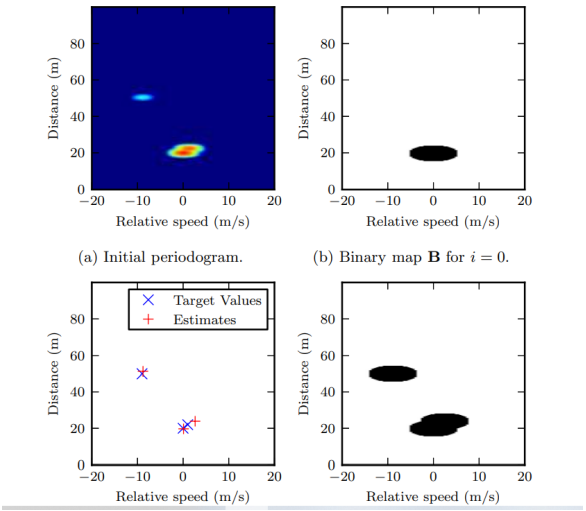
21/11/2021:

- I've done most of the slides for the presentation… I'm now endeavouring to come up with a very basic emulation of a radar system through use of a OFDM scheme…
- ~ Page 77 in the PhD thesis that my supervisor has sent on seems to be useful in this regard:

The algorithm works as follows:

1. Initialize all elements of the binary map to one, $(\mathbf{B})_{k,l} = 1$, and an index value $i = 0$.
2. Find the largest peak
$$(\hat{n}_i, \hat{m}_i) = \arg\max_{n,m} \mathrm{Per}_{\mathbf{F}}(n, m) \quad \text{s.t.} \quad (\mathbf{B})_{n,m} = 1$$
3. If $\mathrm{Per}_{\mathbf{F}}(\hat{n}_i, \hat{m}_i) < \eta$, stop searching.
4. Identify range, Doppler and RCS of the i-th target through a suitable interpolation algorithm from $(\hat{n}_i, \hat{m}_i)$, e.g. quadratic interpolation, and add those to the list of targets.
5. Set

Page 78 is excellent it seems!



(a) Initial periodogram. (b) Binary map $\mathbf{B}$ for $i = 0$.

A lot of focus on pages 35 onwards should be given… This is where the discussion of radar, and the estimation of target distance and velocity is introduced…

22/11/2021:

- The Doppler shift of EM waves differs from that of sound… crucial point!

Making the updates to my presentation based on my supervisor's comments:

(1) Have updated the project motivation
(2) Have changed the word "sense" or "sensing" to "estimate" or "estimating"...
(3) I've added the small hats to the quantities that are being estimated (channel coefficients in the frequency domain)... I've also added noise, represented by Z:

$$Y_{p_1} = X_{p_1}\hat{H}_{p_1} + Z \qquad \therefore \hat{H}_{p_1} = \frac{Y_{p_1}}{X_{p_1}} - \frac{Z}{X_{p_1}}$$

$$Y_{p_2} = X_{p_2}\hat{H}_{p_2} + Z \qquad \therefore \hat{H}_{p_2} = \frac{Y_{p_2}}{X_{p_2}} - \frac{Z}{X_{p_2}}$$

Noise

(4) I've updated the constellation plots to mention which modulation scheme is being used (64 QAM), and what size FFT and IFFT was being used… I also have mentioned that the channel is an AWGN… I changed the colours so that they are hopefully more contrasting… and easier to make out
(5) I've added some of the KPIs to the slide on Radars…
(6) I did some small resizing to some plots in order to make them bigger…

- I can easily changed my code so that it generates just 1s:

data

3069x1 double

	1	2
1	1	
2	1	
3	1	
4	1	
5	1	
6	1	
7	1	
8	1	
9	1	
10	1	

All 1s means that we end up with the same QAM symbols being loaded on the subcarriers… this *might* be what our OFDM Radar frame looks like…

- I (luckily) just realised that RCS (Radar Cross Section) is a measure of an object's reflectivity… the reflected energy divided by the incident energy… Although this webpage seems to look at it similar to how I was thinking of it:

$$\text{dBsm} = 10 \times \log_{10}(\text{RCS}/1\text{m}^2)$$

dBsm: Radar Cross Section of target in decibels.
RCS: Radar Cross Section of target in square meters.

- The PhD thesis paper has good tables showing the values of the various parameters for different standards:

Standard	802.11a	802.11p
f_C	5.5 GHz	5.9 GHz
$P_{\max}$	20-30 dBm (depending on region)	33 dBm
Δf	$20\,\text{MHz}/64 = 312.5\,\text{kHz}$	$5\,\text{MHz}/64 = 78.125\,\text{kHz}$
T_G	$1/4T$	$1/4T$
N	52	52
M	≤ 1365	≤ 1365

The above is from page **88**

r

2060x1 double

	1	2	3	4	5	6	7	8
1	-8.7969e-12							
2	-2.1618e-10							
3	-6.0280e-11							
4	-3.7088e-10							
5	-7.3454e-11							
6	-5.0443e-10							
7	-9.7208e-11							
8	-8.0199e-10							
9	-1.5271e-10							
10	-2.1631e-09							
11	2.5148e-09							

y

2060x1 double

	1	2
1	-0.0012	
2	-0.0306	
3	-0.0085	
4	-0.0525	
5	-0.0104	
6	-0.0714	
7	-0.0138	
8	-0.1134	
9	-0.0216	
10	-0.3060	
11	0.3557	

y is the signal that is transmitted *after* it passes through the channel. *r* is the reflected signal… it will pass through the channel again… (as can be seen above, the signal values are heavily attenuated due to the signal hitting the target…)

Below is shown *r2*: this is the signal that arrives at the receiver, *after* it passes through the channel on its return journey…

r2

2062x1 double

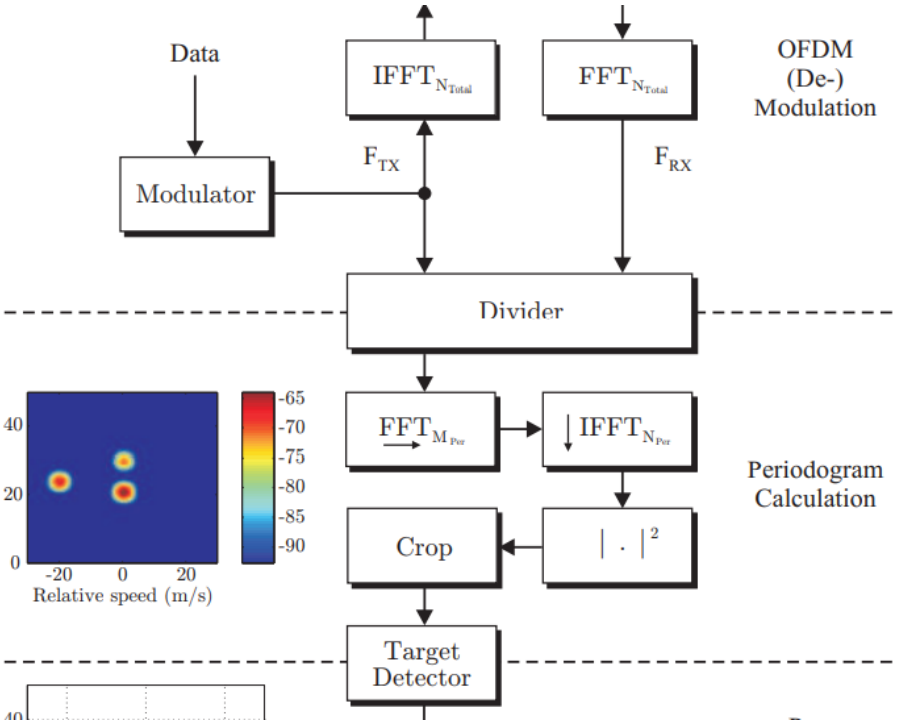
	1
1	-7.9172e-12
2	-1.9632e-10
3	-1.0101e-10
4	-4.3232e-10
5	-1.6440e-10
6	-6.1703e-10
7	-2.1775e-10
8	-9.4301e-10
9	-3.3672e-10
10	-2.2981e-09
11	1.7696e-09

r

2060x1 double

	1
1	-8.7969e-12
2	-2.1618e-10
3	-6.0280e-11
4	-3.7088e-10
5	-7.3454e-11
6	-5.0443e-10
7	-9.7208e-11
8	-8.0199e-10
9	-1.5271e-10
10	-2.1631e-09
11	2.5148e-09

I think I've understood a little better the ideas mentioned in the section on periodograms:



The divider divides the received frame by the sent one (we must remember that the Rx knows what frame was sent *because it sent it* - the Rx and Tx are the same! It sends the frame out and waits to see if there is a reflection…)

Then, from the diagram, the FFT with the → suggests that we should take the FFT of each row, before then taking the IFFT of each column… Once this is done we should get the magnitude of each point… before cropping and passing it on to the target detector…

The received frame still looks like it has Hermitian symmetry:

1019	3.9209e-09 - 3.9218e-09i
1020	4.0822e-09 - 4.0025e-09i
1021	3.7920e-09 - 3.6836e-09i
1022	3.8229e-09 - 3.7242e-09i
1023	4.0096e-09 - 3.9413e-09i
1024	3.9127e-09 - 3.7529e-09i
1025	-1.2918e-10 + 0.0000e+00i
1026	3.9127e-09 + 3.7529e-09i
1027	4.0096e-09 + 3.9413e-09i
1028	3.8229e-09 + 3.7242e-09i
1029	3.7920e-09 + 3.6836e-09i
1030	4.0822e-09 + 4.0025e-09i
1031	3.9209e-09 + 3.9218e-09i

We now want to divide the Rx frame by the sent one… but we should be careful, element-wise division is what I think is needed, not normal or standard division (they are different):

```
a =

     1     2
     4     5

>> b = [2 1; 3 3]

b =

     2     1
     3     3
```

```
>> a./b

ans =

    0.5000    2.0000
    1.3333    1.6667

>> a/b

ans =

   -1.0000    1.0000
   -1.0000    2.0000
```

Optimisation principles seem to be coming into play…

Going back to the two-dimensional periodogram, and given coarse estimates $(\hat{n}, \hat{m})$, the fractional indices are obtained by solving the optimization problem

$$(\tilde{n}, \tilde{m}) = \arg\max_{(n,m)} \mathrm{Per}_{\mathbf{F}}(n, m)$$
$$\text{subject to} \qquad (3.89)$$
$$\hat{n} - 1 \leq n \leq \hat{n} + 1 \text{ and } \hat{m} - 1 \leq m \leq \hat{m} + 1$$

F

2048x1 complex double

	1	2
1	Inf + 0.0000e+00i	

As can be seen above, I am getting an "Inf" in row 1 of **F (where F = F_{rx}/F_{tx})** … I think this makes sense, as element 1 corresponds to the DC subcarrier, which is 0… and dividing 0 by 0 is undefined…

The middle one (DC subcarrier for second sequence) is also giving an "Inf", albeit a negative one:

1025	-Inf + 0.0000e+00i

```
>> d = fft(a, 3, 2)

d =

     3     0     0
```

23/11/2021:

- I've been taking more detailed notes on estimation using the periodogram (my reference being the PhD thesis of Braun Martin)
- I've also done some more reading up on the Doppler effect (shift), and have read up on the differences between the Doppler effect of sound, as opposed to light. One equation that I came across which seems to correspond with that given in the PhD paper is:

  $f_D = (2v\cos(\theta))/\lambda$

Where θ represents the angle between the target's direction of travel, and the radar's line of sight with the target. *v* represents the target's relative velocity. In the PhD's paper, it's assumed that the radar system need not estimate the angle of arrival, and I therefore assume that θ = 0 degrees. Plugging this in, we get:

$f_D = 2v/\lambda$

And λ is simply c/*f* - plugging this in gives:

$f_D = 2f(v/c)$

As stated in the thesis paper.

In the paper as well, the periodogram is expressed in terms of the FFT of a discrete-time signal, s(*k*). The FFT is fast and efficient, meaning the periodogram can be used to search for sinusoids in the discrete-time signals present in the received frame.

- I think I might finally understand the concept of normalised frequency, and what the units are: cycles/sample, or rads/sample…
- I also think I understand why zero padding improves the frequency resolution…

24/11/2021:

- Today I mainly focused on the periodogram: what it is, and some of the details of consideration based on the PhD paper that I'm using as a reference…

25/11/2021:

- Below are some questions which I think I might get asked (or are questions which I should have answers for…)

(1) What frequency band do radars in cars use?
76-81 GHz is the band mostly used.
76-77 GHz (BW of 1 GHz); automotive long-range radar band
77-81 GHz; automotive SRR (short-range radar)...

In European nations, the 24 GHz band is currently being used, but this is being discontinued going forward; here are some advantages of the 77-81 GHz band:

- **Larger Available BW and better resolution… improves the range and velocity resolution of the Radar**
- **The difference in phase between the Tx signal and the Rx signal is used to measure the relative velocity of the targets in a Radar's field. As the wavelength decreases (which corresponds to an *increase* in the frequency), the resolution and accuracy of the velocity measurements increases proportionally… so going from a band around 24 GHz to one around 77 GHz leads to an improvement of, roughly, 3x…**
- **In general, Antenna sizes decrease with an increase in the frequency that needs to be transmitted… so moving up to a higher frequency allows for smaller antennas to be used…**
- **Higher Tx power permitted with smaller and higher-frequency antennas…**

(2) What frequency range do Short-range Radar systems use?
~ 24 GHz

(3) In modern-day 4G and 5G, how many subcarriers are used?
802.11 has 52 subcarriers, 48 of which are data ones, and 4 are pilot tones…
The center DC or "Null" subcarrier is not used…

802.11a Timing Related Parameters

Parameter	Value
Total subcarriers N_{ST}	52
Data subcarriers N_{SD}	48
Pilot subcarriers N_{SP}	4 (subcarriers -21, 7, 7, 21)
Subcarrier Frequency Spacing F_{SP}	312.5 KHz (20MHz/64)
Symbol Interval Time T_{SYM}	4 us (T_{GI} +T_{FFT})
Data Interval Time T_{DATA}	3.2 us (1/F_{SP})
Guard Interval (GI) Time T_{GI}	0.8 us (T_{FFT}/4)
IFFT/FFT Period T_{FFT}	3.2 us (1/F_{SP})
SIGNAL Symbol TIme T_{SIGNAL}	4 us (T_{GI} +T_{FFT}
Preamble $T_{PREAMBLE}$	16 us (T_{SHORT} +T_{LONG})
Short Training Sequence T_{SHORT}	8 us (10xT_{FFT}/4)
Long Training Sequence T_{LONG}	8 us (T_{GI2} + 2xT_{FFT})
Training symbol GI T_{GI2}	1.6 us (T_{FFT}/2)
FFT sample size	64 point

In the OFDM scheme, each frame is made of *subframes*. For example, a subframe may contain 20 OFDM symbols, and there may be as many as 7 subframes per OFDM frame - giving us:

7 x 20 = 140 OFDM symbols per frame!

(4) What are some of the reasons for the middle DC or "Null" subcarrier?
It allows for the use of simple (and therefore easy to design and build - and are not too expensive) zero-IF (IF standing for Intermediate Frequency) RF receivers. No transmission typically occurs on the DC subcarrier, as a strong interfering signal is usually seen here: this is either due to leakage of the local oscillator into the output, as well as or because of the non-zero DC offset in the ADC at the Rx side

A DCR (Direct-conversion Receiver): these are sometimes also known as "homodyne" receivers, or "Zero-IF" receivers, are receiver designs which demodulate the incoming radio signal using synchronous detection driven by a local oscillator, the frequency of which is the same (or very close to) as the carrier frequency of the incoming signal…

These receivers are used to convert the signal to "baseband"...

***baseband* is the range of frequencies occupied by a signal which has not been modulated to *higher frequencies*… hence the "base" in the term…**

*** A baseband signal may have components going all the way down to DC (0 Hz)…**

Looking online, it seems that "dyne" comes from French, and the Greek equivalent means "Force"...

"Homodyne" would then mean "One Force"? This *might* make sense given that Homodyne or Zero-IF receivers convert to baseband in *one* step…

(5) What is *coherence bandwidth*? **The bandwidth of frequencies over which the channel is more or less flat (attenuation remains constant); all frequency components are affected in a similar way and behave identically…**

(6) What is *coherence time*? The time for which the channel is constant (its impulse response is not varying)… The coherence time is the inverse of the *Doppler spread*… really the *coherence time* can be used as a measure of how quickly the channel is changing. *Doppler spread* is the widening of the spectrum of a narrowband signal as it passes through a multipath channel…

(7) What are some examples of the bit rates that can be achieved with the OFDM system? **64-QAM can achieve up to 54 Mbps, while QPSK is closer to 12 Mbps**

(8) How is the multipath delay estimated or calculated?

(9) What are the main differences between OFDM and FDM?

The OFDM scheme differs from traditional FDM in the following interrelated ways:

1. Multiple carriers (called subcarriers) carry the information stream,
2. The subcarriers are orthogonal to each other, and
3. A guard interval is added to each symbol to minimize the channel delay spread and intersymbol interference.

(10) What are some examples of guard times for an OFDM system?
Example would be 0.8 µS for the guard time or interval…

Symbol time interval might be 4 µS (3.2 + 0.8)...

(11) What are some examples of subcarrier spacings used in OFDM schemes?
Δf = 312.5 kHz

(12) Why is it beneficial to combine Radar and Communication…
Less hardware
Reduced costs
Possibility for use of the same, or largely the same, portion of the bandwidth
Many vehicles have VANETs and Radars… OFDM radar lends itself very nicely to these…
V2V communication (data about traffic situation, speed, message to other vehicles about when brakes are applied, when one vehicle is slowing down etc.); if time is of the essence, then direct V2V communication is preferable to going via a base station…

(13) What are some examples of things becoming more connected due to the IoT movement?

Smart Mobiles, smart refrigerators, smartwatches, smart fire alarms, smart door locks, smart bicycles, medical sensors, fitness trackers, smart security system, etc., are few examples of IoT products. 1 nov 2021

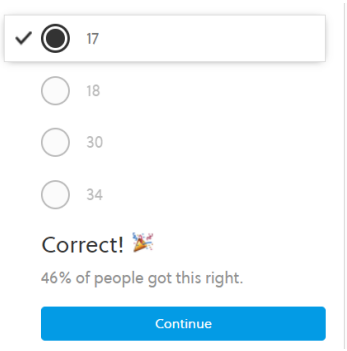
With reliable connectivity to the cellular networks over various networks such as 2G, 3G, and 4G/LTE, IoT sensors-embedded in vehicles send signals and trigger warning alerts for low battery, coolant temperature or engine maintenance. IoT solutions in the automotive industry allow the automation of the various processes and trip planning in fleet management. Moreover, this helps the fleet management segment to gain more customer satisfaction through on-time deliveries and high quality of service.

01/12/2021:

- I'm looking into MLEs, and some of the ideas surrounding them…

An airline has numbered their planes $1, 2, \ldots, N$, and you observe the following 3 planes, which are randomly sampled from the N planes:

What is the maximum likelihood estimate for N? In other words, what value of N would, according to conditional probability, make your observation most likely?



This example in which we assume that we are randomly sampling three planes from a group of N, where the planes are labelled 1 to N. The question is, What is the MLE for N? As the question clarifies, What value of N would, according to conditional probability, make your observation most likely.

The fewer planes there are, the more likely I am to pick one of the given three (either plane 4, 9, or 17)... therefore, if I minimise the number of planes, I maximise the likelihood of picking 4, 9, or 17. Therefore N = 17; N cannot be less than 17, as then how would I have picked plane 17!

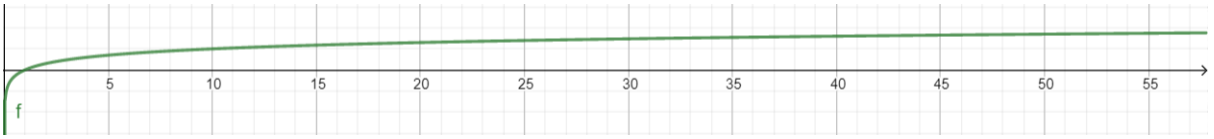
If $N < 17$, the probability is, of course, 0. If $N \geq 17$, the probability of seeing these three planes is $3! \times \frac{1}{N} \times \frac{1}{N-1} \times \frac{1}{N-2}$. Note that as N grows, this probability decreases, so the maximum likelihood estimate for N is 17.

This should strike you as slightly weird -- intuitively, 17 seems like a bad guess for N since, well, "what are the odds I happened to see the largest plane?" This illustrates an issue with maximum likelihood estimators (finding the parameter which makes the observed outcome most likely). A better estimate might be the minimum-variance unbiased estimator*, which is

$$m \cdot \left(1 + \frac{1}{k}\right) - 1 = 17 \cdot \left(1 + \frac{1}{3}\right) - 1 \approx 21.67.$$

- I read up some more and did some examples of MLE problems involving coin flips…
- Because taking a logarithm of a product expression, returns a sum expression, it is often used to simplify the process of maximising the likelihood function. The logarithm function is said to be monotonically increasing (this means, as a function, it is always increasing or remaining constant - and **never decreasing**…)

y = log(*x*)...



To maximise a sum of terms is often easier to maximise the product of terms (we can set the derivatives of each separate term to zero, and apply various methods to work out the max. and min. values)

02/12/2021:

- I've been doing some reading up and thinking about *unbiased* estimators; different ways to interpret or think about the definition, and why the name makes sense given the mathematical idea of a *bias* in some predictor model…
- As mentioned in the paper, an important idea will be to measure the probability that the MLE values are correct, and no due to noise (which can happen if the SNR drops below a *threshold* value)
- When increasing the bandwidth (B = NΔf), it is usually preferable to increase N rather than the subcarrier spacing (Δf)...
- It seems that the symbol := roughly means "define as"...
- I've learned a little bit about the *Q* function, as it appeared in the thesis paper that I am reading at the moment…

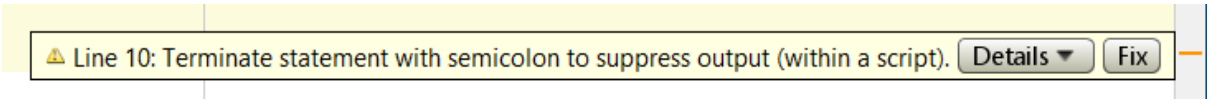
05/12/2021:

- The randi() function in Matlab is very good at producing a random binary matrix in Matlab. Each row can be considered 1 data bit stream, and will constitute an OFDM symbol:

```
%% OFDM Radar:
N = 512; % Size of the IFFT block available at Tx
M = 64; % Order of QAM modulation being used at the Tx...
k = log2(M); % The number of bits each symbol will represent...
codeRate = 0.5; % The code-rate being used by the convolutional encoder at the Tx...
lenData = (N-2)*(1/2)*(k)*(codeRate); % The amount (or length) of the data we should generate...

frameSize = 3; % The number of OFDM symbols per frame...

dataStreams = randi([0 1], lenData, frameSize);
```

- It turns out that the orange squiggly line under the "=" sign is just flagging that the output (of the large binary matrix) can be suppressed by using a semicolon:

```
dataStreams = randi([0 1], lenData, frameSize)
```




```
dataStreams = randi([0 1], lenData, frameSize);
```

10/12/2021:

- Yesterday I was looking into window functions, and why they're used. I also investigated how the number of bins or buckets in the FFT relates to the frequency resolution.
- I've done some code which creates a transmit frame (quite efficiently *I think*):

Ftx

266x3 double

	1	2	3
1	0.0165	0.0266	0.0933
2	-0.0094	0.0504	-0.0796
3	0.0279	-0.0434	-0.0090
4	-0.0079	0.0379	-0.1104
5	-0.0333	0.0342	0.0674
6	-0.0492	-0.0349	0.0855

ofdmSymbol

2058x1 double

	1	2
1	0.0213	
2	0.0230	
3	-0.0106	
4	-8.4031e-04	
5	-0.0253	
6	0.0055	
7	0.0097	

The image directly above is of the single OFDM symbol that I'm transmitting in my OFDM_channel_2 script… as can be seen, the first element need not be 0 (in fact the first few elements of the OFDM symbol will be copies of the elements near the end of the symbol, as they constitute the *cyclic prefix*.)

- Quantization error is the difference between the analog signal and the nearest available digital representation level; these errors lead to or constitute **quantization noise**…
- Scalloping loss occurs when the frequency of the sinusoid which is input to the FFT is not an integer multiple of f_s/N… this means that the frequency component does not line up directly with a bin, but instead between two bins…

11/12/2021:

- The Nelder-Mead simplex algorithm seems to be implemented by MATLAB via the fminsearch function.
- cf. roughly means "compare"...

08/01/2022:

- I'm working on implementing the basic binary target detection algorithm discussed in Braun Martin's PhD paper.
- Shown below is part of a five symbol frame, representing what is transmitted or sent out:

Ftx

266x5 double

	1	2	3	4	5
1	-0.0542	-0.0769	-0.0140	-0.0204	-0.1497
2	-0.0620	-0.0305	-0.1334	0.0378	-0.0410
3	-0.0156	-0.1253	0.0383	0.0198	-0.0676
4	-0.0629	-0.1025	0.0452	1.4855e-04	0.1010
5	0.1115	0.0351	0.0559	-0.1338	-0.0436
6	0.0186	-0.0586	-0.0505	0.0334	0.1454
7	-0.0627	-0.0482	0.0987	-0.1210	0.0709
8	0.0203	0.0157	0.0507	0.0250	-0.0158
9	0.0988	-0.0957	0.0020	-0.0242	0.0096
10	0.0131	0.0091	-0.0049	0.0493	-0.0329
11	0.0422	0.0542	0.0591	0.1049	-0.0181

I'm trying to write the code so that it's easy to change. Below is shown part of a frame which contains 14 symbols…

Ftx

266x14 double

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	-0.0544	0.0044	0.0199	-0.0504	0.0459	-0.0033	-0.0314	-0.0198	-0.0077	-0.0508	-0.1005	-0.0330	0.0343	-0.0293
2	-0.0250	0.0012	-0.0847	-0.1008	-0.0698	-0.0050	-0.0516	-0.0646	-0.1079	-0.0028	0.0461	-0.0807	-0.0503	0.0094
3	-0.0762	0.0663	-0.0789	-0.1634	0.0051	-0.0801	-0.0927	0.0109	0.0457	0.0120	0.0112	-0.0380	-0.0953	0.1195
4	-0.0153	0.0204	-0.0533	0.0865	-0.0235	-0.0573	-0.0446	0.0796	-0.0431	0.0352	0.0039	0.0943	-0.0081	0.0733
5	0.1286	0.0479	0.0614	-0.0286	-0.1248	0.0108	-0.0291	0.0177	-0.0100	-0.0098	-0.0020	-0.0519	-0.0121	-0.0510
6	0.0053	-0.0592	-0.0629	0.1353	-0.0440	-0.0211	-0.0994	0.0095	-0.0288	-0.0803	-0.0280	-0.0380	-0.1120	0.0018
7	-0.0514	0.0050	0.0178	-0.1097	-0.0143	-0.0088	-0.0342	0.0710	0.0986	0.0303	-0.0755	0.0308	-0.0310	-0.0045
8	0.0452	-0.0462	0.0136	-0.0742	-0.0136	-0.0462	0.0336	0.0345	-0.0085	-0.0114	0.0739	0.0112	-0.0263	-0.0245
9	0.0358	-0.0171	0.0556	0.0042	-0.0973	-0.0088	-0.0636	-0.0151	-0.0875	0.0595	-0.0618	-0.0712	-0.0082	0.0141
10	0.0182	-0.0819	-0.0327	0.0448	-0.0080	-0.0353	-0.0405	0.0288	-0.0297	-0.0469	0.0031	0.0317	0.1169	0.0639
11	0.1193	-0.0012	-0.1266	-0.0422	-0.0253	-0.0784	-0.1603	-0.1338	0.0518	-0.0157	-0.0181	0.0229	-0.0084	0.1555

There are 266 rows, as the FFT size is 256, and the CP length is 10… giving a total of 266…

Frx

268x5 double

	1	2	3	4	5	6	7
1	-0.0318	-0.0210	0.1153	0.0215	0.0317		
2	-0.0915	0.1690	-0.0877	0.0024	0.0067		
3	0.0180	0.0480	-0.0309	-0.0759	-0.1243		
4	0.0916	0.0089	-0.0801	-0.1234	-0.1243		
5	0.0048	-0.0717	-0.0367	-0.0295	-0.0068		
6	0.0693	-0.0041	0.0707	0.1715	0.0321		
7	0.0367	-0.0748	0.0765	-0.1007	0.0883		
8	-0.0049	0.0361	0.0018	0.0853	-0.0653		

Ftx

266x5 double

	1	2	3	4	5
1	-0.0318	-0.0210	0.1153	0.0215	0.0317
2	-0.0915	0.1690	-0.0877	0.0024	0.0067
3	0.0180	0.0480	-0.0309	-0.0759	-0.1243
4	0.0916	0.0089	-0.0801	-0.1234	-0.1243
5	0.0048	-0.0717	-0.0367	-0.0295	-0.0068
6	0.0693	-0.0041	0.0707	0.1715	0.0321
7	0.0367	-0.0748	0.0765	-0.1007	0.0883
8	-0.0049	0.0361	0.0018	0.0853	-0.0653

In the above, I'm comparing the transmit and receive frames for *perfect* channel conditions… hence why there is **no difference** between the two…

It also turns out that the "1" is necessary when trying to write complex exponentials in MATLAB:

```
dopplerTerm = exp(1i*2*pi.*l*To*fD);
```

l is a vector, going from 0 to (M - 1), where M is the frame size (the number of OFDM symbols in each frame…)

Loop variables should be initialized immediately before the loop.

This improves loop speed and helps prevent bogus values if the loop does not execute for all possible indices.

```
result = zeros(nEntries,1);
for index = 1:nEntries
   result(index) = foo(index);
end
```

I've been keeping the above point in mind…

09/01/2022:

- I've modelled the noise as AWGN:

```
% Now we'll transmit our frame through the channel:
for column = 1:frameSize
    % Two convolutions: the first is for when the frame
    % is travelling towards the target; the second is for
    % when the reflected wave is passing back through the
    % channel towards the transmitter.

    % Adding White Gaussian Noise...
    y = awgn(conv(Ftx(:, column), h), 30, 'measured');
    % The reflected EM wave passes through the same channel defined by
    % 'h'...
    y = awgn(conv(y, h));
    Frx(:, column) = y; % Filling in each column of our received frame...
end
```

- And I'm now in the process of modelling the effect of the EM wave reflecting off the target:

```
% Index vectors: l is for the columns, k is for the rows
l = 0:frameSize-1;
k = 0:(N-1) + 2*(length(h) - 1) + cpLen;
To = 3.2e-6; % The OFDM symbol period
fC = 5.5e9; % The frequency of the centre subcarrier
Vrel = 10; % The relative velocity of our target to the transmitter
c = 3e8; % The speed of light
d = 20; % The distance between the target and the transmitter
subcarrierSpacing = 312.5e3; % The subcarrier spacing

% Working out the approximate Doppler shift caused when the EM wave impinges on the
% target:
fD = 2*fC*(Vrel/c);

% And the time delay:
timeDelay = 2*d/c;

% This term occurs in the expression for the received frame:
dopplerTerm = exp(1i*2*pi.*l*To*fD);

% This term also occurs in the expression for the received frame:
delayTerm = exp(-1i*2*pi.*k*timeDelay*subcarrierSpacing);
```

```
Command Window
>> a = ones(2, 2)

a =

     1     1
     1     1

>> b = 0.25*a

b =

    0.2500    0.2500
    0.2500    0.2500
```

```
% Modeling the effect of the transmitted frame impinging on the target:
for j = 1:frameSize
    Frx(:, j) = b*Frx(:, j).*delayTerm*dopplerTerm(j);
end
```

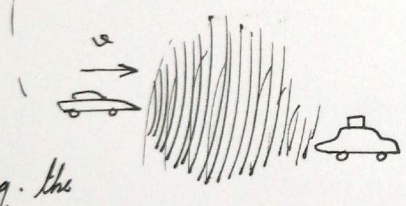
- I found a good pdf doc online which shows how the equation for the Doppler shift that is stated in the PhD paper might have come about… attached below are some derivations that I did out:

→ noise ✓
→ doppler effect & delay ✓
→ attenuation.

E.M. waves:

$$f_o = f_s \left[1 \pm \frac{v_{rel.}}{c} \right]$$

observed freq. — Source freq.



freq. the red car observes

change or difference in freq. is:

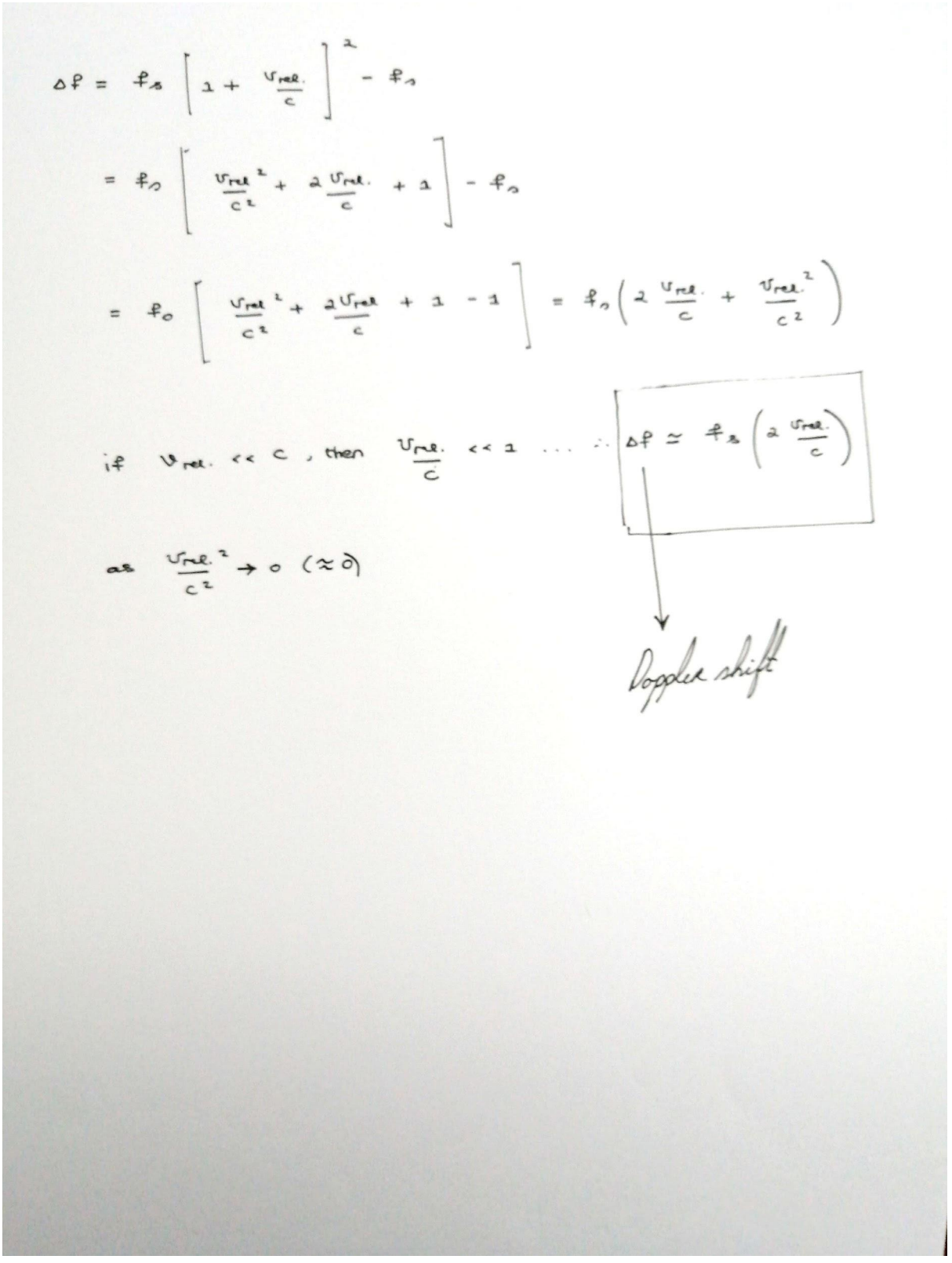
$$\Delta f = f_o' - f_s$$

freq. of source — frequency observed by police car

$$f_o' = f_o \left[1 + \frac{v_{rel.}}{c} \right]$$

$$\therefore \Delta f = f_o \left[1 + \frac{v_{rel.}}{c} \right] - f_s$$

$$\frac{f_s}{} \left[1 + \frac{v_{rel.}}{c} \right]$$




```
% Remove CP and tail from each OFDM symbol (column in the received frame):
Frx = Frx(cpLen+1:end, :); % Removing the CP
Frx = Frx(1:end-2*(length(h) - 1), :); % Removing the tails of each OFDM symbol
```

- TxSymbols is a matrix of all the symbols which, when passed through the IFFT block, go towards forming a *frame* (after the CPs have been added, of course!)...

Frx | TxSymbols

256x5 complex double

	1	2	3	4	5
1	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i
2	0.1543 - 0.7715i	-0.1543 - 1.0801i	0.4629 + 0.7715i	0.7715 - 0.4629i	0.7715 + 0.1543i
3	-1.0801 + 1.0801i	-0.1543 - 0.4629i	0.7715 + 0.7715i	0.4629 + 0.4629i	0.4629 + 0.1543i
4	-0.7715 + 0.7715i	-1.0801 + 1.0801i	-0.7715 + 0.7715i	-1.0801 + 0.7715i	-0.1543 - 0.1543i
5	0.1543 - 0.4629i	-0.4629 + 0.7715i	-0.7715 + 0.4629i	0.4629 - 1.0801i	1.0801 + 0.1543i
6	0.7715 - 1.0801i	-0.1543 - 0.1543i	1.0801 + 0.4629i	0.1543 - 0.1543i	0.4629 - 1.0801i
7	1.0801 + 0.4629i	-0.7715 + 1.0801i	1.0801 + 1.0801i	-0.7715 - 0.7715i	-0.4629 - 0.4629i

```
channelConditions = "Perfec";

switch(channelConditions)
    case "Perfect"
        h = [1; 0; 0]; % Channel coefficients...
    case "Decent"
        h = [0.8; 0.2; 0.4];
    otherwise
        disp("Check status of 'channelConditions' above.")
        return
end

% The length of the cyclic prefix required is related to the
% length of the channel...
cpLen = 3*length(h) + 1;
```

Command Window

```
>> ofdm_radar
Check status of 'channelConditions' above.
>>
```

I think this might be the first “switch” case that I’ve used in MATLAB… 👀 I’m using it so that I can easily switch between various channel conditions during simulation…

10/01/2022:

- “et al” means, roughly, “and others”
- The picture below is showing, I think, that taking the FFT of each row, and then the IFFT of each column, is *the same* as working out the IFFT of each column, and then the FFT of each row:

```
>> b = ifft(a)

b =

    1.5000    2.0000
    0.5000    1.0000

>> b= fft(b, 2)

b =

     2     3
     1     1

>> c = fft(a, 2)

c =

     3     4
     1     2

>> c = ifft(c)

c =

     2     3
     1     1

fx >>
```

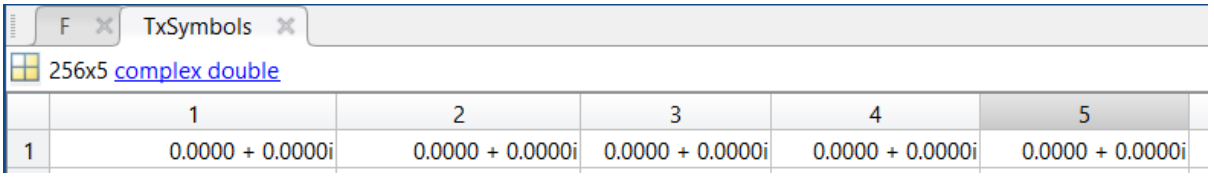
- I'm currently running into an issue when working out **F**:

```
F = Frx./TxSymbols; % As suggested in Braun Martin's paper...
```

F

256x5 complex double

	1	2	3	4	5
1	Inf + Infi	-Inf - Infi	-Inf - Infi	-Inf + Infi	-Inf - Infi
2	-1.4327e-0...	1.8352e-09...	8.3083e-10...	2.5848e-10...	-2.0433e-0...



F | TxSymbols

256x5 complex double

	1	2	3	4	5
1	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i

The “Inf” terms are resulting from the element-wise division by 0…

Each row of $\mathbf{F}_{\mathrm{Tx}}$ represents a sub-carrier; each column represents an OFDM symbol of the transmitted frame. If sub-carriers are left empty (such as the DC carrier), the corresponding elements $c_{k,l}$ are set to zero, although empty sub-carriers at the edge may be discarded (therefore, the matrix has N rows even if $N < N_{\mathrm{Total}}$).

11/01/2022:

- Circular Gaussian Noise: [0, 2π) // This means that “0” is included, but “2π” isn’t…
- The point below is very important (all of this is in Appendix A):

First, the phase is considered. As $\tilde{\mathbf{Z}}$ was complex, circular Gaussian noise, $\varphi_{\tilde{z}}$ is uniformly distributed within $[0, 2\pi)$. Adding or subtracting a random phase will not change the statistics of the noise’s phase.

For the amplitude, things are different. Only if c is a symbol from a PSK alphabet, then $b_c = 1$, and the amplitude is unchanged. For phase-amplitude modulation alphabets this can be problematic, as the symbol’s amplitude varies within several, discrete steps, and therefore the PDF if z is non-Gaussian.

Fig. A.1 shows the PDF of the amplitude of $(\mathbf{Z})_{k,l}$ obtained through simulation. It is generated by plotting the histograms of amplitudes of $\tilde{\mathbf{Z}}$ after dividing it by random symbols from diverse modulation alphabets.

Two effects can be observed: First, BPSK indeed does not affect the noise’s statistics, and second, the PDF of b_z is clearly not Gaussian for higher modulation alphabets. Also shown is a Gaussian distribution which has the same variance as the noise after the division with the 16-QAM symbols. This is the noise distribution a radar system will assume

- The diagram below hopefully captures the ambiguity about what to do:

$$F_{Tx} = \begin{bmatrix} \boxed{0 & 0 & \cdots & 0} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,M-1} \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ c_{N-1,0} & c_{N-1,1} & \cdots & c_{N-1,M-1} \end{bmatrix} \leftarrow DC\ subcarrier$$

$$F_{Rx} = \begin{bmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,M-1} \\ b_{1,0} & b_{1,1} & \cdots & b_{1,M-1} \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ b_{N-1,0} & b_{N-1,1} & \cdots & b_{N-1,M-1} \end{bmatrix}$$

$$\rightarrow F = \frac{F_{Rx}}{F_{Tx}} = \begin{bmatrix} Inf. & Inf. & \cdots & Inf. \\ a_{1,0} & a_{1,1} & \cdots & a_{1,M-1} \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ Inf. & Inf. & \cdots & Inf. \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ a_{N-1,0} & a_{N-1,1} & \cdots & a_{N-1,M-1} \end{bmatrix}$$

Idea number 1: do the element-wise division, but ignore the rows in the transmitted frame with 0 + 0*j…

Frx

256x5 complex double

	1	2	3	4	5	6	7
1	6.4941e-10...	-1.0201e-0...	2.0147e-09...	-3.3515e-1...	1.3133e-09...		
2	2.3965e-09...	4.3466e-10...	-1.3547e-0...	2.6469e-09...	2.2909e-09...		
3	1.0844e-09...	3.3001e-09...	2.1367e-09...	4.1457e-10...	-1.6930e-0...		
4	-5.2337e-1...	-2.1434e-0...	-2.6862e-0...	-2.2555e-1...	7.3719e-11...		
5	-1.5988e-1...	4.8639e-10...	6.3973e-10...	-1.3584e-0...	-2.4477e-0...		
6	-1.2771e-1...	4.6324e-10...	8.7409e-10...	7.1267e-10...	-2.8124e-1...		
7	-7.6773e-1...	1.5052e-09...	1.8531e-09...	1.4493e-09...	-7.3004e-1...		

F

256x5 complex double

	1	2	3	4	5	6	7
1	6.4941e-10...	-1.0201e-0...	2.0147e-09...	-3.3515e-1...	1.3133e-09...		
2	1.1000e-09...	-1.9557e-0...	2.4366e-09...	-7.8385e-1...	1.5709e-09...		
3	-1.0785e-0...	4.6044e-10...	1.4050e-09...	2.4066e-09...	4.0165e-09...		
4	-3.0829e-0...	2.1464e-11...	1.5213e-09...	-1.7097e-1...	-3.0060e-1...		
5	7.6600e-10...	4.7751e-10...	1.9457e-09...	-1.2304e-0...	5.9630e-10...		
6	-1.9518e-1...	-9.2666e-1...	9.0729e-10...	7.4928e-10...	1.4678e-10...		
7	2.0783e-10...	-1.6745e-0...	2.4054e-09...	-1.9454e-0...	-1.9505e-1...		

As can be seen above, F and Frx have the same first row now…

Command Window

```
>> clear
>> ofdm_radar
>> find(isinf(F))

ans =

  0×1 empty double column vector

fx >>
```

```
a =

     1     1     1
     1     1     1
     1     1     1

>> a = a + 1*i

a =

   1.0000 + 1.0000i   1.0000 + 1.0000i   1.0000 + 1.0000i
   1.0000 + 1.0000i   1.0000 + 1.0000i   1.0000 + 1.0000i
   1.0000 + 1.0000i   1.0000 + 1.0000i   1.0000 + 1.0000i

>>
```

MatLab is powerful…

```
   1.0000 + 1.0000i   1.0000 + 1.0000i   1.0000 + 1.0000i
   1.0000 + 1.0000i   1.0000 + 1.0000i   1.0000 + 1.0000i
   1.0000 + 1.0000i   1.0000 + 1.0000i   1.0000 + 1.0000i

>> b = abs(a)

b =

    1.4142    1.4142    1.4142
    1.4142    1.4142    1.4142
    1.4142    1.4142    1.4142
```

```
>> c = abs(a).^2

c =

     2.0000    2.0000    2.0000
     2.0000    2.0000    2.0000
     2.0000    2.0000    2.0000
```

Now we can apply the square-of-the-magnitude step mentioned in Braun Martin's paper:

```
P = abs(F).^2; % Assigning to P, in order to retain F...
```

F P

256x5 double

	1	2	3	4	5
1	3.7353e-19	2.6438e-20	1.0190e-18	2.4529e-19	6.4439e-19
2	2.6712e-19	1.6219e-19	2.0523e-19	2.3377e-19	2.8276e-19
3	3.8116e-19	4.4821e-20	9.9963e-20	4.0035e-19	2.5701e-19
4	3.6635e-19	4.5677e-20	3.4287e-19	7.9046e-20	4.1127e-20
5	2.1559e-19	3.9488e-19	1.0879e-19	4.9385e-21	8.9819e-20

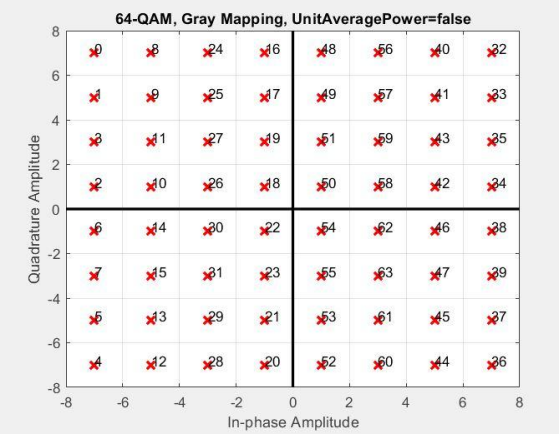
As can be seen, the numbers are incredibly small…

$$\tau(n) = \frac{n}{N_{\text{Per}}\Delta f}, \tag{3.50}$$

$$f_{D,}(m) = \frac{m}{M_{\text{Per}}T_O}. \tag{3.51}$$

12/01/2022:

- I'm looking briefly into QPSK. It seems that each QPSK symbol represents 2 bits. And there is 1 symbol in each quadrant.
- I've got another, shorter program, which randomly fills in a frame to be transmitted with complex symbols (this avoids me needing to generate bits, encode them etc.):




```
% Generate the signal set for the 64-QAM scheme:
M = 64; % Order of QAM scheme being used
d = (0:M-1)';
signalSet = qammod(d, M);
```

signalSet

64x1 complex double

	1	2
1	-7.0000 + 7.0000i	
2	-7.0000 + 5.0000i	
3	-7.0000 + 1.0000i	
4	-7.0000 + 3.0000i	
5	-7.0000 - 7.0000i	
6	-7.0000 - 5.0000i	
7	7.0000 - 1.0000i	

I'm going to randomly take symbols from this set, populate the transmit frame, and then create the received frame in accordance with the equation below:

$\tilde{\mathbf{Z}} \in \mathbb{C}^{N \times M}$ is the matrix representation of the AWGN. Its elements are i.i.d. complex random variables from a circular, zero-mean Gaussian distribution with variance σ^2. As f_0 and $\tilde{\varphi}_0$ are constant, define $\varphi_h := \tilde{\varphi}_h - 2\pi f_0 \tau_h$ and thus simplify (3.17) to

$$(\mathbf{F}_{\mathrm{Rx}})_{k,l} = b_0 (\mathbf{F}_{\mathrm{Tx}})_{k,l} \cdot e^{j2\pi T_O f_{D,0} l} \cdot e^{-j2\pi\tau_0 \Delta f k} \cdot e^{j\varphi_0} + (\tilde{\mathbf{Z}})_{k,l}. \quad (3.18)$$

```
   5.0000 - 3.0000i

>> randsample(signalSet, 1)

ans =

   5.0000 - 3.0000i

>> randsample(signalSet, 1)

ans =

   1.0000 - 3.0000i

>> randsample(signalSet, 1)

ans =

  -1.0000 - 3.0000i
```

signalSet | Ftx

128x14 complex double

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....
2	-3.0000 + 7...	7.0000 + 3....	-5.0000 - 1....	5.0000 + 5....	-3.0000 - 3....	-5.0000 + 5...	1.0000 + 1....	-7.0000 - 1....	-1.0000 + 1...	-5.0000 - 3....	7.0000 + 5....	-7.0000 - 5....	-3.0000 + 5...	-7.0000 - 1....
3	-7.0000 - 3....	1.0000 + 1....	-1.0000 + 1...	7.0000 - 1....	3.0000 - 5....	-3.0000 + 3...	5.0000 - 7....	1.0000 + 7....	-3.0000 + 3...	5.0000 + 5....	-7.0000 - 3....	3.0000 + 3....	-5.0000 + 3...	-1.0000 + 5...
4	1.0000 + 3....	-3.0000 - 3....	1.0000 + 5....	-3.0000 + 5...	-1.0000 + 5...	-7.0000 + 1...	5.0000 + 3....	-3.0000 + 3...	-3.0000 - 7....	7.0000 - 3....	-7.0000 + 3...	-1.0000 - 7....	1.0000 + 5....	5.0000 - 7....
5	-5.0000 + 7...	-5.0000 + 7...	-7.0000 - 5....	-7.0000 + 7...	-3.0000 + 3...	5.0000 + 5....	5.0000 - 1....	7.0000 + 5....	-7.0000 - 1....	5.0000 + 7....	-5.0000 + 7...	-5.0000 + 7...	-7.0000 - 1....	-5.0000 + 5...
6	-5.0000 + 1...	-5.0000 - 7....	-1.0000 - 7....	-1.0000 - 7....	-5.0000 - 5....	-1.0000 + 7...	3.0000 + 5....	5.0000 - 5....	7.0000 + 3....	-5.0000 + 3...	-5.0000 - 5....	-7.0000 - 7....	3.0000 + 1....	5.0000 - 5....
7	7.0000 + 3....	-1.0000 - 7....	-5.0000 + 1...	7.0000 - 3....	3.0000 - 3....	-5.0000 + 1...	-1.0000 + 7...	-3.0000 + 5...	-7.0000 - 7....	5.0000 + 3....	-3.0000 + 5...	3.0000 - 1....	-3.0000 + 5...	7.0000 - 3....
8	-5.0000 + 5...	-3.0000 + 7...	-5.0000 + 1...	1.0000 + 7....	1.0000 - 3....	-1.0000 - 1...	5.0000 + 3....	-1.0000 + 1...	7.0000 + 5....	1.0000 - 5....	7.0000 - 1....	-1.0000 - 5....	-1.0000 + 3...	-3.0000 - 7....
9	-3.0000 + 3...	-1.0000 - 3....	7.0000 + 3....	5.0000 - 3....	-3.0000 + 3...	-3.0000 + 3...	-7.0000 - 3....	-7.0000 + 5...	-1.0000 + 1...	-1.0000 - 7....	5.0000 + 5....	3.0000 - 5....	3.0000 + 3....	-3.0000 - 5....
10	-5.0000 - 3....	1.0000 + 7....	1.0000 + 7....	5.0000 - 3....	5.0000 - 3....	-7.0000 - 1....	5.0000 + 3....	-3.0000 - 5....	-5.0000 - 5....	-7.0000 - 1....	1.0000 - 7....	-5.0000 + 3...	-5.0000 + 1...	5.0000 + 1....
11	3.0000 + 5....	7.0000 + 5....	5.0000 - 7....	-5.0000 + 5...	3.0000 - 5....	7.0000 + 1....	5.0000 + 3....	-7.0000 + 1...	1.0000 + 3....	5.0000 - 3....	-7.0000 - 3....	7.0000 + 5....	-1.0000 - 7....	7.0000 + 1....
12	-3.0000 + 5...	-3.0000 + 1...	-5.0000 + 3...	-1.0000 + 7...	-7.0000 + 5...	3.0000 + 3....	5.0000 + 5....	3.0000 + 3....	-5.0000 + 1...	3.0000 + 1....	1.0000 + 1....	7.0000 - 7....	-3.0000 - 7....	-1.0000 + 7...

I form the top half first, and concatenate a slightly modified version to the bottom, to form the entire frame:

```
Ftx = zeros(N/2, frameSize); % The transmit frame, which is currently empty

for row = 2:N/2
    Ftx(row, :) = randsample(signalSet, frameSize, true);
end

Ftx = [Ftx; zeros(1, frameSize); conj(flip(Ftx(2:end, :), 1))];
```

62	-5.0000 - 3.0000i	5.0000 + 5.0000i	-1.0000 - 7.0000i	-7.0000 - 1.0000i	7.0000 + 1.0000i
63	-1.0000 - 7.0000i	7.0000 + 1.0000i	1.0000 - 3.0000i	-7.0000 + 3.0000i	7.0000 + 7.0000i
64	5.0000 + 7.0000i	3.0000 - 5.0000i	7.0000 + 1.0000i	-3.0000 - 1.0000i	1.0000 + 1.0000i
65	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i
66	5.0000 - 7.0000i	3.0000 + 5.0000i	7.0000 - 1.0000i	-3.0000 + 1.0000i	1.0000 - 1.0000i
67	-1.0000 + 7.0000i	7.0000 - 1.0000i	1.0000 + 3.0000i	-7.0000 - 3.0000i	7.0000 - 7.0000i
68	-5.0000 + 3.0000i	5.0000 - 5.0000i	-1.0000 + 7.0000i	-7.0000 + 1.0000i	7.0000 - 1.0000i

Below is shown a full frame, when N = 16, and frameSize = 4; this allows one to more easily verify that the intended operations have been performed correctly!

Ftx

16x4 complex double

	1	2	3	4
1	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i
2	3.0000 + 1.0000i	5.0000 + 3.0000i	3.0000 - 7.0000i	-7.0000 - 1.0000i
3	7.0000 + 7.0000i	-7.0000 - 3.0000i	7.0000 + 1.0000i	5.0000 - 7.0000i
4	-5.0000 + 5.0000i	1.0000 + 5.0000i	-3.0000 + 5.0000i	3.0000 + 5.0000i
5	-1.0000 + 3.0000i	-7.0000 + 3.0000i	-5.0000 - 1.0000i	-7.0000 - 5.0000i
6	3.0000 - 7.0000i	-7.0000 + 5.0000i	-7.0000 - 3.0000i	-7.0000 + 7.0000i
7	-5.0000 - 5.0000i	-7.0000 + 7.0000i	5.0000 + 5.0000i	7.0000 + 5.0000i
8	-5.0000 - 3.0000i	-5.0000 - 7.0000i	-7.0000 - 5.0000i	-1.0000 - 3.0000i
9	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i
10	-5.0000 + 3.0000i	-5.0000 + 7.0000i	-7.0000 + 5.0000i	-1.0000 + 3.0000i
11	-5.0000 + 5.0000i	-7.0000 - 7.0000i	5.0000 - 5.0000i	7.0000 - 5.0000i
12	3.0000 + 7.0000i	-7.0000 - 5.0000i	-7.0000 + 3.0000i	-7.0000 - 7.0000i
13	-1.0000 - 3.0000i	-7.0000 - 3.0000i	-5.0000 + 1.0000i	-7.0000 + 5.0000i
14	-5.0000 - 5.0000i	1.0000 - 5.0000i	-3.0000 - 5.0000i	3.0000 - 5.0000i
15	7.0000 - 7.0000i	-7.0000 + 3.0000i	7.0000 - 1.0000i	5.0000 + 7.0000i
16	3.0000 - 1.0000i	5.0000 - 3.0000i	3.0000 + 7.0000i	-7.0000 + 1.0000i

13/01/2022:

- I've generated Z (I think!), the matrix representing the AWGN:

Z

16x4 complex double

	1	2	3	4
1	0.1804 + 0.1775i	0.5510 + 0.5290i	0.6143 + 0.4399i	0.6364 + 0.0109i
2	0.3578 + 0.4356i	0.6604 + 0.3186i	0.0597 + 0.2482i	0.2611 + 0.0304i
3	0.4943 + 0.3347i	0.0919 + 0.0593i	0.2827 + 0.3629i	0.0786 + 0.1195i
4	0.6300 + 0.2487i	0.4022 + 0.1619i	0.1838 + 0.2841i	0.5517 + 0.4590i

Only part of it is shown above…

- I've also added in the random phase offset caused by the target:

```
targetPhi = 2*pi*rand; % The random phase offset caused by the target

*exp(targetPhi*1i)
```

- Page 153, and in and around there, needs reading! It seems like it holds valuable info.
- Page 152 contains the following table:

Transmit power	$P_{\mathrm{Tx}} = 20\,\mathrm{dBm}$
Antenna gain	$G = 0\,\mathrm{dB}$
Noise figure	$\mathrm{NF} = 5\,\mathrm{dB}$
Maximum distance	$d_{\max} = 400\,\mathrm{m}$
Maximum velocity	$v_{\mathrm{rel,max}} = 150\,\mathrm{m/s}$
Periodogram dimensions	$N_{\mathrm{Per}} = 4N$, $M_{\mathrm{Per}} = 4M$

Table 5.1: Hardware- and estimator related simulation parameters

- It seems to suggest that the dimensions of the periodogram are considerably bigger than the dimensions of the frame (N by M)
- The maximum distance of 400 m lines up roughly with what I had calculated a day or two ago, which is good!
- Section 5.2 shows:

5.2.1 Single-target accuracy

The first results are created by simulating the case where there is only one valid target, and no clutter. The signal processing matrix thus only consists of the sinusoids corresponding to this target and AWGN, as in (3.20):

$$(\mathbf{F})_{k,l} = b_0 e^{j2\pi l T_O f_c \frac{2v_{\mathrm{rel},0}}{c_0}} e^{-j2\pi k \frac{2d_0}{c_0}\Delta f} e^{j\varphi_0} + (\mathbf{Z})_{k,l}. \qquad (5.7)$$

In all of these simulations, results were calculated for a value of $d_0 = 10\,\mathrm{m}$ up to $d_0 = 400\,\mathrm{m}$, increasing the distance in steps of $1\,\mathrm{m}$. At every value of d_0, 10000 iterations were run. In one iteration, the transmit matrix $\mathbf{F}_{\mathrm{Tx}}$, the noise $\mathbf{Z}$ and phase φ_0 were generated randomly. The attenuation b_0 was calculated from the current value of d_0 according to (3.13). This creates a unique value of the SNR at every distance. Note that this affects the results, as the estimation quality depends on the SNR, which is in turn a function of the distance (the quantity to be estimated).

- A crucial detail which my project supervisor pointed out to me: when taking the transpose of a matrix of complex values, the *signs* of the complex numbers are changed

```
F =

   0.4826 + 0.2018i   1.0000 + 0.4181i
   1.0000 + 0.4181i   0.3304 + 0.1382i
   0.9850 + 0.4118i   0.5911 + 0.2471i

>> F'

ans =

   0.4826 - 0.2018i   1.0000 - 0.4181i   0.9850 - 0.4118i
   1.0000 - 0.4181i   0.3304 - 0.1382i   0.5911 - 0.2471i
```

In my code, I had:

```
119
120     % Periodogram calculation:
121     % (1): Taking the FFT of each row
122 -   F = F'; % Transposing F, as the fft() function, when used on a matrix, takes the
123     % FFT of each column... so by transposing the matrix, we turn the rows into
124     % columns...
125 -   F = fft(F); % Set size here
126
127 -   F = F'; % Turning the columns back into rows
128 -   F = ifft(F); % IFFT of each column
```

On line 122, when I take the transpose, I'm *also* changing the signs of the complex numbers, meaning when I take the FFT I'm not operating on the same data that was originally in F, before the transpose operation.

To avoid this problem, the '.' operator can be specified in tandem with '

```
F =

   0.4826 + 0.2018i   1.0000 + 0.4181i
   1.0000 + 0.4181i   0.3304 + 0.1382i
   0.9850 + 0.4118i   0.5911 + 0.2471i

>> F.'

ans =

   0.4826 + 0.2018i   1.0000 + 0.4181i   0.9850 + 0.4118i
   1.0000 + 0.4181i   0.3304 + 0.1382i   0.5911 + 0.2471i
```

- I've made the changes that I discussed with my professor:

```
F = Frx./Ftx;

% And now we can remove the rows with "Inf" values, which were caused
% by dividing by "0"s in the transmit frame (Ftx)
F(1, :) = 0;
F(N/2 + 1, :) = 0;

% Complex periodogram calculation:
Cper = F;
Cper = Cper.';
Cper = fft(Cper, 4*frameSize);
Cper = Cper.';
Cper = ifft(Cper, 4*N);

% Periodogram calculation:
Per = 1/(N*frameSize)*abs(Cper).^2;
heatmap(Per)
```

- I found a reference to the issue of division by 0 in the paper:

Prior to transmission, an active carrier list $\mathbf{N}$ is generated by randomly selecting K values from the range of total carriers, $\mathbf{N} \in \{0, \ldots, N-1\}^K$. The transmit matrix then has the following form:

$$(\mathbf{F}_{\mathrm{Tx}})_{k,l} = \begin{cases} \sqrt{U} c_{k,l} & \text{if} \quad k \in \mathbf{N} \\ 0 & \text{otherwise.} \end{cases} \tag{3.115}$$

Only the randomly selected sub-carriers are utilized, the rest are initialized with zeros (as in the previous section, the amplitude of the carriers utilized is increased by $\sqrt{U}$ to achieve the same total transmit power).

The radar processing as described in Section 3.3 is unaffected by the fact that some sub-carriers are idle, only the division (3.19) needs to be adapted to avoid a division by zero,

$$(\mathbf{F})_{k,l} = \begin{cases} \frac{(\mathbf{F}_{\mathrm{Rx}})_{k,l}}{(\mathbf{F}_{\mathrm{Tx}})_{k,l}} & \text{if} \quad k \in \mathbf{N} \\ 0 & \text{otherwise.} \end{cases} \tag{3.116}$$

This affects the estimation of the number of sinusoids and their frequencies: They now have to be estimated from a set of *non-regularly sampled* data.

- I'm checking out one of Martin Braun's papers, which contains more discussion on transmit and receive frames!

Fig. 1. OFDM Radar System Setup

Below

TABLE I
RADIO SYSTEM PROPERTIES

B	f_C	G	NF	T	σ_{RCS}
91.5 MHz	24 GHz	10 dB	20 dB	290 K	10 m^2

During transmission, the receiver is active. If at least one reflecting target is in range, a time- and frequency-shifted signal is received. This received noisy time domain signal is

$$r(t) = \sum_{h=0}^{H-1} b_h s(t - \tau_h) e^{j2\pi f_{D,h} t} + w_{\sigma^2}(t). \tag{2}$$

H is the number of reflecting targets. $b_h = |b_h| e^{j\tilde{\varphi}_h}$ is a complex attenuation factor for the h-th target. Without loss of generality, assume $|b_0| = 1$. $w_{\sigma^2}(t)$ is complex white Gaussian noise of variance σ^2.

- I think I might finally have a concrete idea of how Nmax and Mmax are being calculated:

(I)FFT$_{k,L}\{x(k)\}$ denotes a length-L (inverse) FFT of $x(k)$ with respect to k. In the equation above, the element-wise matrix product is first subject to an FFT on every column, the result is then subject to an IFFT on every row.

From assumptions 1) and 2), the resulting matrix can be cropped prior to further processing. If G is the fraction of the OFDM symbol used as cyclic prefix, and D is the fraction of the sub-carrier distance the Doppler shift can maximally be, then maximum values for $\hat{n}$ and $\hat{m}$ are defined as

$$m_{\max} = \lceil D \cdot M_{\text{FFT}} \rceil, \qquad n_{\max} = \lceil G \cdot N_{\text{FFT}} \rceil. \tag{21}$$

1) T_G is larger than the propagation time of the reflected signal to the furthest target and back.
2) Δf is at least one order of magnitude larger than the largest Doppler shift.
3) The signal's centre frequency is several orders of magnitude larger than its total bandwidth, so the Doppler shift is assumed constant over the entire bandwidth.
4) The target speed is small enough to allow for the assumption that its position can be assumed constant during one measurement.

- I've found a function to generate a matrix of WGN values, and you can specify for the output to be complex:

Command Window

```
>> wgn(32, 4, -6, 'complex')

ans =

  -0.4634 + 0.1800i  -0.2675 + 0.1090i   0.8330 + 0.1655i  -0.3147 + 0.1571i
  -0.1537 + 0.0999i   0.4856 - 0.4455i  -0.2182 - 0.0743i   0.0355 + 0.1389i
   0.1214 + 0.0119i  -0.6065 - 0.3067i   0.2651 + 0.2216i  -0.1930 - 0.4432i
   1.2682 - 0.4726i  -0.0362 - 0.0626i  -0.0682 + 0.0649i   0.1076 - 0.3360i
   0.9815 + 0.3996i  -0.0856 + 0.2805i   0.3149 - 0.3649i  -0.2128 - 0.2626i
  -0.4784 + 0.1241i   0.1131 - 0.4721i  -0.2711 + 0.3364i   0.1736 - 0.1800i
```

And you can specify the rows and columns…
The values in the above are much smaller than those in Z (the noise matrix that I created):

Z

16x4 complex double

	1	2	3	4	5
1	0.0203 + 0.1060i	0.2823 + 0.6508i	0.0905 + 0.3028i	0.0762 + 0.4949i	
2	0.3464 + 0.4664i	0.3726 + 0.0372i	0.7065 + 0.3408i	0.6409 + 0.4515i	
3	0.1187 + 0.3667i	0.2947 + 0.5217i	0.1210 + 0.0853i	0.6220 + 0.0238i	
4	0.6920 + 0.6880i	0.4645 + 0.1903i	0.0231 + 0.4168i	0.5782 + 0.0487i	
5	0.5040 + 0.4589i	0.4440 + 0.2990i	0.3968 + 0.1599i	0.1844 + 0.2260i	

14/01/2022:

- I'm going to try and implement the idea for cropping, and I'm going to change how I define Z, the AWGN noise matrix…

```
Z = wgn(N, frameSize, -30, 'complex');

noiseVar = 1; % The variance of the AWGN

% Now to model equation 3.18 from Braun Martin's paper:
for j = 1:frameSize
    Frx(:, j) = b*Ftx(:, j).*delayTerm*dopplerTerm(j)*exp(targetPhi*1i) + Z(:, j);
end
```

- This is an interesting idea for finding the position of the maximum element in a matrix:

```
>> [max_num, max_idx] = max(a(:))

max_num =

    10


max_idx =

     7
```

```
>> [x, y] = ind2sub(size(a), max_num);
>> [x, y] = ind2sub(size(a), max_num)

x =

     1


y =

     4
```

However my answers for x and y, the position of the max value of 10, are not correct:

```
a =

     1     1    10
     1     1     1
     1     1     1
```

x and y should be 1, and 3… as far as I can see…

I saw this comment on some forum, and the result looks better!

```
>> [x, y] = ind2sub(size(a), find(a == max_num))

x =

     1


y =

     3
```

a(:) seems to turn a matrix into a line or array:

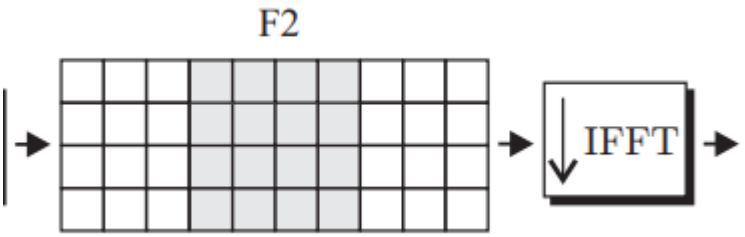
```
>> a(:)

ans =

     1
     1
     1
     1
     1
     1
    10
     1
     1
```

- I'm getting a more reasonable value for the distance:

```
tau =

   3.9375e-07


distance =

   59.0625
```

and the time delay is one order of magnitude closer to the actual time delay…

- Another thing, I noticed in Martin Braun's PhD paper, he had 10 columns for Mper



More cropping is done after the IFFT block…

In working out N_{max} and M_{max}, I chose D = ⅛, as it divides into 4*32, which is my frame size.

```
% Complex periodogram calculation:
Nper = 4*N;
Mper = 4*frameSize;
G = 1/4;
D = 1/8;
Nmax = G*Nper;
Mmax = D*Mper;
```

But as can be seen below (a different paper by Martin Braun), Martin suggests a typical value of D being 1/10…

the sub-carrier distance the Doppler shift can maximally be, then maximum values for $\hat{n}$ and $\hat{m}$ are defined as

$$m_{\max} = \lceil D \cdot M_{\mathrm{FFT}} \rceil, \qquad n_{\max} = \lceil G \cdot N_{\mathrm{FFT}} \rceil. \tag{21}$$

The ranges $-m_{\max} \ldots m_{\max}$ and $0 \ldots n_{\max}$ shall be called *search ranges*. Typical values are $G = 1/4$ and $D = 1/10$.

This is for assumption 2 not to be violated:

2) Δf is at least one order of magnitude larger than the largest Doppler shift.

Maybe that's why ten is there, as an order of magnitude difference usually means bigger or smaller by a factor of 10…

If I set my D = 1/10, then I need Mper to be divisible by 10

- Another point of importance seems to be that the modulation scheme is normalised to unit power:

> 6) The modulation system is normalized to unit power, i.e. $\mathrm{E}\left\{|(\mathbf{F}_{\mathrm{Tx}})_{k,l})|^2\right\} = 1$. This is always the case for constant-modulus alphabets such as phase shift keying (PSK).

We can work out the mean of our transmit frame by using (:) with the mean() function:

```
>> a(1, 3) = 10

a =

     1     1    10
     1     1     1
     1     1     1

>> mean(a)

ans =

     1     1     4

>> mean(a(:))

ans =

     2
```

- It looks like I don't have unit power after the modulation

```
ans =

    6.0094
```

```
>> mean(q(:))

ans =

    0.9196
```

Much better now…

> As an example, a system with the following parameters shall be analysed: $M = 128$, $N = 1200$, $\Delta f = 76.25$ kHz and $T_G = 1/(4\Delta f)$. The radio system is parametrized as in Table I. D and G are fixed to $1/10$ and $1/4$, respectively, resulting in $m_{\max} = 7$ and $n_{\max} = 300$. The following calculations are

From the above, I can see how nmax is 300 (as that is what one gets when they do (1200)(¼)... however, I'm not sure where the value of 7 for mmax is coming from…

- The answer that I get for d when using my value for tau is the same as that which I get from the equation for d:

```
tau =

   1.1875e-07


distance =

   17.8125
```

```
tau = (n_hat - 1)/(Nper*subcarrierSpacing)

distance = (n_hat - 1)*c/(2*subcarrierSpacing*Nper)
velocity = (m_hat - 1)*c/(2*fC*To*Mper)
```

When using the cropped periodograms as described in Section 3.3.2, the factor NM is replaced by $N_{\max}(2M_{\max}+1)$ in (3.95) and (3.97). However, when zero-padding is used, these values are *not* increased by the interpolation factor. It is true that if there are more bins in the periodogram due to zero-padding there are potentially more bins to cause

Equations 3.95 and 3.97 look like the following:

$$\eta = \sigma_N^2 \ln(1-(1-p_{\text{FA}})^{\frac{1}{NM}}). \tag{3.95}$$

If the requirement is a certain FAR, which is calculated by

$$\text{FAR} = NMp_{\text{FA, bin}}, \tag{3.96}$$

the threshold is set by

$$\eta = -\sigma_N^2 \ln \frac{\text{FAR}}{NM}. \tag{3.97}$$

1. We first skip CP samples, then take a buffer of N samples and transform that into the frequency domain using a N-point FFT.
2. Since we know the transmit signal and its frequency representation, we can just divide out the transmit symbols and receive an estimate of the channel influence in the frequency domain. Note that if the carrier was unused, it obviously can't be divided by, and is just set to zero. Because our system has a small doppler shift compared to `delta_t`, we can assume minimal inter-carrier interference.

The above is from an implementation of a OFDM radar scheme that I found on GitHub…

In mathematics, the **arguments of the maxima** (abbreviated **arg max** or **argmax**) are the points, or elements, of the domain of some function at which the function values are maximized.[note 1] In contrast to global maxima, which refers to the largest *outputs* of a function, arg max refers to the *inputs*, or arguments, at which the function outputs are as large as possible.

General Parameters: 802.11a/g uses an $N = 64$ point FFT in a 20MHz channel, $\delta_f = 1/T_s = 312.5$kHz, $T_s = 3.2\mu s$, 4 μs time block is used, with cyclic prefix, 52 of the 64 subcarriers are populated, 4 are pilots (for phase and frequency training and tracking), 48 actually carry data.

The 802.11g packet structure includes the following:

Short training field (STF):
8 μs; 10x repetition of 0.8 μs symbol. Uses 12 subcarriers; good autocorrelation property and low peak to average ratio; also used for automatic gain control (AGC)
Long training field (LTF):
8 μs, composed of two 3.2μs training symbols and prepended by a 1.6μs cyclic prefix. Used for time acquisition and channel estimation.
Signal field (SiG):
4 μs (3.2 + 0.8 cyclic prefix), contains 24 bits BPSK describing transmit rate, modulation, coding, length. Forms together with the training field the preamble (totaling 20 μs).
Data Field:
includes service field (16 bits: 7 used to synchronize descrambler, 9 reserved for future use), data bits, and tail bits, and optionally padding bits. The Data field consists of a stream of symbols, each 4 μs (3.2 + 0.8 cyclic prefix), transmitted over 48 subcarriers, and 4 pilots.

It seems that in the 802.11 standard, a 64-point (I)FFT is used, where only 52 of the channels are used for transmission.

This is the section that I need to read thoroughly:

5.2.1 Single-target accuracy

The first results are created by simulating the case where there is only one valid target, and no clutter. The signal processing matrix thus only consists of the sinusoids corresponding to this target and AWGN, as in (3.20):

$$(\mathbf{F})_{k,l} = b_0 e^{j2\pi l T_O f_c \frac{2v_{\mathrm{rel},0}}{c_0}} e^{-j2\pi k \frac{2d_0}{c_0}\Delta f} e^{j\varphi_0} + (\mathbf{Z})_{k,l}. \tag{5.7}$$

In all of these simulations, results were calculated for a value of $d_0 = 10\,\mathrm{m}$ up to $d_0 = 400\,\mathrm{m}$, increasing the distance in steps of $1\,\mathrm{m}$. At every value of d_0, 10000 iterations were run. In one iteration, the transmit matrix $\mathbf{F}_{\mathrm{Tx}}$, the noise $\mathbf{Z}$ and phase φ_0 were generated randomly. The attenuation b_0 was calculated from the current value of d_0 according to (3.13). This creates a unique value of the SNR at every distance. Note that this affects the results, as the estimation quality depends on the SNR, which is in turn a function of the distance (the quantity to be estimated).

Because only one target was active, there was no need to implement a multi-target detector. Instead, the largest peak of the (cropped) periodogram was chosen as the estimate. This will highlight the need for a threshold and demonstrate the previously discussed threshold effect.

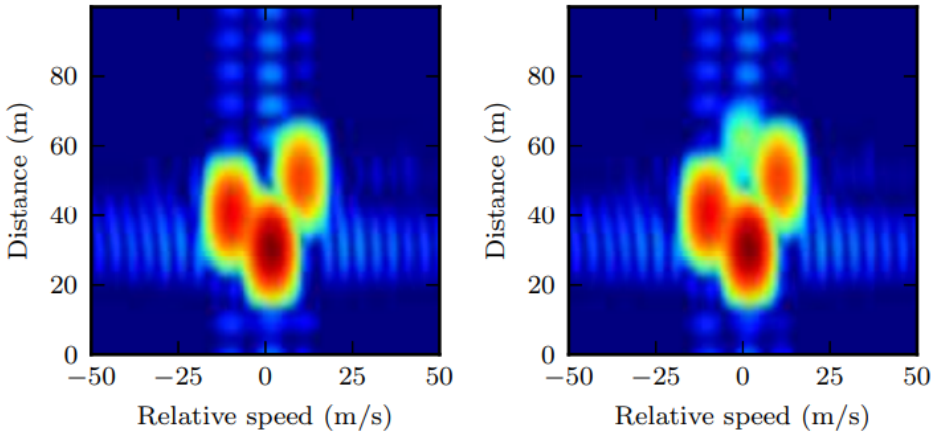
From the above, it seems that the periodogram-based estimation was used. And the peak of the cropped version was taken as the estimate…

Standard	802.11a	802.11p
f_C	$5.5\,\mathrm{GHz}$	$5.9\,\mathrm{GHz}$
$P_{\max}$	$20\text{-}30\,\mathrm{dBm}$ (depending on region)	$33\,\mathrm{dBm}$
Δf	$20\,\mathrm{MHz}/64 = 312.5\,\mathrm{kHz}$	$5\,\mathrm{MHz}/64 = 78.125\,\mathrm{kHz}$
T_G	$1/4T$	$1/4T$
N	52	52
M	≤ 1365	≤ 1365

I need to understand this better, and implement these standards into my code…
I'm already using the same subcarrier spacing and the same centre frequency, but my N and M values are way off… Below is shown a more detailed table:

802.11a Timing Related Parameters

Parameter	Value
Total subcarriers N_{ST}	52
Data subcarriers N_{SD}	48
Pilot subcarriers N_{SP}	4 (subcarriers -21, 7, 7, 21)
Subcarrier Frequency Spacing F_{SP}	312.5 KHz (20MHz/64)
Symbol Interval Time T_{SYM}	4 us (T_{GI} +T_{FFT})
Data Interval Time T_{DATA}	3.2 us (1/F_{SP})
Guard Interval (GI) Time T_{GI}	0.8 us (T_{FFT}/4)
IFFT/FFT Period T_{FFT}	3.2 us (1/F_{SP})
SIGNAL Symbol TIme T_{SIGNAL}	4 us (T_{GI} +T_{FFT}
Preamble $T_{PREAMBLE}$	16 us (T_{SHORT} +T_{LONG})
Short Training Sequence T_{SHORT}	8 us (10xT_{FFT}/4)
Long Training Sequence T_{LONG}	8 us (T_{GI2} + 2xT_{FFT})
Training symbol GI T_{GI2}	1.6 us (T_{FFT}/2)
FFT sample size	64 point



(a) Three targets, with different Doppler.

(b) A fourth, large target at a distance appears as an aliased image.

Figure 3.20: Periodogram using an 802.11a signal parametrization, using a Hamming window. All three main targets have a RCS of $\sigma_{\mathrm{RCS}} = 10\,\mathrm{m}^2$.

I'm trying to implement the 802.11a standard, and simulate a situation with one target.

Ftx

53x128 complex double

	1	2	3	4	5	6	7	8	9	10	11
25	0.3162 - 0....	-0.9487 - 0....	-0.9487 - 0....	-0.9487 + 0....	-0.9487 - 0....	0.3162 - 0....	-0.3162 + 0....	-0.3162 + 0....	0.9487 - 0....	-0.3162 + 0....	-0.3162 -
26	-0.9487 + 0....	0.3162 + 0....	0.3162 + 0....	-0.3162 + 0....	-0.3162 - 0....	0.3162 + 0....	0.9487 - 0....	-0.3162 + 0....	0.3162 + 0....	-0.9487 + 0....	-0.3162 -
27	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 +
28	-0.9487 - 0....	0.3162 - 0....	0.3162 - 0....	-0.3162 - 0....	-0.3162 + 0....	0.3162 - 0....	0.9487 + 0....	-0.3162 - 0....	0.3162 - 0....	-0.9487 - 0....	-0.3162 +

- I'm using a double or nested for-loop in order to try and see if my issue with the periodogram estimation is to do with how I've coded up equation (3.19)...
- The results seem more promising, at least for distance and time delay:

```
tau =

   1.0000e-07


distance =

    15
```

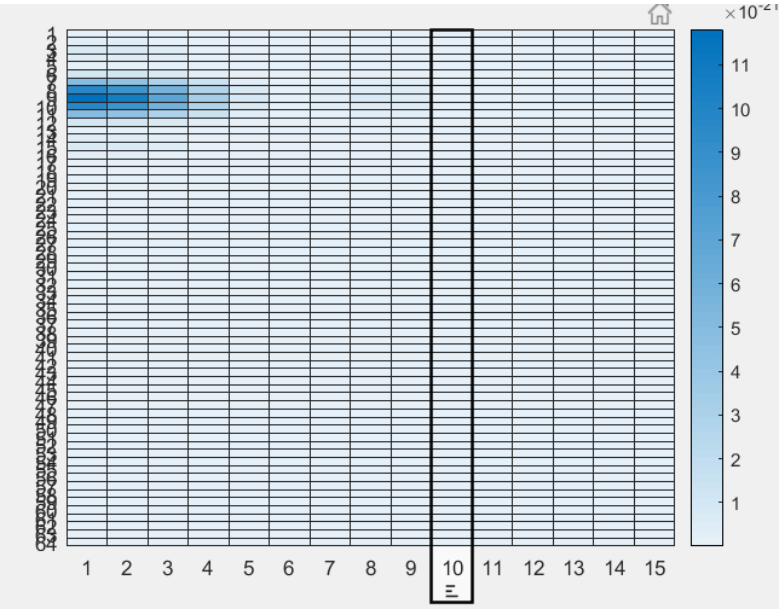
```
d = 15; % The distance between the target and the transmitter
```

timeDelay 1.0000e-07

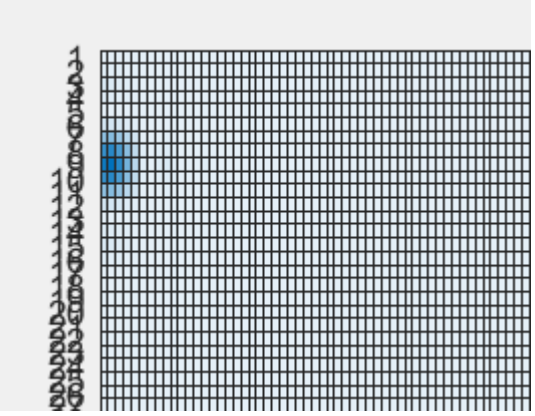
```
m_hat = m_hat - (Mmax + 1)
tau = (n_hat - 1)/(Nper*subcarrierSpacing)

distance = (n_hat - 1)*c/(2*subcarrierSpacing*Nper)
velocity = (m_hat - 1)*c/(2*fC*To*Mper)
```

```
velocity =

 -974.0260
```

But the velocity is still way off… I think this must be to do with how I'm working out the *m* indexes…



The periodogram looks a lot better though!



And I can increase M, the frame size… still, my velocity value is way off…

```
41      % Target parameters:
42 -    Vrel = 35; % The relative velocity of our target to the transmitter
43 -    d = 15; % The distance between the target and the transmitter
44 -    targetRCS = 10; % The radar cross section of the target
```

Command Window

```
   1.0000e-07


distance =

    15


velocity =

   36.6211
```

I think I might have mistakenly fixed it

```
41      % Target parameters:
42 -    Vrel = 90; % The relative velocity of our target to the transmitter
43 -    d = 15; % The distance between the target and the transmitter
44 -    targetRCS = 10; % The radar cross section of the target
```

Command Window

```
   1.0000e-07


distance =

    15


velocity =

   89.8881
```

15/01/2022:

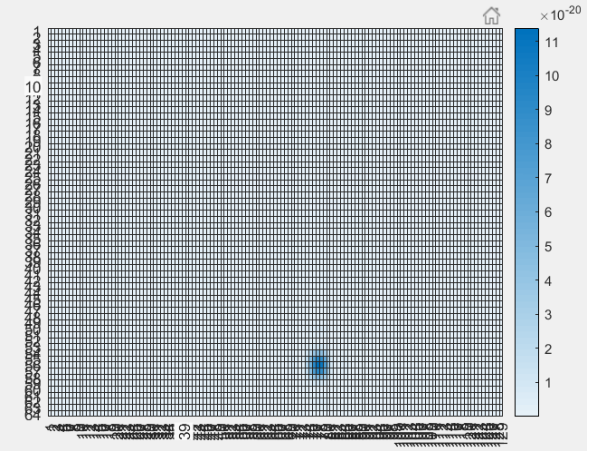
- I've just checked that I can reverse or flip my Per matrix (I tested out the sequence of steps on a smaller matrix):

```
a =

    0.7711    0.3513    1.0000    0.8582    0.2600
    0.8297    0.9105    1.0000    0.9268    0.7028
    0.0943    0.6348    1.0000    0.6260    0.8495
    0.5301    0.7505    1.0000    0.4829    0.7786
    0.5806    0.1680    1.0000    0.2795    0.0185

>> b = [a(:, floor(5/2)+2:end) a(:, floor(5/2)+1) a(:, 1:floor(5/2))]

b =

    0.8582    0.2600    1.0000    0.7711    0.3513
    0.9268    0.7028    1.0000    0.8297    0.9105
    0.6260    0.8495    1.0000    0.0943    0.6348
    0.4829    0.7786    1.0000    0.5301    0.7505
    0.2795    0.0185    1.0000    0.5806    0.1680
```

It also works if the matrix a has an even number of columns:

```
a =

    0.6838    0.4225    1.0000    0.0695    0.2058    1.0000
    0.9891    0.6994    1.0000    0.3058    0.2333    1.0000
    0.0786    0.2552    1.0000    0.4679    0.5321    1.0000
    0.3706    0.4963    1.0000    0.5579    0.3182    1.0000
    0.9288    0.4433    1.0000    0.3109    0.1957    1.0000

>> b = [a(:, floor(5/2)+2:end) a(:, floor(5/2)+1) a(:, 1:floor(5/2))]

b =

    0.0695    0.2058    1.0000    1.0000    0.6838    0.4225
    0.3058    0.2333    1.0000    1.0000    0.9891    0.6994
    0.4679    0.5321    1.0000    1.0000    0.0786    0.2552
    0.5579    0.3182    1.0000    1.0000    0.3706    0.4963
    0.3109    0.1957    1.0000    1.0000    0.9288    0.4433
```

The heat map looks better now, but it’s still not correct:



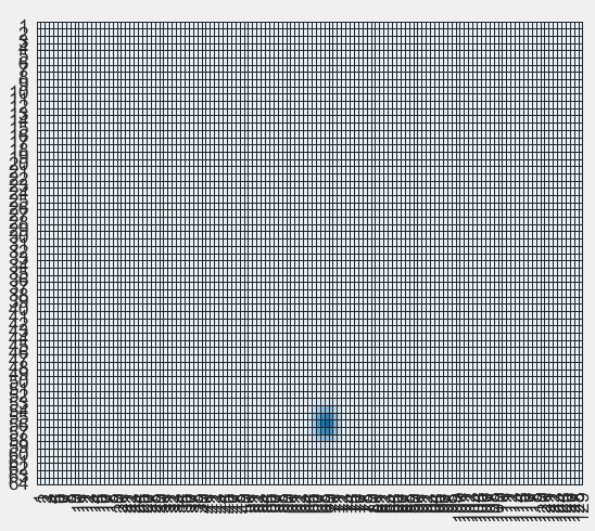
I also flipped the matrix upside down, as I want small distances to be near the bottom, and large distances to be near the top…

- I think I might need to use fftshift, to centrally shift the *m* indexes… instead of my method of rearranging the matrix…

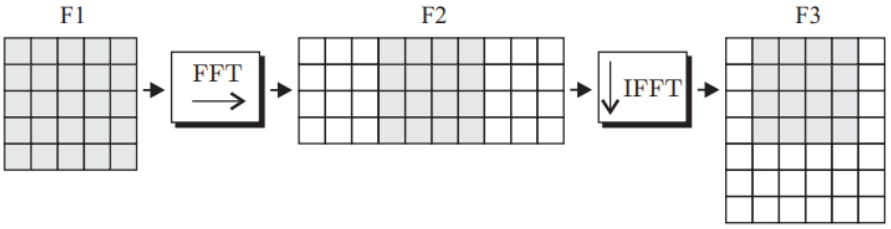
```
a =

     2     4     3     6     7

>> b = fftshift(a)

b =

     6     7     2     4     3
```

```
a =

     2     4     3     6     7

>> b = fftshift(a)

b =

     6     7     2     4     3

>> c = [a; b]

c =

     2     4     3     6     7
     6     7     2     4     3

>> d = fftshift(c, 2)

d =

     6     7     2     4     3
     4     3     6     7     2
```

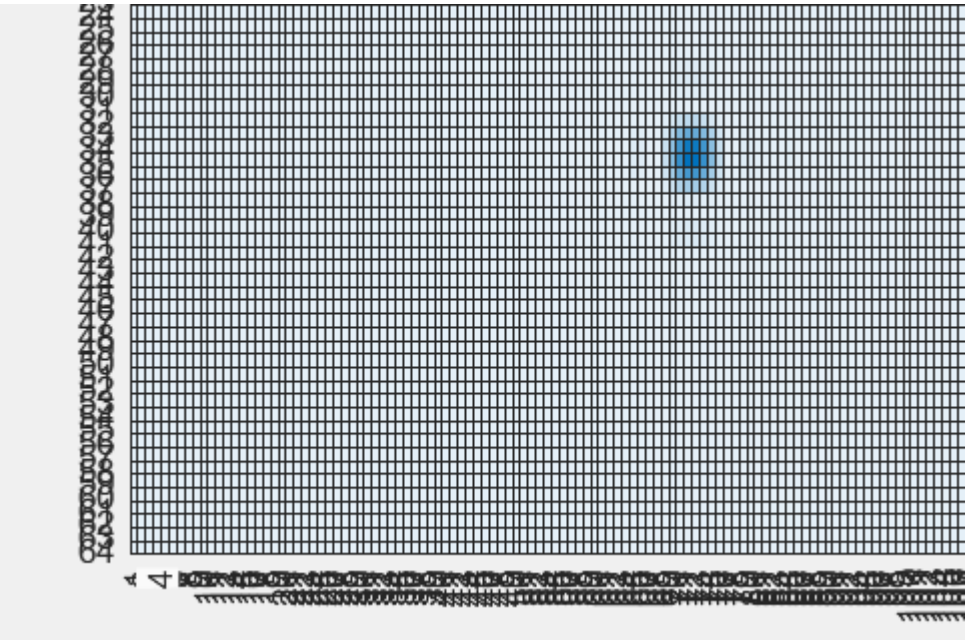
I might be able to do the above (the application of fftshift to c) on my Periodogram matrix. Shown below is the result (distance = 15, velocity = 50), which looks promising… maybe…



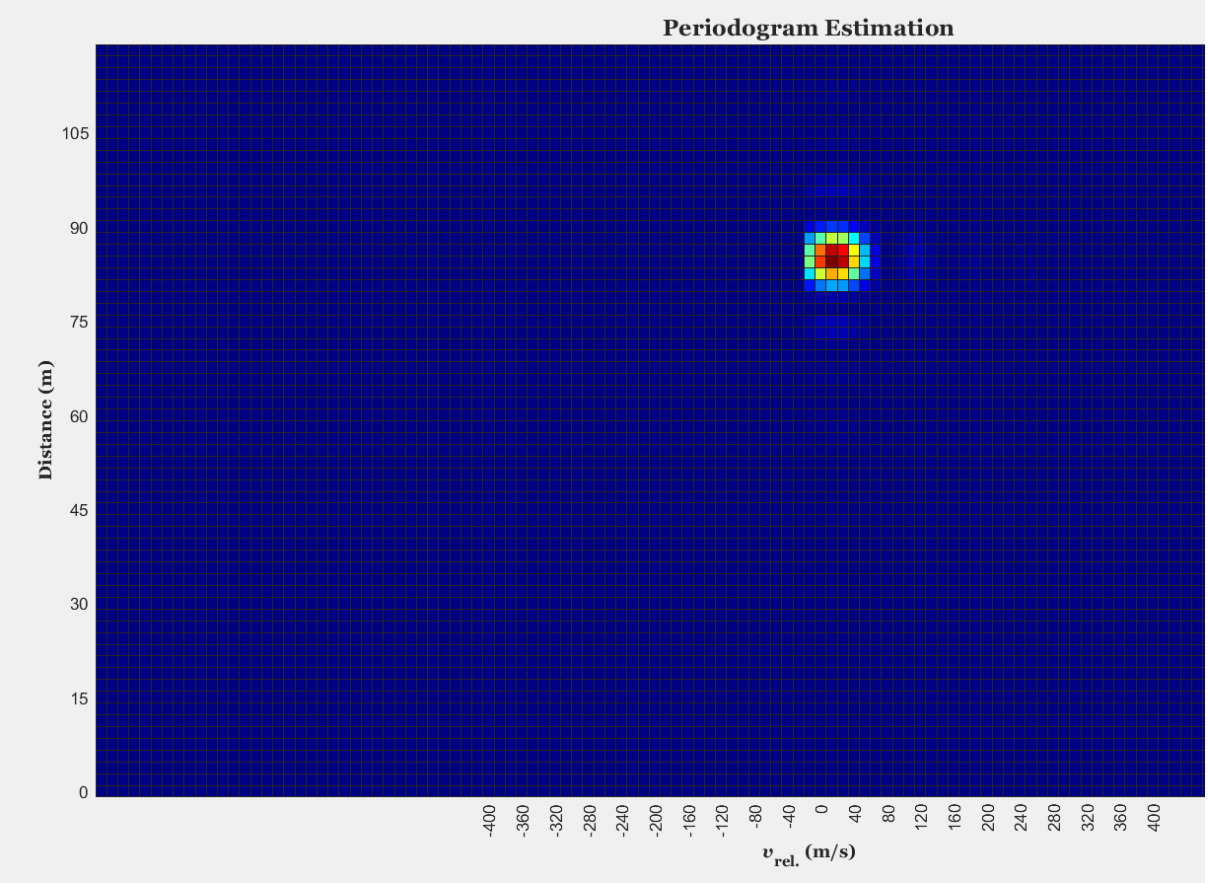
- I think when I'm doing the fft of each row (of length Mper), I need to apply the padding on both the left and right sides of the M columns, as indicated in a figure in the PhD paper by Braun… when I do fft(Cper, Mper), MATLAB might only be padding on the end of the each row… to the right…



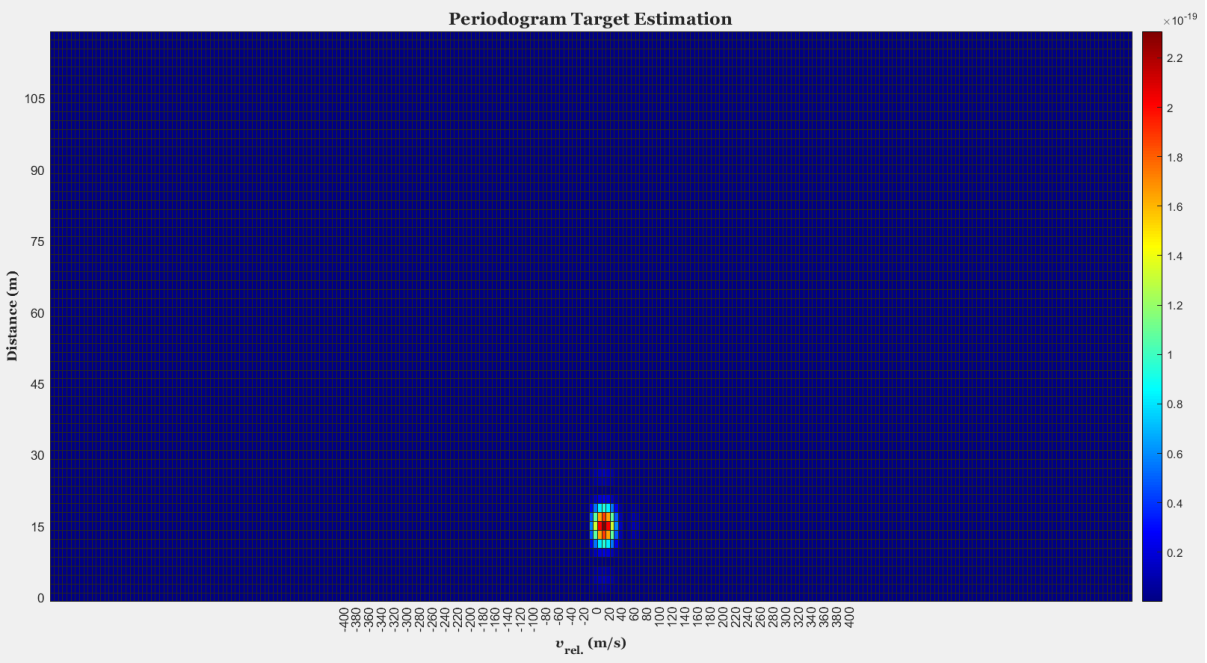
Still, it seems to work well with positive values, and increasing the distance from 15 to 55 causes the blue collection of cells to move upwards… indicating that the target is a greater distance away…



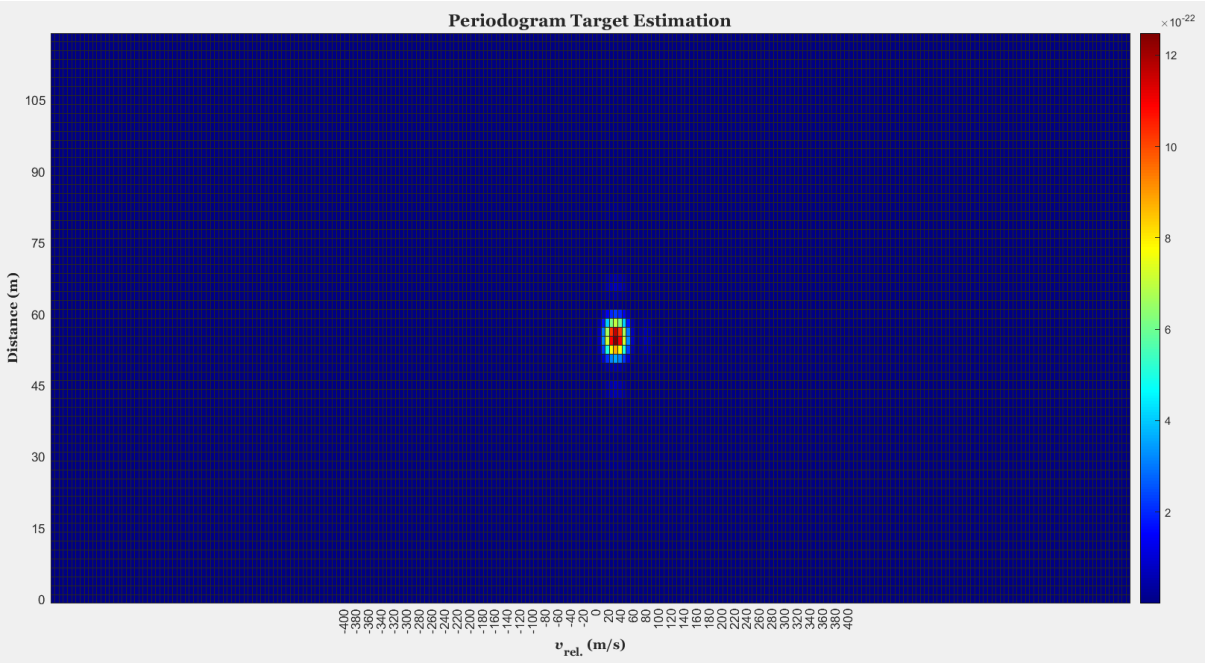
- I've made some changes to the periodogram (although I've learned that working with the heatmap() function is anything but easier: changing the axis labels and tick values is a serious pain…):

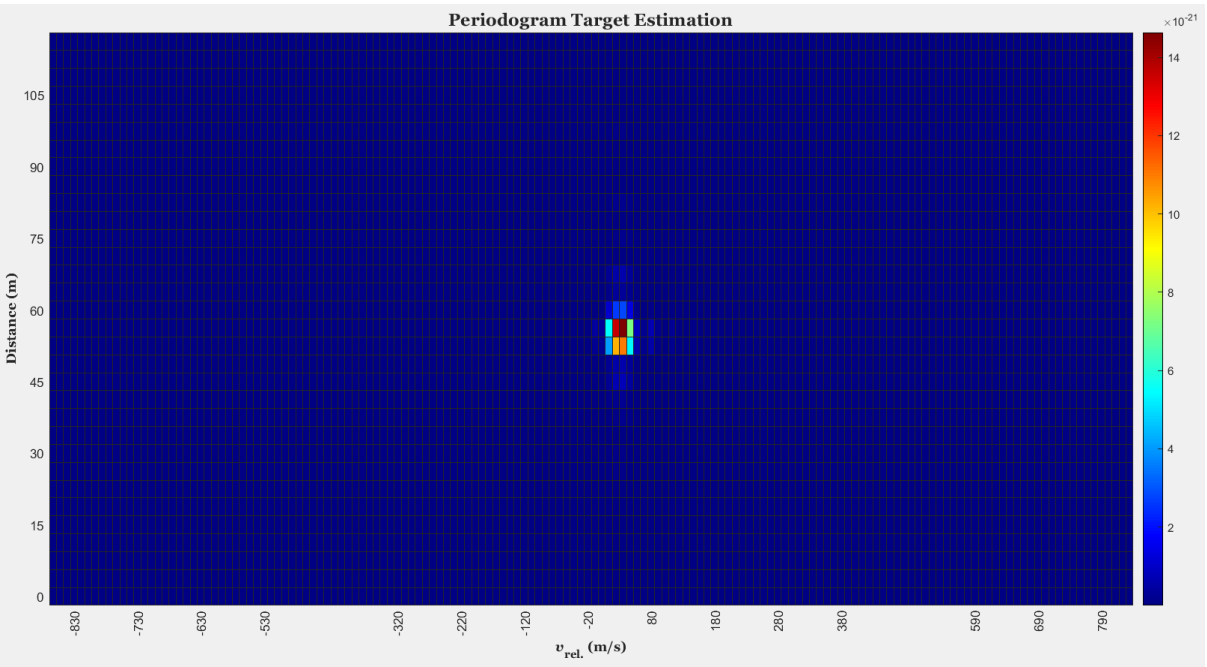


- Here are some results which I plan on including in the interim report:
  d = 15
  v = 20



d = 55
v = 40





- I will want to keep the following in mind, while writing the report:

Grade	Criteria
A-/A/A+	**Excellent:** Extremely well written report. Very comprehensive research literature review. Excellent knowledge and understanding of the problem. Very well-developed project plan, and thoughtful consideration of ethical/sustainability implications. Initial results showing very good promise.
B-/B/B+	**Very Good:** Very well written report with thorough research literature review. Very good knowledge and understanding of the subject matter demonstrated. Well-developed, realistic plan, with demonstrated awareness of ethical/sustainability implications. Initial results showing strong promise.

- The value of G = ¼, given in the paper by Braun et al, seems to match up with the value you get when T_G is divided by T:

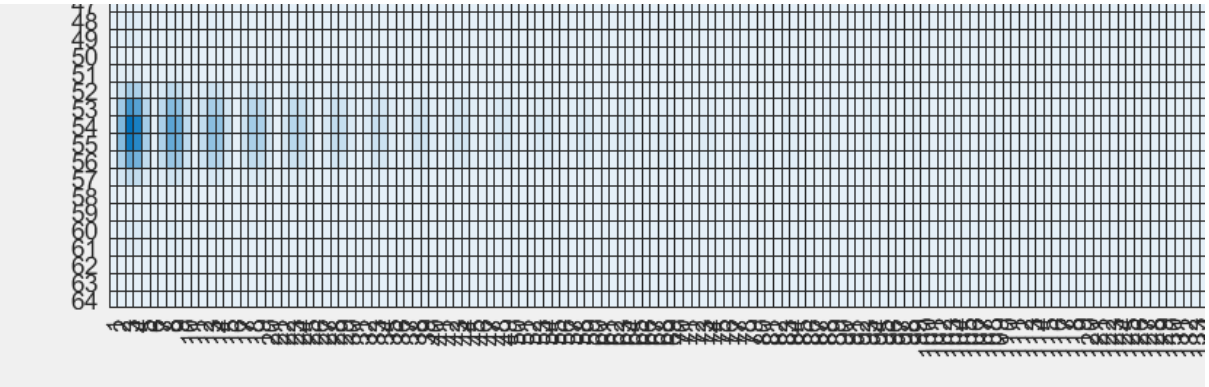
Guard Interval (GI) Time T_{GI}	0.8 us ($T_{FFT}/4$)
IFFT/FFT Period T_{FFT}	3.2 us ($1/F_{SP}$)

  0.8/3.2 = ¼

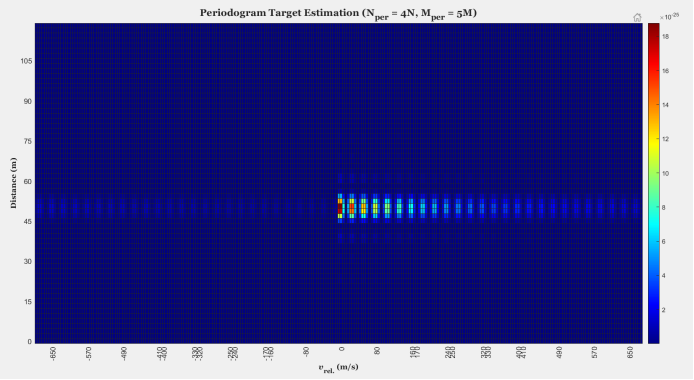
  This means G = T_G/T, and **not $T_O = T + T_G$**

18/01/2022:

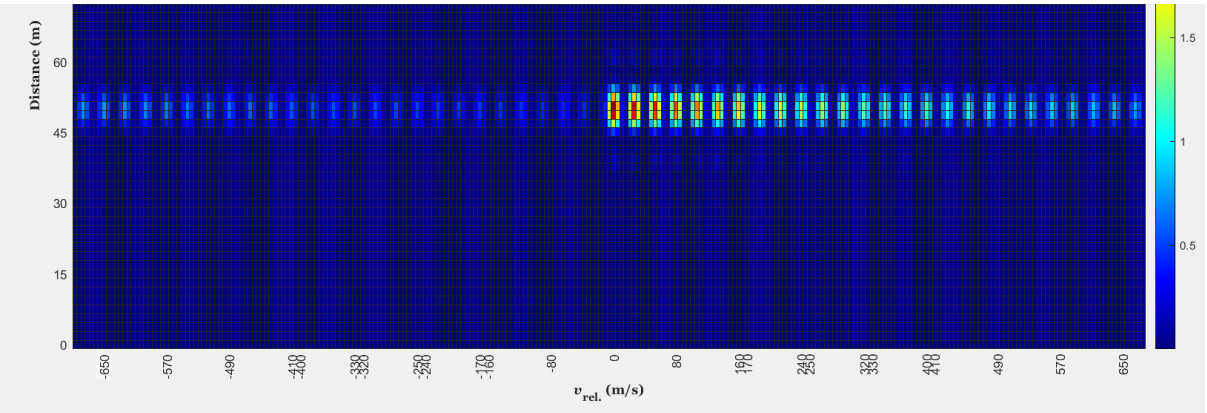
- When I make the relative velocity value negative, the peak in the periodogram goes somewhat strange:



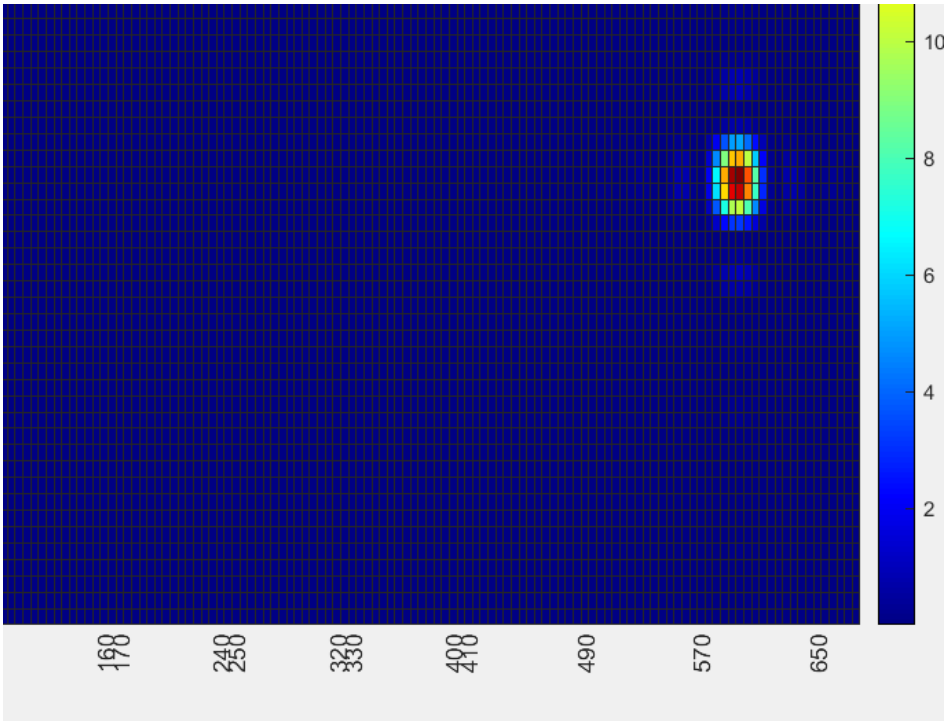
- This next diagram shows it again…



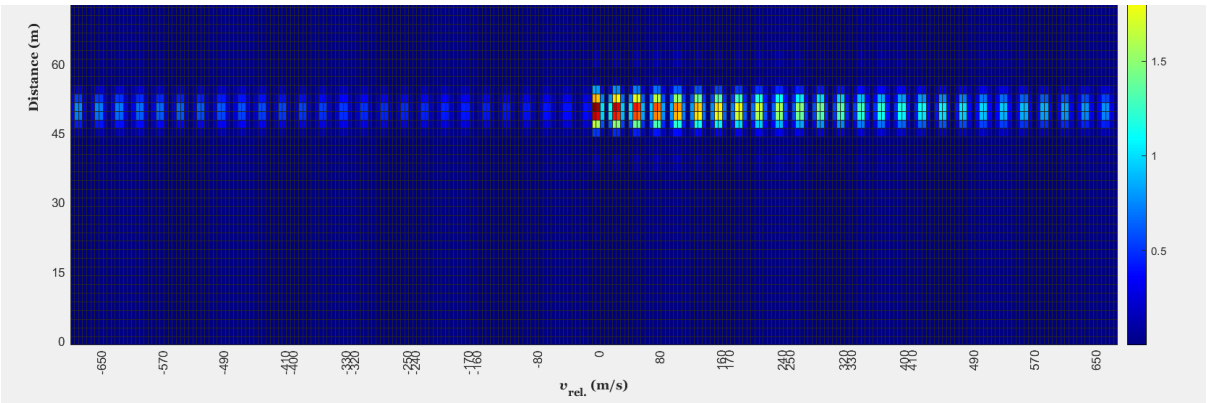
- The diagram below is for V = -700



And the result below is for V = 600 m/s…



But negative velocities aren't showing up correctly…



I think the equation below:

$$\mathrm{Per}_{\mathbf{F}}(n,m) = \frac{1}{NM}\left|\overbrace{\sum_{k=0}^{N_{\mathrm{Per}}-1}\left(\overbrace{\sum_{l=0}^{M_{\mathrm{Per}}-1}(\mathbf{F})_{k,l}e^{-j2\pi\frac{lm}{M_{\mathrm{Per}}}}}^{N\text{ FFTs of length }M_{\mathrm{Per}}}\right)e^{j2\pi\frac{kn}{N_{\mathrm{Per}}}}}^{M_{\mathrm{Per}}\text{ IFFTs of length }N_{\mathrm{Per}}}\right|^2 \quad (3.30)$$

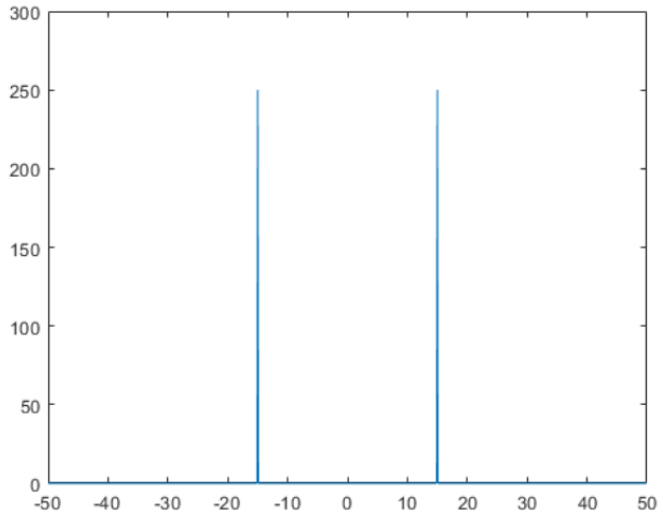
$$= \frac{1}{NM}\left|\mathrm{CPer}_{\mathbf{F}}(n,m)\right|^2. \quad (3.31)$$

Which contains the small index *m*, which means the way in which I'm computing the FFTs of each row needs to change…

- I think I'm on the right track when searching up the fftshift() function:

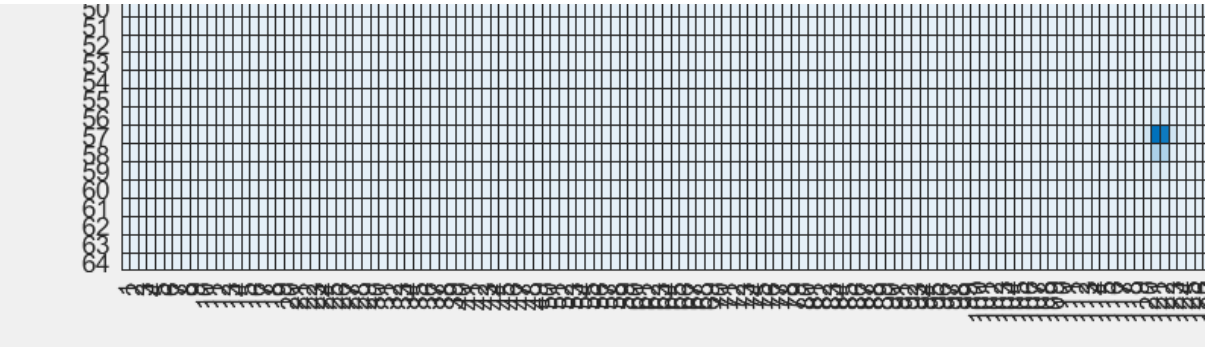
Shift the zero-frequency components and plot the zero-centered power.

```
Y = fftshift(X);
fshift = (-n/2:n/2-1)*(fs/n); % zero-centered frequency range
powershift = abs(Y).^2/n;     % zero-centered power
plot(fshift,powershift)
```



As can be seen above, the fshift vector is going from -n/2 to n/2 - 1… similar to the *m* index in the periodogram…

I've reduced N_{per} and M_{per} to be equal to N and M… when I make the velocity negative, I notice that the *m* index is *below* the midway point on the horizontal axis… this would correspond to a negative *m* index…



When I make the velocity positive, it goes above the midway point on the horizontal axis… this corresponds to a positive *m* index…

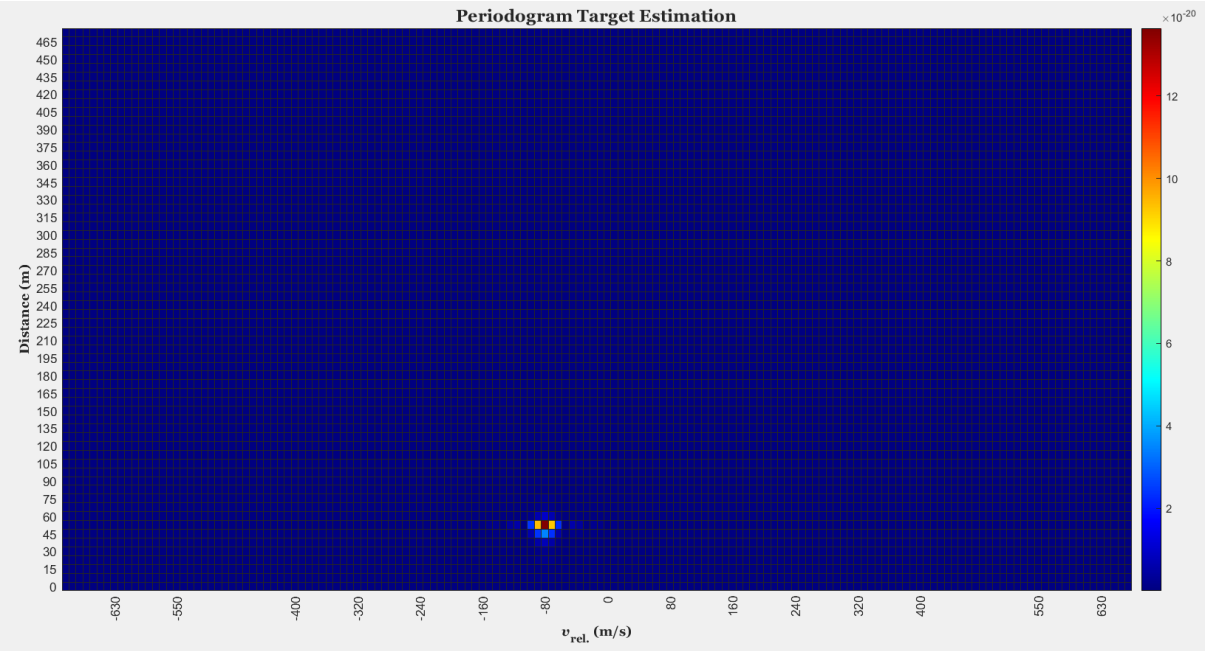
The graph below was generated when d = 50, v = -400. The x-axis index was 114… this means with respect to the centre, the *m* index is (114 -256/2 + 1 = -15). Subbing this into the equation for velocity:

$$\hat{d} = \frac{\hat{n}c}{2\Delta f N_{per}} \quad and \quad \hat{v} = \frac{\hat{m}c}{2f_C T_O M_{per}}$$

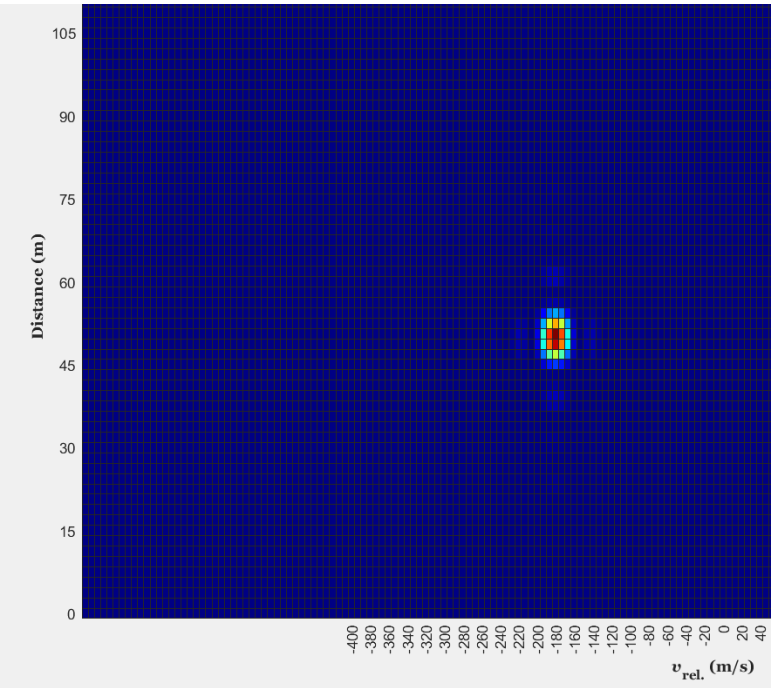
gives -399.5 m/s… so this *seems* to be checking out…

19/01/2022:

- I've got a graph which I *think* is more correct than before. When I set the target's parameters to d = 50 and v = -80, it seems to show a brighter spot (indicating the peak) at the correct position…



- I'm also, it seems, able to change the dimensions of the periodogram (N_{per} and M_{per})...



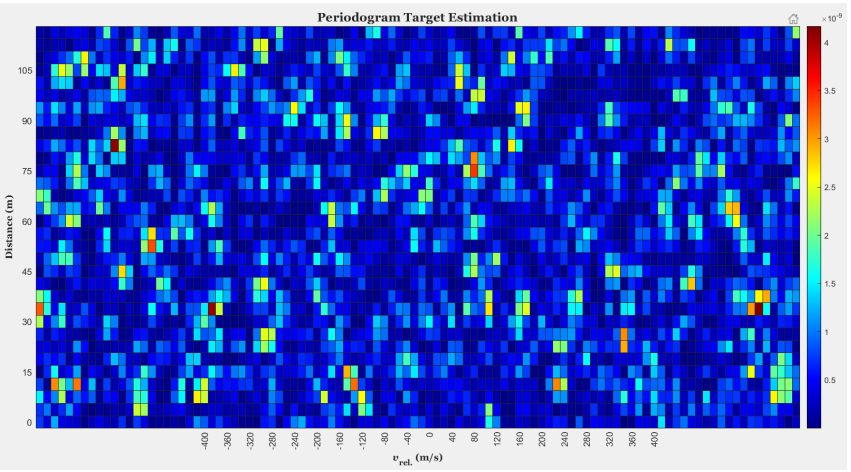
20/01/2022:

- I've printed out some more papers to use as references. I've also started a new document where the main findings of reading each paper will be noted… this can be quickly referenced in the future…

25/01/2022:

- I've added the *noise* matrix into the expression that calculates F_{Rx}, the receive frame:

```
*timeDelay*subcarrierSpacing) + Z(row, column);
```



As can be seen above, the picture is much more chaotic. As well as that, the values for the distance and the velocity are incorrect:

```
distance =

   33.7500


velocity =

  585.9375
```

These values match up with the darkest point or cell on the periodogram (the cell with the highest value.)

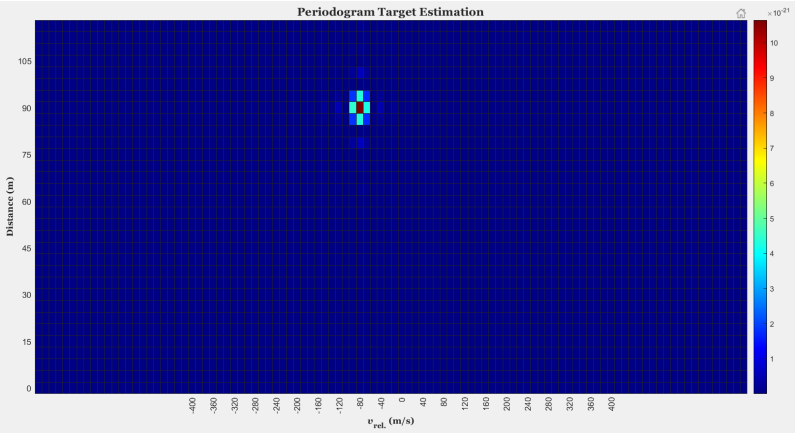
Changing the *strength* or *power* of the noise seems to improve things:

```
Z = wgn(N, frameSize, -50, 'complex');
```

to…

```
Z = wgn(N, frameSize, -200, 'complex');
```

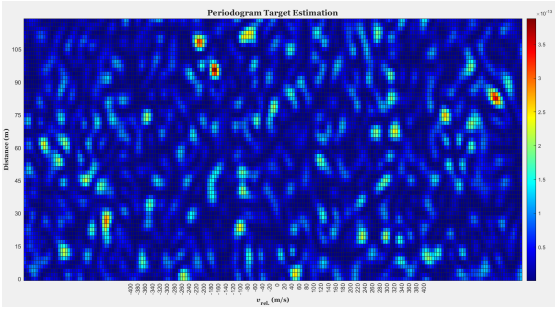
Results in…



However, the value of -200 for the matrix *Z* is **very unrealistic**, I think (I need to check this…)

For example, a value of “-6” would correspond to: $10^{-6/10}$ = 0.25 watts

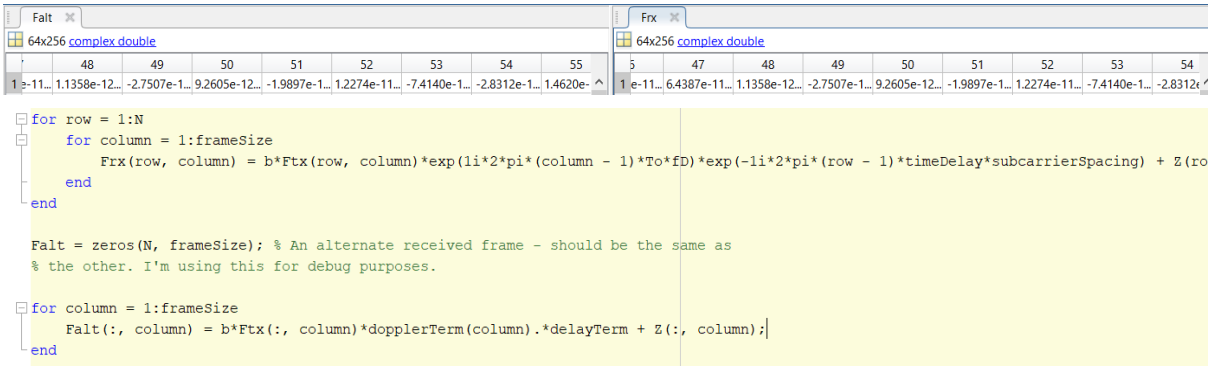
So a value of -200 would correspond to: $10^{-200/10} = 1 \times 10^{-20}$ watts



Increasing the size of the periodogram doesn’t help so much with reducing the noise’s effect… The SNR isn’t good enough at the moment…

26/01/2022:

- It seems that the first row of F_{Rx} is calculated correctly when using my slightly more compact calculation for F_{Rx}:

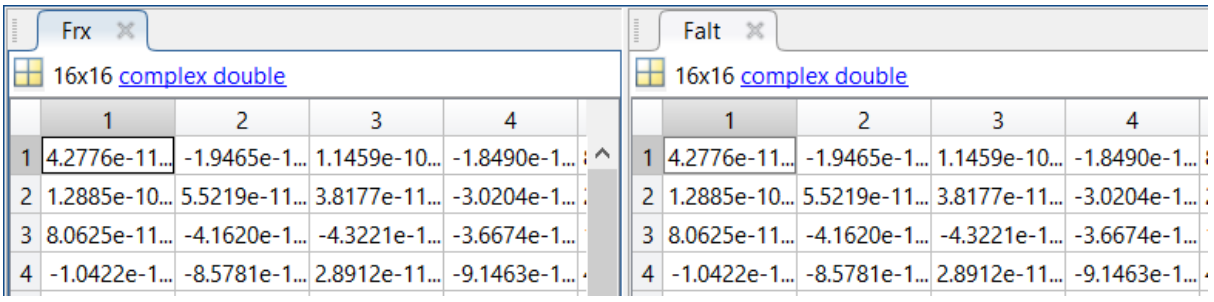


- Finally I know the issue! It turns out the transpore operator was causing the problem:

```
dopplerTerm = exp(1i*2*pi*l*To*fD)';
```

l is a vector, and so, therefore, is *dopplerTerm*. It's also *complex*, meaning when you take the transpose, the sign of the imaginary part of each entry **changes**… (I don't why 👀). The solution was to do the following:

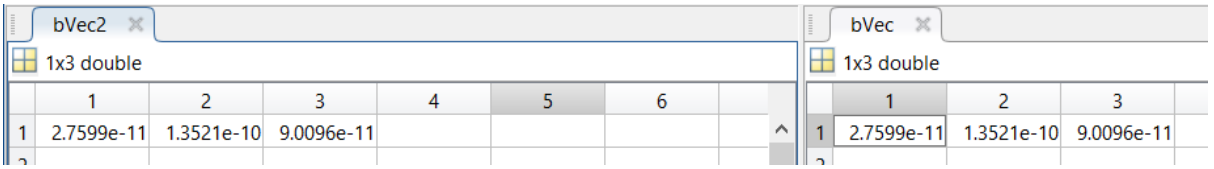
```
dopplerTerm = exp(1i*2*pi*l*To*fD).';
```



	1	2	3	4
1	4.2776e-11...	-1.9465e-1...	1.1459e-10...	-1.8490e-1...
2	1.2885e-10...	5.5219e-11...	3.8177e-11...	-3.0204e-1...
3	8.0625e-11...	-4.1620e-1...	-4.3221e-1...	-3.6674e-1...
4	-1.0422e-1...	-8.5781e-1...	2.8912e-11...	-9.1463e-1...

Finally they are the same!
*That's the 3rd or 4th time that I've been caught out by the operation of the transpose operator…

- I'm still working on this script which will hopefully emulate the situation in which there are *many*, or *multiple* targets in the field of view of the radar.



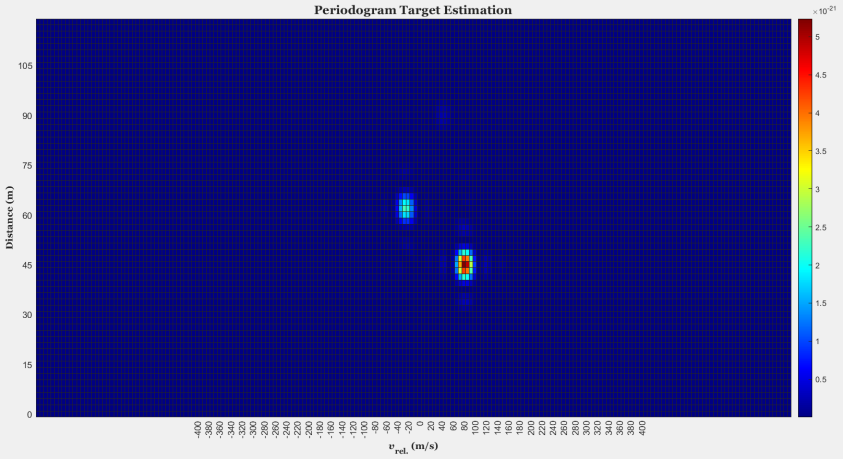
	1	2	3	4	5	6
1	2.7599e-11	1.3521e-10	9.0096e-11			

	1	2	3
1	2.7599e-11	1.3521e-10	9.0096e-11

The two vectors contain the three attenuation factors for three targets. As can be seen, the two vectors are the same, although the ways they are got or calculated are different:

```
bVec = sqrt((c.*targetRCSs)./((4*pi)^3*(targetDistances.^4)*(fC^2))); % The attenuations associated with the

bVec2 = zeros(1, 3);
for j = 1:3
    bVec2(j) = sqrt((c*targetRCSs(j))/((4*pi)^3*(targetDistances(j)^4)*(fC^2)));
end
```

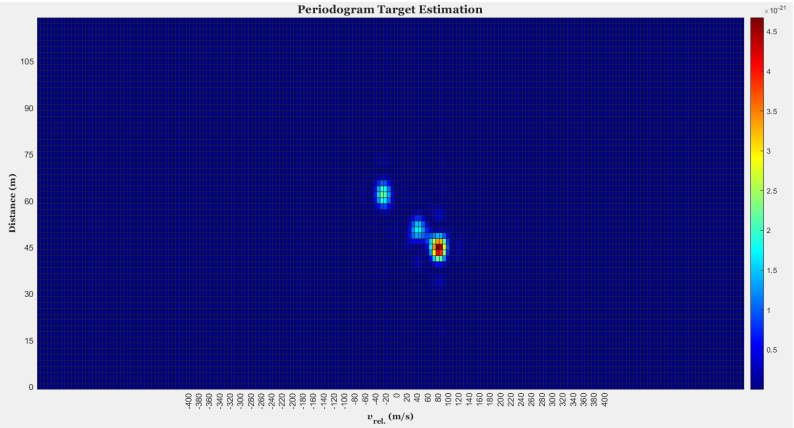
I've got something basic up and running, but further testing will need to be done to verify that the behaviour is correct:

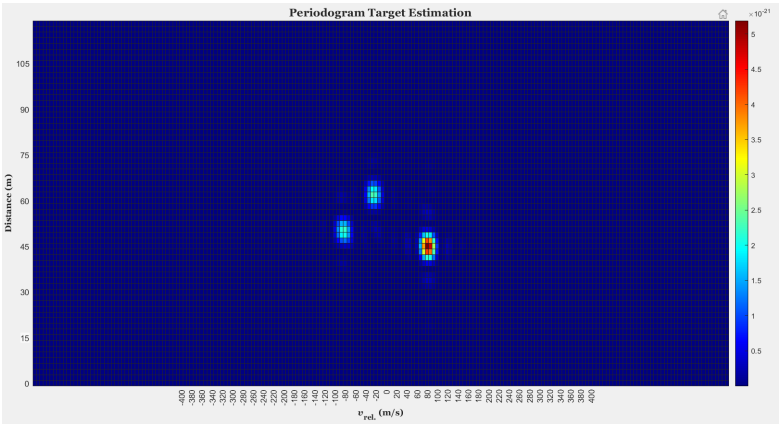


The parameters of the targets were:

```
% Parameters of targets:
targetDistances = [90 45 62];
targetRelVelocities = [43 82 -24];
targetRCSs = [10 15 24];
```

Another test, with the first target a little bit closer, shows:

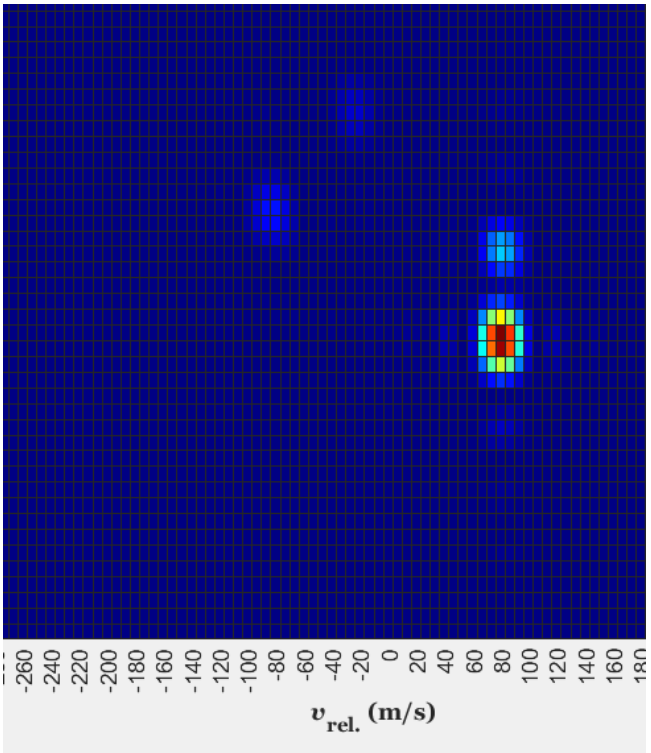




The parameters of the targets were:

```
% Parameters of targets:
targetDistances = [50 45 62];
targetRelVelocities = [-83 82 -24];
targetRCSs = [10 15 24];
```

In the example below, there are **four targets**:




```
% Parameters of targets:
targetDistances = [50 45 62 35];
targetRelVelocities = [-83 82 -24 80];
targetRCSs = [20 15 24 30];
```

28/01/2022:

- I have been doing some light reading up on Digital-Analog conversion, and Digital-Analog converters… I want to have a slightly better or more intuitive sense of what this block in an OFDM system is doing…
- I have also got a better feel or understanding of *why* the following equation might be true:

  $T = 1/\Delta f$ (This equation is often mentioned in papers or literature on OFDM - Δf is the **subcarrier spacing** of the OFDM system)

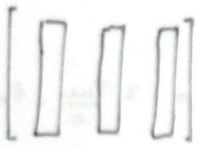
If I have a sine wave, of, say, 100 Hz, its period is simply the inverse ($T = 1/f$), so: 1/100 or 0.01 seconds. Δf is *also* just a frequency… but it is more a *band of frequency*… but the relationship between frequency and period still applies. For example, the sine wave of 100 Hz could be thought of as occupying the band 0 - 100 Hz, so the period would be: $T = 1/(100 - 0)$ = 0.01 seconds.

- I've been thinking about a way to model the following equation:

$$r(t) = \sum_{h=0}^{H-1} b_h s(t - \tau_h) e^{j2\pi f_{D,h} t} e^{j\tilde{\varphi}_h} + \tilde{z}(t).$$

Below are some sheets that I've done out:

(28/01/22) Project



frame of 8 symbols → ifft → add cp

↓

concatenate

↓

$s(n)$

discrete-time signal

↓

DAC

for one antenna there is only one target:

$$r(t) = b_0\, s(t-\tau_0)\, e^{j2\pi f_0 t} + \underbrace{\tilde{z}(t)}_{\text{awgn}}$$

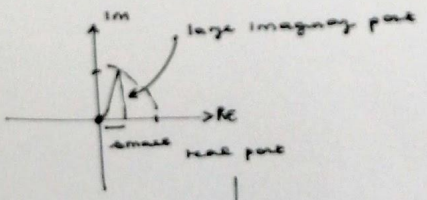
b_0: attenuation factor; τ_0: time delay

$r(t)$ has a real and complex part: $e^{j2\pi f_0 t} = \cos(2\pi f_0 t) + j\sin(2\pi f_0 t)$

→ we can plot: $s(t)$ vs $\mathrm{Re}\{r(t)\}$

$s(t)$ vs $\mathrm{Im}\{r(t)\}$

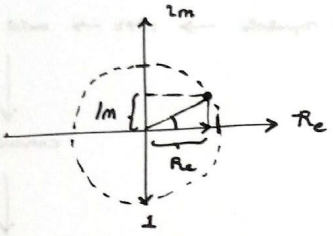
// and change f_0 to see the effect...



The real part of $r(t)$ is what has Physical significance...

↓

This means the correlation between $s(t)$ & $r(t)$ will be low...

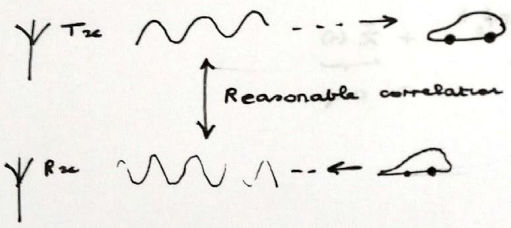


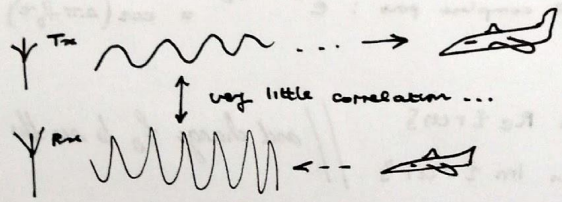
$$f_D = 2 \frac{v_{rel.}}{c} \cdot f_c$$

$v_{rel.} \downarrow \Rightarrow f_D \downarrow$

$\Downarrow$

r(t) is similar to s(t)





- I've created a program so that I can begin trying to create a graph of *r(t)*, and understand *how* it changes when f_D changes:

ofdmFrame2

19x16 double

	1	2	3	4	5	6	7	8	9
1	-0.5111	-0.1826	0.2413	0.0087	0.3705	0.0562	0.8369	0.1567	-0.0506
2	0.2286	0.5089	0.8624	0.3018	-0.1982	1.3018	-1.8928	-0.3321	-0.0214
3	-0.6871	-0.1149	0.2476	-1.1509	0.3388	-0.6535	-0.9199	-0.5103	0.6456
4	-0.1250	0.6250	-1.1250	-1.1250	0.6250	-0.1250	0.1250	1.3750	-0.6250
5	-0.5074	0.3354	-0.5473	0.7973	0.6680	-0.5948	0.1837	0.4019	-2.0060
6	-1.1857	-0.1982	0.0947	-0.0518	0.9482	0.1553	0.2286	-0.2714	-0.2286
7	-0.5249	-0.8497	-0.3242	0.3448	-0.9947	-0.6557	0.4405	-0.7016	1.0283
8	0.3750	1.3750	0.6250	1.1250	0.3750	-0.8750	-0.3750	0.3750	-0.6250
9	0.9075	0.9326	-0.4913	0.7413	-0.4741	0.9009	-0.0869	-0.1139	0.7148
10	-0.4786	-1.2589	-1.6124	-0.0518	-0.5518	0.9482	1.6428	1.0821	-0.7286
11	-0.4165	-1.3422	0.7095	-0.3062	-0.2352	-0.3036	-0.5372	-0.2397	0.1044
12	-1.1250	-0.8750	-0.1250	-0.1250	-1.3750	-0.1250	1.1250	-1.1250	1.3750
13	1.6110	0.4146	0.2973	-0.0473	-0.0645	0.1377	0.5663	1.0552	-0.1582
14	0.9357	-0.5518	1.1553	0.3018	1.3018	-0.9053	-0.4786	-0.9786	0.4786
15	0.1284	0.8068	-0.1329	-0.3877	0.3911	1.1128	-0.4834	-0.0484	-0.2783
16	1.3750	0.3750	0.1250	-0.3750	-1.1250	-0.3750	-0.3750	-0.1250	0.3750
17	-0.5111	-0.1826	0.2413	0.0087	0.3705	0.0562	0.8369	0.1567	-0.0506
18	0.2286	0.5089	0.8624	0.3018	-0.1982	1.3018	-1.8928	-0.3321	-0.0214
19	-0.6871	-0.1149	0.2476	-1.1509	0.3388	-0.6535	-0.9199	-0.5103	0.6456
20									

ofdmFrame

16x16 double

	1	2	3	4
1	-0.1250	0.6250	-1.1250	-1.1250
2	-0.5074	0.3354	-0.5473	0.7973
3	-1.1857	-0.1982	0.0947	-0.0518
4	-0.5249	-0.8497	-0.3242	0.3448
5	0.3750	1.3750	0.6250	1.1250
6	0.9075	0.9326	-0.4913	0.7413
7	-0.4786	-1.2589	-1.6124	-0.0518
8	-0.4165	-1.3422	0.7095	-0.3062
9	-1.1250	-0.8750	-0.1250	-0.1250
10	1.6110	0.4146	0.2973	-0.0473
11	0.9357	-0.5518	1.1553	0.3018
12	0.1284	0.8068	-0.1329	-0.3877
13	1.3750	0.3750	0.1250	-0.3750
14	-0.5111	-0.1826	0.2413	0.0087
15	0.2286	0.5089	0.8624	0.3018
16	-0.6871	-0.1149	0.2476	-1.1509
17				
18				
19				
20				

  The above is just showing that the last 3 elements (an example cp length) are being prefixed to each column… I think in an OFDM system these columns would then be concatenated before being passed to a digital-analog converter for transmission…

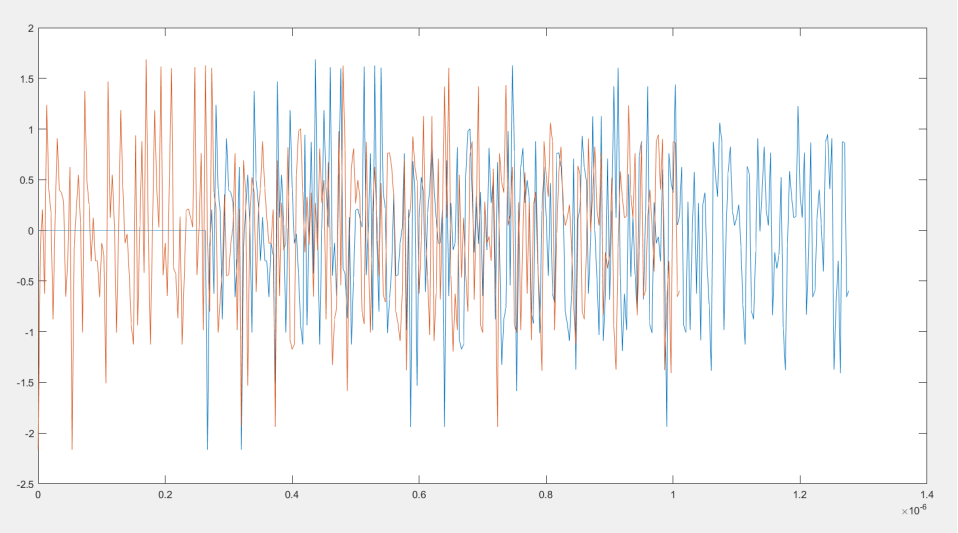
- It seems that the (:) operator in MATLAB is just what I need in order to quickly concatenate all the columns of the OFDM frame to be transmitted…

```
a =

     1     2     5
     2     5     5
     3     2     6

>> a(:)

ans =

     1
     2
     3
     2
     5
     2
     5
     5
     6
```

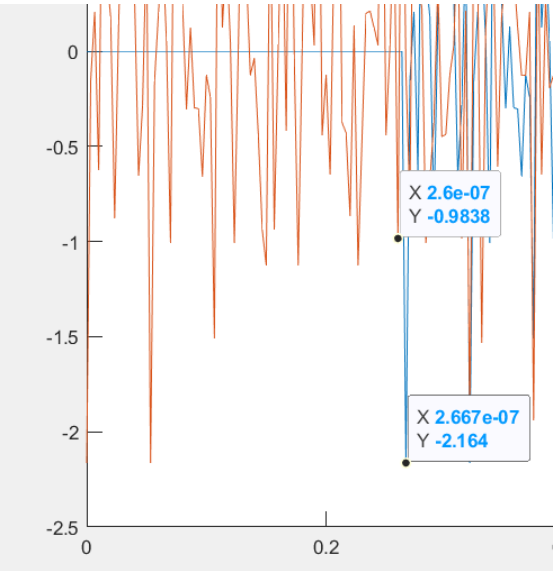
- It seems, at least in some of the literature, that computing the IFFT of complex symbols is considered OFDM modulation… Demodulation is then performed by the FFT on the receiver side…
- I've created the signal s(t) to transmit, by concatenating all the columns of the OFDM transmit frame… as shown above. In reality, I think this discrete-time signal would be sampled by a DAC, which would then result in an EM wave to be sent out via some antenna…

29/01/2022:

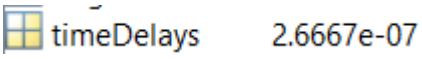
- I've created a basic graph:



The orange signal is s(t), the signal we send out. The blue signal is r(t), the signal we receive. The flat or horizontal blue line is the time we have to wait before we start sensing the reflected signal at the receiver…



The value, from reading off the horizontal axis (representing *time*), is around 0.2667 μS… this **agrees** with the expected time delay from the script:



KEY POINT: the above graph was got by modelling *r* as:

```
for j = padLength+1:length(r) % We start at index "padLength+1" for the reason stated above...
    r(j) = sTx(j-padLength)*cos(2*pi*dopplerShifts*t(j));
end
```

It **did NOT** include the factor *b*, which represents the amount by which s(t) is attenuated when it impinges on the target…

- It seems that *cross correlation* is something that I should look into:

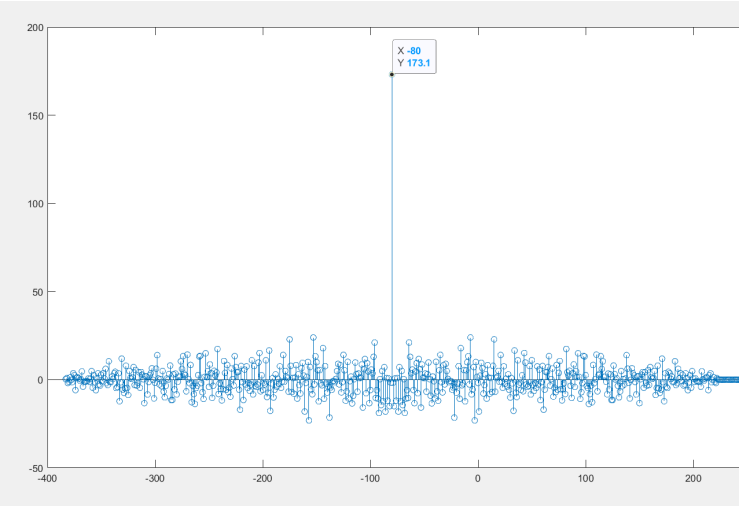
Description

`r = xcorr(x,y)` returns the cross-correlation of two discrete-time sequences. Cross-correlation measures the similarity between a vector `x` and shifted (lagged) copies of a vector `y` as a function of the lag. If `x` and `y` have different lengths, the function appends zeros to the end of the shorter vector so it has the same length as the other.

Interestingly, *autocorrelation* (“auto” representing “self” - think of an *autobiography* being a book about oneself) is the correlation of a signal with a *delayed* copy ***of itself***…

- I just (very hastily) used the xcorr() function in MATLAB, in order to try and work out the cross correlation between sTx and r:

```
[c, lags] = xcorr(sTx, r);
stem(lags, c)
```



Interestingly, at *x* = -80 (the *lag* value), the correlation is maximum. 80 is significant in this case: it is the number of samples in the delay part of the signal *r*…

01/02/2022:

- Over the last few days I have been reading some more literature and papers, and taking notes.
- I have also gained a slightly better understanding of how the range resolution of a radar is determined by:

  ½(cT), where T is the symbol or pulse duration…

  Looking at the above equation, if one wishes to **increase** the resolution, then the **pulse width** in time (**T**) must be reduced…
- I have also been reading up on the cross correlation, as well as the convolution theorem… and how it can be used to compute the cross correlation…
- I’ve been looking more into the cross-correlation function…

```
n = 0:15; % Some vector
s1 = (0.55).^n; % Signal s1...
s2 = circshift(s1, 3); % Signal s2...

[corr, lag] = xcorr(s1, s2)
```

s2

1x16 double

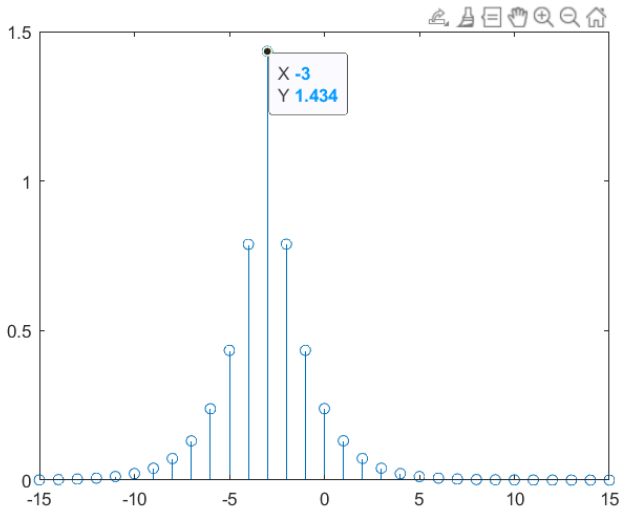
	1	2	3	4	5	6	7
1	4.2142e-04	2.3178e-04	1.2748e-04	1	0.5500	0.3025	0.1
2							

s1

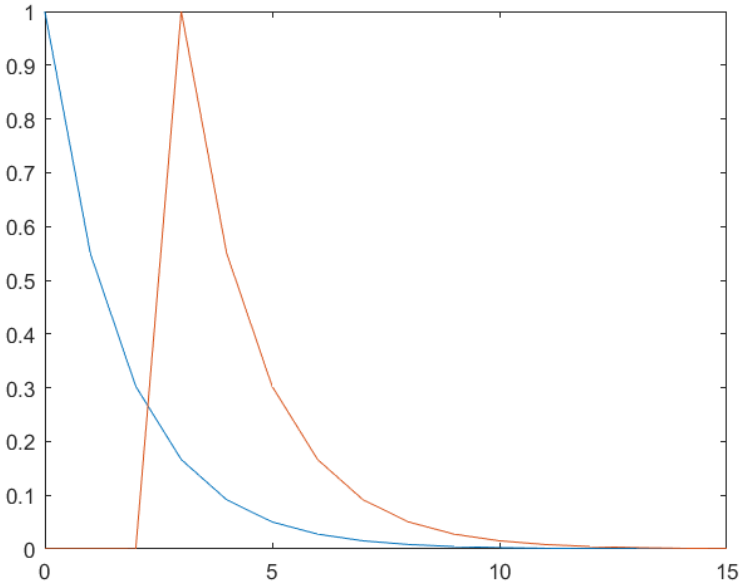
1x16 double

	13	14	15	16
1	7.6622e-04	4.2142e-04	2.3178e-04	1.2748e-04
2				

From the above, it can be seen that the *circshift* function is a very useful one. Because s2 is a circularly shifted version of s1, where the shift constant is 3, the first **3** elements of s2 are the same as the last **3** elements of s1…



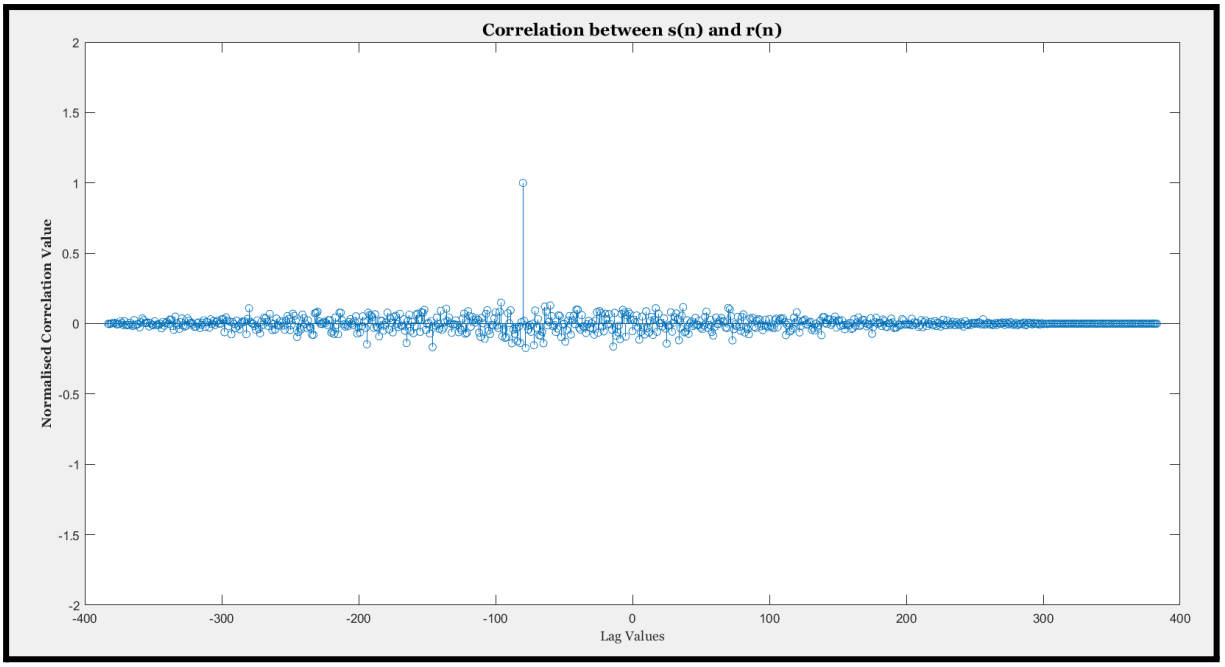
The peak at -3 indicates that the **maximum** correlation occurs if one of the signals is delayed by 3…

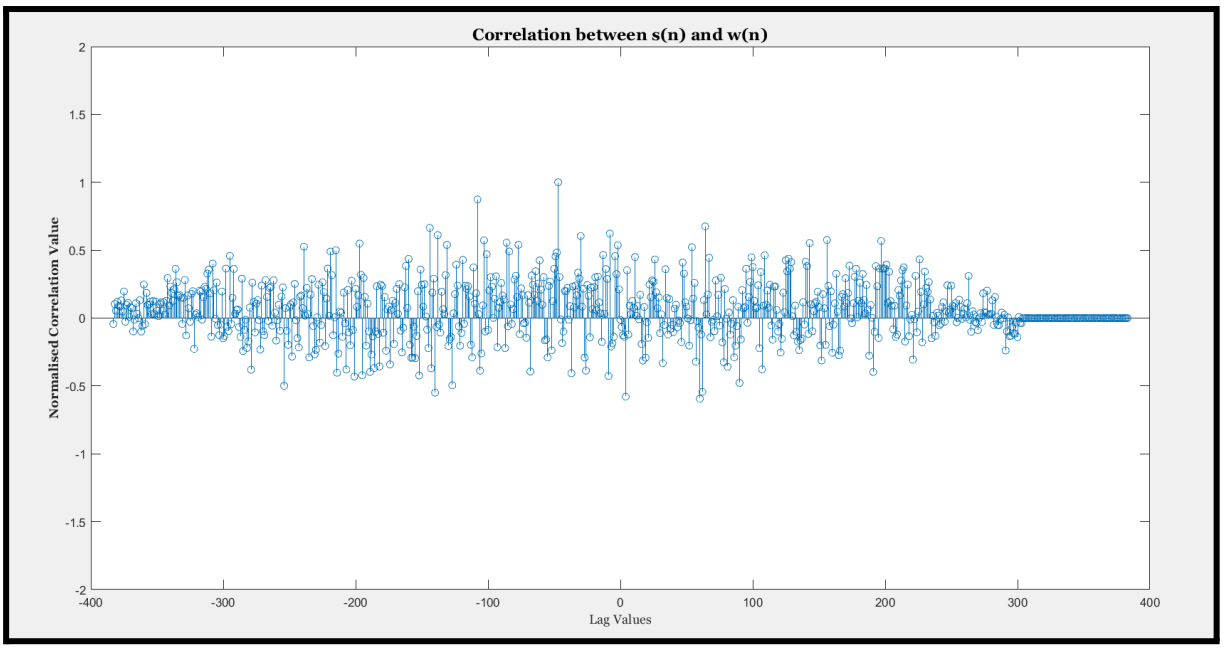


It can be seen that if the **blue** curve were delayed or shifted by 3 to the right, it would match up with the orange one…

04/02/2022:

- Yesterday I was just testing my intuition, and getting a slightly better idea of the *correlation*. Below are two plots: one shows the correlation between the transmitted signal s(n), and the received signal r(n). The other plot shows the correlation between s(n) and w(n), where w(n) represents White Gaussian noise.





- From the first plot, the delay (in *discrete time samples*) can be estimated. From this the *time delay* can be estimated as: $n*f_s$, where n is the number of discrete time samples in the delay (80 in the case above)

05/02/2022:

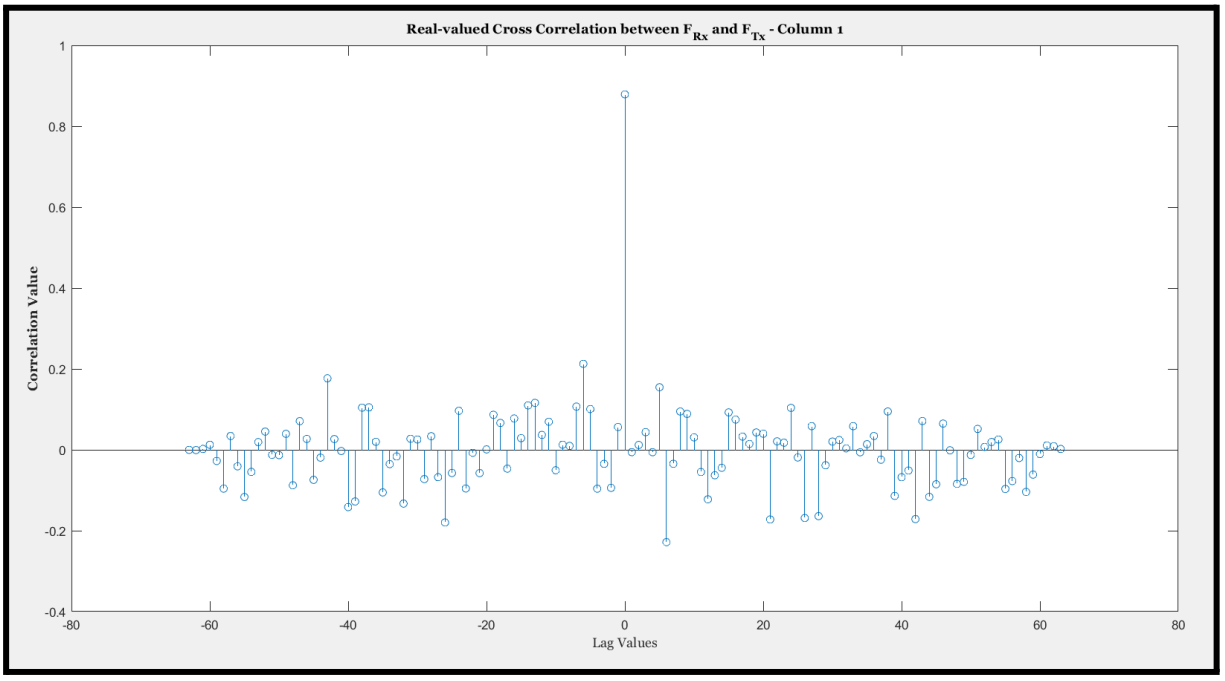
- I'm working on taking the two dimensional correlation of the receive frame, to see if I can use it to discern both the time delay and frequency shift…
- While reading through the main paper (M. Braun) again, I can see that values for the *guard interval* are actually discussed:

Cyclic Prefix In a final step, a *guard interval* is inserted before every OFDM symbol. It is of duration T_G, which is chosen as an integer fraction of the symbol duration. Typical values are $T_G/T = 1/4$ or $T_G/T = 1/8$; choosing a valid guard interval duration is discussed in Section 3.6. Of course, this increases the total OFDM symbol duration to $T_O = T + T_G$.

- I've been refreshing the concept or idea of *variance* in my mind, and doing some example problems and derivations, to feel more comfortable with it… in short, ***variance*** is an indicator of **how much deviation** there is in a probability distribution. **Deviation** is the tendency for an outcome of a **random variable** to **differ** from its **expected value** (**mean**).
- I have also been reading up on the **Cramér-Rao Lower Bound**. This quantity in **statistics and estimation theory** gives a **lower bound** value for the **variance** of an **unbiased estimator**. An estimator which achieves this lower bound value is fully *efficient*… *efficiency* (in statistics and estimation theory) is a way of measuring the **quality** of an estimator…
- I have also continued to list the advantages or drawbacks of various techniques for estimating the target's position and velocity. So far I'm still focusing on periodogram-based estimation techniques…

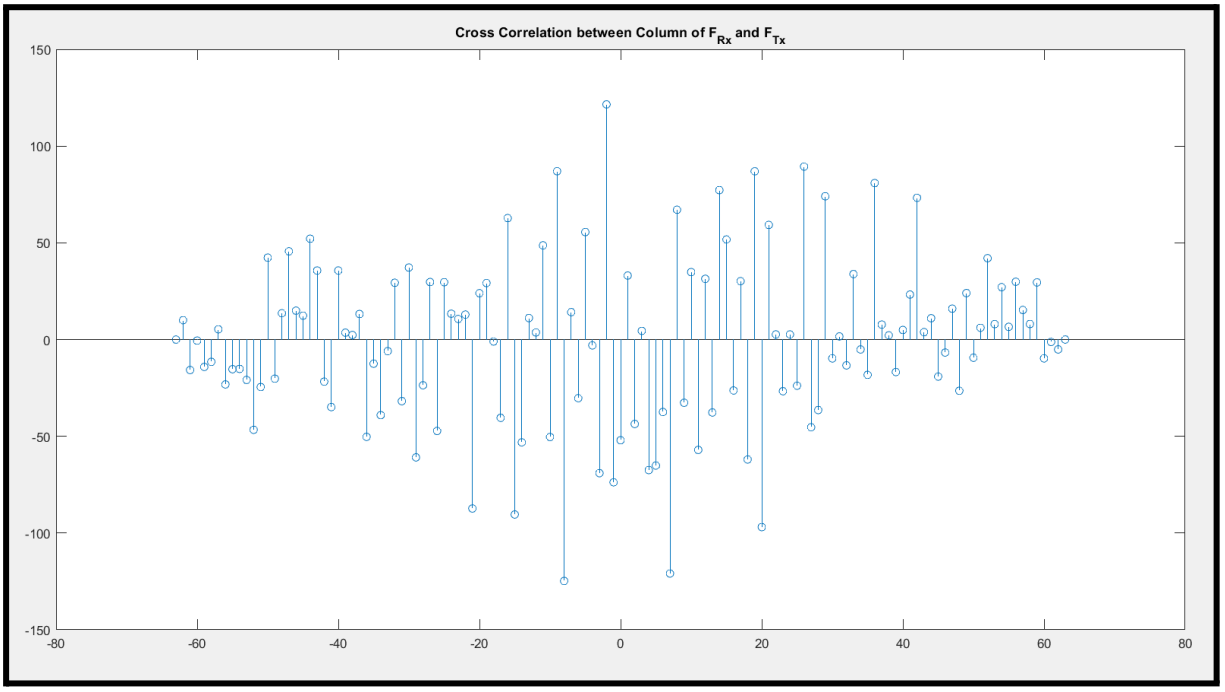
07/02/2022:

- Currently I'm computing the correlation between each column of the transmitted and received frame. The question is, Can I work out the Doppler shift that the moving target imposed on the transmitted frame? The other question is, Should I be computing the correlation between columns that contain real values, or complex ones?
- When I use the real-valued signals (or the real components of s(n) and r(n)) in the **xcorr** function, I see that the maximum correlation between the columns occurs at a lag of 0…



As can be seen, this graph seems to suggest that column 1 of F_{Rx} and F_{Tx} are most correlated when **neither** is lagging the other… when the lag value is **zero**. Because of

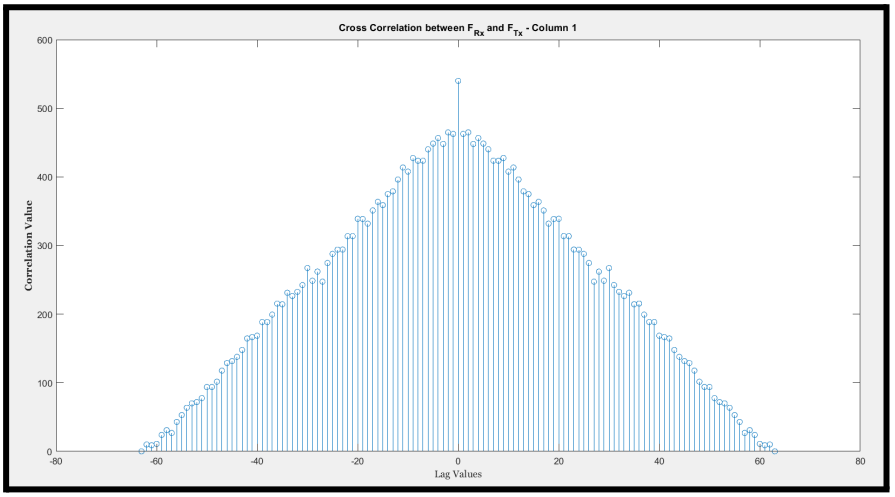
this I cannot get any information about the frequency spread caused by the Doppler effect. As a result, I cannot compute the velocity of the target. When I compute the FFT of each column of F_{Rx} and F_{Tx} and ***then*** use the correlation, the result is different:



Since the values in each matrix are now **complex**, MATLAB gives me the following warning message:

```
Warning: Using only the real component of complex data.
> In getRealData (line 52)
  In stem (line 40)
  In ofdm_radar_2D_cross_correlation_method (line 72)
fx >>
```

When I proceed to take the absolute value of each complex element in the columns of F_{Rx} and F_{Tx} **before** computing the **correlation**, I see:



- “Control” + “F” is **really brilliant**. When I search for “cross-” in M. In Braun's main paper, I find he mentions ideas of using the cross-correlation. In section 3.5 for example:

3.5 Non-spectral estimation based algorithms

All estimation techniques presented make use of the sinusoidal form of $\mathbf{F}$, i.e. the fact that target detection is turned into a sinusoidal detection problem.

Other possible methods of performing radar estimation have also been suggested. One way to obtain a radar image is to cross-correlate the received signal with the transmitted signal both in time and frequency, such as discussed in [37]. The resulting two-dimensional cross-correlation function is then further subjected to a peak detector in the same way as the periodogram to determine the number of peaks as well as their corresponding range and Doppler.

A detailed comparison of this method with the periodogram method is given in [18, Chap. 3]. The two biggest disadvantages of the cross-correlation are the increased computational complexity and the lower peak-to-spur ratio. The former is due to the fact that the efficient FFT-based algorithm for the periodogram presented in Section 3.3 cannot be as effectively applied to the two-dimensional cross-correlation. The in-

- I’ve downloaded the relevant papers mentioned in this section.
- I’m currently reading up on ***spurs*** to get a better idea or understanding of them.
- I was doing some more quick study/revision on the variance and expected values of an **average** of i.i.d random variables. The author mentions this when computing ***an average Cramér-Rao Lower Bound***:

  Var[d] = (1/M)var[$\widehat{d}$]

  // The variance of the average is simply the variance of each divided // by the total number of random variables, M

08/02/2022:

- I’m currently doing some paper analysis on *Comparison of Correlation-based OFDM Radar Receivers*
- The **convolution theorem** is very important:

  Suppose $g = f * h$

  F{g} = F{f}F{h}

  // i.e. convolution in one domain is multiplication in another…

  Therefore: **$g = F^{-1}(F\{f\}F\{h\})$**

- The FFT and IFFT can **also** be used to compute the **correlation**:
  Correlation between ***f*** and ***h*** can be computed as:
  $C(f, h) = F^{-1}(F^{-1}\{f\}F^{-1}\{h\})$

- I'm becoming more competent at LaTex:

Random Equations

$\mathscr{L}\{f(t)\} = F(s)$

```
\section*{Random Equations}
$\mathscr{L}\{f(t)\}=F(s)$
\end{document}
```

- I've been learning about the **ambiguity function** χ

  (1) It's a two variable function, where the arguments are the propagation delay and Doppler frequency: $\chi(\tau, f)$
  (2) It's complex-valued
  (3) The correlation between $s_{a,b}$ and $S_{c,d}$ = **$\chi(a-c, b-d)$**

- If s(t) is sent out at the Tx, then the received signal in perfect conditions with no noise or distortion would be:

$$r(t) = s(t - \tau)e^{i2\pi ft}$$ // Double check this negative signed needed in the exp term…

It's delayed, and shifted in frequency…

- I'm going to print out the following papers and read them in person, taking notes on the key findings of each:

Ambiguity Function Analysis for OFDM Radar Signals

Comparison of Correlation-Based OFDM Radar Receivers

09/02/2022:

- s(t), the transmit signal, is complex
- The **ambiguity function** is an **effective** tool for analysing the **range** and **velocity** resolution of the radar waveform
- An expression for the **ambiguity function** is:

$$\chi(\tau, f_d) = \int_{-\infty}^{+\infty} s(t)s^*(t-\tau)e^{j2\pi f_d t}dt$$

- The paper gives a way of redefining this as:

$$\chi(\tau, f_d) = \chi_{auto}(\tau, f_d) + \chi_{cross}(\tau, f_d)$$

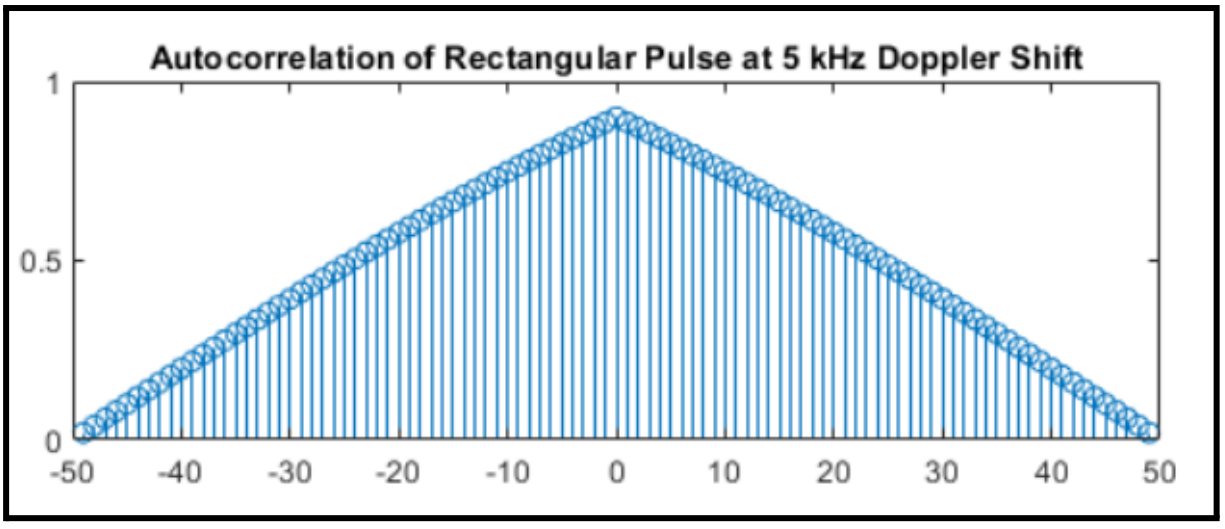
- It seems taking the **autocorrelation** of corresponding **subcarriers** might offer some insight; the **cross-correlation** component is between **different subcarriers**, and shows as **interference** between the **adjacent channels**.
- MATLAB has an implementation of the **ambiguity function**:

ambgfun

Ambiguity and crossambiguity function

```
afmag = ambgfun(x,Fs,PRF)
afmag = ambgfun(x,y,Fs,PRF)
[afmag,delay,doppler] = ambgfun( __ )
[afmag,delay,doppler] = ambgfun( __ ,'Cut','2D')
[afmag,delay] = ambgfun( __ ,'Cut','Doppler')
[afmag,delay] = ambgfun( __ ,'Cut','Doppler','CutValue',V)
[afmag,doppler] = ambgfun( __ ,'Cut','Delay')
[afmag,doppler] = ambgfun( __ ,'Cut','Delay','CutValue',V)
ambgfun( __ )
```

- I must remember that the **ambiguity function** is **built** out of the **cross-correlation** and **auto-correlation** functions, or computations…
- In the MATLAB documentation, I see a similar plot to the one I got a few days ago when I computed the **cross-correlation** (maybe the effect of what I did was to work out the **autocorrelation** of F_{Tx}… since F_{Rx} *is* F_{Tx}, just delayed in time and shifted in frequency) of column 1 of F_{Tx} and F_{Rx}…

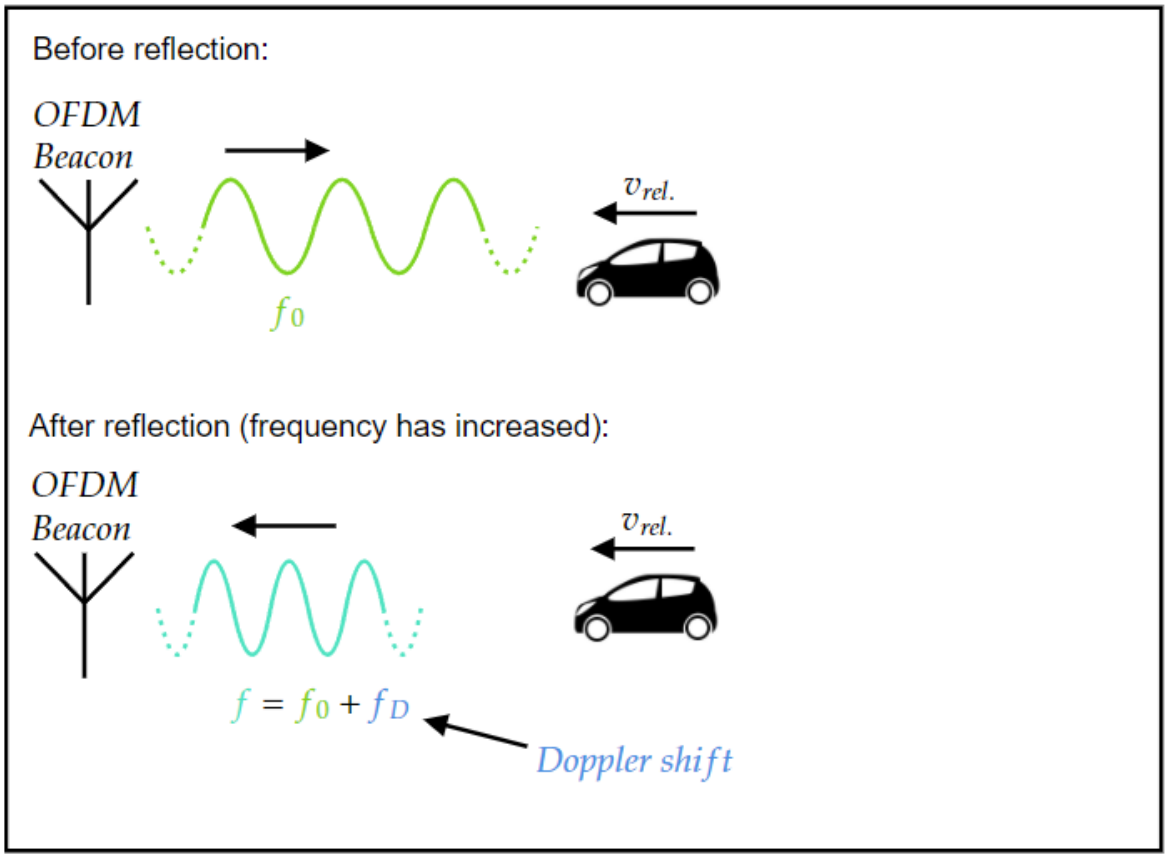


- It seems that **it's reasonable** to talk about calculating the **correlation** of **complex sequences**…

- The inputs ***x*** and ***y*** to the ambgfun() in MATLAB should be **complex-valued** according to the documentation…

12/02/2022:

- I did some reading up on the **inner product**, and how it is a **generalisation** of the **dot product**. There are **4** conditions for the **inner product**:
  - (1) <u + v, w> = <u, w> + <v, w>
  - (2) <αu, w> = α<u, w>
  - (3) <v, w> = <w, v>
  - (4) <v, v> ≧ 0
- The **fourth (4th) condition** is known as the ***positive-definite*** condition.
- Some books or authors consider the first **three** conditions as the main ones, and the **4th** condition is a *weaker* condition.
- In many books and resources, the **inner product** is considered a function. **Inner product** functions which **do not** meet **condition four (4)** above, are sometimes called **indefinite inner products**.
- **Indefinite inner products** can leads to “norms” which yield an **imaginary magnitude** for certain **vectors** (and these vectors are sometimes called **spacelike**)
- “Norms” are **functions** from a **real** or **complex vector space** to the **nonnegative real numbers**; the Euclidean distance of a vector from the **origin** is a norm known as the **Euclidean norm** or **2-Norm**
- I have done up some diagrams which make me more comfortable or confident in the expressions for the reflected signal given in many of the papers:



$$s(t) = cos(2\pi f_0 t) = \frac{e^{j2\pi f_0 t} + e^{-j2\pi f_0 t}}{2} \quad \leftarrow Euler's\ identity$$

$$r(t) = s(t)e^{j2\pi f_D t} = \left(\frac{e^{j2\pi f_0 t} + e^{-j2\pi f_0 t}}{2}\right)e^{j2\pi f_D t} = \frac{e^{j2\pi f_0 t}e^{j2\pi f_D t} + e^{-j2\pi f_0 t}e^{j2\pi f_D t}}{2}$$

The moving target introduces a ***frequency shift*** to the signal that was transmitted s(t)...

$$r(t) = \frac{e^{j2\pi(f_0+f_D)t} + e^{-j2\pi(f_0+f_D)t}}{2} = cos(2\pi(f_0 + f_D)t)$$

frequency of the reflected wave is *higher*...

- In the paper I'm reading it shows that the correlation between two **neighbouring** subcarriers is **0**… this is calculated using the *correlation* expression which I have been hearing about in the Digital Communications class:

$$\frac{1}{T}\int_0^T e^{j2\pi(f_0+k\Delta f)t} e^{-j2\pi(f_0+l\Delta f)t} dt$$

And I *think* that **because** each **signal is complex**, when we are computing the **correlation** one of them is **conjugated**… There is an excellent post about this on the signal processing side of StackExchange… [Cross-Auto Correlation with Complex Signals]
I just learned how to add **hyperlinks** 😀

- The LaTex cheat sheet that I downloaded a week or so ago is very good, and has tips such as:

Displaystyle

To get full-sized inline mathematical expressions use `\displaystyle`. Use this sparingly. Typing

```
I want this $\displaystyle \sum_{n=1}^{\infty}
\frac{1}{n}$, not this $\sum_{n=1}^{\infty}
\frac{1}{n}$. yields
```

I want this $\displaystyle \sum_{n=1}^{\infty} \frac{1}{n}$, not this $\sum_{n=1}^{\infty} \frac{1}{n}$.

- Watched a good video discussing how cross-correlation and auto-correlation can be used to find the time delay between two signals.

- I also watched another video which talked a little bit more about correlation and its use in GPS and satellite-based technology. “Gold” codes came up as well.

  > Coarse/acquisition code
  >
  > The C/A PRN codes are **Gold codes with a period of 1023 chips transmitted at 1.023 Mchip/s**, causing the code to repeat every 1 millisecond. They are exclusive-ored with a 50 bit/s navigation message and the result phase modulates the carrier as previously described.

  It seems that a “Chip” is a pulse of a direct-sequence spread spectrum code…

13/02/2022:

- I’ve been doing analysis of the papers about the cross-correlation-based approaches to OFDM radar…

15/02/2022:

- A random process is **stationary** if a timeshift **does not** change its **statistical properties**…
- I just did one *fairly long* derivation of the “Matched Filter” concept; it was worth it though (I think).
- The matched filter is the **filter** which **maximises** the **SNR** at the output…
- The matched filter **acts** as a **correlator** whose output at time t_o (the sampling time) is the **autocorrelation** of the **desired signal**, s(t):

  $R_{ss}(t - t_0)$, when $t = t_0$, we get $R_{ss}(t_0 - t_0) = R_{ss}(0)$

$$R_{ss}(0) = \int_{-\infty}^{+\infty} s(t)s(t)dt = \int_{-\infty}^{+\infty} s^2(t)dt = E$$

- The derivation also reintroduced me to the Cauchy-Schwarz Inequality.
- I’m doing more analysis of the paper on Correlation-based receivers for OFDM radar
- I’ve learned that the letter $\boldsymbol{\nu}$ is **typically** used for the **frequency** of **electromagnetic waves**… whereas ***f*** is used for the frequency of ordinary waves which **require a medium**…
- I understand certain aspects of the equations given in the current paper that I am reading (On correlation-based OFDM radar receivers) slightly better now.
- Proximate-Matched Filtering (PMF) reduces the computation complexity of the normal or standard Matched filtering approach. The PMF expression or equation can be **implemented** via the FFT
- There is also a discussion on PMF-CP, which is basically **applying** the PMF approach **after** you remove the **CP** from each **block** or **symbol** (*m*)
- Another tweak or slightly different algorithm is the PMF-CC, where the CC stands for **circular correlation**.

In plain english, (11)–(13), therefore, mean that the PMF-CC basically:

1) estimates the time-frequency shifted elementary symbol in each subband of the received signal, as in a regular linear OFDM communication receiver;
2) multiplies it by the conjugate of the (known) elementary symbol actually transmitted in that subband;
3) computes a two-dimensional FFT, transforming the subband-slow time domain into the range-Doppler domain of interest.

- **Unitary FFT:** An FFT that is normalised by a constant being multiplied into each element of the FFT result…

$$\chi_m^{(\check{r},s_0)}(l,0) = \frac{L}{K}\mathrm{FFT}_K^{-1}\left\{\mathrm{FFT}_K\{\boldsymbol{r}_m\} \odot \mathrm{FFT}_K^*\{\boldsymbol{s}_m\}\right\}[l], \quad l \in \mathcal{I}_K \quad (12)$$

16/02/2022:

- I'm still reading the main paper on **correlation-based** OFDM **radar receivers**…
- The author introduces **pulse-ambiguity** functions too:

$$A^{(g,g)}(l,f) \triangleq \frac{1}{K}\sum_{p=0}^{L-1} g[p]g^*[p-l]e^{j2\pi fp} \quad (15)$$

$$A^{(g,\check{g})}(l,f) \triangleq \frac{1}{K}\sum_{p=0}^{L-1} g[p]\check{g}^*[p-l]e^{j2\pi fp} \quad (16)$$

]

- The **presence** of **multiple** FFTs in the **PMF-CP** and **PMF-CC** receivers drastically improves the computational speed or efficiency
- The **paper** is **very hard** to read for me… the language is quite verbose, and the mathematical notation is heavy or steep (you need to read it very carefully)

TABLE III
Waveform and Channel Parameters for SINR Simulations

Parameter	Variable	Value
Number of subcarriers	K	128
Cyclic prefix length	Δ	16
Number of blocks	M	32
Constellation	$c_{k,m}$	QPSK
Target power	$\mathbb{E}\{\lvert\alpha\rvert^2\}$	0 dBW
Input noise power	σ^2	0 dBW

- When **unit-variance** modulation schemes are used then the **reciprocal** and **matched** filtering approaches are **equivalent**.
- It seems from this paper that **if** the **target** is **stationary** then **attempting** to compensate for the **non** existing **Doppler phase** within each **block** seems to be **detrimental** for the MF approach.

  “Especially, since the target is static here, attempting to compensate the nonexistent rotating Doppler phase within each block even seems to be detrimental for the MF, as hinted by Fig. 1(b)”

- I better understand this equation or expression for the transmitted **OFDM** signal s(t):

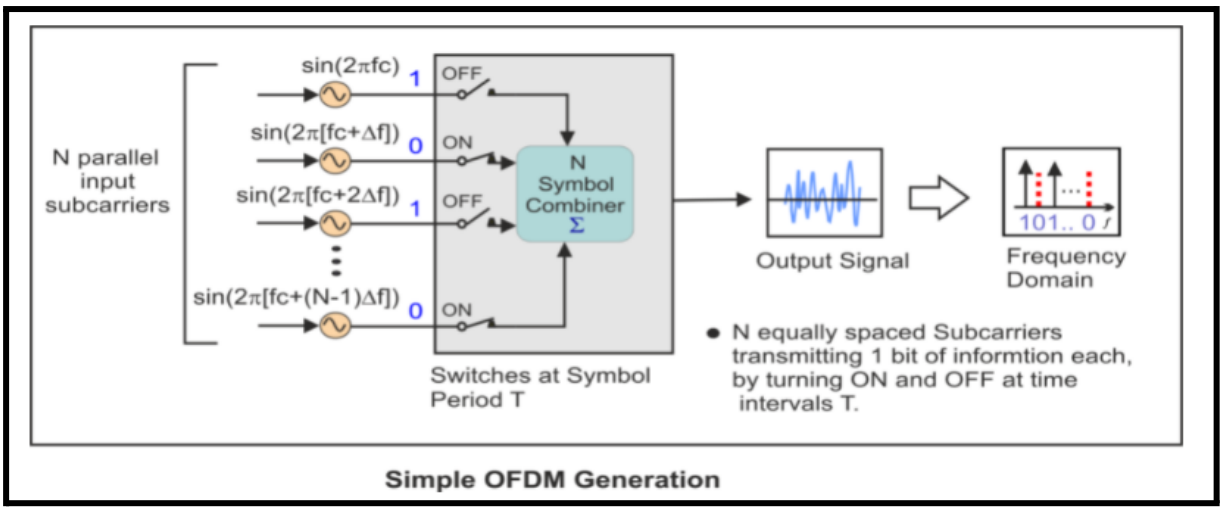
$$s_l(t) = \sum_{l=0}^{M-1}\sum_{k=0}^{N-1} e^{j2\pi\left(f_0 + k\Delta f\right)\left(t - lT'_{sym}\right)} p\left(t - lT_{sym}\right) \qquad (3)$$

p(t - *l*T) are pulses which will be **delayed** by a **certain amount depending** on the value of **index *l***, representing the OFDM symbol that is being transmitted. E.g., if *l* = 0, then:

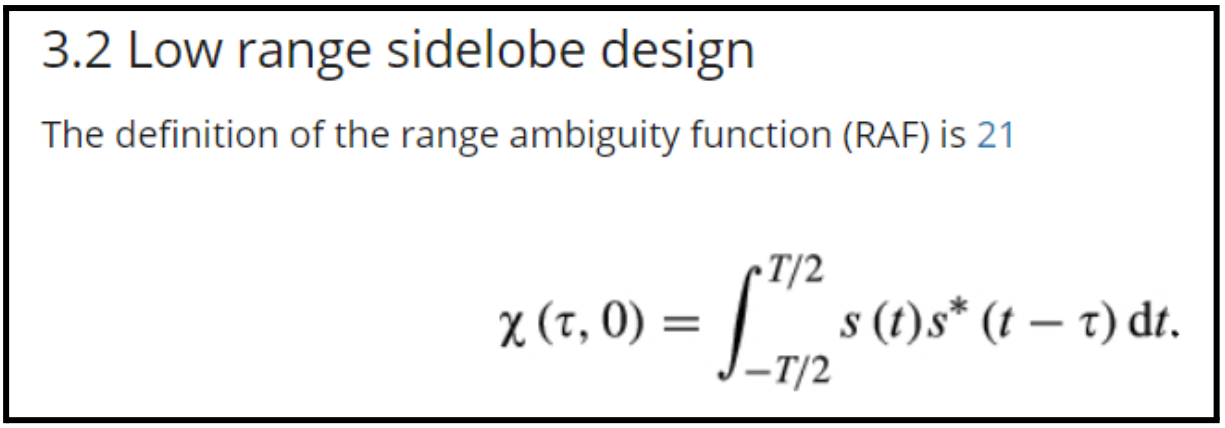
p(t - *l*T) = p(t)

Also, for each **value** of the index ***l***, $s_l(t)$ evaluates to the **sum** of **N** (as ***k*** goes from **0** to **N-1**) **complex sinusoids**.

This **idea** or **interpretation agrees** (I ***think***) with what this diagram below shows where the subcarriers are **summed together**:



- I found a website which gives:



Presumably the exponential term with the Doppler frequency is gone as substituting in 0 for f_D would bring the complex exponential to 0…

- I'm just doing some more reading, slow and steady, on the **ambiguity function** in radar. The **ambiguity function** can be thought of as the **output** of the **matched filter**…
- Ambiguity function is used by radar engineers to study different waveforms and their usefulness for **radar applications**…
- I'm going to study some more properties of the ambiguity function… **for example**, it seems that the **ambiguity** function has **symmetry** about the **origin**…

17/02/2022:

- I think the effect of dividing the **received** symbols by the **transmitted** symbols, might *also* be achieved by **multiplying** the **received** symbols by the **complex conjugate** of the **transmitted symbols**.
- My supervisor has just **cleared up** a lot about some of the equations that are stated early on in the paper on correlation-based OFDM radar receivers:

$$\chi^{(r,s)}(l, \nu) \triangleq \frac{1}{\sqrt{KM}} \sum_{p=0}^{LM-1} r[p] s^*[p-l] e^{-j2\pi \nu p/L}. \quad (4)$$

In the above equation which involves a sum, the values of v and l are **fixed**, and do not change. Therefore, if I want to know the value of this function at a *particular* point (l, v) then I simply substitute them into the equation, and compute the sum (which is actually computed the **cross correlation**.

This is where I was getting stuck: I didn't realise that I *choose* which values to use for v and l. Picking a range of values and then computing the function's values will then result in a **map**: one axis will correspond to the quantity l and the other will represent v…

When different values of l and v are put into the sum expression, the overall result (cross correlation) will be some value. If it is **high**, then it suggests that there is a target at l which caused a Doppler shift of f_D. If it is low, then it suggests that there is not a target at l moving at whatever speed corresponds to f_D.

- I had a basic idea of how I might compute the Doppler shift by inspecting the transmitted and received symbols:

$c_{k,l}$: the transmit symbol

$b_{k,l}$: the receive symbol

$$b_{k,l} = c_{k,l} e^{j2\pi l T_o f_D} e^{-j2\pi k \tau \Delta f}$$

$$c_{k,l} = e^{j\alpha}$$

Now we can multiply the received symbol by the complex conjugate of the transmit symbol:

$$c^*_{k,l} c_{k,l} e^{j2\pi l T_o f_D} e^{-j2\pi k \tau \Delta f}$$

...and:

$$c^*_{k,l} c_{k,l} = e^{j\alpha} e^{-j\alpha} = 1$$

...so:

$$c^*_{k,l} c_{k,l} e^{j2\pi l T_o f_D} e^{-j2\pi k \tau \Delta f} = e^{j2\pi l T_o f_D} e^{-j2\pi k \tau \Delta f} = e^{j(2\pi l T_o f_D - 2\pi k \tau \Delta f)}$$

$$2\pi l T_o f_D - 2\pi k \tau \Delta f = Somenumber$$

And we can solve for f_D assuming when we know τ, which we *know* from the time-domain correlation calculation...

19/02/2022:

- I think today is the first time that I have ever put a **matrix** inside an **exponent expression**:

```
a =

     1     1
     1     1
     1     1

>> y = exp(1i*2*a)

y =

  -0.4161 + 0.9093i  -0.4161 + 0.9093i
  -0.4161 + 0.9093i  -0.4161 + 0.9093i
  -0.4161 + 0.9093i  -0.4161 + 0.9093i

fx >>
```

I'm currently trying to calculate and plot the **ambiguity function**, and being able to do this will make the calculations more compact and efficient (I think)

20/02/2022:

- I'm currently trying to compute χ_m for all values of m (that is, for each OFDM symbol):

```
for block = 1:frameSize
    s = Ftx(:, block); % The signal s
    r = Frx(:, block); % The signal r

    %x2 = exp(-1i*2*pi*300*p/L);
    x = exp(-1i*2*pi*p_nu_matrix/L);

    r_dash = r.*x.';

    % Now to calculate the correlation:
    Xm = zeros(length(nu), 2*L-1);

    for j = 1:length(nu)
        Xm(j, :) = xcorr(r_dash(:, j), s);
    end

    Xm = (1/sqrt(L))*Xm;
    Xm_storage(5*block-4: 5*block, :) = Xm;
end
```

The above for loop computes the expression for each *m* and for all the values of frequency contained inside the *nu* vector:

```
L = N;
p = 0:L-1; % The vector p, as described in the paper referenced at the top of the program
nu = [-2000 -1000 0 1000 2000]; % The Greek letter "Nu", which looks like a 'v', is standing fo

% The very big matrix below will store all the
% Xm matrices:
Xm_storage = zeros(frameSize*length(nu), 2*L-1);

% The matrix below is the result of multiplying the p vector
% by each value in the nu vector, and storing the results in
% the rows of this "p_nu_matrix" matrix. This is simply being
% used to do the computations compactly.
p_nu_matrix = p.*nu';
```

- I'm doing some more basic sanity checks on my results:

```
% Good work: not check 1 or 2 of the values for Xm (e.g. m = 1, m = 2 -
% frequencies of 2000 and 1000)
x1 = exp(-1i*2*pi*1000*p/L);
r1 = Frx(:, 2);
s1 = Ftx(:, 2);
r1_dash = r1.*x1.';
c1 = (1/sqrt(L))*xcorr(r1_dash, s1);
```

c1 | Xm_storage

63x1 complex double

	1
1	1.4678e-05 - 2.0672e-05i
2	0.1657 + 0.0616i
3	-0.0240 + 0.0657i
4	0.1661 + 0.1969i
5	-0.1734 - 0.3146i

8	1.4678e-05 - 2.0672e-05i	-0.0615 + 0.1657i	-0.0631 - 0.1358i	0.2932 + 0.2841i	-0.1975 - 0....	0.3628 - 0....	-0.3204 - 0....	-0.248
9	1.4678e-05 - 2.0672e-05i	0.1657 + 0.0616i	-0.0240 + 0.0657i	0.1661 + 0.1969i	-0.1734 - 0....	0.0787 + 0....	-0.0448 - 0....	0.1983

As can be seen above, column 1 in the first snapshot is the same as row 9 in the second snapshot… I've changed *m* and the value of "1000" in the expression for x_1 in order to verify (to some degree) that my **for loop** is doing what it should…

For each *m* (OFDM symbol, of which there are 64, at least at the runtime of this program) there are **five** rows:

c1 | Xm_storage

320x63 complex double

	1	2	3
1	9.1631e-06 + 5.2621e-06i	-0.0215 - 0.0280i	0.0567 + 0.0378i
2	9.1631e-06 + 5.2621e-06i	-0.0281 + 0.0216i	-0.0371 - 0.1866i
3	9.1631e-06 + 5.2621e-06i	0.0215 + 0.0281i	-0.1116 + 0.1670i
4	9.1631e-06 + 5.2621e-06i	0.0280 - 0.0215i	0.0921 - 0.0183i
5	9.1631e-06 + 5.2621e-06i	-0.0215 - 0.0280i	0.0567 + 0.0378i
6	1.4678e-05 - 2.0672e-05i	0.0615 - 0.1657i	-0.0461 + 0.0214i
7	1.4678e-05 - 2.0672e-05i	-0.1657 - 0.0615i	0.1332 + 0.0486i
8	1.4678e-05 - 2.0672e-05i	-0.0615 + 0.1657i	-0.0631 - 0.1358i
9	1.4678e-05 - 2.0672e-05i	0.1657 + 0.0616i	-0.0240 + 0.0657i
10	1.4678e-05 - 2.0672e-05i	0.0615 - 0.1657i	-0.0461 + 0.0214i
11	-5.6740e-06 + 2.6708e-07i	-0.0751 + 0.1600i	0.0126 - 0.1130i
12	-5.6740e-06 + 2.6708e-07i	0.1601 + 0.0751i	-0.2478 + 0.1979i
13	-5.6740e-06 + 2.6708e-07i	0.0751 - 0.1601i	0.3327 + 0.0372i
14	-5.6740e-06 + 2.6708e-07i	-0.1601 - 0.0751i	-0.0976 - 0.1222i
15	-5.6740e-06 + 2.6708e-07i	-0.0751 + 0.1600i	0.0126 - 0.1130i

where each row corresponds to a **different** value of ν

```
nu = [-2000 -1000 0 1000 2000]; % The Greek letter "Nu", which looks like a 'v', is
```

I can add more elements to this vector “Nu”... to plot χ_m for more frequency values…

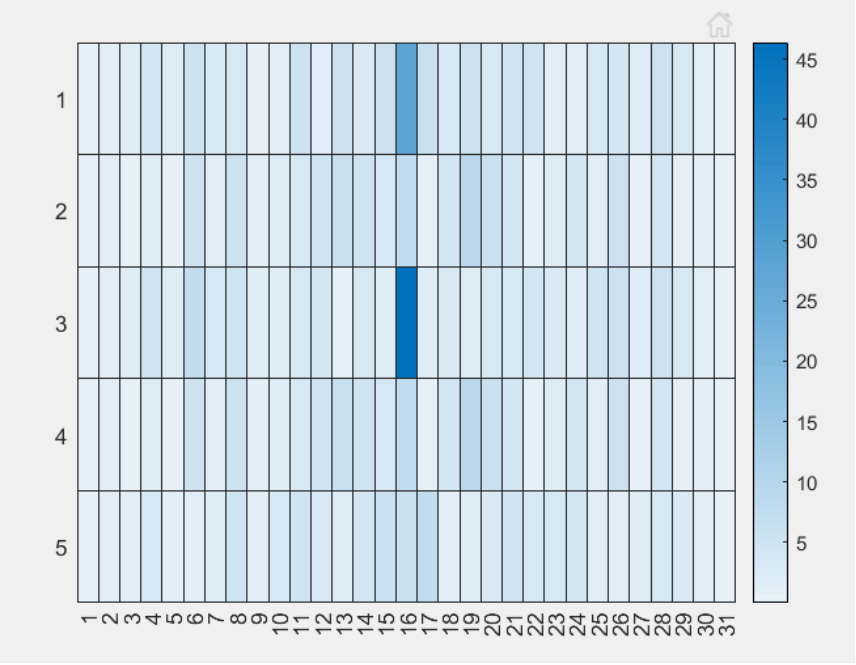
21/02/2022:

- I think I’m nearly there with regards to plotting the ambiguity function:

```
b =

     3
     7
     5

>> a.*b

ans =

     3    12    15
    28    28    42
    10    10     5

>> a = [1 4 5; 4 4 6; 2 2 1]

a =

     1     4     5
     4     4     6
     2     2     1

fx >>
```

I'm just adding the above to have as a quick reference

- I've got a heatmap up anyway:

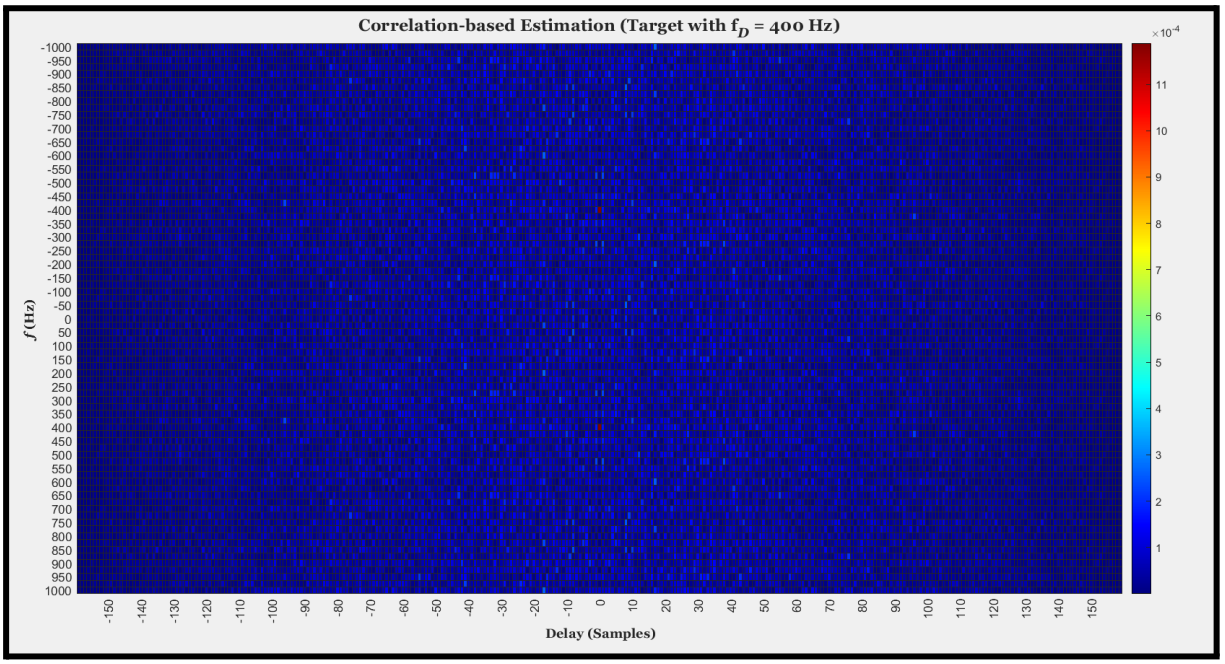


22/02/2022:

- I've created some heatmaps, but they aren't functioning correctly yet… the results are off.

23/02/2022:

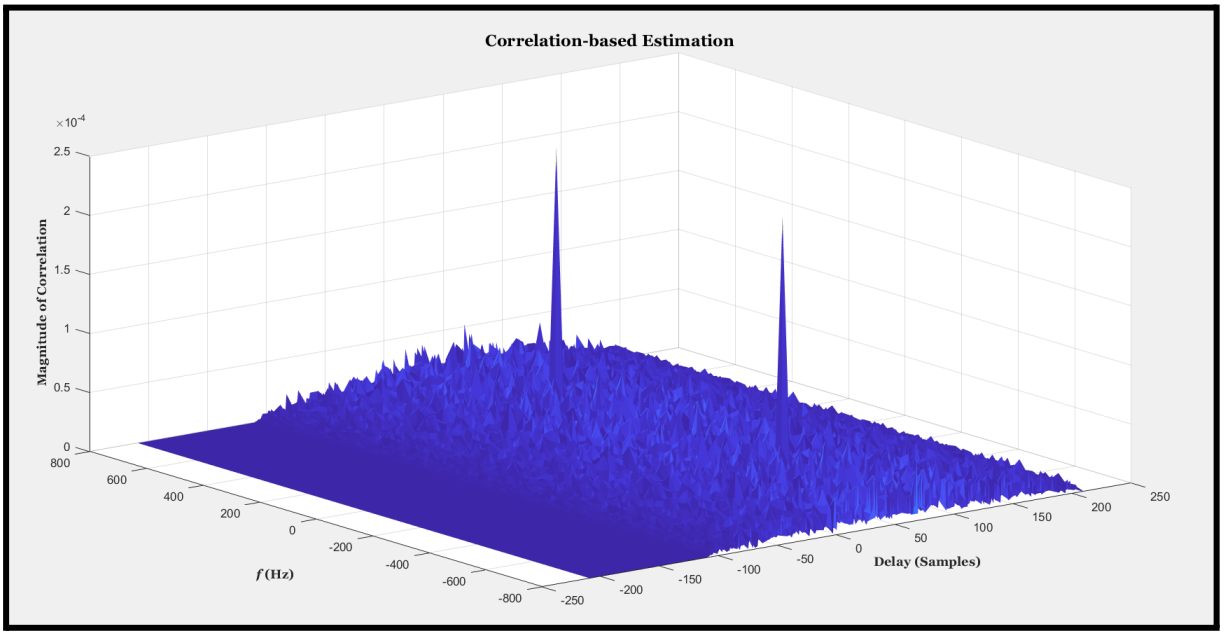
- I have finally been able to get some code running for the correlation-based method!



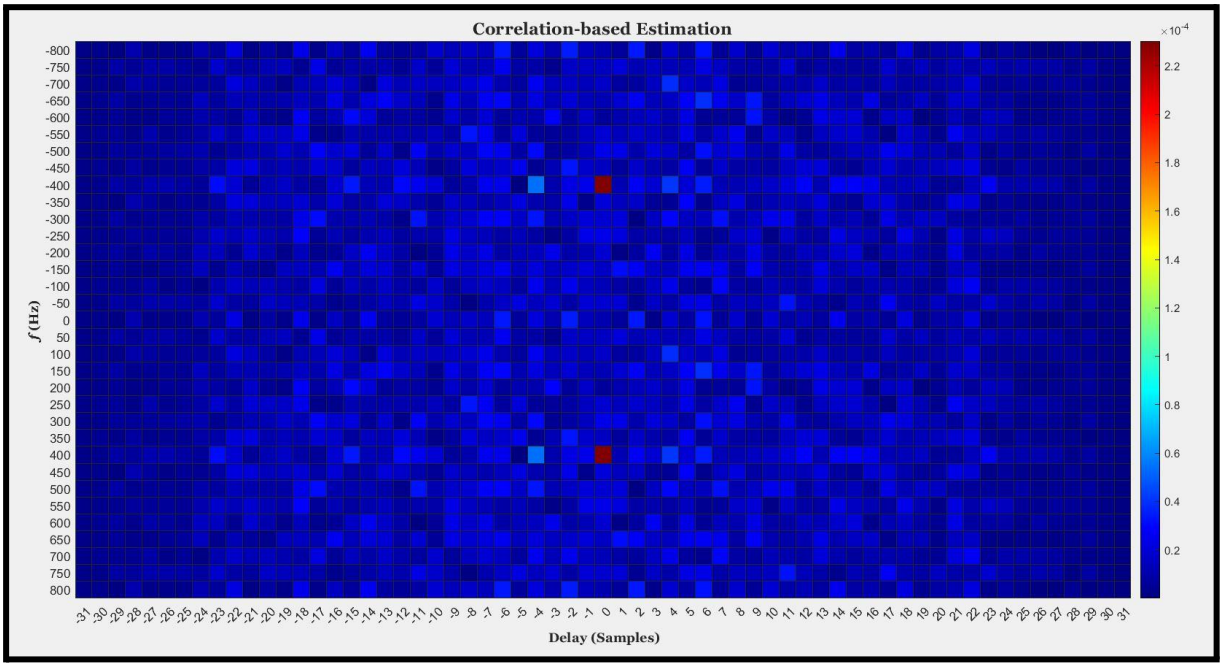
As the diagram above mentions, the target **causes** a **Doppler shift** of 400 Hz. On the heatmap, the cells with the **largest values** (corresponding to when the correlation between ***r*** and ***s*** is maximum) have ***y*** values **of 400 Hz** and **-400 Hz**. I have tried changing the value of f_D, and that graph updates correctly!

26/02/2022:

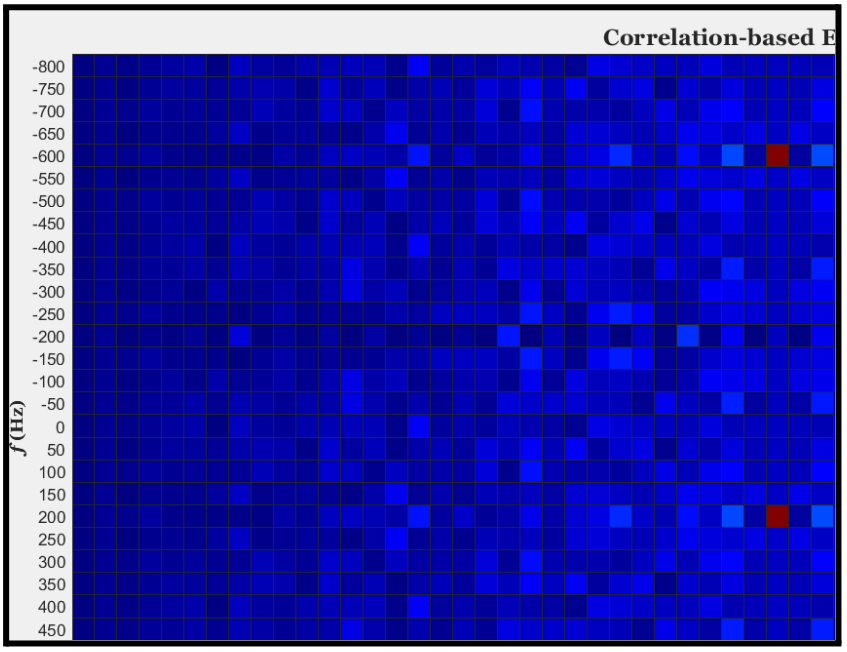
- I forgot to update this with the work I got done last Wednesday and Thursday morning, before the meeting with my supervisor.
- Below is shown a **surface plot** that I got by implementing equation 4 of the paper "*Comparison of Correlation-based OFDM Radar Receivers*"



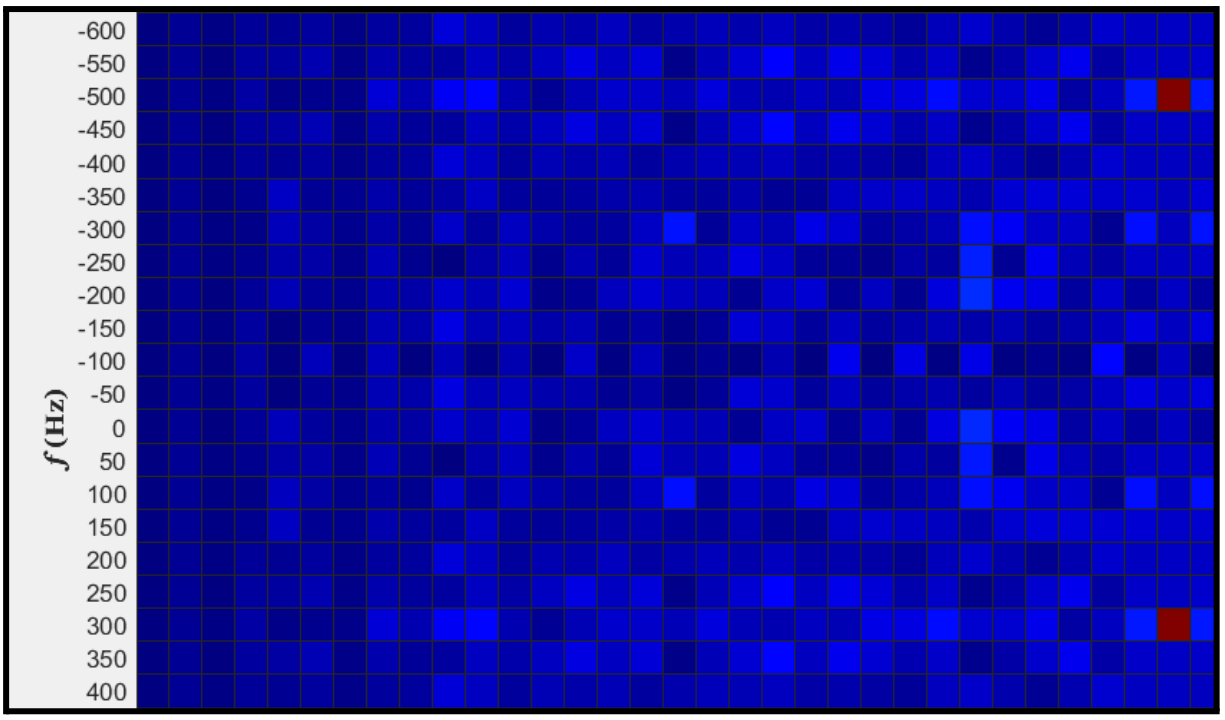
- I am now trying to correct the code I have for the implementation of equation 5 and 6…
- By modelling the **received** signal in the same way as discussed in the paper (which is also what allowed me to generate the above graph or plot for equation 4) I seem to have fixed the issues I was seeing when trying to compute equation 5 and equation 6!:



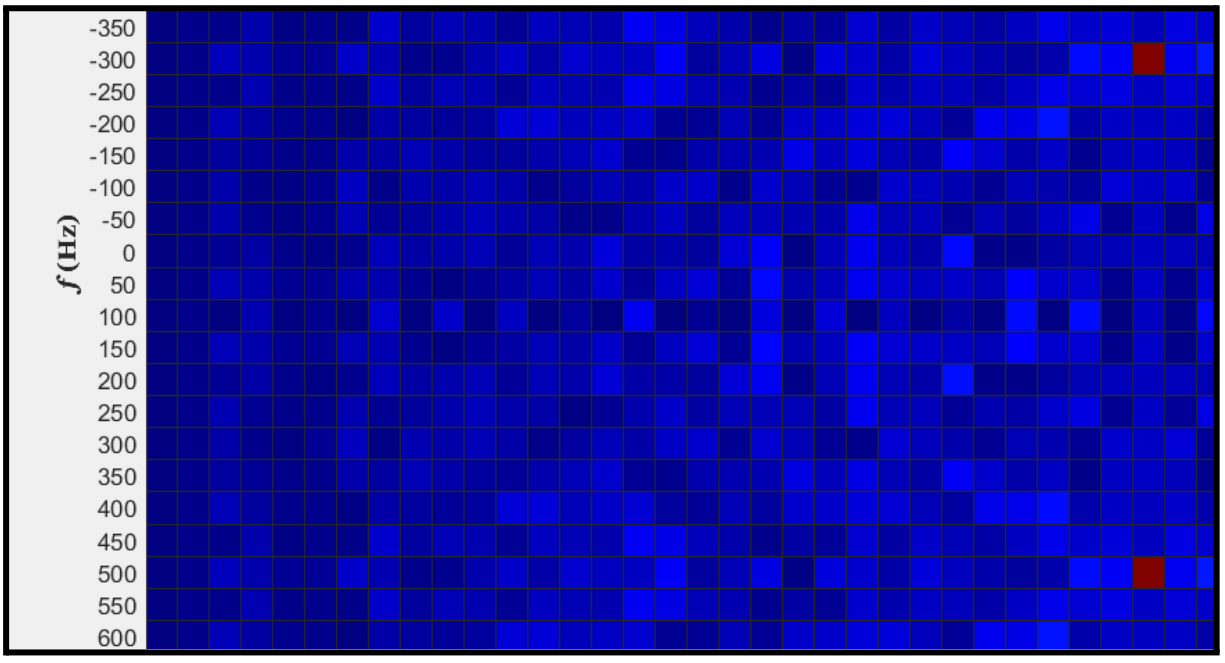
- I have noticed something small: there is always two bright spots, but unlike before, they are not **always** equidistant from the midline (*f* = 0):



In the example above, the Doppler shift value in the program was set to **200 Hz**. And as can be seen, there is a **deep** red square along the **f = 200** line. The other **red square** appears on the opposite side at **f = -600**.



In this example, the Doppler shift was set to 300 Hz, and a **peak** (**red square**) appears along that line. But the other appears at **-500**. What I notice, is that the **sum** of the magnitude of these numbers seems to always be **800**



As a last example, I set the Doppler shift to be **-300 Hz** (indicating that the target is moving away from the OFDM beacon). As can be seen, a peak is seen along the line **f = -300**. The other is along **f = 500**…

- From an article on the **ambiguity function**, it seems that these extra **peaks** are not *totally* unexpected, but are simply **false positives**… They are **ambiguous results**.

The ambiguity occurs specifically when there is a high correlation between $s_{\tau,f}$ and $s_{\tau',f'}$ for $(\tau, f) \neq (\tau', f')$.

And as for why the sum of the **magnitudes** of the frequency values of each peak add together to give **800**, I think it has to do with the upper (or lower) bound of the **frequency Vector**:

```
nu = -800:50:800;
```

Although changing the limits to be -1000 and 1000 did not change anything else…

- I'm reading again about the **proximate matched filtering** approach. And noticed this part in particular:

In other words, the PMF reduces (6) to a single cross-correlation—or range compression—for each sweep m, prior to computing a Fourier transform over the sweeps

$$\widetilde{\chi}^{(r,s)}(l, \nu) \triangleq \frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} \chi_m^{(r,s)}(l, 0) e^{-j2\pi \nu m}. \qquad (7)$$

And shown below is the expression for X_m, when $\nu = 0$ the exponential term will also be 1… so it's a **single cross-correlation**:

$$\chi_m^{(r,s)}(l, \nu) = \frac{1}{\sqrt{K}} \sum_{p=0}^{L-1} r[p + mL] s^*[p + mL - l] e^{-j2\pi \nu p/L}.$$

Importantly though, if we look at **equation** 7, is this just the **Fourier transform** of X_m?

- VHF and UHF stand for **Very High Frequency** and **Ultra High Frequency**
- I'm doing some paper analysis
- An **SFN** is a **Single-Frequency Network**…
- I'm reading this paper, and it references an idea that was discussed in the digital communications class:

for various reasons (protection of bandwidth edges, Doppler estimation, etc.) and, also, a complete Null symbol is inserted periodically for synchronization (all $s_i[n]$ are zero). The baseband signal is upconverted to the carrier frequency

$$\tilde{x}(t) = \mathrm{Re}\left\{e^{j2\pi f_c t} x(t)\right\}. \qquad (4)$$

The baseband signal is **complex**. This is shifted in frequency up to f_c (via multiplication with the complex exponential)… and the **transmitted signal** is real-valued…

- Coherent **Integration Gain** is the **increase** in **SNR** achieved by coherently **integrating** multiple **measurements** of a signal in the presence of **additive** noise…

28/02/2022:

- *Proximate* means “immediate” or “close in nature”; it also means “nearly accurate” or “approximate”...
- I’m trying to implement the proximate filter, but my results don’t seem to make much sense at the moment:
- I was learning a little bit more about ***fast*** and ***slow*** time in Radar. The FFT along the “slow-time” axis transform data into the Doppler domain.

01/03/2022:

- r: the **reference signal** (received signal)
- s: the **surveillance signal** (transmitted signal)
- In practice, ***r*** and ***s*** are **sampled**, and the matched filter is implemented for a **window** of **data**. The size of the window is called the **Coherent Processing Interval** (**CPI**)
- I’m watching some very good lectures on **radar** and signal processing (MIT course)

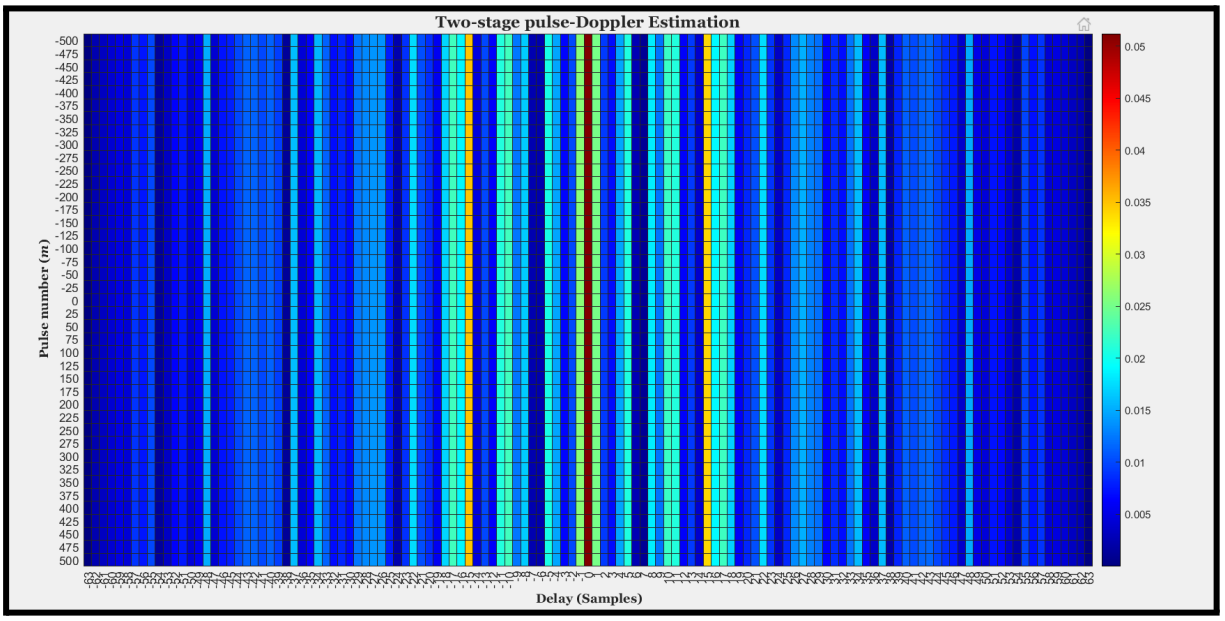
02/03/2022:

- Studied another one of the online lectures on radar
- I’m currently analysing a paper titled *Evaluation of the Ambiguity Function for Passive Radar with OFDM Transmission* that is referenced in *Comparison of Correlation-based OFDM Radar Receivers*
- The *raising* of the “floor” of the **ambiguity** surface is not good it seems (from the paper (*Evaluation of the Ambiguity Function for Passive Radar with OFDM Transmissions*). This is because as the floor is raised, return signals from weak targets are masked or not picked up by the radar receiver. It also reduces the dynamic range of the radar receiver.

05/03/2022:

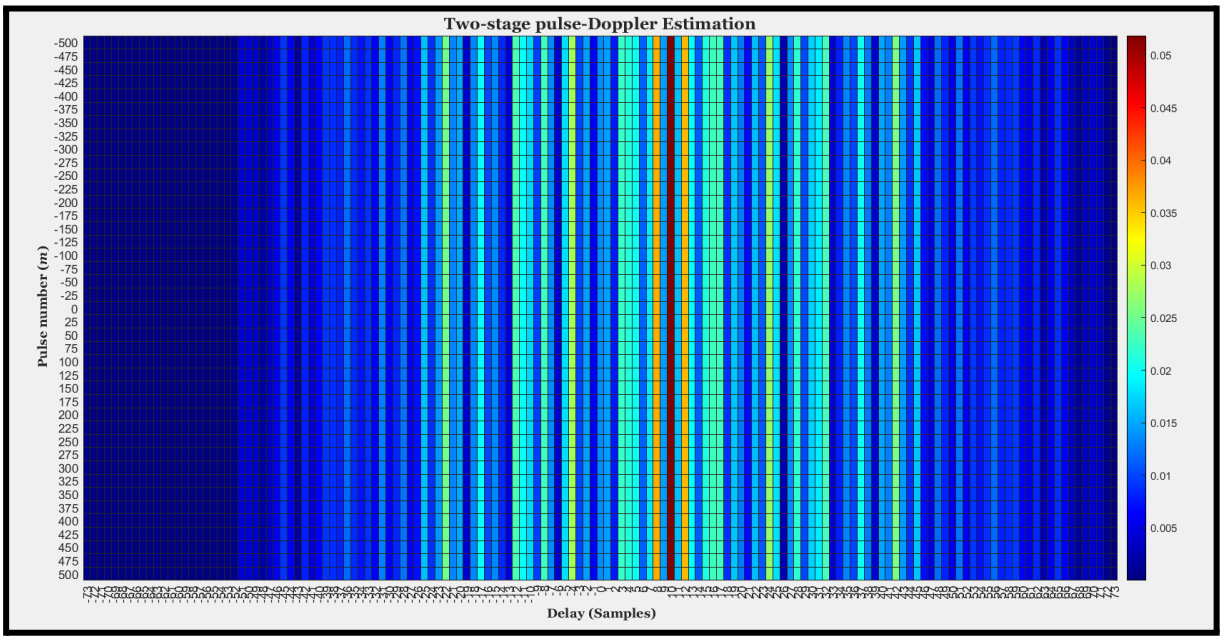
- I’m currently doing further analysis of *Evaluation of the Ambiguity Function for Passive Radar with OFDM Transmissions*. Taking notes, and trying to better understand the formulae that are used in the paper.
- I’m able to work out the 1D cross-correlation between the blocks or symbols of ***s*** and ***r***. But gaining information about the Doppler **dimension** has been unsuccessful so far. I have tried taking the FFT of each column, but the results don’t seem correct. Instead of using the FFT function in MATLAB, I have tried to implement the expression that I am seeing in these papers directly:

```
for block = 1:M
    Xm_dash = Xms(block, :).*negDopplerShift_m(:, block);
    X(1:length(nu), :) = X(1:length(nu), :) + Xm_dash;
end
```



Looking at the horizontal axis, the value is maximum when **delay** is 0. This is what the **delay** variable is set to.

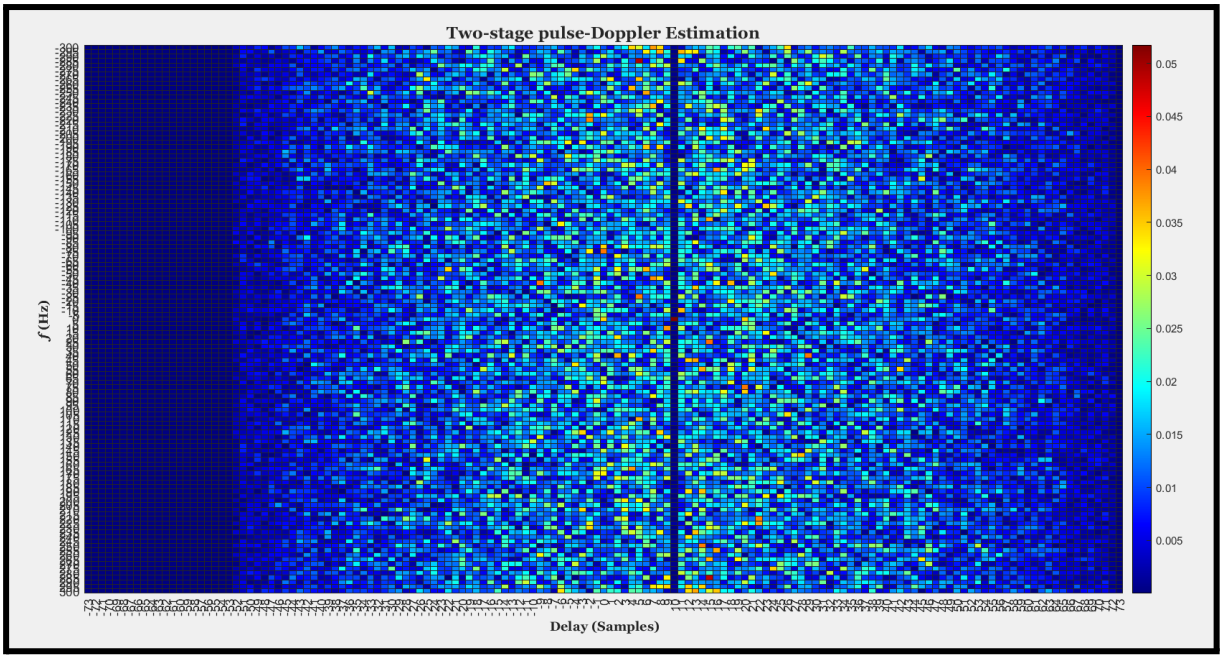
Changing **delay** to 10, the **dark column** seems to track this.



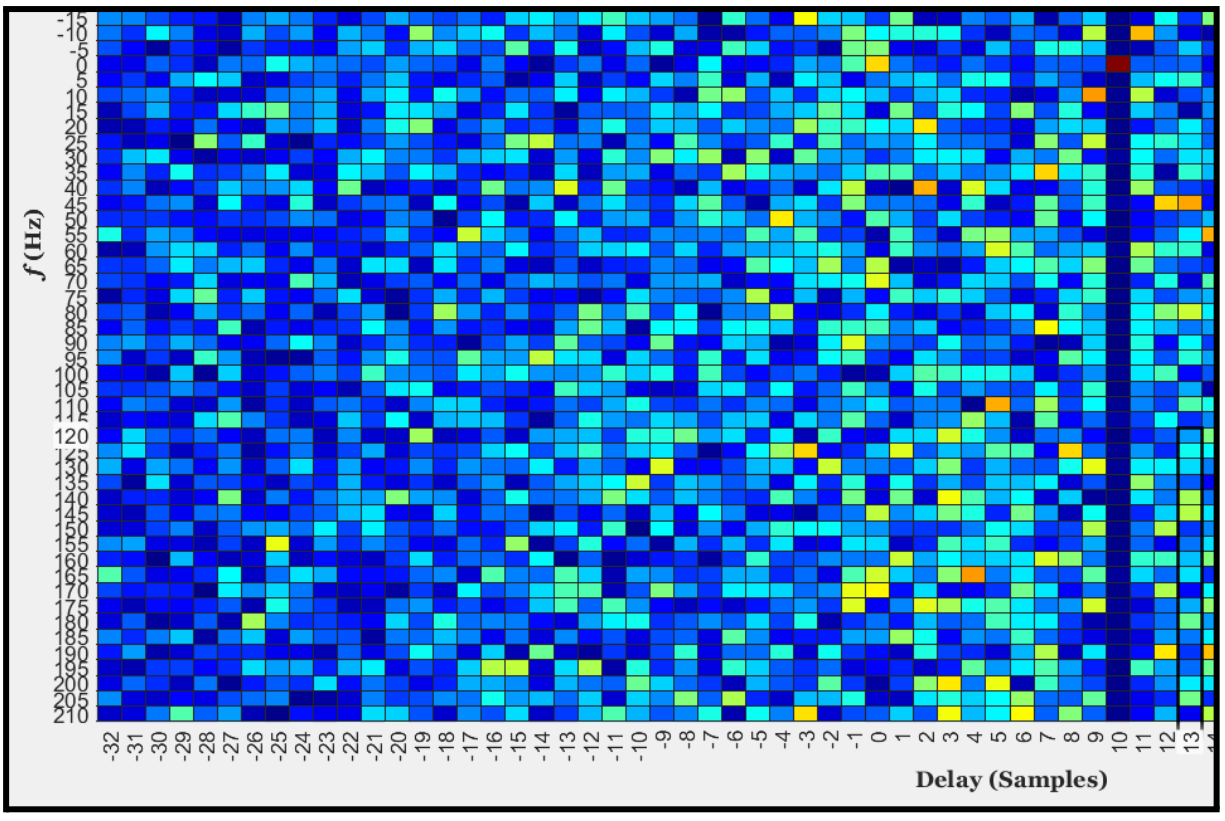
- The results change drastically based on whether I divide by M in the expression below:

```
negDopplerShift_m = exp(-1i*2*pi*nu'.*(0:M-1)/M);
```

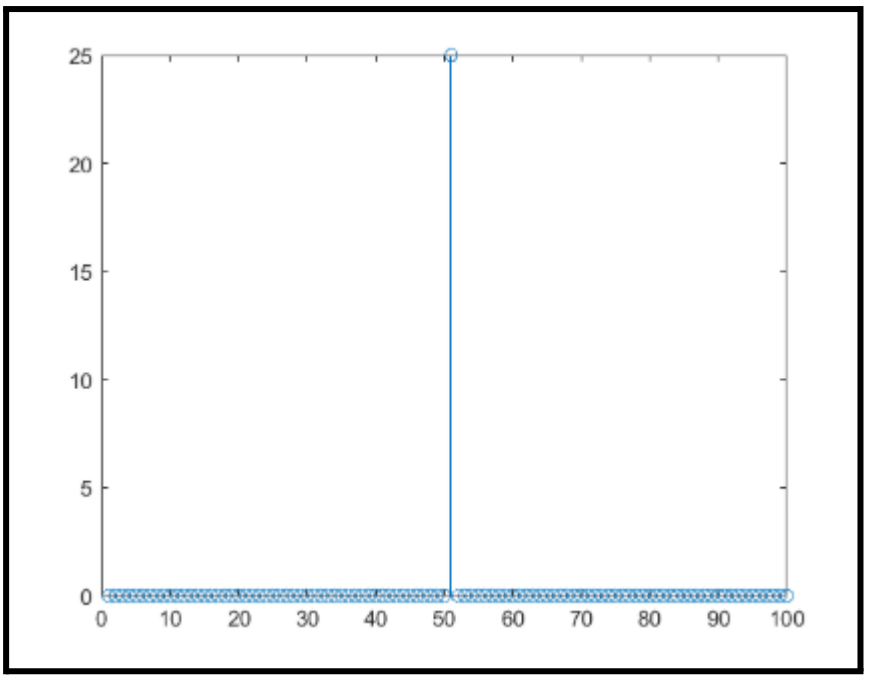
- The graph below is from my attempt to implement the expression shown in the paper:



In the centre crossing (at delay = 10 and frequency = 0) there is a strong peak:

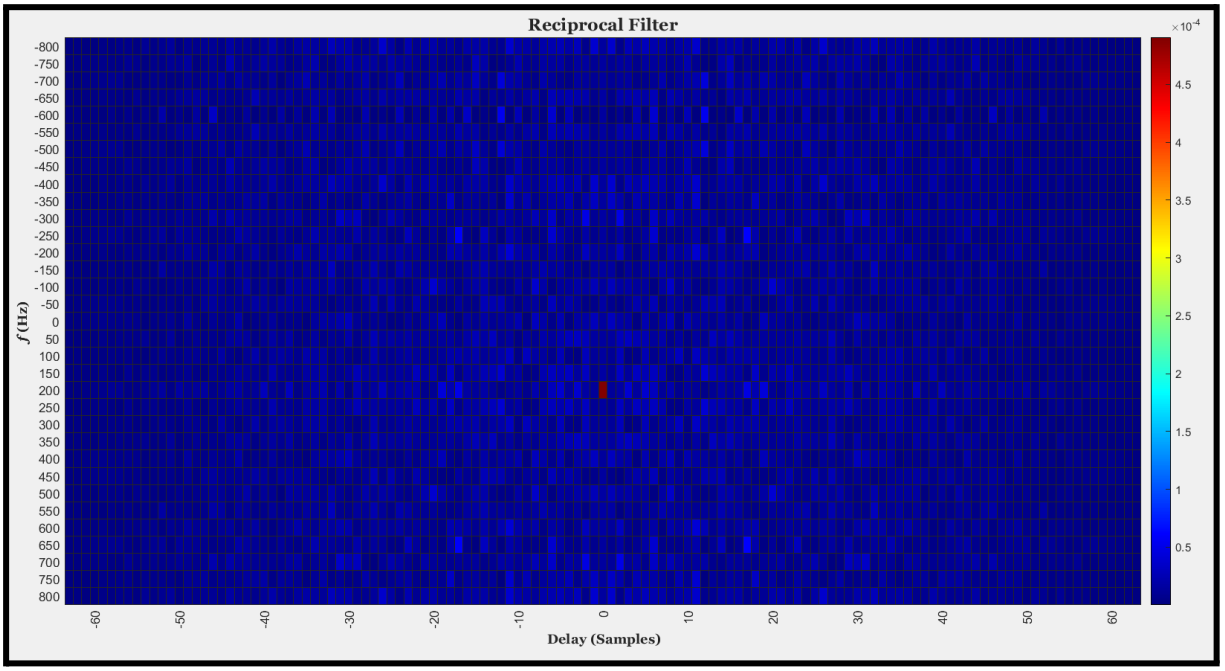


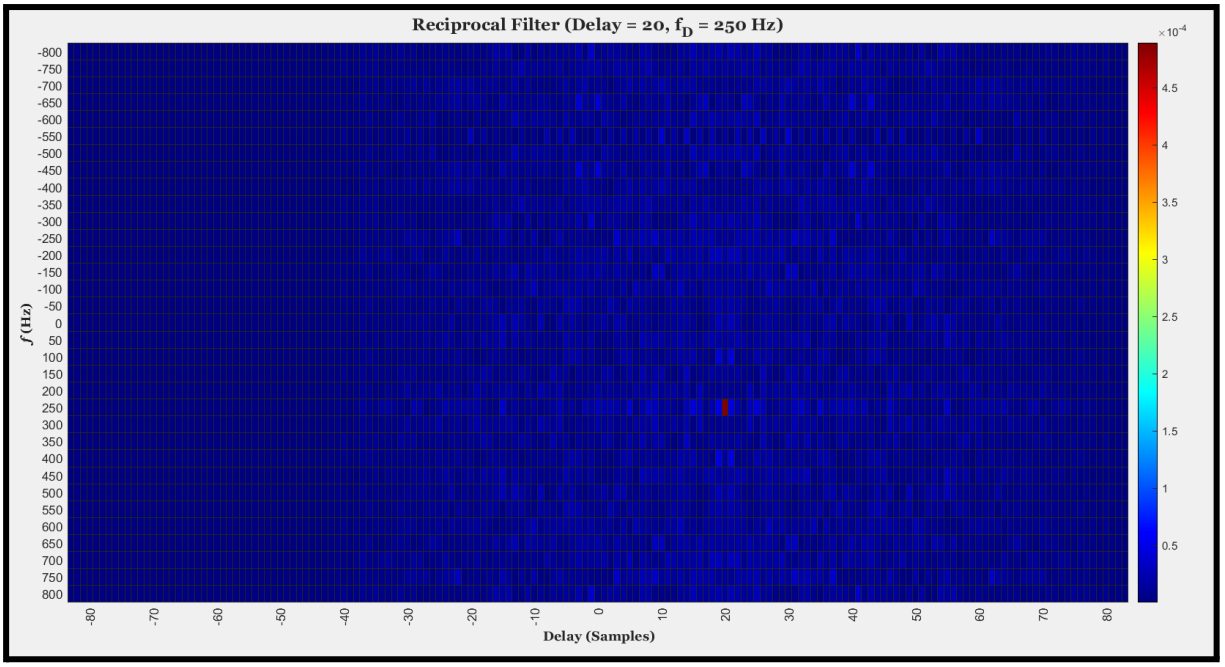
- The graph below is the DFT of a sequence of the same constant (0.25) number:



The peak is in the middle (DC bin), and it has a magnitude of 0.25*100, where 100 is the number of elements…

- I’ve been able to get some promising looking results for the reciprocal filtering approach:



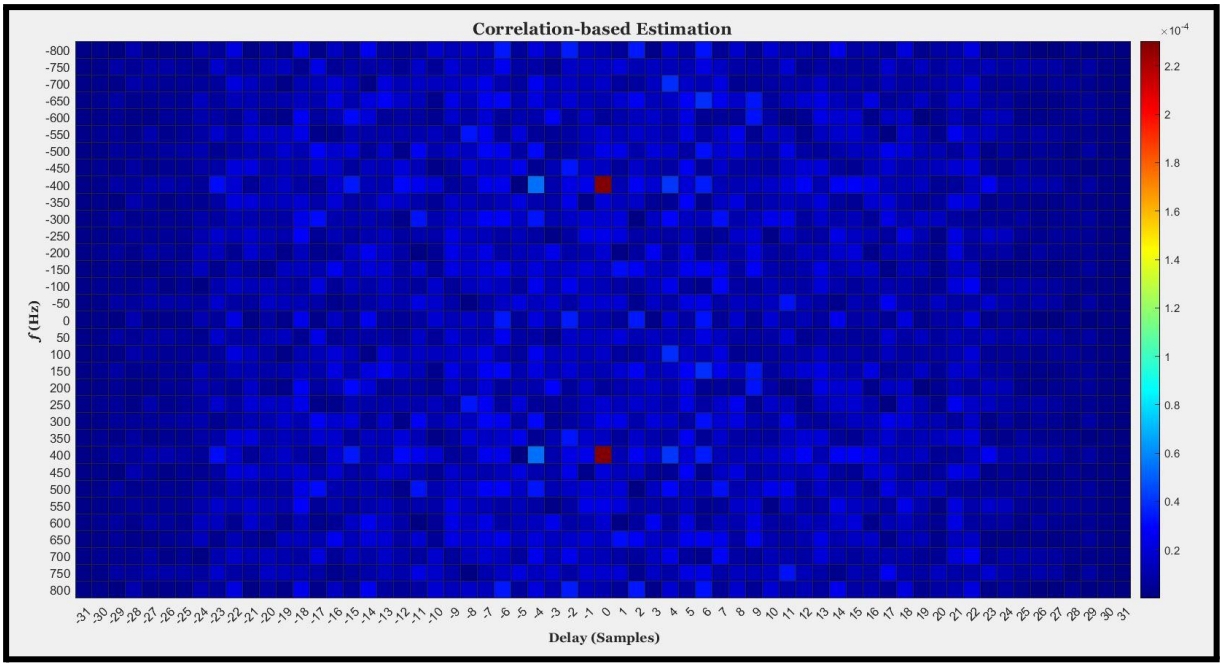


- I also found a way to easily include variable values in the title of the plot:

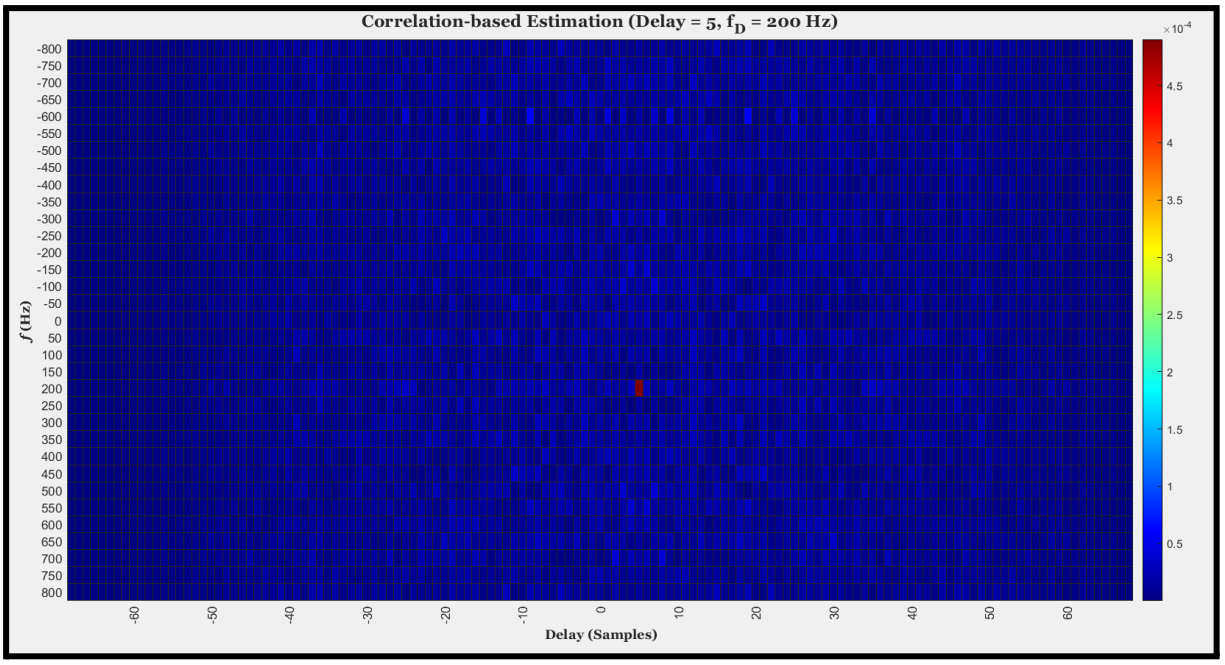
```
title("\fontsize{14}\fontname{Georgia}Reciprocal Filter (Delay = " + delay + ", f_{D} = " + fD + " Hz)");
```

As well as that, when I went to make this change in my other programs, I also discovered that by using the same values for **K** and **M** as I was using the reciprocal filter program, **ambiguous** peaks stopped appearing - for example:

(K = 32 M = 10)



(K = 64 M = 10)



06/03/2022:

- I've created a second version of the reciprocal filtering program. It allows for the cyclic prefix to be easily included or excluded.
- I'm currently reading some more of "DVB-T Passive Radar Signal Processing", a paper that I downloaded a few weeks ago.
- The paper discusses in more detail the **CPI** (Coherent Processing Interval). In one of the lectures on radar that I was listening to online, I came across the following formula for the coherent processing time:

  $T_{cpi} = MT_r$

  Where M is the number of pulses being processed. And T_r is the repetition interval, or time between pulses.

  In this paper, the same principle seems to be applied, although the variables or notation differs:

  $T_{cpi} = 68T_S$

  68 is the number of **OFDM symbols**, and $\mathbf{T_S}$ is the symbol time (equal to: $T_U + T_g$). This would also suggest that T_S is equivalent to the repetition interval, T_r. Thinking about it, this makes sense: every T_S seconds, another OFDM symbol is transmitted - the OFDM symbol *is like* the pulse in this case.

- The paper also mentions how the values in the map are normalised relative to the zero-delay, zero-Doppler peak. This suggests (as I was seeing in my MATLAB simulations) that the centre of the delay-Doppler map will also contain the maximum peak. Now that I come to think about it, this seems to make sense: the signal that

was transmitted **was not shifted in frequency**, nor **was it delayed** in time. *So* a received signal which has zero Doppler and zero delay ***would*** have the highest correlation with the signal that was sent out - so I think it makes sense that the centre of the Doppler-range map has the maximum peak.

- This paper gives an expression for ν as:

  $\nu = q(1/T_S)$, where q is an integer

So it seems that ν is related to $(1/T_S)$ which is equivalent to the pulse repetition frequency. Does this mean that, when I have a heatmap and I frequency-axis index of a peak as being ***q***, then in order to turn that into a frequency I simply need to multiply by $(1/T_S)$?

08/03/2022:

- I have made the updates to the to the correlation-based estimation program, and saved the updated code under ver5.m
- The main change is the following:

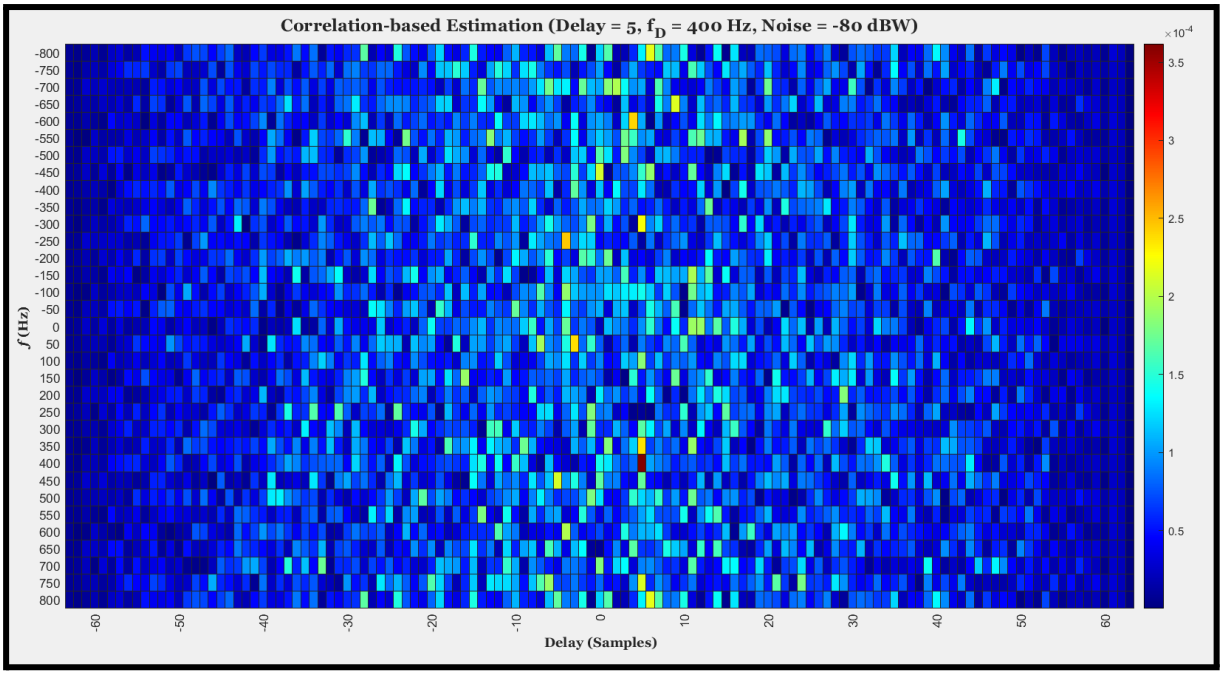
  The **propagation delay** (or simply **delay**) is added once, while the sequence **r** is formed.

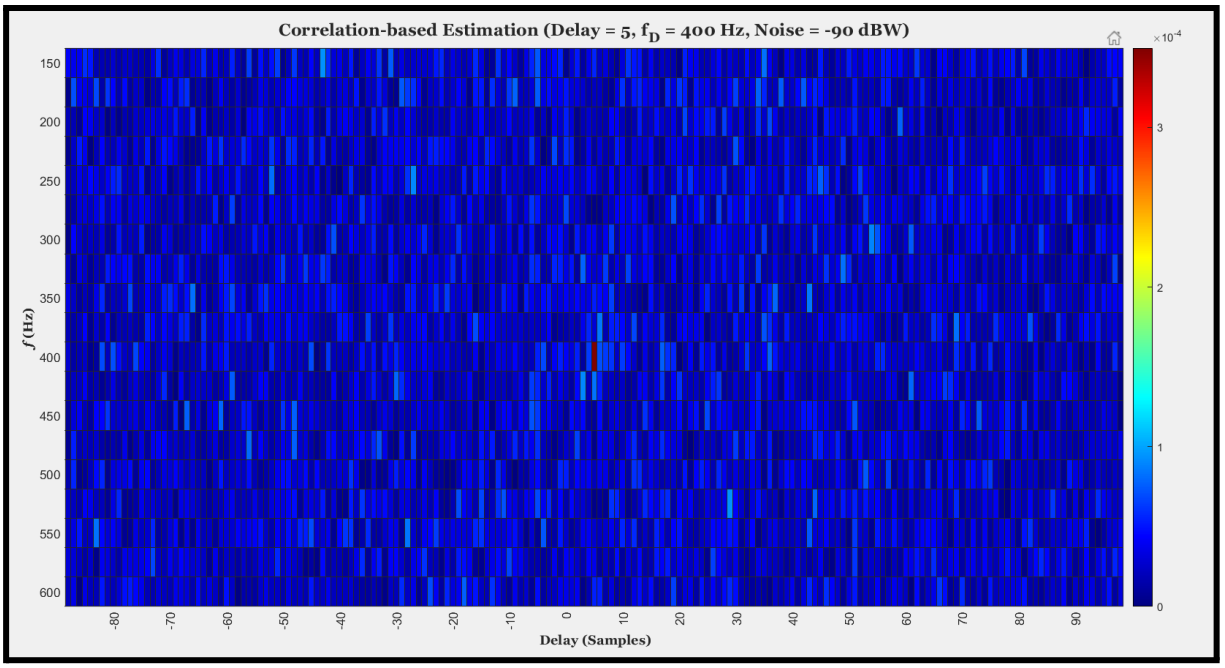
```
r = [zeros(delay, 1); dopplerShift.*s + Z]; % The received signal; adding the delay...
```

- Another part that was changed was the for loop in which the block-to-block correlation was being computed:

```
% This for loop works out Xm, as stated in the paper:
for block = 1:M
    n = block - 1; % This is an index variable used for forming rm and sm

    % sm stands for the part of s which forms OFDM symbol or
    % "block" m. rm stands for the part of r which is considered as the
    % block or symbol m at the receiver:
    sm = s(1+L*n: L*n + L); % The signal sm
    rm = r(1+L*n: L*n + L); % The signal rm
    r_dash = rm.*negDopplerShift.';

    % Now to calculate the correlation:
    Xm = zeros(length(nu), 2*L-1);

    % Computing Xm by using the xcorr() function
    for j = 1:length(nu)
        Xm(j, :) = xcorr(r_dash(:, j), sm);
    end
```

- Overall, the resulting heatmap (or intensity map/range-Doppler map) looks good:

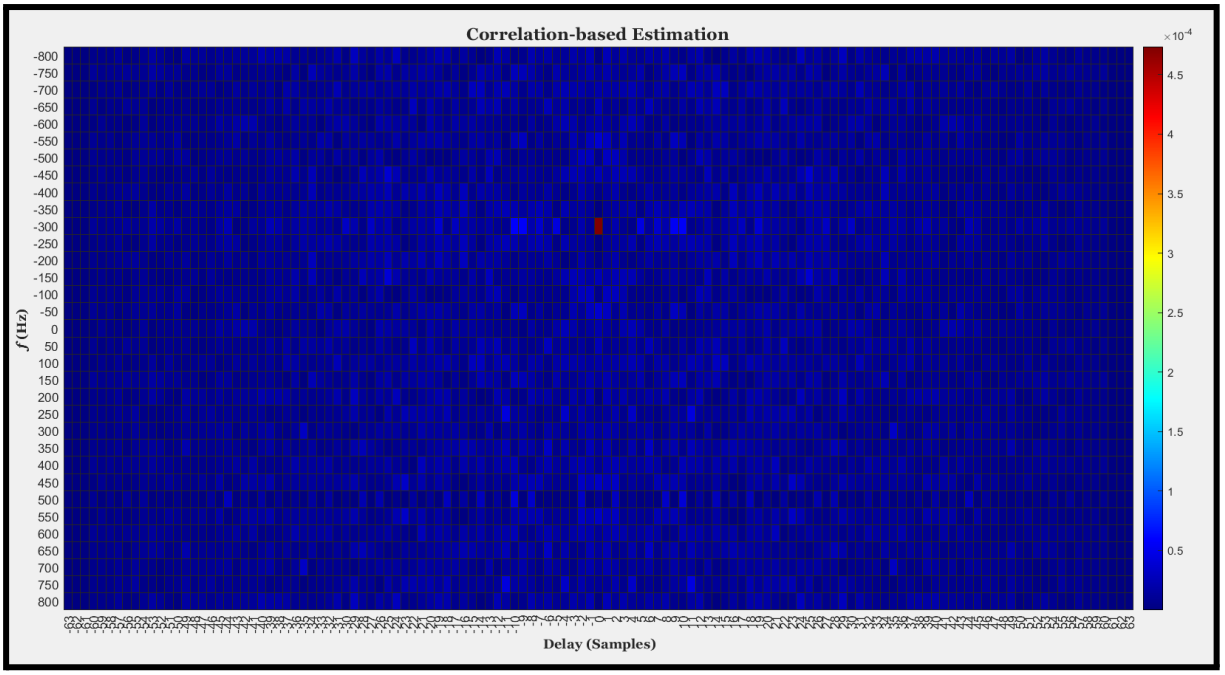




- I have changed most of the programs so that they use this formulation. I have also added the possibility of using or not using the cyclic prefix by changing the value of **G**

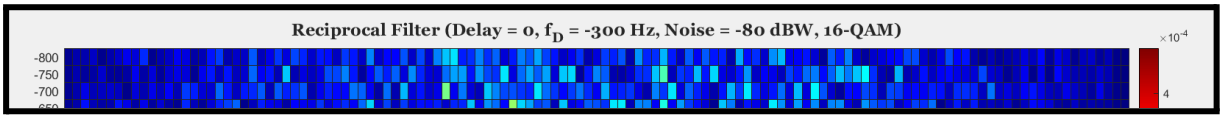
```
G = 0; % The fraction of the OFDM symol which is used as a guard interval
cpLen = G*K; % The cyclic prefix (CP) length
```

- The graph was obtained from changing K from 32 to 64. When K was 32, there were two red peaks. One lying along the -300 Hz line, and the other at 500 Hz line.



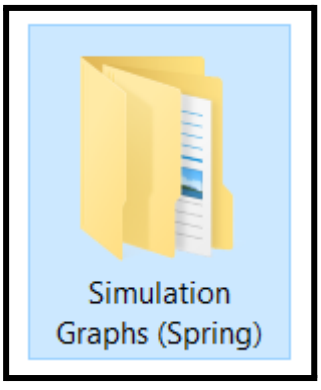
09/03/2022:

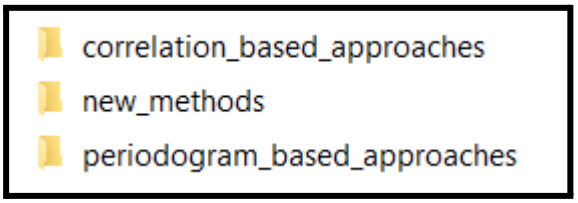
- I've updated the way in which I call the title() function in my .m scripts. It now includes the modulation type and order. This will make it clearer when it comes to inspecting graphs at some point in the future:

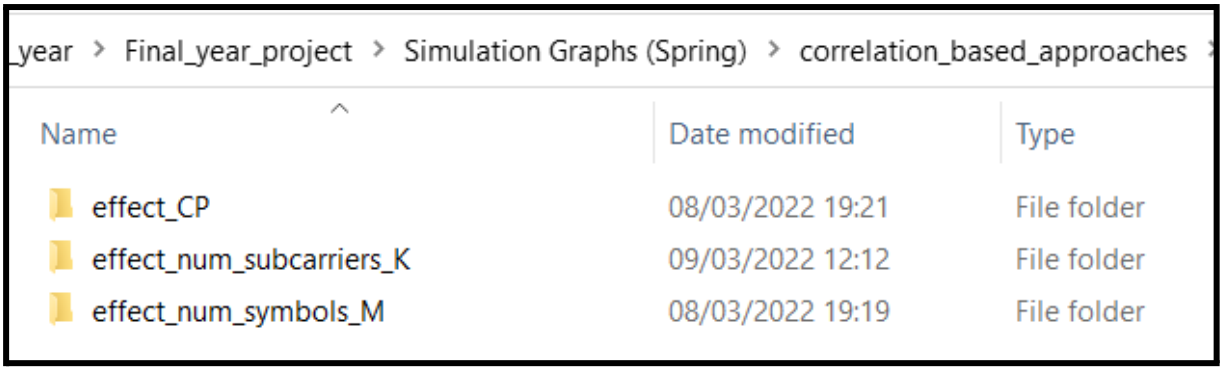



```
+ modOrder + "-" + modulationType + ")");
```

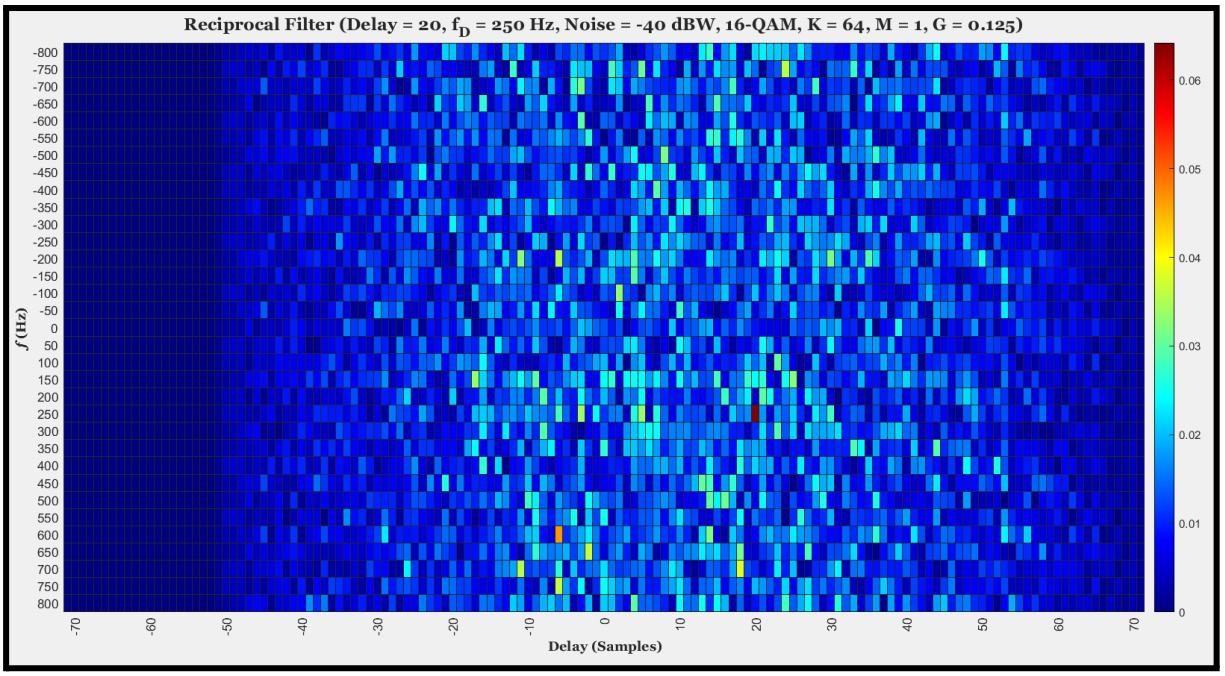
- My results also seem consistent, even though I made a fairly large change to how I was modelling the delay in the signal ***r***.
- I have also created the following folders into which my simulation results will go:



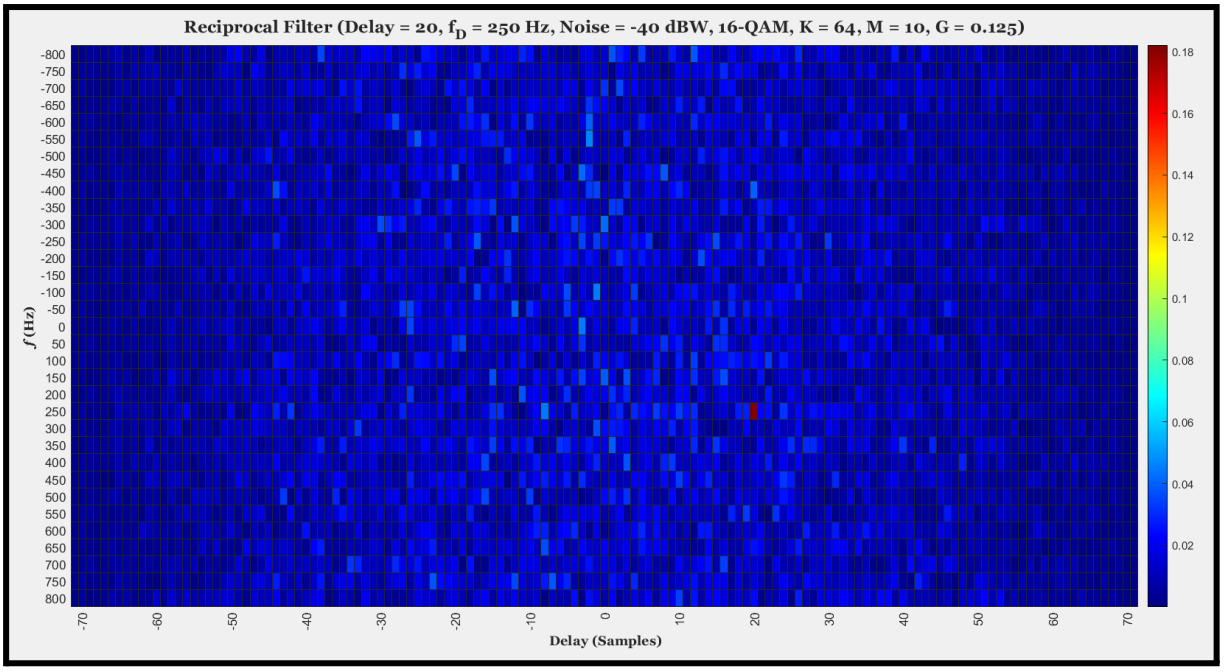




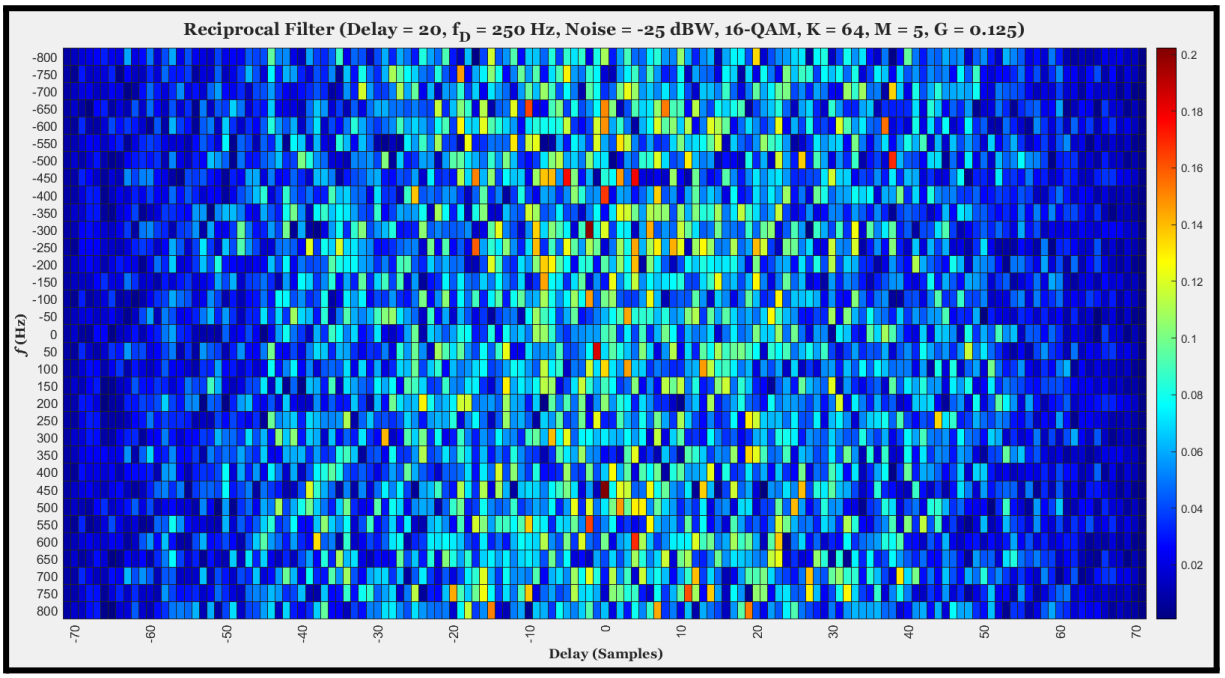
- I am doing some more analysis on the effects of changing **K** and **M** on the performance of the reciprocal filter. One thing that I notice is the following:



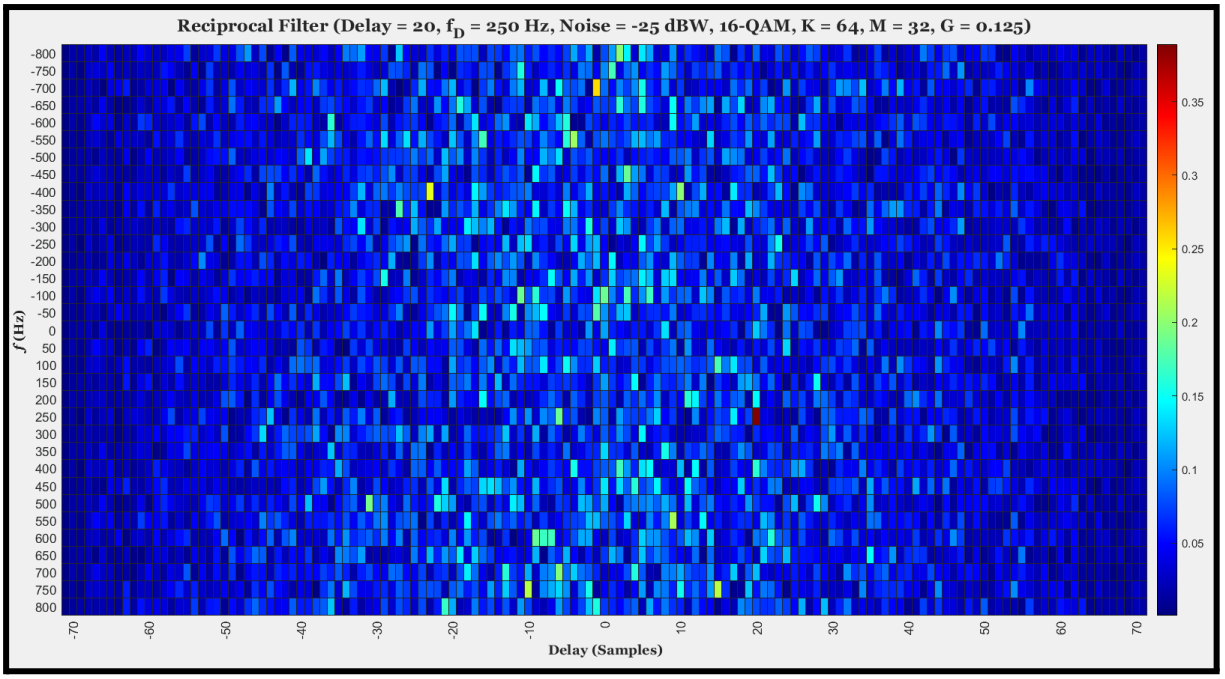
As stated in the title of the graph, M = 1, meaning there was only **1 symbol** in the OFDM frame. The delay and the Doppler shift are identified correctly. Notice the other coloured pixels. These are the result of noise. Now look what happens to the overall colour of the map when I increase **M**, the number of symbols, to 10:



The noise seems to have less of an effect now. There is less chance of picking a wrong point on the heatmap. This suggests that when we average X_m over a larger range (i.e. more symbols) the noise's effect becomes less noticeable. As an example, look at the next two plots, where I increase the noise power from -40 dBW to -25 dBW:

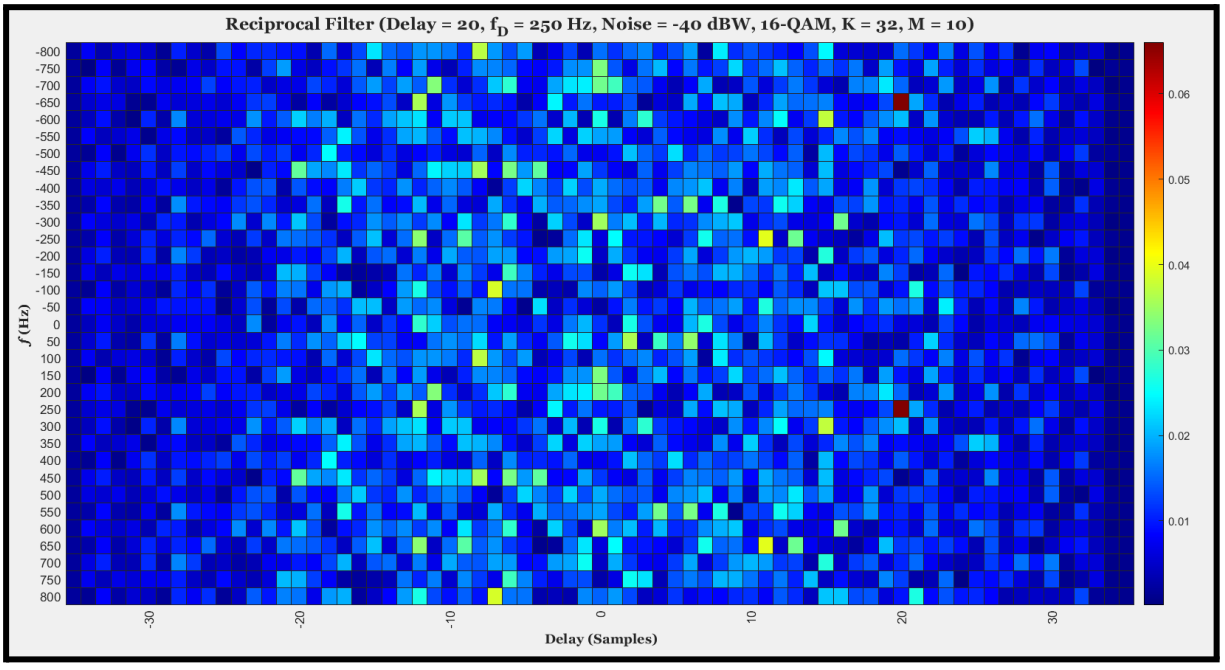


In the above heatmap, no meaningful or correct results can be obtained. But if I pick a larger value of M (the number of OFDM symbols):

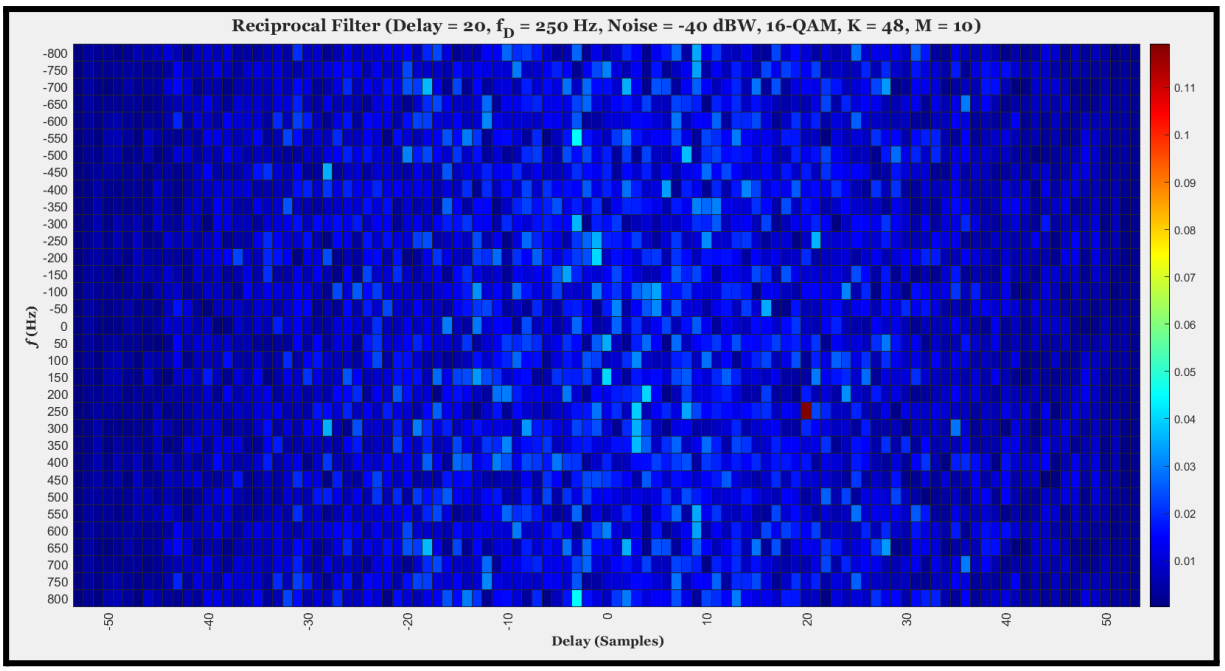


The results are much better! And the Doppler shift and delay can be correctly estimated.

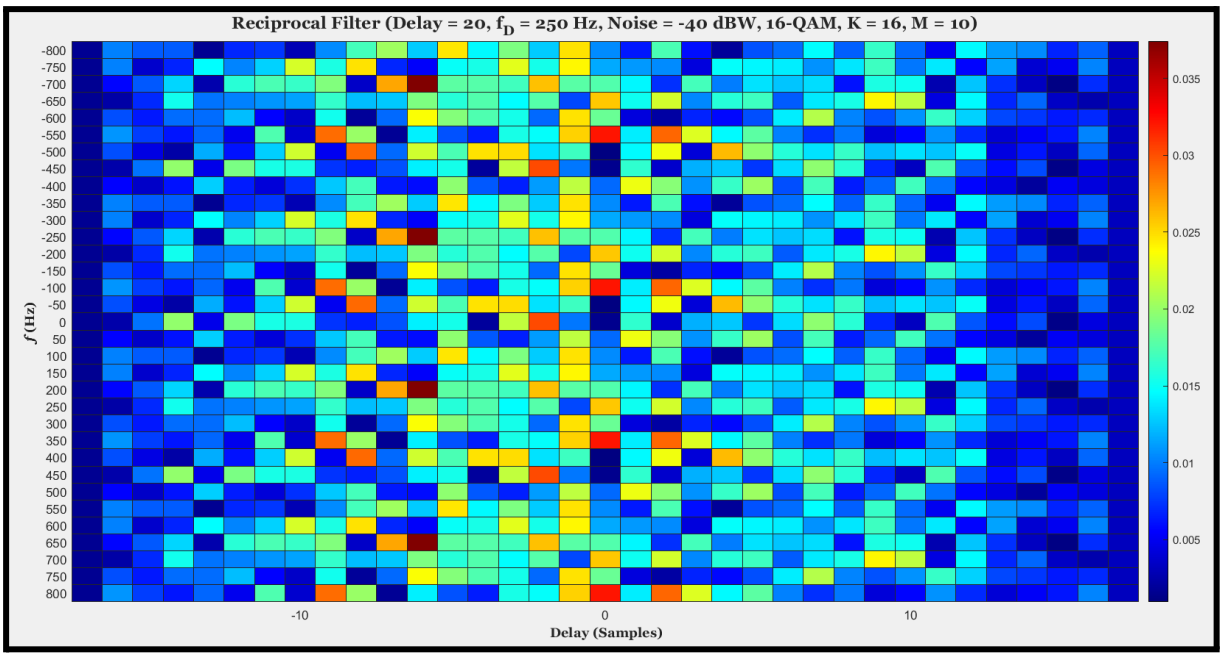
I also checked the effect of changing **K**.



As can be seen up the top, there is an **ambiguity**… When **K** is increased sufficiently it disappears:



Below is another example of what happens if **K** (the number of subcarriers being used) is too low:



We get many ambiguous results, and the measurement of the delay is compromised…

- I am currently writing a program, reciprocal_filter_msc.m, which investigates the effect of using the same symbol on each subcarrier, and for each symbol. The “msc” in the name is meant to be short for “modified subcarriers”.

Ftx

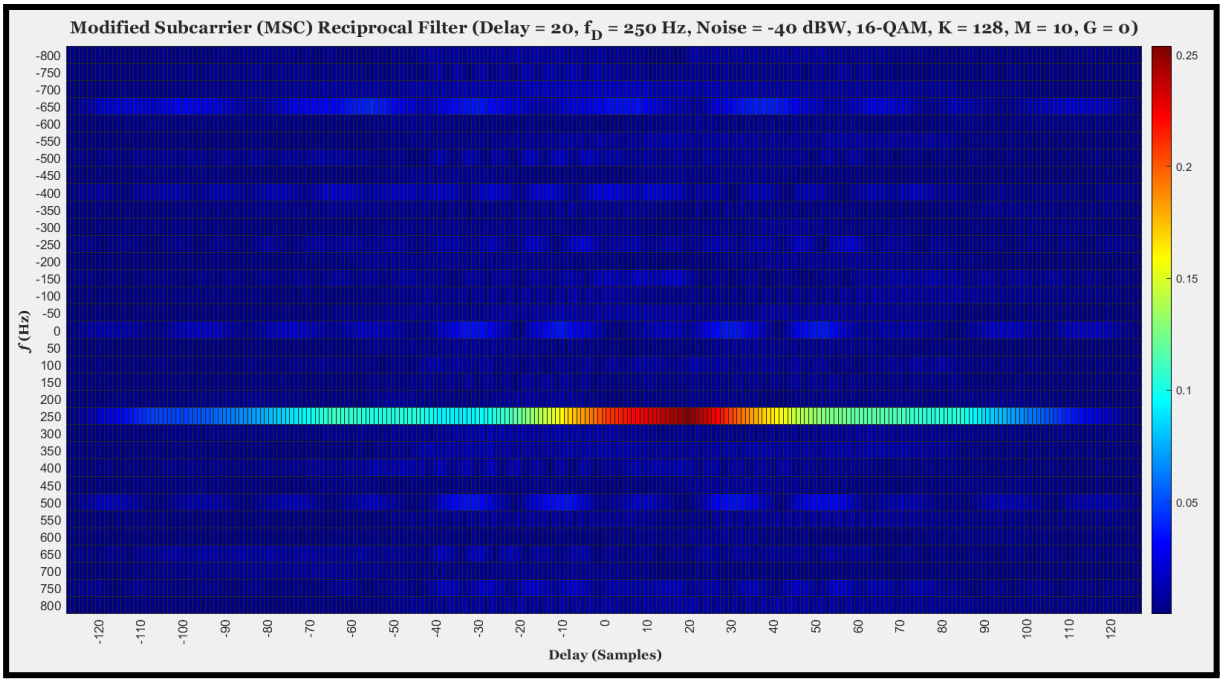
64x10 complex double

	1	2	3
25	-0.9487 + 0.9487i	-0.9487 + 0.9487i	-0.9487 + 0.9487i
26	-0.9487 + 0.9487i	-0.9487 + 0.9487i	-0.9487 + 0.9487i
27	-0.9487 + 0.9487i	-0.9487 + 0.9487i	-0.9487 + 0.9487i
28	-0.9487 + 0.9487i	-0.9487 + 0.9487i	-0.9487 + 0.9487i
29	-0.9487 + 0.9487i	-0.9487 + 0.9487i	-0.9487 + 0.9487i
30	-0.9487 + 0.9487i	-0.9487 + 0.9487i	-0.9487 + 0.9487i
31	-0.9487 + 0.9487i	-0.9487 + 0.9487i	-0.9487 + 0.9487i
32	-0.9487 + 0.9487i	-0.9487 + 0.9487i	-0.9487 + 0.9487i
33	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i
34	-0.9487 - 0.9487i	-0.9487 - 0.9487i	-0.9487 - 0.9487i
35	-0.9487 - 0.9487i	-0.9487 - 0.9487i	-0.9487 - 0.9487i

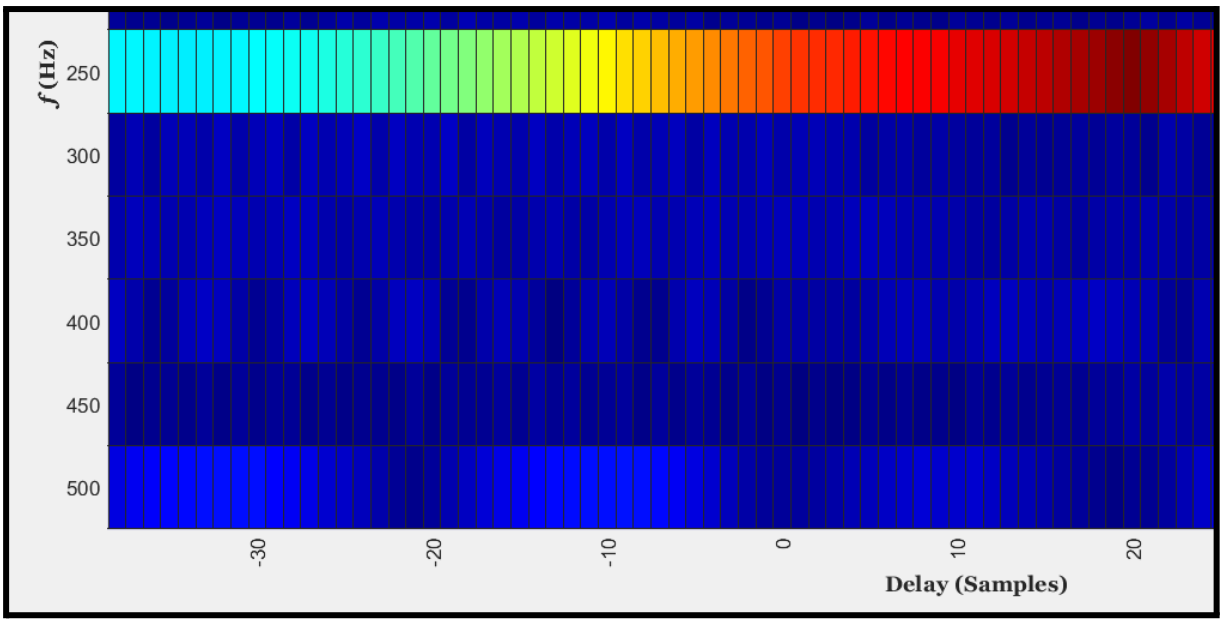
As can be seen, each row and each column is loaded with the same complex symbol. I can control **which** complex symbol from the symbol set is chosen by changing the following variable:

```
symbolNumber = 1; % We can easily change this to see if one symbol is more
% effective for radar than other
```

- Below is shown a graph that was obtained from running the program:



The maximum value actually lines up with the delay of 20 and the Doppler shift of 250 Hz:

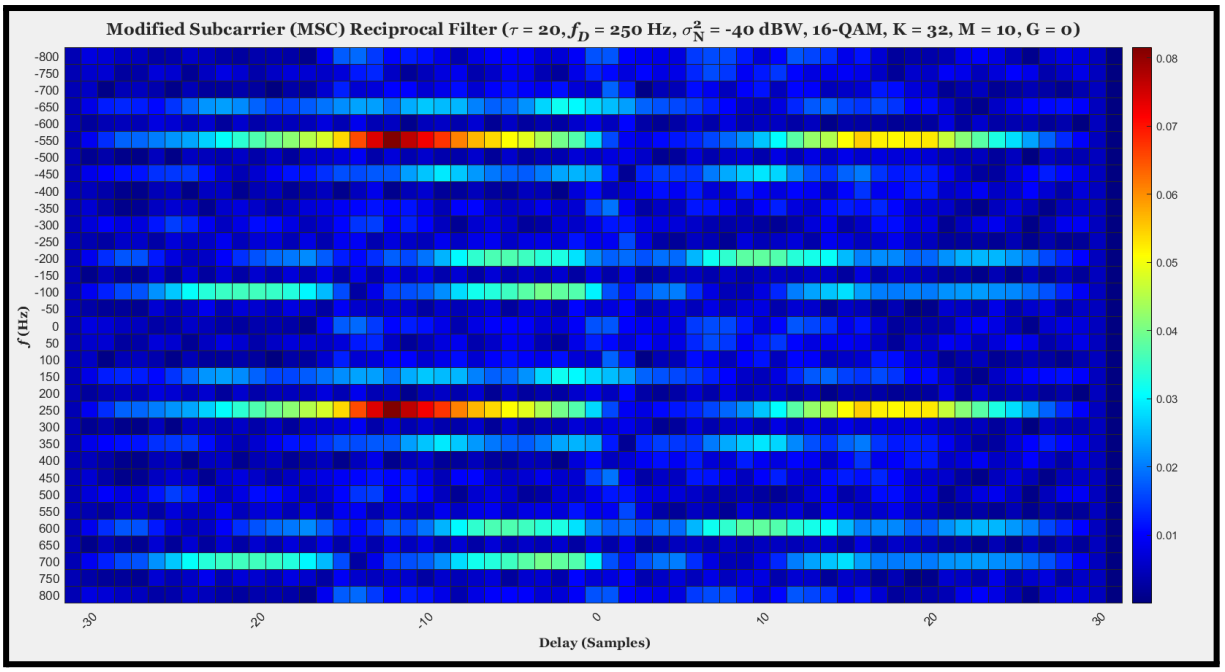


- I am also using sigma^2_N in order to represent the noise power in the title of the MATLAB plots. This is in order to save space:

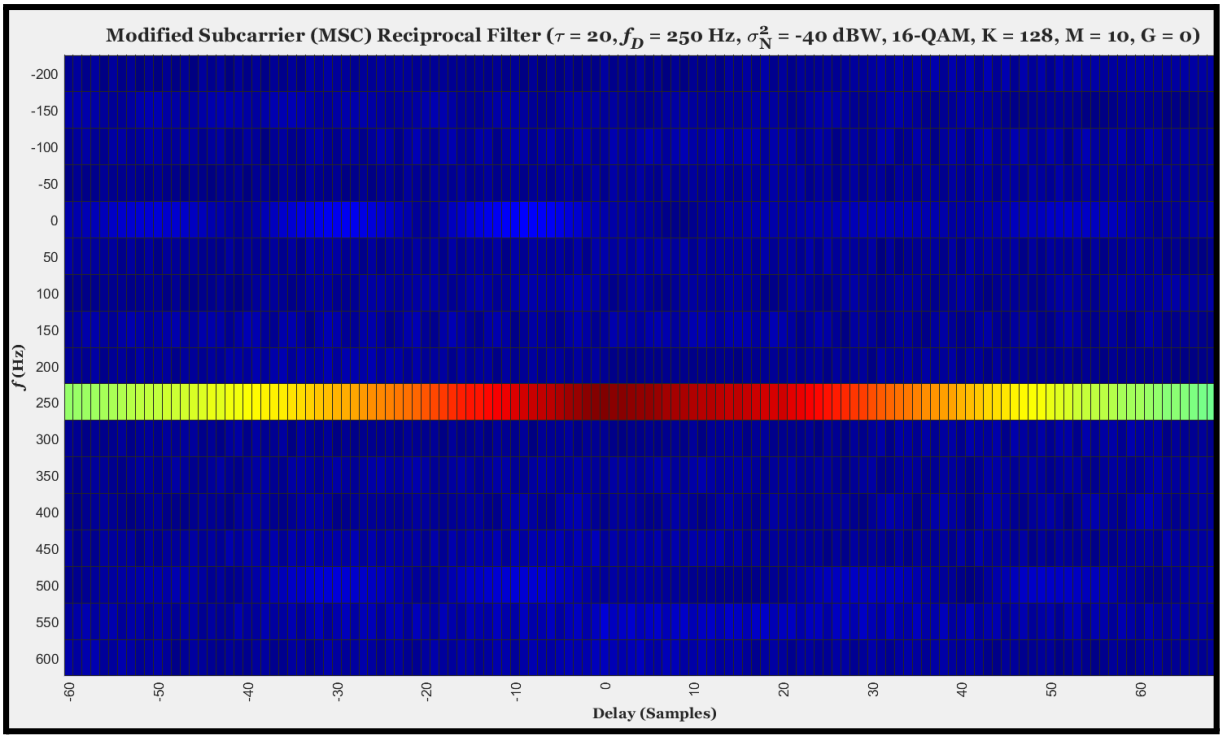
$$(\tau = 20, f_D = 250\ \text{Hz}, \sigma_N^2 = -40\ \text{dBW}$$

I'm also using $\boldsymbol{\tau}$ instead of "delay". It's just to save space as well.

- Like in the previous cases (analysis), reducing the number of subcarriers (K) too much leads to ambiguous results in the range-Doppler map (heatmap):



- And strangely enough, when I change symbolNumber from 1 to **2**, the result for the **delay** is compromised:



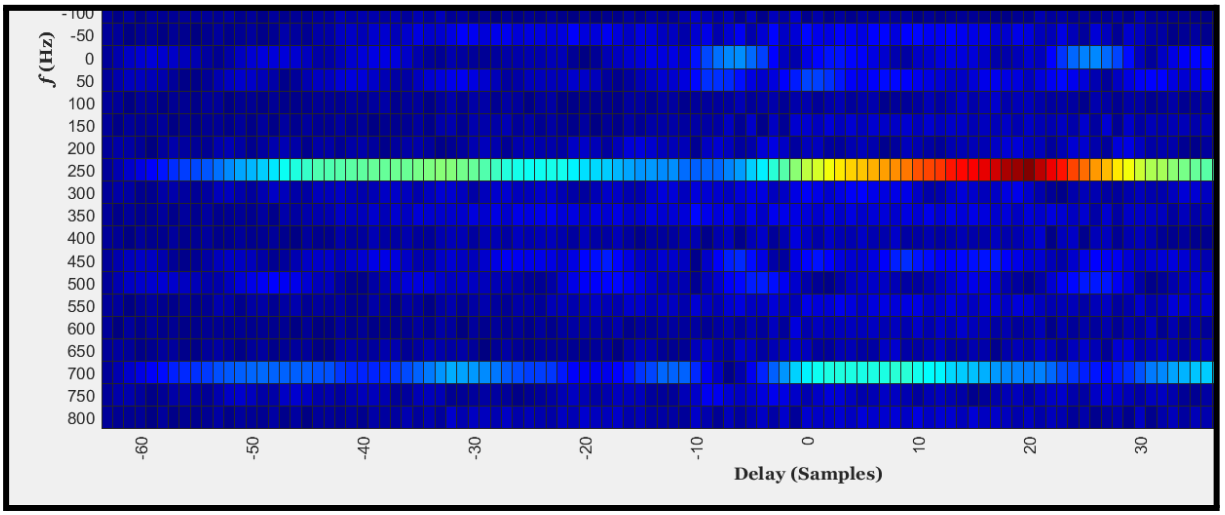
Symbol 2 of the symbol set is:

```
chosenSymbol =

  -0.9487 + 0.3162i
```

Whereas symbol 1 is:

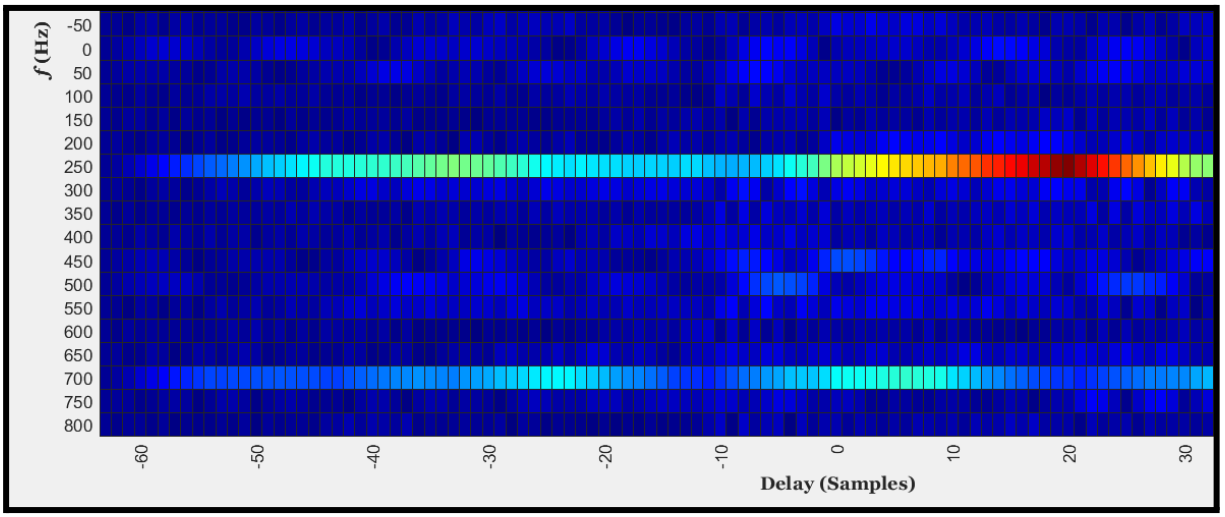
```
chosenSymbol =

  -0.9487 + 0.9487i
```

And the results with symbol 1 are much better:



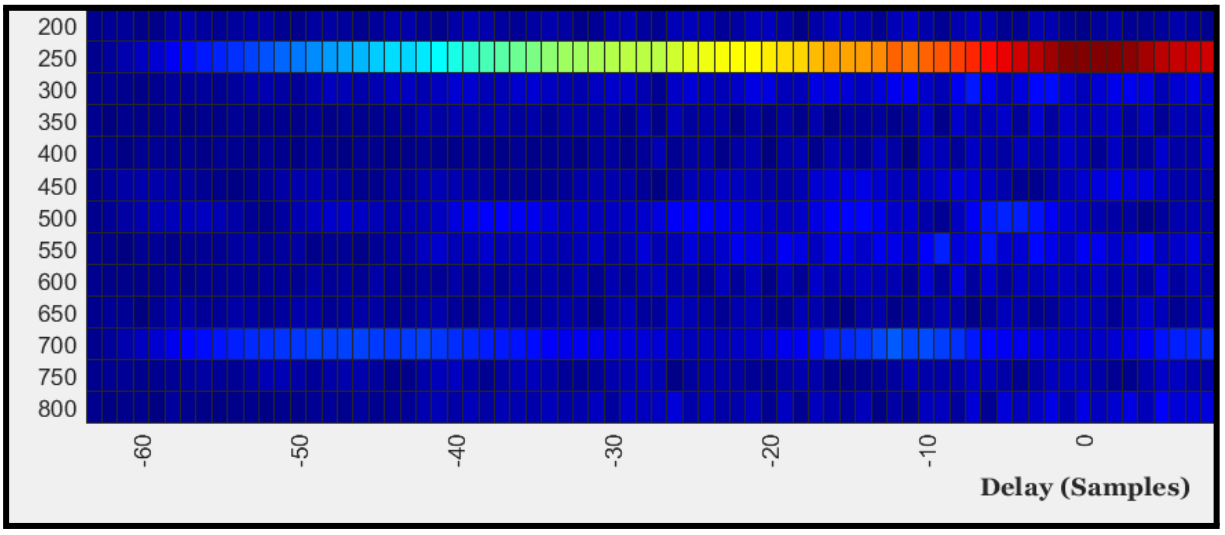
Symbol 3 from the symbol set also produces good results:

```
chosenSymbol =

  -0.9487 - 0.9487i
```



Whereas symbol 4 from the set of possible transmit symbols does not produce good results (at least as far as delay is concerned):

```
chosenSymbol =

  -0.9487 - 0.3162i
```

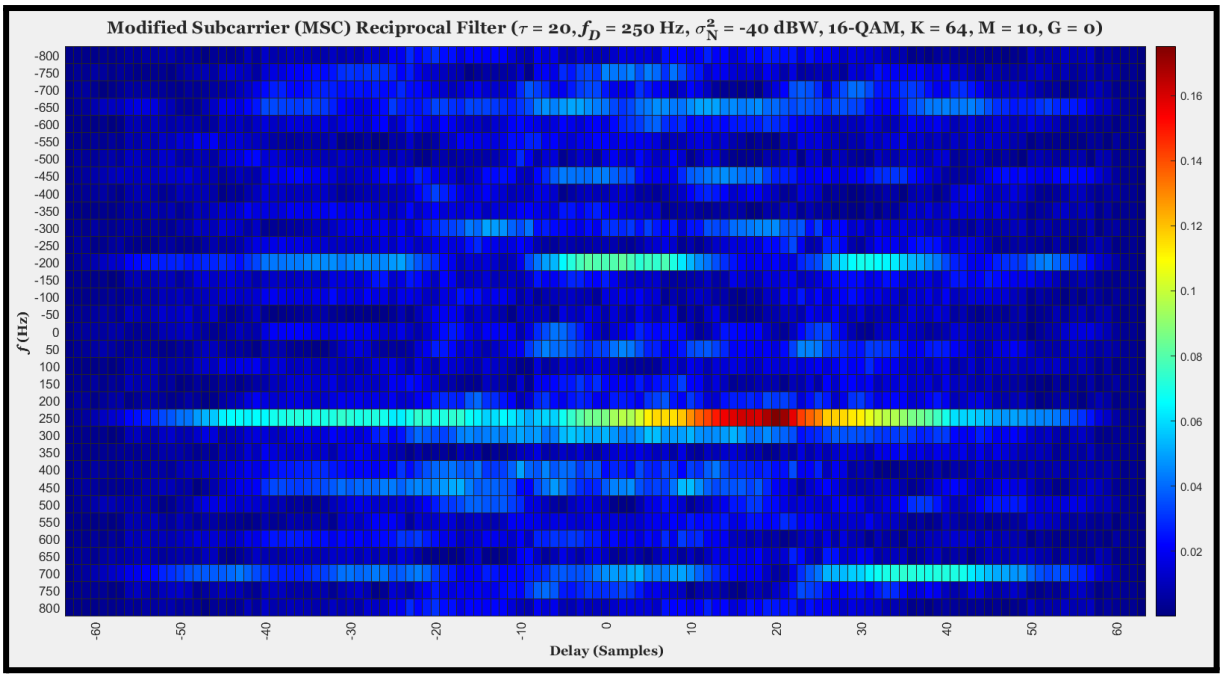


I do not want to jump to conclusions, but it seems as though symbols with an even index (e.g. 2, 4 etc.) do not produce good results. These symbols also have real and imaginary parts which have different magnitudes. Contrastingly, symbols with an odd index do seem to give both the correct delay and Doppler shift. These symbols have real and imaginary parts that have equal magnitudes.

After playing around some more, it seems that if the magnitudes of the real and imaginary components of the transmit symbols are the same, then the results are promising or useful:

```
chosenSymbol =

  -0.3162 + 0.3162i
```

The above corresponds to symbol **6** of the symbol set (for 16-QAM)



- I'm currently trying to analyse the computations involved in the correlation computation or calculation:

```
>> [c, lag] = xcorr(r, s)

c =

    6.0000   12.0000   20.0000   24.0000   18.0000   12.0000    4.0000


lag =

    -3    -2    -1     0     1     2     3

>> [c2, lag2] = xcorr(s, r)

c2 =

    4.0000   12.0000   18.0000   24.0000   20.0000   12.0000    6.0000
```

The above shows that if we have two sequences, say r and s, then the correlation between r and s is the flipped version of the correlation between s and r.

- I now understand better what this comment in the paper on correlation-based OFDM radar receivers means:

> [2]Given the received signal model (3), this implies discarding the last l_0 useful samples of r. In a realistic scenario where $M \gg 1$, the impact of such truncation is however negligible and will therefore be omitted in our development.

$$r[p] = \alpha \exp(j2\pi f_D p/L)s[p - l_0] + w[p] \qquad (3)$$

14/03/2022:

- I am currently reading up on DFT spread or DFT precoding. It seems it can be used to reduce the peak-to-average-power ratio of a signal or waveform. It also seems that it is especially useful with OFDM systems…
- From the paper that I have read so far, it seems that:
  (1) OFDM signal has a high PAPR
  (2) OFDM with CP has been adopted by 5G (we knew this already)
  (3) OFDM schemes have **poor frequency localisation** - this is due to the use of the rectangular pulse-shaping filter
  (4) Null subcarriers are used at the edge of the spectrum to help prevent interference from other wireless systems. But this negatively impacts the **spectral efficiency** of the system…
  (5) High PAPR can cause distortion when non-linear High-Power Amplifiers (HPAs) are used… this distortion can also lead to an increase in the bit error rate (BER)
  (6) If a signal has “weak excursions” then its “envelope” is quite constant or non fluctuating…

- I am currently writing a script which hopefully will allow me to better simulate an OFDM communication link…

```
channelSNR = 20; % the channel's signal-to-noise ratio...

if (usingAsRadar == false)
    r = awgn(s, channelSNR);
end
```

```
ofdmSymbols = ifft(Ftx); % taking the IFFT of each column of Ftx...
ofdmSymbols = [ofdmSymbols(end-cpLen+1:end, :); ofdmSymbols]; % adding the CP to each symbol

% Now to form the transmit sequence (s). This is got by
% concatenating all the OFDM symbols together:
s = ofdmSymbols(:); % the transmit sequence (s)
```

- I have more or less finished the script which models a OFDM communication link. I was having one little problem near the end, and it seems that it was caused by not having specified “UnitAveragePower” as being true when calling the qamdemod() function…

```
payloadData = Frx(2:N/2, :); % extracting the payload from the frame...
rxBits = qamdemod(payloadData(:), modOrder, "OutputType", "bit", "UnitAveragePower", true);
```

```
payloadData = Frx(2:N/2, :); % extracting the payload from the frame...
rxBits = qamdemod(payloadData(:), modOrder, "OutputType", "bit", "UnitAveragePower", true);

numBitErrors = numBits - sum(rxBits == txBits);
```

- With an SNR of 40, the bit error rate is 0:

```
channelSNR = 40; % the channel's signal-to-noise ratio...
```

BER	0

Even with an SNR of 20, the BER is still 0!

```
channelSNR = 20; % the channel's signal-to-noise ratio...
```

BER	0

15/03/2022:

- I am currently doing some paper analysis; the papers are discussing PAPR reduction by using a DFT block on the transmitter side
- I have added some code to my communication link script. It calculates the PAPR:

```
% Let us work out the PAPR:
PAPR = max(abs(Ftx(:)))/mean(abs(Ftx(:)));
```

PAPR	1.4810

- Small amendment: In the above screenshot, I am actually working out the PAPR of the transmit frame, which is made up of the complex symbols. This frame still has **not been** subject to the transmit side **IFFT** block (OFDM modulator) which is what increases the PAPR so much. Look below:

```
% Now to form the transmit sequence (s). This is got by
% concatenating all the OFDM symbols together:
s = ofdmSymbols(:); % the transmit sequence (s)

% Let us work out the PAPR:
PAPR = max(abs(s))/mean(abs(s));
```

PAPR	4.5100
PAPRframe	1.4611

As can be seen, the PAPR ratio of the transmit signal (s) is much higher now (and I am using unit average power). It is also higher than the PAPR of the frame…

```
% Let us work out the PAPR of the complex symbols in the transmit frame:
PAPRframe = max(abs(Ftx(:)))/mean(abs(Ftx(:)));
```

- I am just noticing that if I have a sequence, and I take the FFT of the sequence, the output values (at least some of them) tend to have much larger amplitudes…

```
a =

     2     3     4     5     2

>> b = fft(a)

b =

  16.0000 + 0.0000i  -3.7361 - 0.3633i   0.7361 - 1.5388i   0.7361 + 1.5388i  -3.7361 + 0.3633i
```

- I am trying to get the output of the IFFT block to be real-valued, as I have in my other programs. For some reason, it keeps coming out as complex. I have made a small test program, and just now I am getting better results:

DFTop	60x8 complex do...
Ftx	60x8 complex do...
IFFTip	122x8 complex d...
IFFTop	122x8 double

As can seen, the output from the DFT is of length 60, and is complex. The input into the IFFT block is of length 122, and is still complex. The input to the IFFT is longer… This is why it is called “DFT spreading”. Notice that the output from the IFFT block is real-valued…

```
IFFTip = [zeros(1, M); DFTop; zeros(1, M); conj(flip(DFTop, 1))];
IFFTop = ifft(IFFTip);
```

I am now going to check what happens when I insert more than **one row** of zeros between DFTop and the flipped-and-conjugated version of DFTop…

The output becomes complex again…

IFFTop	124x8 complex d...

- I think the problem is that I am misinterpreting what Hermitian symmetry is:

Ftx

32x8 complex double

	1
1	-1.0000 + 0.0000i
2	-0.9239 - 0.3827i
3	0.7071 + 0.7071i
4	-0.7071 + 0.7071i
5	1.0000 + 0.0000i
6	-0.9239 - 0.3827i
7	-0.3827 - 0.9239i
8	0.3827 - 0.9239i
9	0.3827 - 0.9239i
10	-0.0000 - 1.0000i
11	0.0000 + 1.0000i
12	-0.9239 + 0.3827i
13	-0.3827 - 0.9239i
14	0.9239 - 0.3827i
15	1.0000 + 0.0000i
16	0.3827 - 0.9239i
17	0.3827 + 0.9239i
18	1.0000 + 0.0000i
19	0.9239 + 0.3827i
20	-0.3827 + 0.9239i
21	-0.9239 - 0.3827i
22	0.0000 - 1.0000i
23	0.0000 + 1.0000i

I thought, for example, that the above matrix had Hermitian symmetry, but the output sequence is complex, not real-valued…

> A complex series {y(j),0<=j<=N-1} of length N that satisfies
>
> y(0) real and y(n-j) = conj(y(j)), j = 1, ..., N-1
>
> is said to have Hermitian symmetry, or simply to be a Hermitian series.
> Here conj(z) denotes the complex conjugate of complex number z.
>
> When N is even, y(N/2) is real for Hermitan y.

This page might offer some clues. Firstly, my first row is not real-valued… which might well be a problem…

Making the first row all zeros, and adding a middle row seems to cause the output to become real-valued again:

```
Ftx = [zeros(1, M); Ftx; zeros(1, M); conj(flip(Ftx, 1))];
ofdmSymbols = ifft(Ftx);
```

ofdmSymbols 34x8 double

> Hermitian symmetry is "imposed" on X_k symbols by ensuring that:
>
> $$x_0 = x_{N/2} = 0,$$
> $$x_k = x^*_{N-k} \forall k \in \left(k, \frac{N}{2}\right)$$

The above diagram is simply it: no argument. I must ensure my sequence obeys this rule, if I want the output to be real-valued.

I might *just* have sussed it… but more study and confirmation is needed about this.

16/03/2022:

- I think I have code which does *some* kind of interleaving of the symbols coming out of the DFT block, and going into the IFFT block…

	1	2	3	4	5	6	7	8
1	-6.9570 + 5.0596i	-5.6921 - 3.7947i	16.4438 - 0.6325i	0.6325 - 0.6325i	-1.2649 - 2.5298i	8.2219 + 0....	2.5298 - 2....	-1.2649 + 5...
2	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....
3	5.7901 - 8.8978i	-8.7606 + 6.1951i	9.1844 + 4.2445i	-7.6190 + 14.7016i	-1.0011 - 8.1964i	7.2812 + 2....	-1.7063 + 3...	-5.0023 - 0....
4	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....
5	1.0616 - 0.8361i	2.3068 - 7.3825i	4.0202 - 2.2157i	12.1868 - 1.5548i	3.4202 - 14.8683i	-2.6494 - 4....	-5.0070 - 0....	-3.2366 + 5...
6	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....
7	4.9313 + 4.7950i	-3.2336 + 2.3416i	1.1862 - 6.3331i	-4.7793 + 2.5624i	-3.2971 + 7.8912i	-5.3450 - 1...	-3.8367 - 7....	-10.1767 - ...
8	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0.0000i	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....

In the above example, M = 8 (as I only want 8 symbols in my transmit frame…)

- I think I have also added the option of specifying “localised” mapping:

halfIFFTinput

127x8 complex double

	1	2	3	4	5	6	7	8	9
25	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	
26	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	
27	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	
28	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	
29	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	
30	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	
31	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	
32	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	
33	-9.4868 - 1...	-8.8544 - 1....	-3.1623 - 3....	-0.0000 - 1....	4.4272 - 0....	11.3842 - 1...	-0.6325 - 0....	-0.6325 + 1...	
34	1.1866 - 7....	-1.1653 - 8....	-2.4763 - 8....	7.0766 - 1....	10.0669 - 4...	18.4053 - 5...	9.5348 + 5....	1.4573 + 9....	
35	-0.1630 - 0....	-3.4585 + 2...	5.7994 - 0....	1.8934 + 3....	-1.8262 + 5...	0.6687 - 2....	-3.9654 - 1...	0.5112 + 6....	

halfIFFTinput

127x8 complex double

	1	2	3	4	5	6	7	8
94	4.5983 - 1....	2.4115 - 6....	10.3546 - 4...	-1.0411 - 1....	0.6535 + 0....	-10.7692 + ...	1.4240 + 0....	-1.9054 + 1...
95	7.6797 + 1....	-6.4772 + 7...	-3.9146 + 0...	-2.9439 - 1....	3.3464 - 1....	2.2311 + 2....	-1.3154 - 7....	-9.3555 - 8....
96	6.1979 + 3....	-9.9819 + 4...	-1.4054 - 4....	7.3995 + 3....	-3.2461 + 3...	1.3571 - 1....	5.1788 + 9....	-1.6809 - 0....
97	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....
98	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....
99	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....
100	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....

- I have altered the section which computes the cyclic prefix length before adding it to the frame of OFDM symbols:

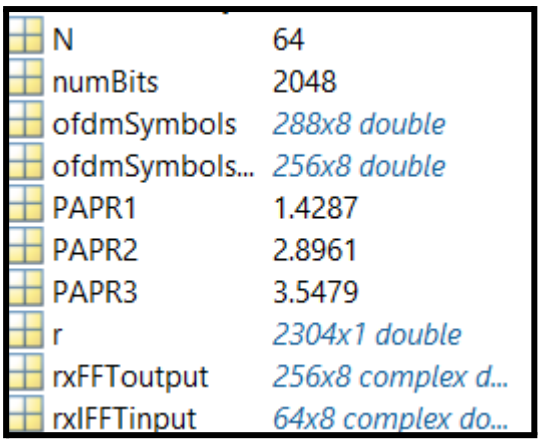
```
% Add the cyclic prefix:
G = 1/8; % the fraction of a symbol that will be used as the cyclic prefix...
cpLen = G*L; % the cyclic prefix length...

ofdmSymbols = [ofdmSymbols(end-cpLen+1:end, :); ofdmSymbols]; % adding the CP to each symbol
```

- I have found this very nice idea online of how to extract only the even index rows of a matrix in MATLAB:

```
r =

    0.3008    0.8378    0.2263    0.5166    0.1551
    0.8389    0.2430    0.9199    0.2852    0.2256
    0.6798    0.0195    0.9360    0.0095    0.8776
    0.3285    0.6075    0.4886    0.2929    0.6057
    0.5383    0.5388    0.6976    0.6665    0.6468

>> r = r(2:2:end, :)

r =

    0.8389    0.2430    0.9199    0.2852    0.2256
    0.3285    0.6075    0.4886    0.2929    0.6057
```

In my case I actually only want to retain the odd index rows… so I am going to use something like:

```
r =

    0.0692    0.6572    0.3985    0.6424    0.2380
    0.2185    0.5197    0.6872    0.5008    0.7599
    0.7800    0.9475    0.0731    0.5301    0.9601
    0.6291    0.0209    0.7580    0.7383    0.1662
    0.0706    0.7287    0.0507    0.2746    0.3786

>> r = r(1:2:end, :)

r =

    0.0692    0.6572    0.3985    0.6424    0.2380
    0.7800    0.9475    0.0731    0.5301    0.9601
    0.0706    0.7287    0.0507    0.2746    0.3786
```



I think on the receiver side I have correctly implemented the FFT and IFFT blocks - I suppose the proof of this will be better seen in the bit error rate!

- I think my channel is working with the DFT spread technique included!

```
channelSNR = 10; % the channel's signal-to-noise ratio...
```


```
channelSNR = 20; % the channel's signal-to-noise ratio...
```

- I just realised that when trying to load data symbols to particular rows of the IFFT, I end up specifying an **arithmetic sequence** expression. For example, suppose I want to load symbols onto the odd rows: 1, 3, 5, 7…

  This can expression generally as: $T_n = 1 + (n-1)2$

  1 is the first term
  2 is the common difference

- Suppose I want to insert three rows of zeros between my data symbols. So I want my data symbols to be on: 1, 5, 9, 13…

  This can be expressed as an arithmetic sequence:

  $T_n = 1 + (n-1)4$

halfIFFTinput

255x8 complex double

	1	2	3	4	5	6	7	8
244	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....
245	-8.8792 + 3...	-1.8495 - 6....	1.7495 + 3....	3.8478 + 6....	9.1988 + 1....	-0.3205 - 2....	-0.7047 - 3....	2.3598 + 1....
246	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....
247	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....
248	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....
249	6.0131 - 10...	-8.8677 + 0...	-3.6496 - 2....	0.7795 + 3....	6.7271 - 4....	-0.8041 - 0....	1.4763 + 0....	1.3461 - 5....
250	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....
251	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....
252	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....	0.0000 + 0....
253	-2.0372 - 8....	-0.5950 + 2...	6.1445 + 1...	-2.9556 - 8....	-5.3094 - 4....	1.7218 + 5....	-0.7020 - 5....	-0.8686 + 5...

```
for i = 1:N
    halfIFFTinput(1+(i-1)*(interPadLength+1), :) = DFToutput(i, :);
```

Instead of using lowercase “n”, I used “i”:

1 + (i - 1)*(interPadLength + 1)

interPadLength was 3 in my case

```
case "interleaving"
    % The variable below is the amount of zeros that we will insert between
    % each symbol on the input pins of the IFFT:
    interPadLength = floor(excess/(N-1));
```

- My program seems to working after my modifications (which hopefully make it more general and more easy to change):

BER	0

```
channelSNR = 20; % the channel's signal-to-noise ratio...
```

```
channelSNR = 10; % the channel's signal-to-noise ratio...
```

Name	Value
BER	0.0015

numBitErrors	3

17/03/2022:

- I am doing some analysis of papers discussing DFT-spread OFDM (DFT SOFDM) before I continue doing simulations in MATLAB.

- One of the papers that I am reading at the moment is "**On Performance Limits of DFT Spread OFDM Systems"**
- UE stands for User Equipment. We want the power consumption of User Equipment to be low. Hence, the high PAPR of OFDM transmission is undesirable for **uplink** transmission.
- For LTE 3G, the adopted **uplink** modulation scheme has been **DFT-Spread OFDM** (DFT SOFDM)
- In standard OFDM transmission, each sub-channel can be thought of as an AWGN channel, with an **SNR determined by** the receiver noise and **channel's response** at that sub-channel only. The same **cannot** be said about **DFT SOFDM**.
- Unitary Fourier Matrix:

  $FF^H = I$
- I am trying to write a cleaner and more concise program that will try to combine DFT-spread OFDM with the reciprocal filtering technique.

  I have been able to more easily define the input sequence to the tx-side IFFT block, which I have called symbolMapOp (as this is the sequence that comes out of the symbol mapper and goes into the IFFT block)

symbolMapOp

256x8 complex double

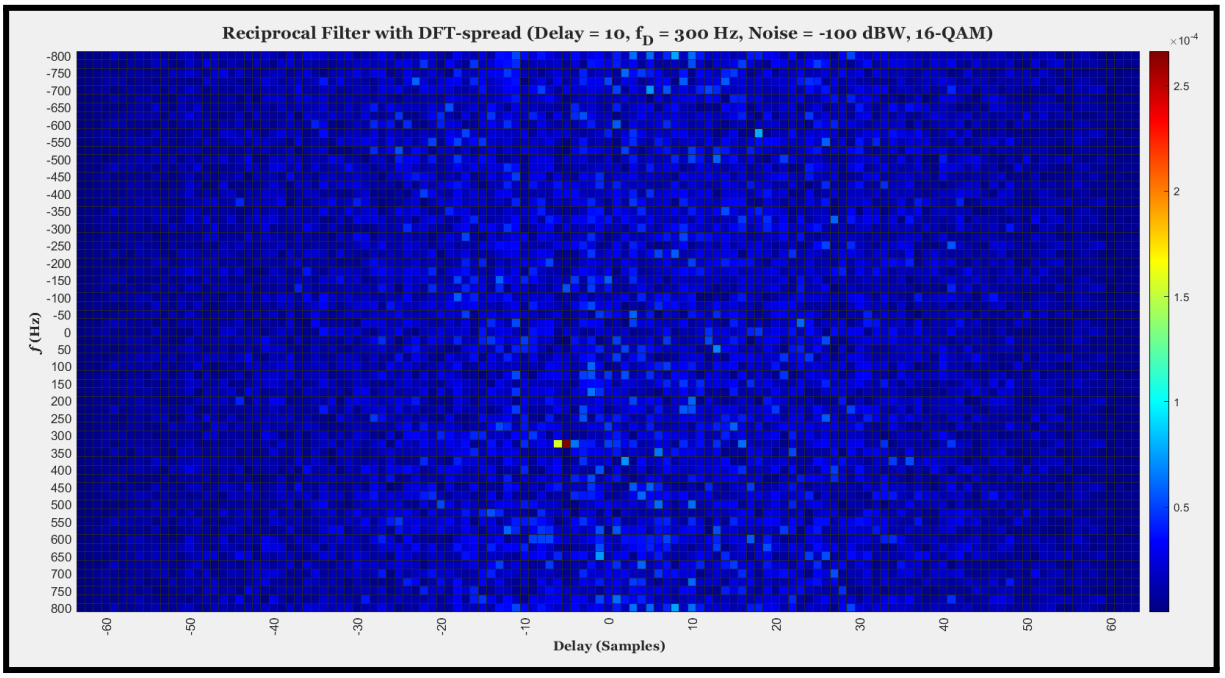
	1	2	3	4
121	0.0000 + 0.0000i	0.0000 + 0....	0.0000 + 0....	0.0000 + 0...
122	-2.8284 - 16.9218i	6.7584 - 10...	10.1229 - 1...	16.5429 - 2...
123	0.0000 + 0.0000i	0.0000 + 0....	0.0000 + 0....	0.0000 + 0...
124	30.9647 + 5.2456i	-13.5806 + ...	27.3633 + ...	20.8467 - 0...
125	0.0000 + 0.0000i	0.0000 + 0....	0.0000 + 0....	0.0000 + 0...
126	-3.1297 - 23.9755i	64.8785 + ...	-3.7211 - 2...	-8.2417 - 1...
127	0.0000 + 0.0000i	0.0000 + 0....	0.0000 + 0....	0.0000 + 0...
128	-16.7897 - 2.6007i	-9.4498 - 1...	-6.8560 - 1...	3.4818 - 6...
129	0.0000 + 0.0000i	0.0000 + 0....	0.0000 + 0....	0.0000 + 0...
130	-16.7897 + 2.6007i	-9.4498 + 1...	-6.8560 + 1...	3.4818 + 6...
131	0.0000 + 0.0000i	0.0000 + 0....	0.0000 + 0....	0.0000 + 0...
132	-3.1297 + 23.9755i	64.8785 - 2...	-3.7211 + 2...	-8.2417 + 1...
133	0.0000 + 0.0000i	0.0000 + 0....	0.0000 + 0....	0.0000 + 0...
134	30.9647 - 5.2456i	-13.5806 - ...	27.3633 - 1...	20.8467 + ...
135	0.0000 + 0.0000i	0.0000 + 0....	0.0000 + 0....	0.0000 + 0...
136	-2.8284 + 16.9218i	6.7584 + 1...	10.1229 + ...	16.5429 + ...

As can be seen, it has Hermitian symmetry. This is also confirmed by the fact that the when this matrix is passed into the IFFT block, the output is real-valued:

```
>> result = ifft(symbolMapOp);
```

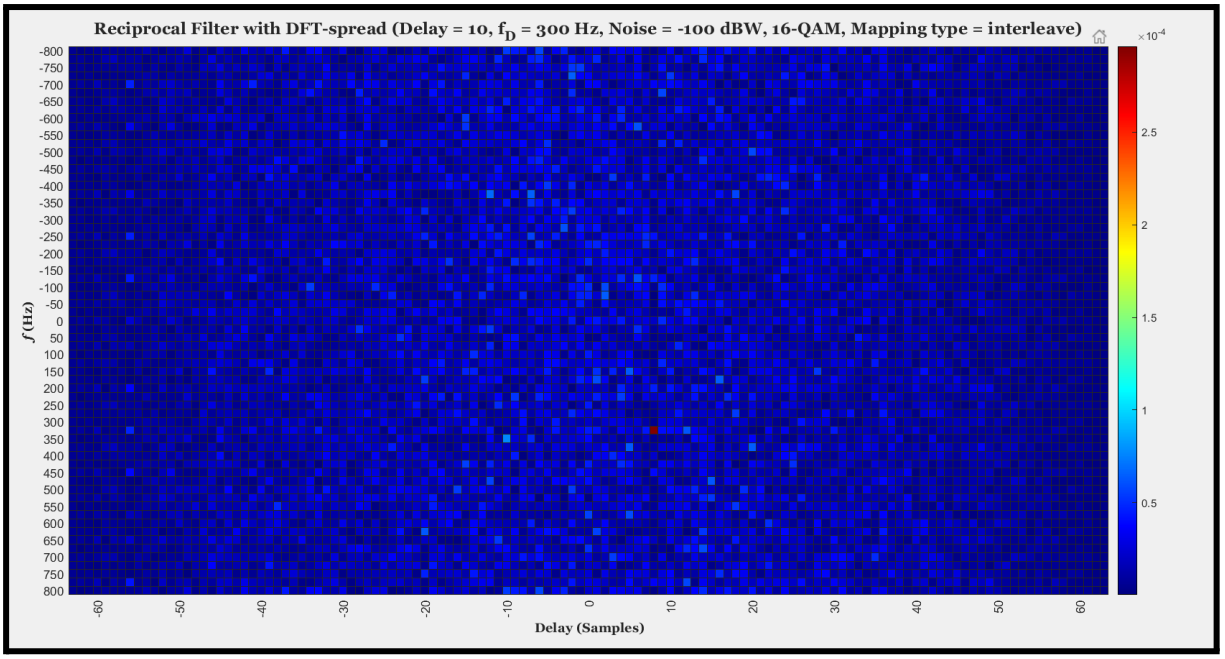
result 256x8 double

- The graph below is from version 2 of my DFT-spread OFDM radar program (DFT_SOFDM_radar_ver2.m) and it doesn't look a million miles off… although more testing is definitely needed as this is just the first graph that I have generated with this program:

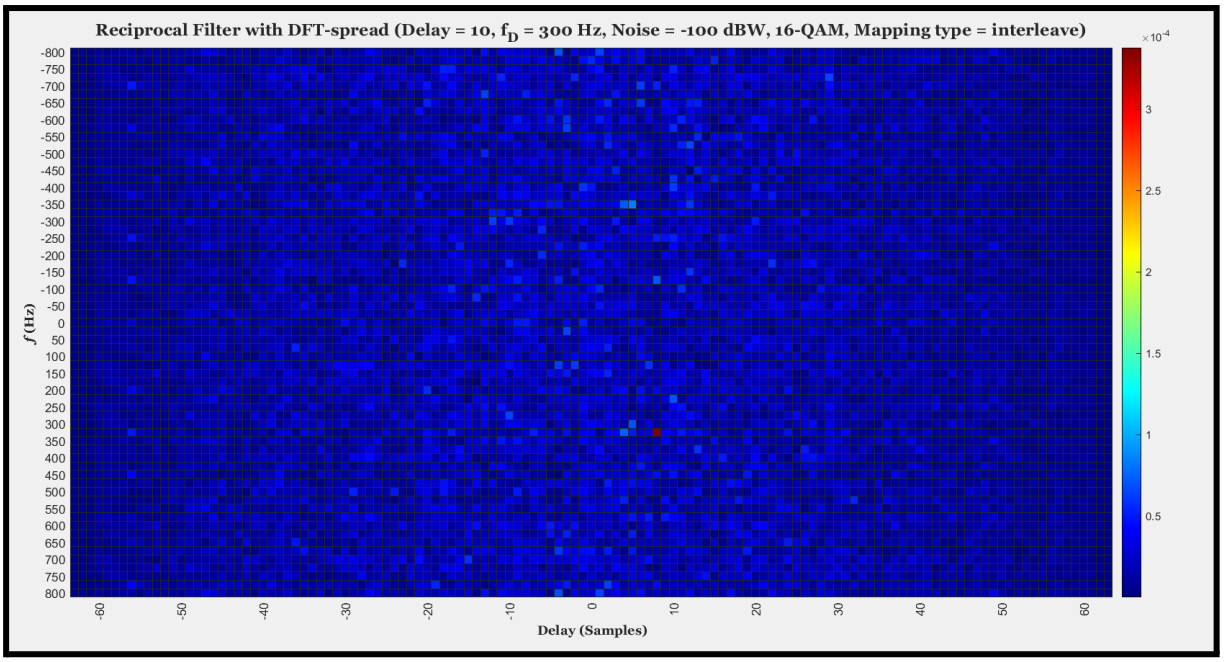



```
% Setup of variables and system parameters:
N = 64; % no. of complex symbols going into Tx-side DFT block
M = 5; % no. of blocks or symbols being used in OFDM frame
L = 4*N; % size of Tx-side IFFT block

% Info. on modulation:
modType = "QAM"; % type of modulation being used
modOrder = 16; % order of modulation

% Info. on mapping from DFT block to IFFT block (Tx side):
mapType = "interleave";
```

The bright spots lie along the line 325 Hz, 25 Hz off the actual Doppler shift value. Also, the above graph was obtained with a guard interval fraction of ⅛… changing it to ¼ I see:



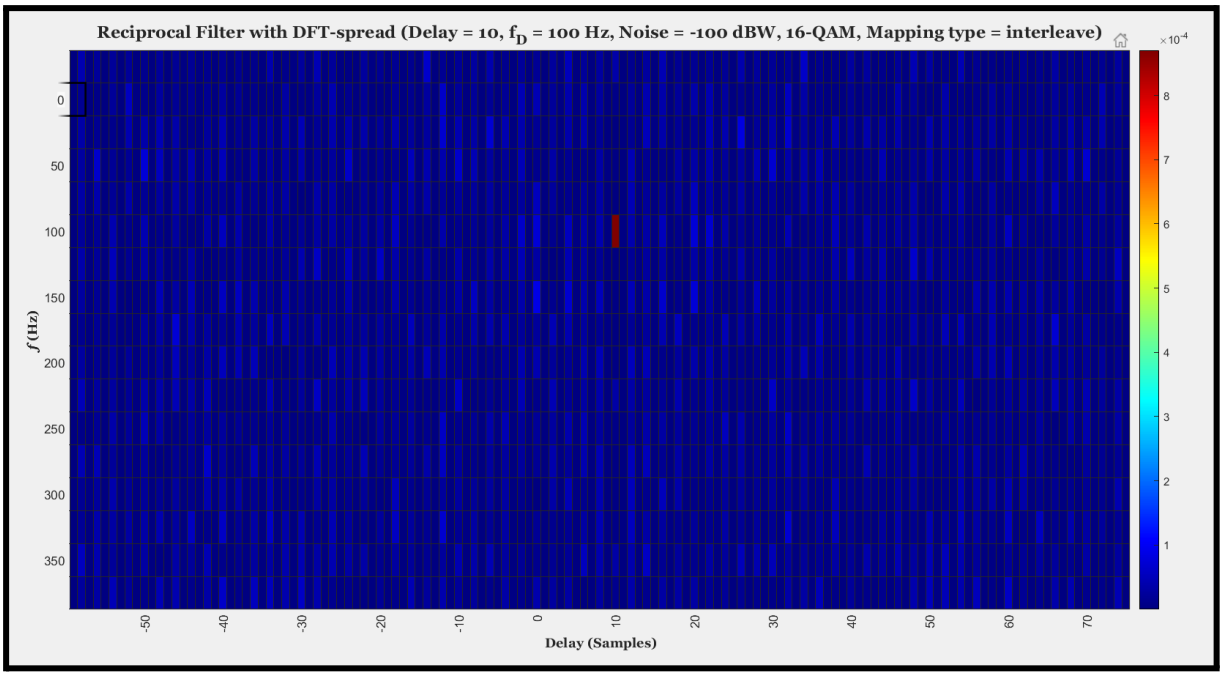
What looks to be a slightly better result…



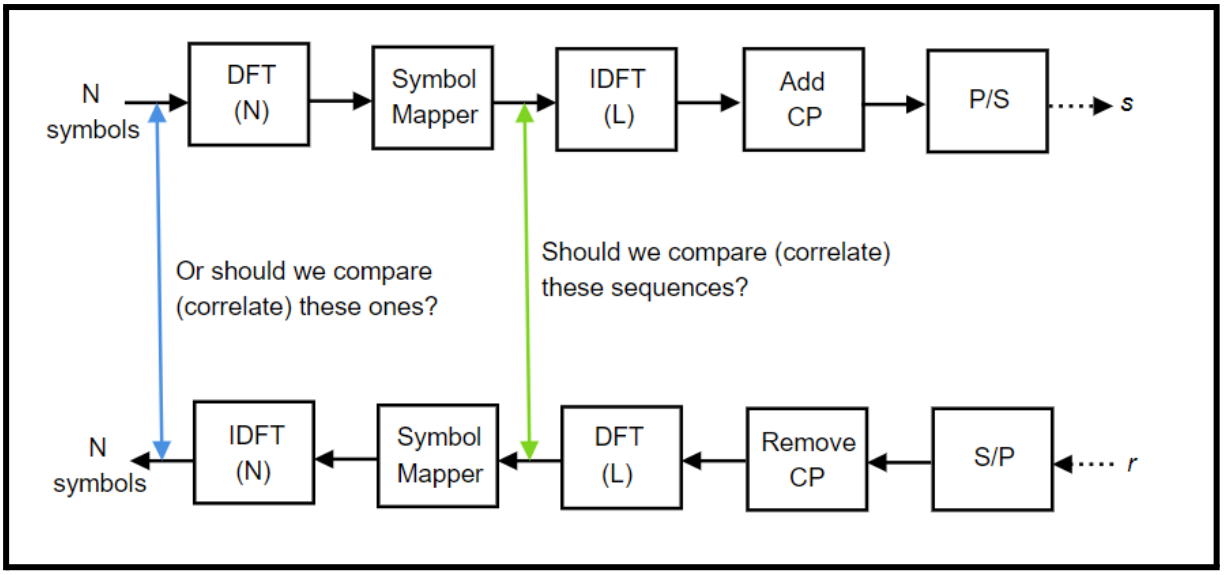
```
% Setup of variables and system parameters:
N = 64; % no. of complex symbols going into Tx-side DFT block
M = 7; % no. of blocks or symbols being used in OFDM frame
L = 4*N; % size of Tx-side IFFT block

% Info. on modulation:
modType = "QAM"; % type of modulation being used
modOrder = 16; % order of modulation

% Info. on mapping from DFT block to IFFT block (Tx side):
mapType = "interleave";
```

18/03/2022:

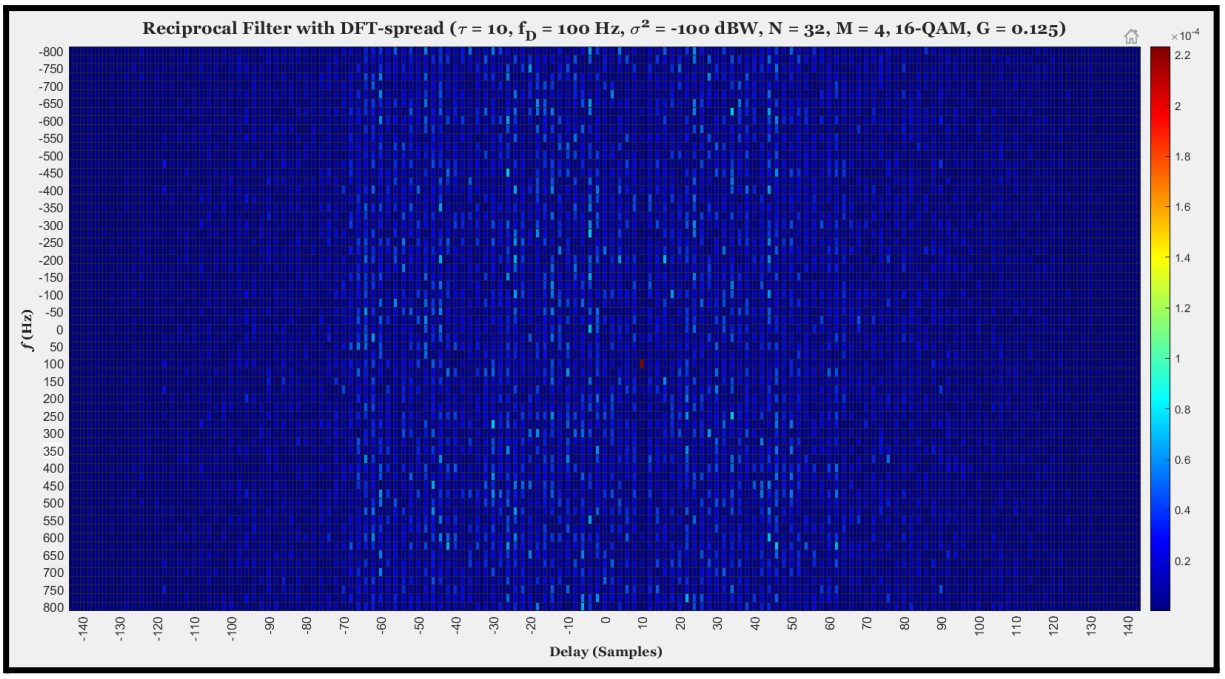
- I am doing some more work on trying to get good results by combining the DFT-spread technique with the reciprocal filtering approach…



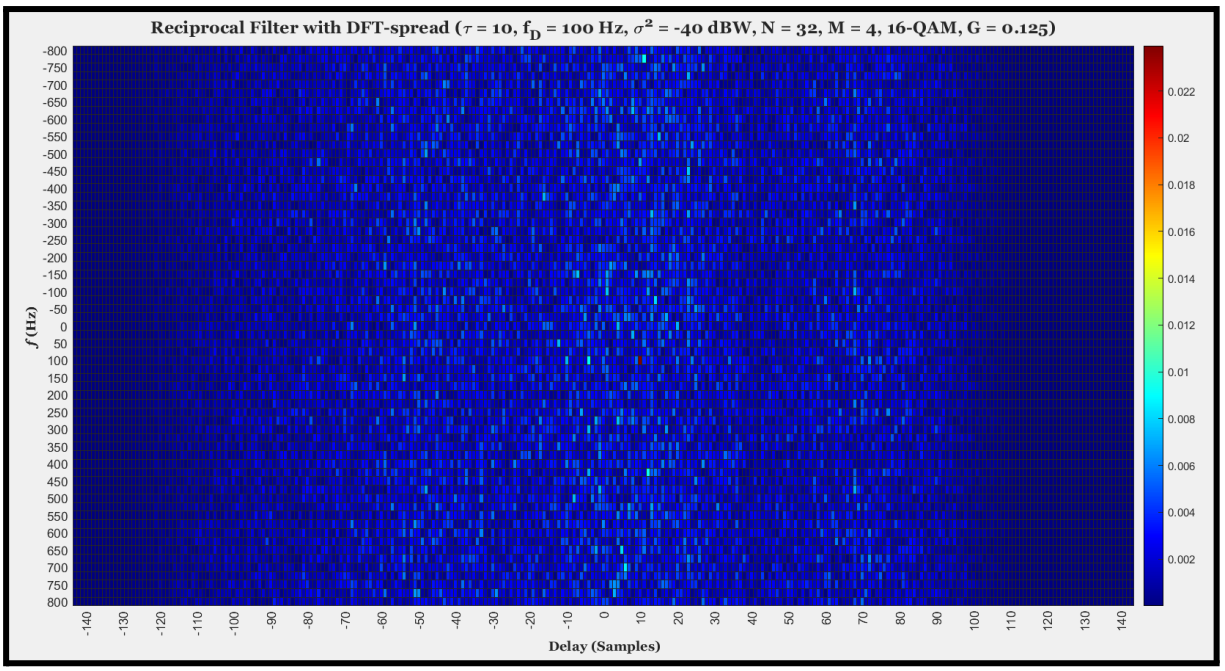
- I think I might experiment with **not removing** the CP… as I am at the moment, which means I am actually implementing the reciprocal filter slightly differently compared to before…
- I have created a program called DFT_SOFDM_radar_ver3.m. It computes the correlation using the sequence that goes into the IDFT on the Tx-side and the sequence that is leaving the DFT block on the Rx-side - shown below as the two sequences connected by the green arrow:



- Some graphs from this program are shown below:



*The method of mapping used between the Tx-side DFT and IDFT block was **interleaving***



*The method of mapping used between the Tx-side DFT and IDFT block was **localised***

- I want to begin to try and compare the radar performance with and without the DFT-spread technique (DFT precoding). First, I am reading up a little bit more on SNR as I want to model the noise in a better way.
- Shown below is some code from a MATLAB document page. It shows how an average (mean) of the magnitudes squared of the symbols is taken in order to obtain the power:

```
c = [-5 -5i 5 5i -3 -3-3i -3i 3-3i 3 3+3i 3i -3+3i -1 -1i 1 1i];
sigpower = pow2db(mean(abs(c).^2));
M = length(c);
```

The **pow2db()** function converts the power levels in the array to **dB**

- Based on the what I see below:

rxSig

2000x1 complex double

	1	2	3	4	5
1	1.2970 - 0.0445i				
2	0.1482 - 1.1709i				
3	-0.0766 + 0.8639i				
4	4.6072 + 0.0442i				
5	0.2375 - 0.9154i				
6	-0.8484 + 0.1747i				
7	-0.6490 - 0.0459i				
8	3.1363 - 2.8315i				
9	-0.0980 - 2.8935i				
10	0.1233 - 5.1095i				
11	3.0595 - 3.1227i				
12	4.6339 + 0.1796i				

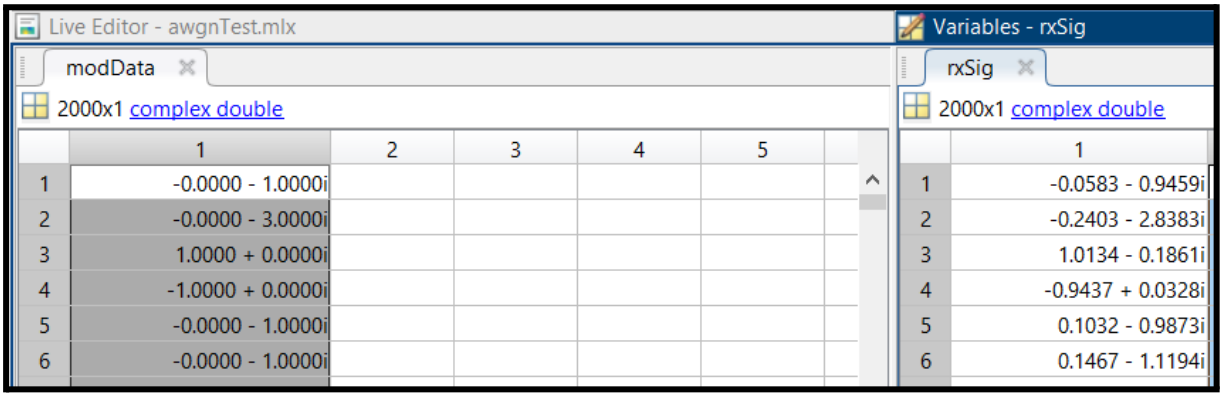
modData

2000x1 complex double

	1
1	1.0000 + 0.0000i
2	-0.0000 - 1.0000i
3	0.0000 + 1.0000i
4	5.0000 + 0.0000i
5	-0.0000 - 1.0000i
6	-1.0000 + 0.0000i
7	-1.0000 + 0.0000i
8	3.0000 - 3.0000i
9	-0.0000 - 3.0000i
10	-0.0000 - 5.0000i
11	3.0000 - 3.0000i
12	5.0000 + 0.0000i

It seems that if the **awgn()** function is given a complex sequence, it will **add noise** to **both components** (both the **real** and **imaginary** components)... I think I will use this function inside my OFDM radar scripts…

Below is the comparison when the SNR is **30 dB** (modData is the transmit sequence):



- To convert power from linear units to deciBels, the we can use:

  $P_{dB} = 10\log_{10}(P)$

```
>> clear
>> pow2db(10)

ans =

    10

>> 10*log10(10)

ans =

    10
```

You can see both the **pow2db(10)** function and **10*log10(10)** give the same answers

- Using the above formula, we can convert SNR from dB to linear units as follows:

  SNR = 20 dB

  $20 = 10\log_{10}(SNR)$

  Therefore SNR = 10^(20/10) = 100

20/03/2022:

- I am doing some more reading up on the benefits and implications of the **DFT-spread** technique. This is so that I am more comfortable writing and discussing why it is used in practice.
- SC FDMA: Single-carrier FDMA (Frequency-division Multiple Access); it deals with the **assignment** of **multiple** users to a **shared communication** resource.
- I am reading an answer given to a question on a DSP forum, and the respondent mentions how an OFDM signal can be viewed as a hybrid signal made up of **many single-carrier ones**. This has given me a slightly different way to look at the OFDM signal.
- It seems that using **less** of the subcarriers should lead to a lower **PAPR**, and power consumption. Likewise, if we want to transmit on, say, 64 subcarrier channels, but still reduce the PAPR, then the DFT-spread technique is a good one.
- I have been reading about **spread-spectrum** techniques, and the benefits of **spreading** a signal over a larger bandwidth.

21/03/2022:

- I have created a program which plots two graphs for the reciprocal filter: one corresponding to when DFT-spread is employed. And the other when it is **not employed**. This will hopefully allow for easier comparison.
- I have generated some graphs for comparison between the reciprocal filter when both DFT-spread is and **is not** used.
- Shown below are the PAPR values for a given setup when **localised mapping** is used:

And below are the results when **interleaving mapping** was chosen:

But it seems to fluctuate greatly (at least PAPR1). Here are values got from a different run of the program (again, interleaving mapping was used).

22/03/2022:

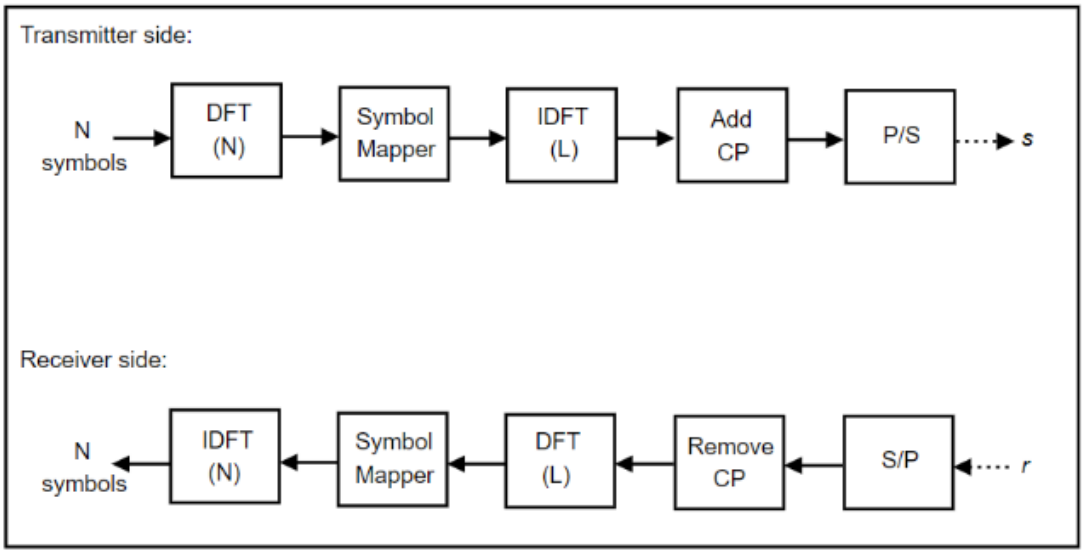
- Yesterday I was finishing a program to compare the results of the reciprocal OFDM radar filter when the DFT-spread technique is used versus when it is not used.
- I had a meeting with my supervisor today, and I have a better idea of what I should contain or add to the conference paper. I will first create a plan of what points I will make and in what order. I will also think about what diagrams I will need.
- I am just reading up on this page OFDM signal structure:
  Some points from my reading:
  (1) During a **burst**, an **entire** frame is transmitted. Then the carrier is left idle for a certain period of time.
  (2) The article wasn't quite short, and not entirely on the topic I needed to read about
- I have some graphs ready to add to the conference paper.

- Reciprocal filter also known as **MCC (Evaluation of the ambiguity function for passive radar with OFDM transmissions)** and **CHAD (A unifying approach for disturbance cancellation and target detection in passive radar using OFDM)**

23/03/2022:

- I have begun writing the first draft of the conference paper:

mitigate the effect of the IDFT block (OFDM modulator). The outputs from this DFT block are then mapped to the inputs of the IDFT block. Figure 1 shows a basic block diagram for an DFT-spread OFDM system:



And I have figured out how to include figures inside my paper… and scale them to the size needed:

```
\begin{figure}[htbp]
\includegraphics[width=\columnwidth,
scale=0.6]{DFT-spread-OFDM_basic_idea.PNG}
\caption{Basic block diagram of DFT-s-OFDM System.}
\label{fig1}
\end{figure}
```

***Abstract*—OFDM radar has become an active area of research in recent years. The possibility of combining a communication and radar system is promising for a few reasons. Firstly, the combined system would most likely entail less hardware. And secondly, it could allow for more efficient use of the frequency spectrum. This paper investigates the effect of using DFT-spread or DFT pre-coding on an OFDM radar's performance. DFT-spread is used to help reduce the PAPR of the OFDM waveform before transmission, thus reducing the demand placed on the high-power amplifier (HPA). The OFDM radar receiver is implemented as a reciprocal filter in this paper.**

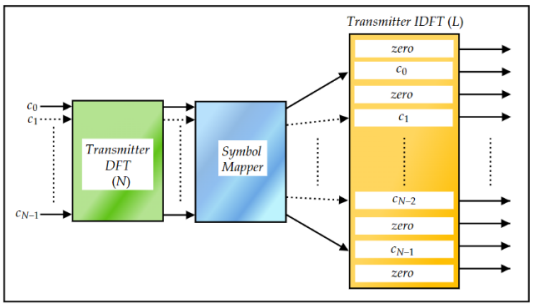
***Index Terms*—component, formatting, style, styling, insert**

I. Background

A. DFT-spread Technique

DFT-spread: One of the main drawbacks of the OFDM waveform is that it has a high peak-to-average power (PAPR) ratio. This can cause unwanted distortions at the output of the transmitter's high-power amplifier (HPA), thereby leading to an increase in the communication system's bit error rate

This means one has some choice when mapping the N DFT outputs to the L IDFT inputs. Two commonly employed mapping rules are interleaving and localised (give some references). A diagram showing the basic characteristic of interleaving and localised mapping can be seen in figure 2 below:



- I am able to slowly write the equations up.

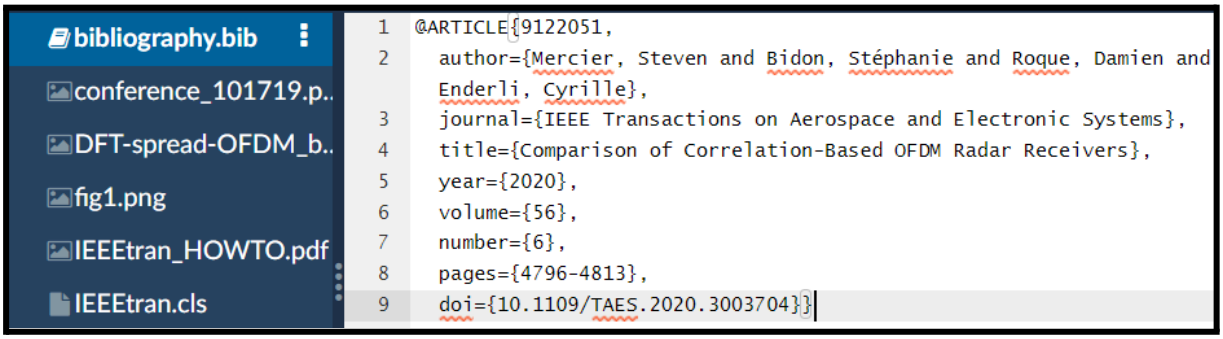
$$s[l] = \sum_{m=0}^{M-1} \left(\frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} c_{n,m} e^{-j2\pi \frac{n}{N}(l-mP)} \right)$$

- I found that using **\begin{align}** and **\end{align}** a good way of getting the above equation to have a number beside it:

```
56 \begin{align}
57 s[l] = \sum_{m=0}^{M-1}\left(\frac{1}{\sqrt{N}}\sum_{n=0}^{N-1}c_{n,m
   }e^{-j2\pi \frac{n}{N}(l-mP)}\right)
58 \end{align}
```

$$s[l] = \sum_{m=0}^{M-1} \left(\frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} c_{n,m} e^{-j2\pi \frac{n}{N}(l-mP)} \right) \quad (1)$$

- I am learning how to use BibTex to make citing other people's work easier:



- It seems that a separate file is needed, and all the relevant information for the citation is kept in here. This separate file (bibliography.bib in my case) is then referenced by the main.tex file.
- I think that I have made my first reference :)

> *C. Reciprocal Filter*
>
> Correlation-based OFDM radar receivers have analysed and discussed before, such as in [1]

- It seems that the IEEE people have preferences on the use of some commands over others:

> *E. LaTeX-Specific Advice*
>
> Please use "soft" (e.g., `\eqref{Eq}`) cross references instead of "hard" references (e.g., `(1)`). That will make it possible to combine sections, add equations, or change the order of figures or citations without having to go through the file line by line.
>
> Please don't use the `{eqnarray}` equation environment. Use `{align}` or `{IEEEeqnarray}` instead. The `{eqnarray}` environment leaves unsightly spaces around relation symbols.
>
> Please note that the `{subequations}` environment in LaTeX will increment the main equation counter even when there are no equation numbers displayed. If you forget that, you might write an article in which the equation numbers skip from (17) to (20), causing the copy editors to wonder if you've discovered a new method of counting.

- I am about 60-70% of the way to being doing the first draft of the conference paper. I am able to create basic tables in LaTeX!

TABLE I
SIMULATION PARAMETERS

Number of subcarriers (N)	16, 32, 64
Number of OFDM symbols (M)	7, 10, 14
Modulation scheme	16-QAM
Noise power (σ^2)	-25, -40 dBW
Fraction of symbol used as CP (G)	0, 1/8
Size of IDFT block (L)	2N, 4N

- I resolved a little issue I was having where an image that I had added did not appear in the correct position:

```
\begin{figure}[htbp]
\includegraphics[width=\columnwidth,
scale=0.6]{RECIPR~1.PNG}
\caption{Reciprocal filter radar receiver with
DFT-precoding on the transmit side.}
\label{fig7}
\end{figure}
```

A. Interleaving Mapping

REFERENCES

[1] Steven Mercier, Stéphanie Bidon, Damien Roque, and Cyrille Enderli. Comparison of correlation-based ofdm radar receivers. *IEEE Transactions on Aerospace and Electronic Systems*, 56(6):4796–4813, 2020.

[2] Stephen Searle, James Palmer, Linda Davis, Daniel W. O'Hagan, and Martin Ummenhofer. Evaluation of the ambiguity function for passive radar with ofdm transmissions. In *2014 IEEE Radar Conference*, pages 1040–1045, 2014.

[3] Ghislain Gassier, Gilles Chabriel, Jean Barrère, Françoise Briolle, and Claude Jauffret. A unifying approach for disturbance cancellation and target detection in passive radar using ofdm. *IEEE Transactions on Signal Processing*, 64(22):5959–5971, 2016.

[4] Klaus Martin Braun. *OFDM radar algorithms in mobile communication networks*. PhD thesis, Karlsruhe, Karlsruher Institut für Technologie (KIT), Diss., 2014, 2014.

[5] Martin Braun, Christian Sturm, and Friedrich K. Jondral. Maximum likelihood speed and distance estimation for ofdm radar. In *2010 IEEE Radar Conference*, pages 256–261, 2010.

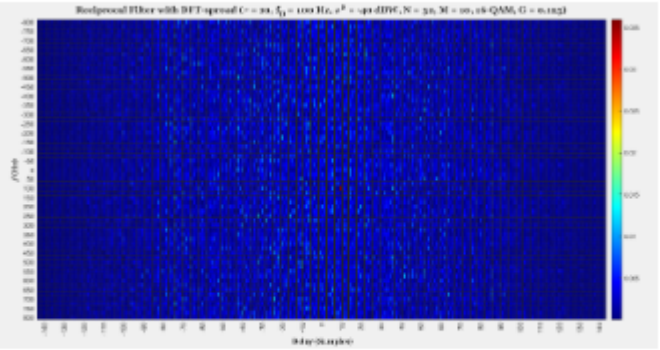


Fig. 7. Reciprocal filter radar receiver with DFT-precoding on the transmit side.

Using what is shown below helped me to resolve the issue

```
\begin{figure}[ht!]
```

25/03/2022:

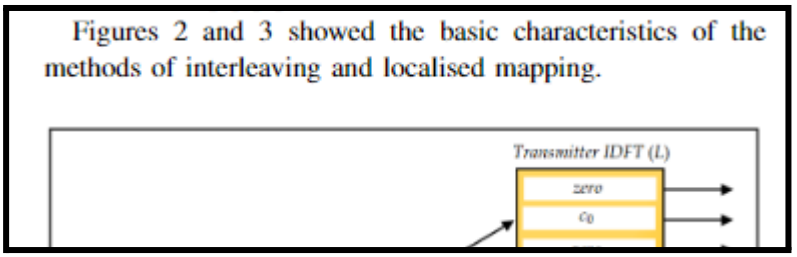
- I have been working on writing up the conference paper for the past 2 days

26/03/2022:

- I am currently writing up the final version of the conference paper.

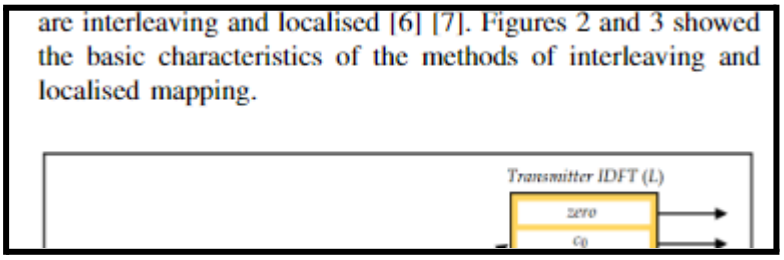
- I am getting a little bit more familiar with LaTeX and Overleaf. I wanted to space out my code, so as to be able to read it better and add comments, but when I had blank lines between text, LaTeX was interpreting them as new paragraphs.

```
DFT outputs to the $L$ IDFT inputs. Two mapping rules often used are interleaving and
localised~\cite{article}~\cite{8877195}.

Figures~\ref{interleaving_mapping} and \ref{localised_mapping} showed the basic
characteristics of the methods of interleaving and localised mapping.
```



It turns out that using a **%** at the start of the blank line will avoid LaTeX doing this!

```
localised~\cite{article}~\cite{8877195}.
%
Figures~\ref{interleaving_mapping} and \ref{localised_mapping} showed the basic
characteristics of the methods of interleaving and localised mapping.
```



- I think I am also getting better at creating and referencing figures in LaTeX:

```
Figures~\ref{interleaving_mapping} and \ref{localised_mapping} showed the basic
characteristics of the methods of interleaving and localised mapping.
%
% Let us add the next figures for interleaving and localised mapping
```

```
% Let us add the next figures for interleaving and localised mapping
\begin{figure}[htbp!]
    \begin{center}
        \includegraphics[width=\columnwidth, scale=0.6]{interleaving_mapping_basic_idea.PNG}
    \end{center}
    \caption{Illustration of the idea of Interleaving Mapping.}
    \label{interleaving_mapping}
\end{figure}
```

- I think I am getting better at using the \ref{} command

```
example in figure \ref{interleaving_mapping} above,
the symbols.
```

```
% Let us describe a received frame
\begin{align}\label{Frx}
\mathbf{F_{Rx}} =
\begin{pmatrix}
b_{0,0} & b_{0,1} & \cdots & b_{0,M-1}\\
b_{1,0} & b_{1,1} & \cdots & b_{1,M-1}\\
\vdots & \vdots & \ddots & \vdots\\
b_{N,0} & b_{N,1} & \cdots & b_{N,M-1}
\end{pmatrix}
\end{align}
%
With regards to equation (\ref{Frx}), $b_{1,1}$ would
```

- I found a way to frame my MATLAB simulation plots:

```
% Let us give this example heat map graph now
\begin{figure}[htbp!]
    \begin{center}
        \frame{\includegraphics[width=\columnwidth,
        scale=0.6]{2D-correlation_receiver_no1.PNG}}
```

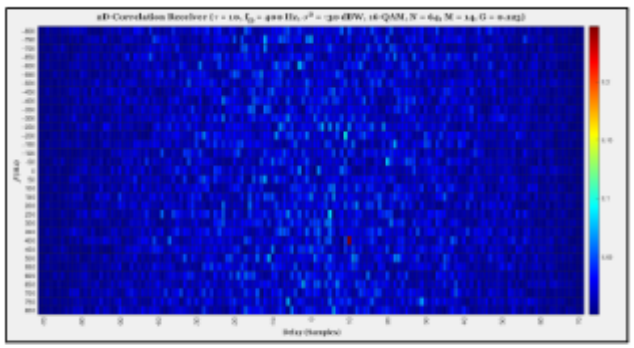


Fig. 5: Heat map generated from the correlation-based receiver defined by equations (8) and (10)

So far, the correlation-based radar receiver discussed in

Cool! :)

- My references are coming out much better now, and are appearing at the end of the document! :)

References

[1] Khaled Tani, Yahia Medjahdi, Hmaied Shaiek, Rafik Zayani, and Daniel Roviras. Papr reduction of post-ofdm waveforms contenders for 5g amp; beyond using slm and tr algorithms. In *2018 25th International Conference on Telecommunications (ICT)*, pages 104–109, 2018.

[2] N. Ohkubo and T. Ohtsuki. Design criteria for phase sequences in selected mapping. In *The 57th IEEE Semiannual Vehicular Technology Conference, 2003. VTC 2003-Spring.*, volume 1, pages 373–377 vol.1, 2003.

[3] Filippo Tosato, Magnus Sandell, and Makoto Tanahashi. Tone reservation for papr reduction: An optimal approach through sphere encoding. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6, 2016.

[4] M. F. Pervej, M. Z. I. Sarkar, T. K. Roy, M. M. Hasan, M. M. Rahman, and S. K. Bain. Analysis of papr reduction of dft-scfdma system using different sub-carrier mapping schemes. In *2014 17th International Conference on Computer and Information Technology (ICCIT)*, pages 435–439, 2014.

[5] Rizwana Ahmad and Anand Srivastava. Papr reduction of ofdm signal through dft precoding and gmsk pulse shaping in indoor vlc. *IEEE Access*, 8:122092–122103, 2020.

[6] Zi-Yang Wu, Gao Yu-Liang, Ze-Kun Wang, Chuan You, Chuang Yang, Chong Luo, and Jiao Wang. Optimized dft-spread ofdm based visible light communications with multiple lighting sources. *Optics Express*, 25:26468, 10 2017.

[7] Khaled Tahkoubit, Adda Ali-Pacha, Hmaied Shaiek, and Daniel Roviras. Papr reduction of bf-ofdm waveform using dft-spread technique. In *2019 16th International Symposium on Wireless Communication Systems (ISWCS)*, pages 406–410, 2019.

[8] Steven Mercier, Stéphanie Bidon, Damien Roque, and Cyrille Enderli. Comparison of correlation-based ofdm radar receivers. *IEEE Transactions on Aerospace and Electronic Systems*, 56(6):4796–4813, 2020.

[9] Stephen Searle, James Palmer, Linda Davis, Daniel W. O'Hagan, and Martin Ummenhofer. Evaluation of the ambiguity function for passive radar with ofdm transmissions. In *2014 IEEE Radar Conference*, pages 1040–1045, 2014.

[10] Ghislain Gassier, Gilles Chabriel, Jean Barrère, Françoise Briolle, and Claude Jauffret. A unifying approach for disturbance cancellation and target detection in passive radar using ofdm. *IEEE Transactions on Signal Processing*, 64(22):5959–5971, 2016.

[11] Klaus Martin Braun. *OFDM radar algorithms in mobile communication networks*. PhD thesis, Karlsruhe, Karlsruher Institut für Technologie (KIT), Diss., 2014, 2014.

[12] Martin Braun, Christian Sturm, and Friedrich K. Jondral. Maximum likelihood speed and distance estimation for ofdm radar. In *2010 IEEE Radar Conference*, pages 256–261, 2010.

31/03/2022:

- It was nice to have submitted the conference paper last Sunday evening.
- I am now thinking about starting to write the initial part of the thesis, the general parts.
- I will keep a structured plan of key points in a separate document
- I think I will need to generate more results. I might need to investigate the effectiveness of the DFT-spread technique with the periodogram-based approach… or something along these lines - I might try the idea of using the same symbol on each of the subcarriers again…
- I had a meeting with my supervisor. Some very good points to keep in mind:
    (1) I must place more focus on the **trade-off** between the **radar** and **communications** functions. Key parameters will be **channel capacity**, **data rate**, **false-alarm** rate, **radar resolution** etc.
    (2) In the beginning of the thesis, I should make **a great effort** to **motivate** the project; **why** is this something worth considering.
    (3) I can also discuss the history of wireless communications and OFDM in the introduction. As well as radar…
    (4) Then comes the motivation for **why combining** these two functions into a single system would be good; why is it a natural question
    (5) Background: give a background on OFDM, why is it used, where is it used; discuss radar algorithms for detecting targets, and the radar's KPIs

(6) Literature review: what has been done thus far? Don't spread yourself too thin. Pick a few resources to drill into for a somewhat detailed discussion - other resources can be referenced very briefly…
(7) Discuss the work you have undertaken so far…
(8) Give solid, and insightful results and comments…

01/04/2022:

- I am generating some results, and considering the impact of using the OFDM system for radar on the **communications** aspect; how is the bit rate affected?; how is the spectral efficiency affected?; how is the resolution affected?
- 802.11a uses the 5 GHz band; 802.11g uses the 2.4 GHz band; pilot tones seems to **always use BPSK**, according to this very useful page that I am reading [OFDM_802.11standards_details]

Data Subcarriers	48
Pilot Subcarriers*	4 (-21, -7, +7, +21) *Always BPSK
DC Subcarrier	Null (0 subcarrier)

- The Advances in Wireless Networking is now proving more useful!

Modulation	NBits	CRate	NChan	SDur (micro sec)	Data Rate (Mbps)
BPSK	1	1/2	48	4	6
BPSK	1	3/4	48	4	9
QPSK	2	1/2	48	4	12
QPSK	2	3/4	48	4	18
16-QAM	4	1/2	48	4	24

- I am adding code to my DFT_SOFDM radar script so that I can compute the data rate. I am also being more precise, and allow the 802.11 standard to be specified:

```
% Let us define the OFDM system parameters based on the IEEE 802 standard
% in use:
switch (IEEEstandard)
    case "802.11a"
        To = 4e-6; % the OFDM symbol duration (period)
        fC = 5.5e9; % 802.11a uses the 5 GHz band
        N = 52; % 48 for data, and 4 pilot tones
        numPilotTones = 4; % no. pilot tones used
        M = 7; % no. of symbols in a frame or burst
        G = 1/4; % the guard interval fraction for 802.11a
        subcarrierSpacing = 312.5e3; % subcarrier spacing for 802.11a
    case "802.11g"
        To = 4e-6; % the OFDM symbol duration (period)
        fC = 2.4e9; % 802.11g uses the 2.4 GHz band
        N = 52; % 48 for data, and 4 pilot tones
```

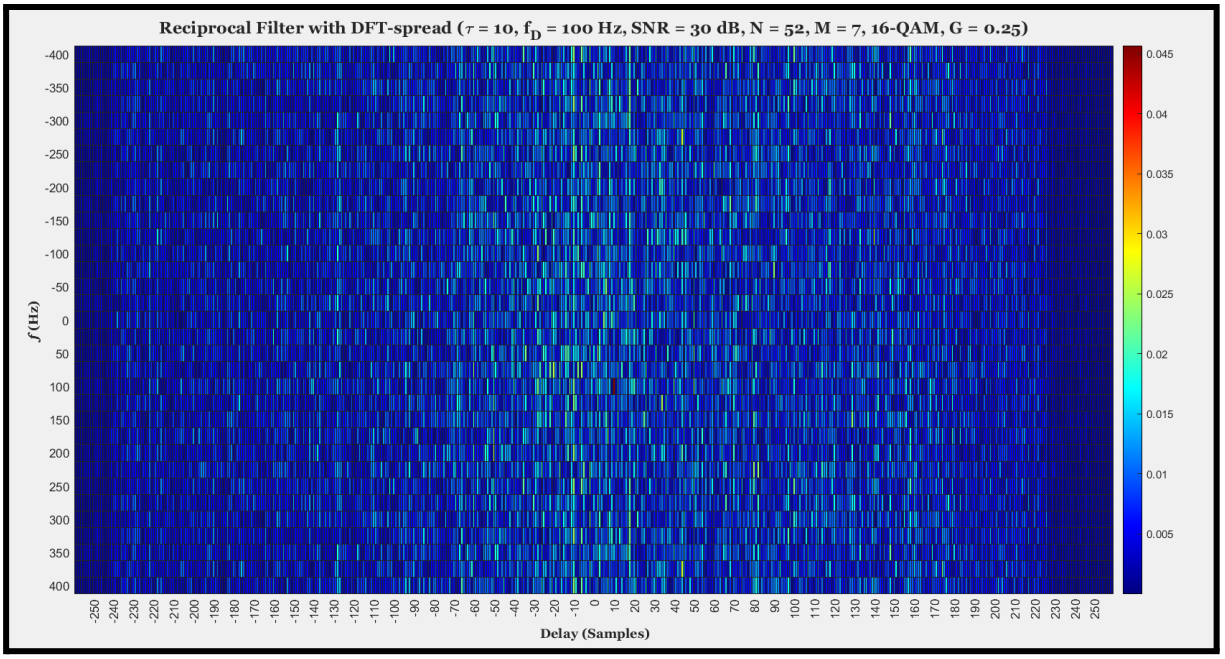
```
bitsPerComplexSymbol = log2(modOrder); % no. bits per complex symbol
bitsPerOFDMsymbol = (N-numPilotTones)*bitsPerComplexSymbol*codeRate;
dataRate = (bitsPerOFDMsymbol/To)/10^6 % data rate in Mbits/s
```

```
ans =

    "The data rate is 24 Mbits/s"
```

The answer that I got above seems to match with what many websites and online resources are giving. It also corresponds to what the tutorial notes in advances in wireless networking say for 16-QAM being used with 802.11a (24 Mbits/s)

Modulation	NBits	CRate	NChan	SDur (micro sec)	Data Rate (Mbps)
BPSK	1	1/2	48	4	6
BPSK	1	3/4	48	4	9
QPSK	2	1/2	48	4	12
QPSK	2	3/4	48	4	18
16-QAM	4	1/2	48	4	24

- I have also switched over to using specifying and displaying the SNR of the channel rather than the noise power:



- It seems that the subcarrier spacing of 312.5 kHz comes from a bandwidth allocation of 20 MHz, for **64** (the size of Tx-side FFT) subcarriers.

  20 MHz/64 = 0.3125 MHz
- Using **%f** is a very nice way to get MATLAB to use display certain values in an easier to read way:

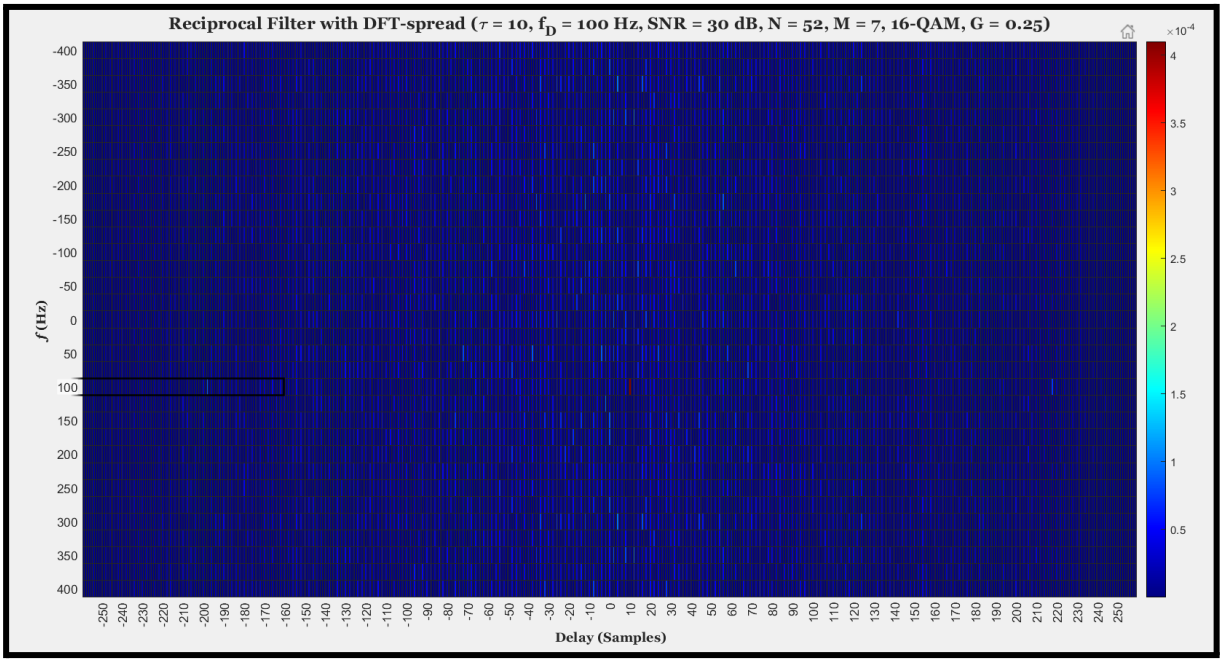
```
fprintf("The bandwidth used by the standard system is %f MHz.\n", bandwidth1)
fprintf("The bandwidth used by the DFT-S OFDM system is %f MHz.\n", bandwidth2)
fprintf("The data rate is %d Mbits/s.\n", dataRate)
```

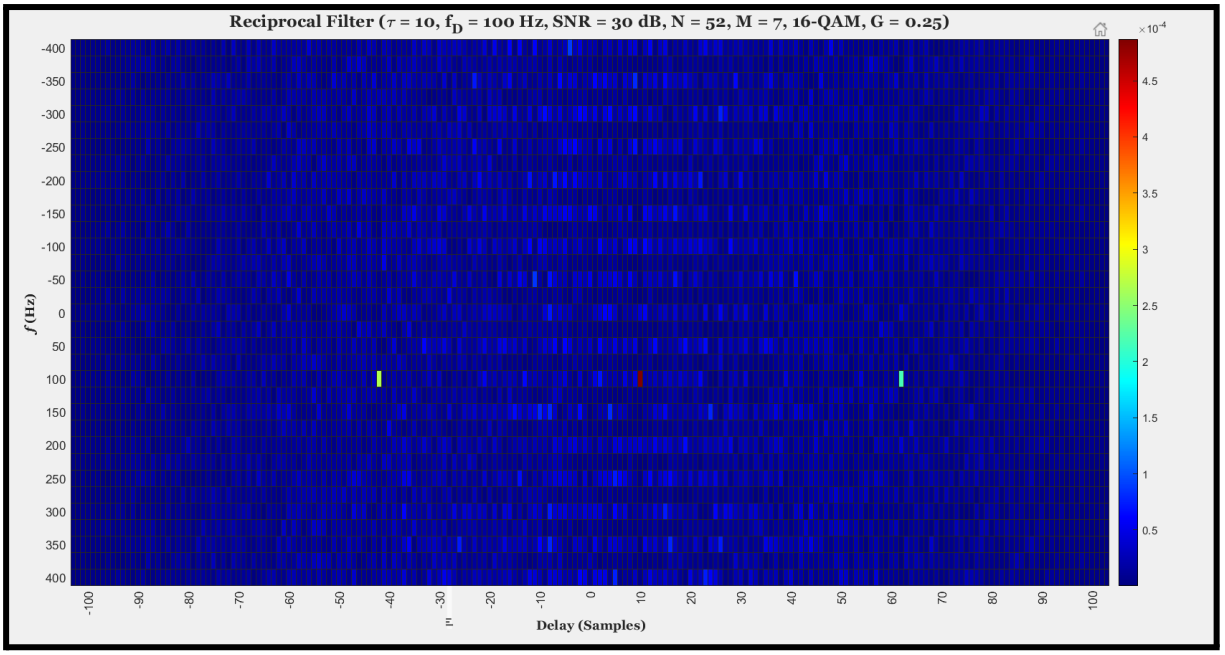
```
>> DFT_SOFDM_radar_ver6
The bandwidth used by the standard system is 16.250000 MHz.
The bandwidth used by the DFT-S OFDM system is 65.000000 MHz.
The data rate is 24 Mbits/s.
```

I think the above is easier to read when compared to:

```
>> DFT_SOFDM_radar_ver6
The bandwidth used by the standard system is 1.625000e+01 MHz.
The bandwidth used by the DFT-S OFDM system is 65 MHz.
The data rate is 24 Mbits/s.
```

- I think working with SNR is better. I am able to change *a*, the attenuation factor, and the results are still good. I think this is because *a* scales both the signal and noise, meaning the ratio is not affected that much. However, a very small value of *a* means the receiver must be more sensitive… (have a greater dynamic range)

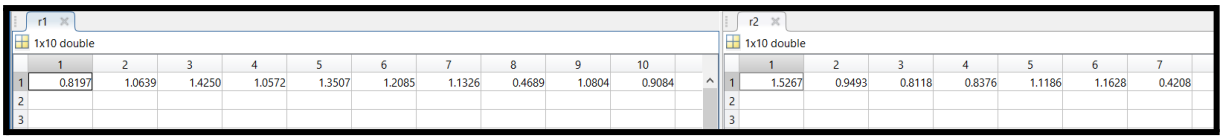




- I am doing **more investigation** into exactly how the **awgn() function** is behaving in MATLAB:
- Consider the following piece of test code:

```
s = ones(1, 10);
sigPower = pow2db(mean(abs(s).^2))
snr = 10;
r1 = awgn(s, snr, sigPower);
r2 = awgn(s, snr, 'measured');
```

The signal s has a power of 1 W, on average. This is equivalent to 0 dB (sigPower). The question is, are the two function calls used to define r1 and r2 the same?



Looking at them it seems like they are…
I can also work out the mean of each sequence:

```
Command Window
>> mean(r1)

ans =

    1.0515

>> mean(r2)

ans =

    1.0912
```

These mean values are more or less the same…

Also, since my signal power (sigPower) is 0 dB, and I have set the **SNR** to be **10**. This means the noise power should be **-10 dB**. The AWGN channel model is:

r = s + n

Which means

n = r - s

Let us define this vector in MATLAB:

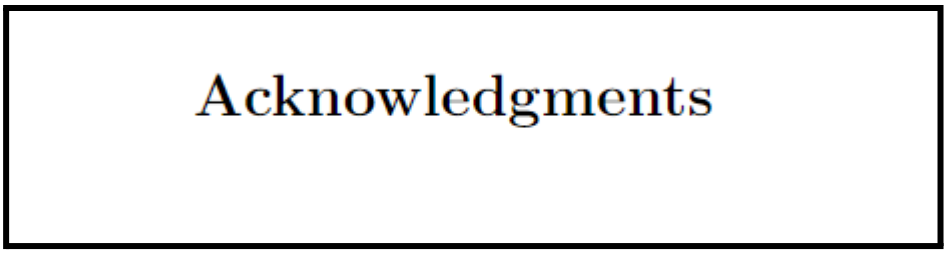
```
% Since the signal power (sigPower) is 0 dB, and the SNR is 10 dB, the
% noise power must be around -10 dB. Also, the AWGN model can be given by:
%
% r = s + n
%
% This means
%
% n = r - s; let us compute this vector
n = r1 - s;
pow2db(mean(abs(n).^2)) % calculating the power of the noise power
```

```
ans = -10.0104
```

02/04/2022:

- I am learning how to create separate files (for different chapters or sections of the thesis) and then how to include them in the main.tex file:

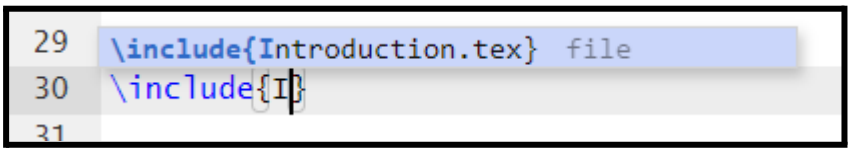
- Also, using \chapter is very handy!




```
\chapter*{Acknowledgments}
```

The "*" stops Acknowledgements from being numbered…

- Overleaf even suggests certain autocompletions :)!



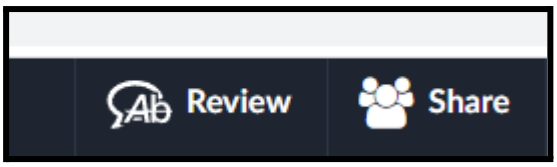
- The \includetableofcontents is another very nice command it seems:

```
\tableofcontents

\include{Introduction}
```

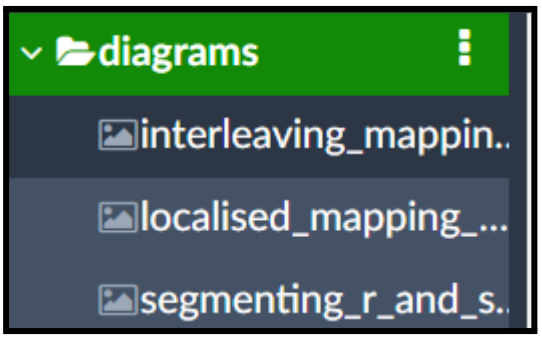
- \appendix is useful for separating the main sections from the appendix items:

```
\include{Introduction}
\include{Conclusions}
\appendix
\include{Appendix}
```

- I will be able to share my document with my supervisor using the share facility (hopefully):

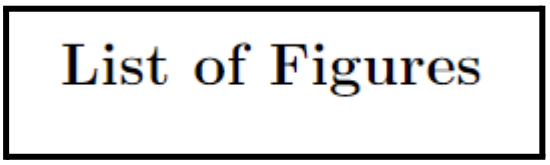


- I can create a separate folder for diagrams!



- I am going to separate diagrams (illustrations made by me) from plots (generated in MATLAB).
- You can also add a list of figures:

```
\tableofcontents
\listoffigures
```



- Footnotes; these seem to be those little numbers above a certain section of text that indicate that the reader should consult the bottom of the page or section for further details.

05/04/2022:

- Yesterday I was doing some of the writing of the thesis. Today I am trying to better understand the role of the cyclic prefix in the radar performance. I am editing some existing MATLAB programs and seeing the effect of different changes. For example, of including or removing the cyclic prefix.

15/04/2022:

- The FloatBarrier command from the placeins package is brilliant!

```
% cyclic prefix needs to be long enough
\begin{figure}[htbp!]
    \begin{center}
        \includegraphics[width=\columnwidth,
        scale=0.6]{diagrams/cyclic_prefix_not_long_enough.PNG}
    \end{center}
    \caption{CP is not sufficiently long.}
    \label{cp_not_long_enough}
\end{figure}
\FloatBarrier
%
```

It is helping the images to appear more coherently on the document!

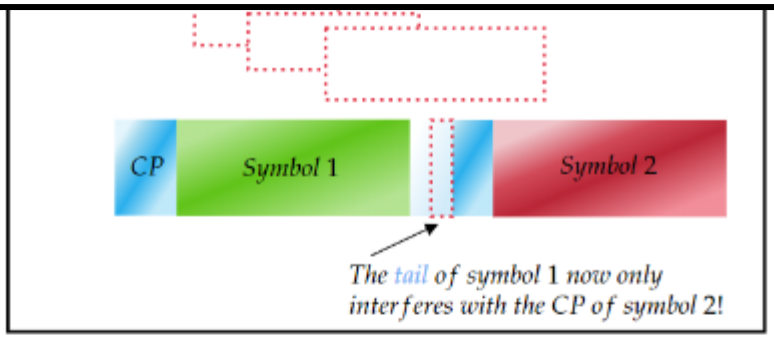


Fig. 2.6. The CP guards the payload data of symbol 2.

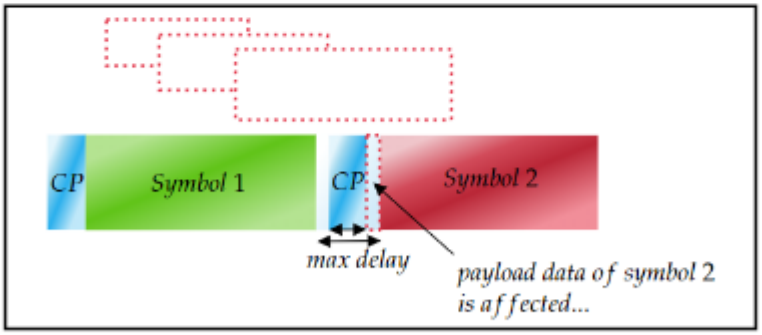


Fig. 2.7. CP is not sufficiently long.

Pilot Tones and Frequency-domain Equalisation

As opposed to something like:

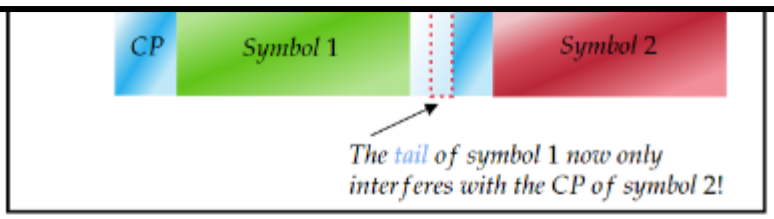


Fig. 2.6. The CP guards the payload data of symbol 2.

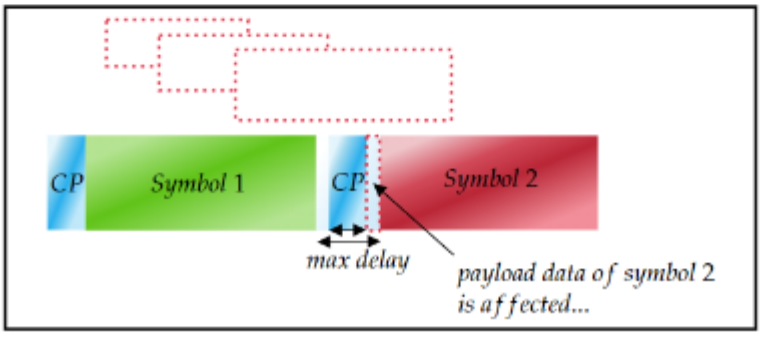
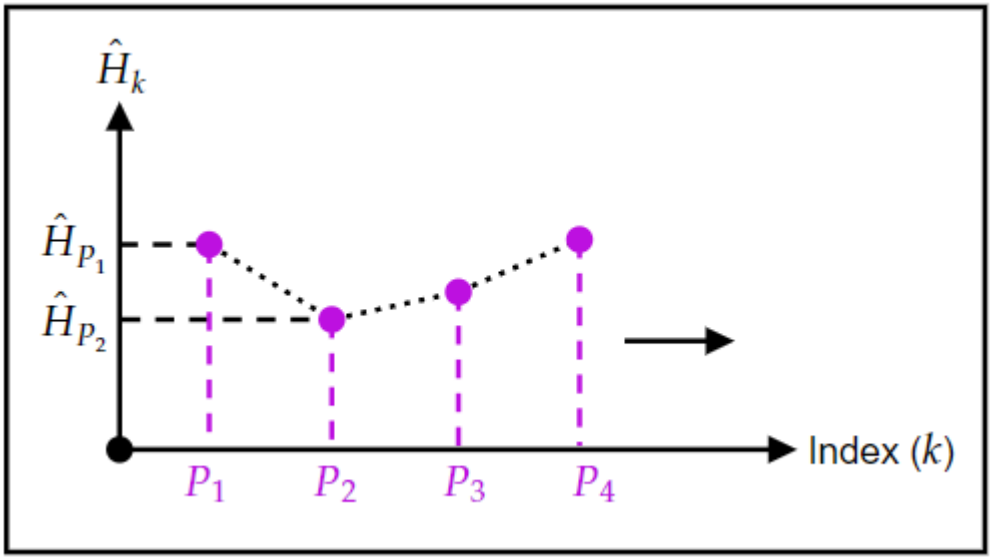


Fig. 2.7. CP is not sufficiently long.

- I am quite happy with this diagram below:



16/04/2022:

- I only yesterday learned that you can give your references more meaningful names in the ref.bib file - this will make it much easier when figuring out which reference I want to cite at a given moment. I have already started adding some names which are more explanatory (when compared to strings of numbers)

```
@INPROCEEDINGS{A_seminal_work_JCAS,
author={Lellouch, G. and Nikookar, H.},
booktitle={2007 14th IEEE Symposium on Communicat
Vehicular Technology in the Benelux},
title={On the Capability of a Radar Network to Su
Communications},
year={2007},
volume={},
number={},
pages={1-5},
doi={10.1109/SCVT.2007.4436249}}

@INPROCEEDINGS{Sturm_first_true_JCAS,
```

18/04/2022:

- The FloatBarrier command is very useful again:

```
\end{figure}
\FloatBarrier
If the symbol $Y_{k}$ is
```

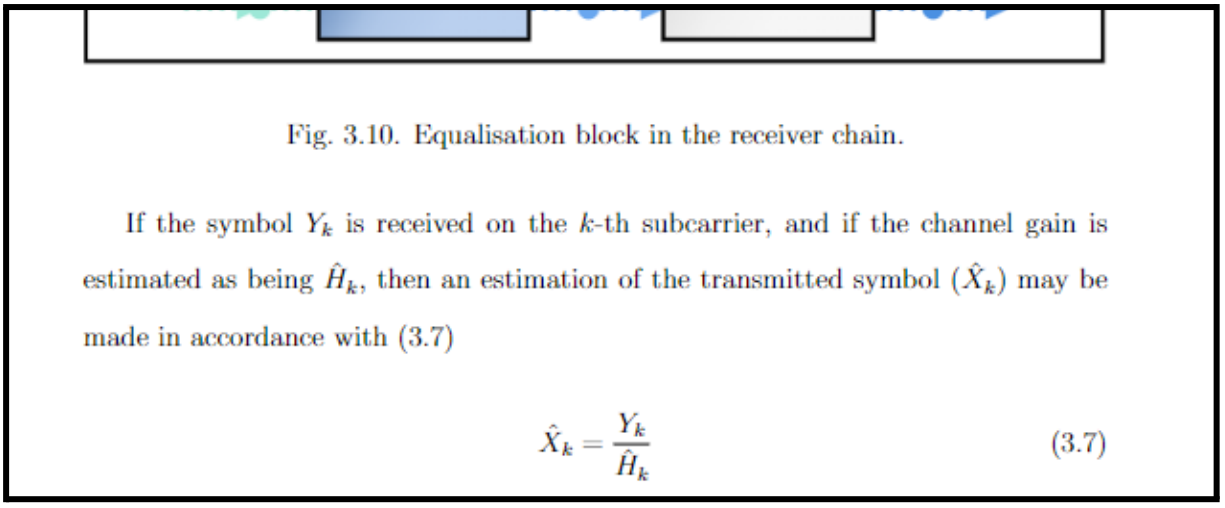


Fig. 3.10. Equalisation block in the receiver chain.

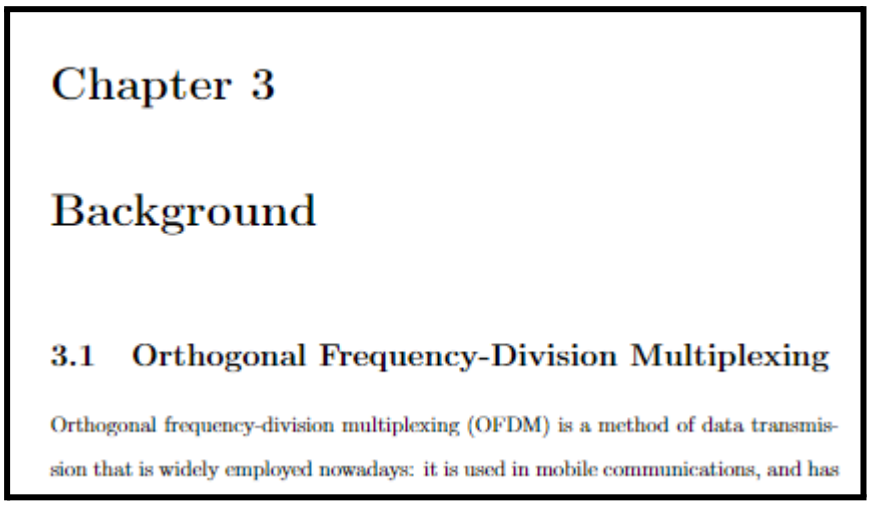
If the symbol Y_k is received on the k-th subcarrier, and if the channel gain is estimated as being $\hat{H}_k$, then an estimation of the transmitted symbol ($\hat{X}_k$) may be made in accordance with (3.7)

$$\hat{X}_k = \frac{Y_k}{\hat{H}_k} \tag{3.7}$$

It helps me keep the text above *below* figure 3.10

19/04/2022:

- I can reference chapters by giving them labels and using \ref

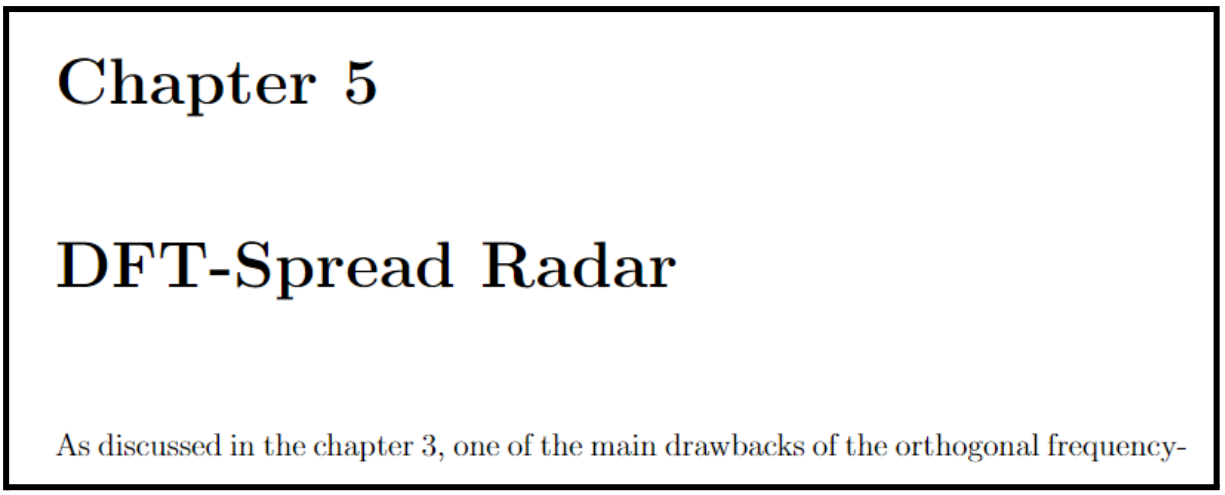


Chapter 3

Background

3.1 Orthogonal Frequency-Division Multiplexing

Orthogonal frequency-division multiplexing (OFDM) is a method of data transmission that is widely employed nowadays: it is used in mobile communications, and has



Chapter 5

DFT-Spread Radar

As discussed in the chapter 3, one of the main drawbacks of the orthogonal frequency-

24/04/2022:

- ***¡La tesis está completada y entregada! 😀😀🥂🍾🌆🌃**

17/06/2022:

- I received an “A-” for my final thesis report, which I’m fairly happy with :) 😃. This means that overall I received an “A-” in the final-year project! This seemed like a far-off possibility at many points throughout the year, especially around Christmas time…