Lecture 9

- Projects
- Multi-carrier systems (OFDM)

LUND UNIVERSITY

Project

- Study one existing or proposed link
  - If choosing a standard, also choose an application example.
- Write a description of it, including
  - Technical details on speed, modulation, equalisation, antennas etc.
  - Link budget, both numerical and graphical
    - » Use well motivated assumptions where no data can be found.
  - 2-4 pages.
- Deadline: 25 May 2018
- Format: pdf
- Email: ajn@eit.lth.se, with subject:"EITN75 report"
- Make sure report itself includes all necessary info, including participants names.
- Reports will be run through Urkund.

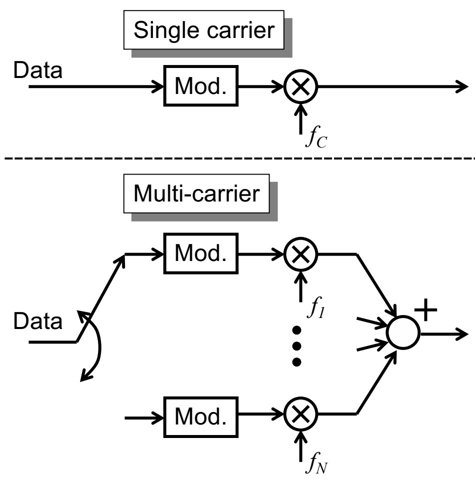
LUND UNIVERSITY

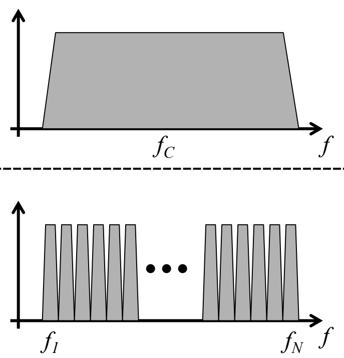
Ideas

- Sattellite links, such as Sattelite TV, INMARSAT, GPS, Irridium,
- Space probes such as Mars probes, including rovers, New Horizons (Pluto), Pioneer, Voyager, Cassini etc.
- DAB-radio, Terrestial digital TV,
- Domestic system: *GSM, 4G, 5G*, Blutetooth, Bluetooth LE, WiFi, *LORA*, WiMax,
- Medical applications: MICS, Bluetoth LE
- *Submarine communication*

LUND UNIVERSITY

Single/Multi-carrier



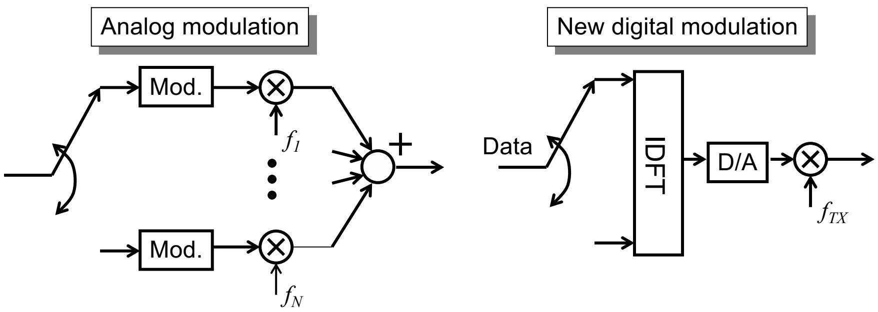


- Using *N* cubcarriers increases the symbol length by *N* times.
- The ISI is reduced by the same amount (in symbols).

LUND UNIVERSITY

Analog vs. digital implementation

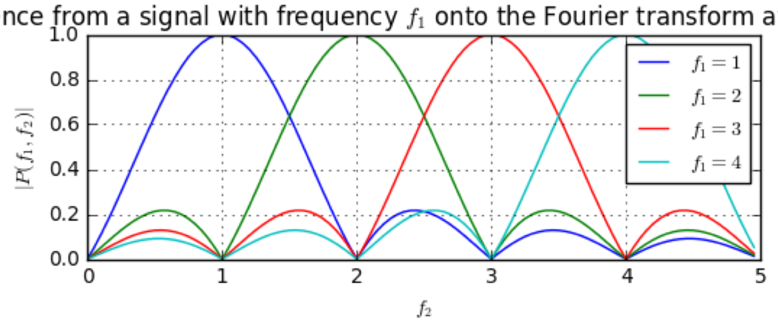
1970's: Digital modulation of subcarriers



LUND UNIVERSITY

OFDM
Orthogonal Frequency-Division Multiplexing

Interference from a signal with frequency f_1 onto the Fourier transform at frequency f_2.



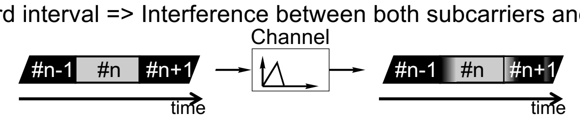
LUND UNIVERSITY

http://dspillustrations.com/pages/posts/misc/intuitive-explanation-of-ofdm.html

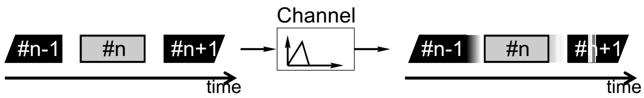
Addition of guard intervals and cyclic prefix

1980's: Improved digital circuits increses interest

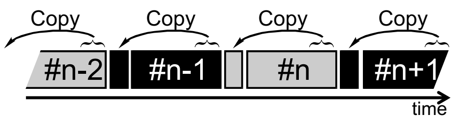
No guard interval => Interference between both subcarriers and symbols



Guard interval => No interference between symbols

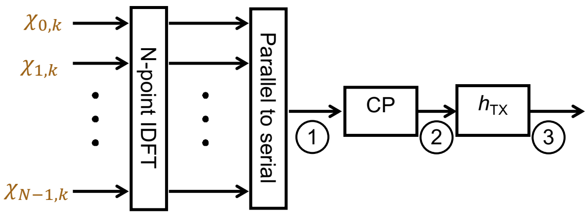


Cyclic prefix => No interference between neither subcarriers nor symbols

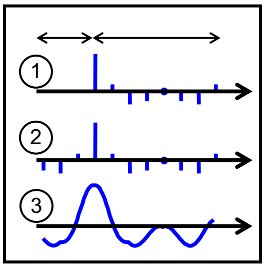


LUND UNIVERSITY

N-channel transmitter



k	– symbol
m	– sample
n	– subcarrier
L	– CP length
T_{samp}	– sampling period
h_{TX}	– TX filter



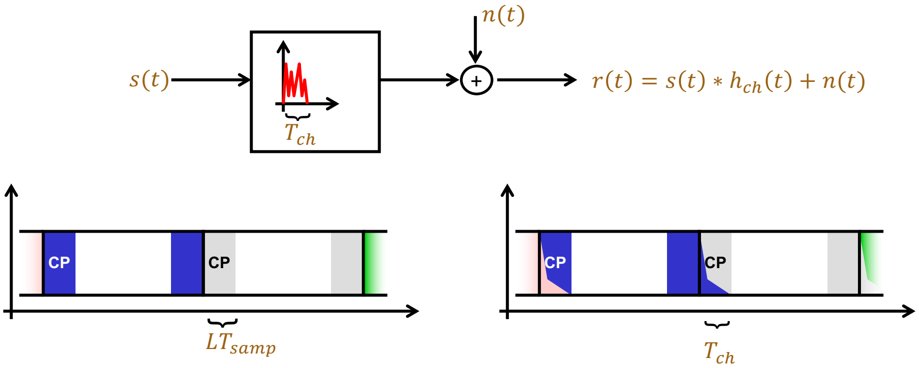
N-point IDFT: $s_{m,k} = \frac{1}{N}\sum_{n=0}^{N-1} x_{n,k}\, exp\left(j2\pi\frac{mn}{N}\right) \; for\; 0 \le m \le N-1$

Adding CP: $s_{m,k} = s_{N+m,k}\,\delta\left(t - (k(N+L)+m)T_{samp}\right)$

TX filtering: $s(t) = h_{TX}(t) * \left(\sum_k \sum_{m=-L}^{N-1} s_{m,k}\,\delta\left(t-(k(N+L)+m)T_{samp}\right)\right)$

LUND UNIVERSITY

Channel effects

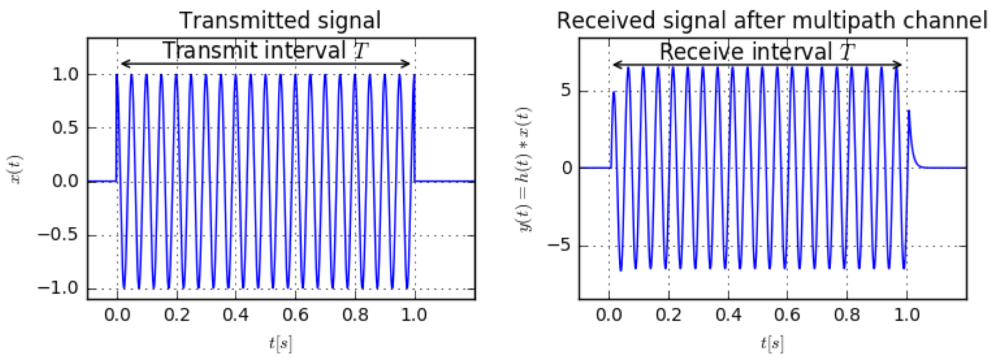


As long as the CP is longer than the delay spread of the channel, $LT_{samp} > T_{ch}$, it will absorb the ISI.

By removing the CP in the receiver, the transmission becomes ISI free.

LUND UNIVERSITY

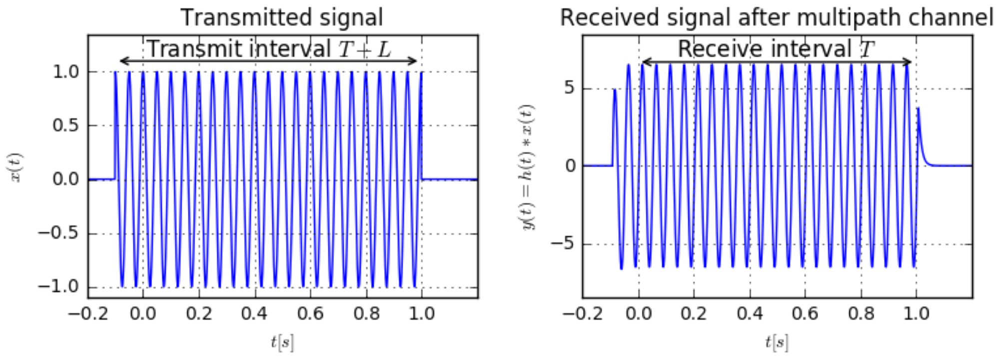
Cyclic prefix



http://dspillustrations.com/pages/posts/misc/intuitive-explanation-of-ofdm.html

LUND UNIVERSITY

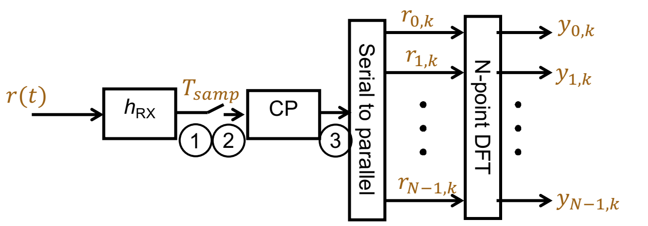
Cyclic prefix



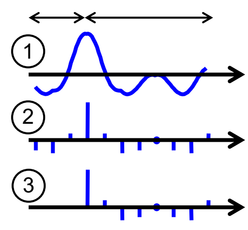
http://dspillustrations.com/pages/posts/misc/intuitive-explanation-of-ofdm.html

LUND UNIVERSITY

N-subcarrier reciever



q – symbol
p – sample
n – subcarrier
L – CP length
T_{samp} – sampling period
h_{RX} – RX filter



RX filtering: $\tilde{z}(t) = h_{RX}(t) * r(t)$

Sampling: $z_k = \tilde{z}(kT_{samp})$

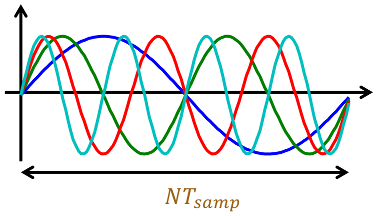
Removing CP: $r_{p,q} = z_{q(N+L)+p}$ $for\ 0 \le p \le N-1$

N-point DFT: $y_{n,q} = \sum_{p=0}^{N-1} r_{p,q} exp\left(-j2\pi \frac{np}{N}\right)$ $for\ 0 \le n \le N-1$

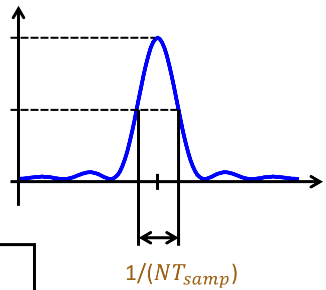
LUND UNIVERSITY

Modulation spectrum

Transmitted OFDM symbol decomposed into different subcarriers (ideal case, 4 subcarriers shown, no CP)



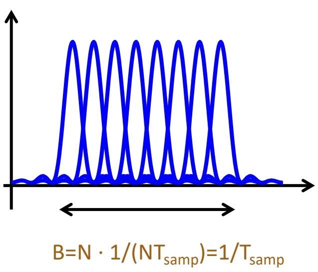
Power spectrum of one subcarrier transmitted at f_n Hz.



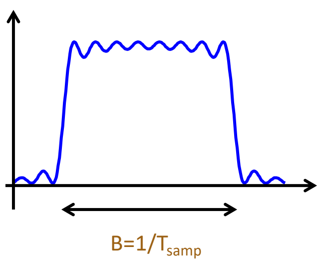
N : Subcarriers
T_{samp} : sampling period

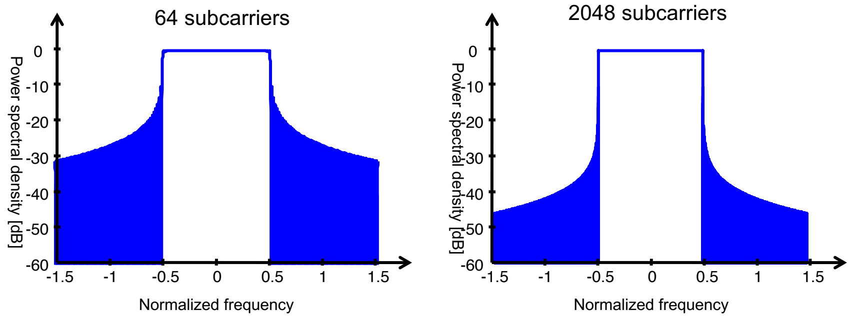
$$sinc(x) = \frac{\sin(\pi x)}{\pi x}$$

The distance between each subcarrier becomes which is the same as the 3 dB bandwidth of the individual subcarriers. Using all *N* subcarriers (8 in this case) we get:



The total modulation spectrum is a sum of the individual subcarrier spectra (assuming independent data on them).





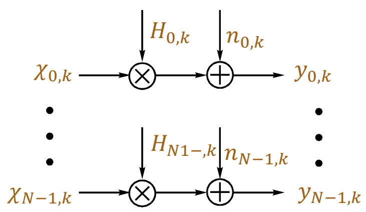
Normalized freq.:

$$\upsilon = T_{samp} f = \frac{f}{B}$$

LUND UNIVERSITY

Simplified model

Simplified model under ideal conditions (no fading and sufficient CP)



Total filter in the signal path:

$$h_{tot}(t) = h_{TX}(t) * h_{ch}(t) * h_{RX}(t)$$

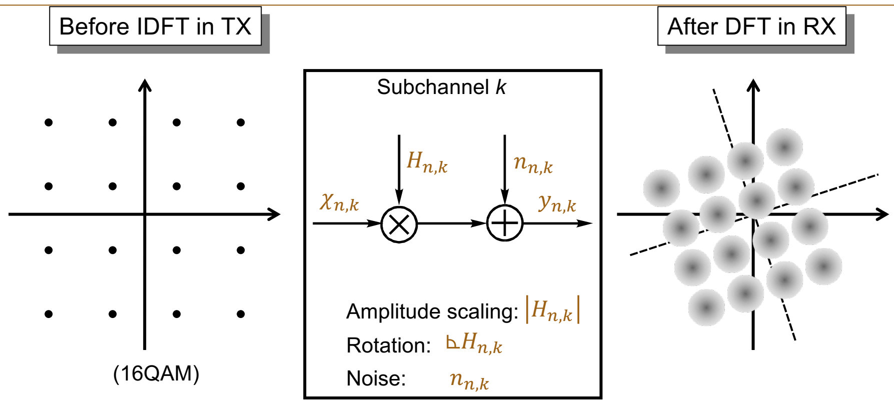
$$H_{tot}(f) = H_{TX}(f) \times H_{ch}(f) \times H_{RX}(f)$$

Given that subcarrier n is transmitted at frequency f_n the attenuations become:

$$H_{n,k} = H_{tot}(f_n)$$

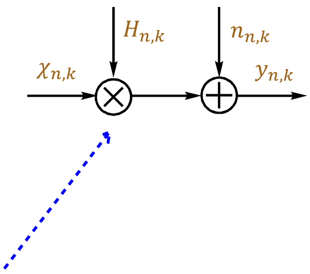
LUND UNIVERSITY

One subchannel



- Simple equalization of each subchannel: Back-rotate and scale

Uncoded performance



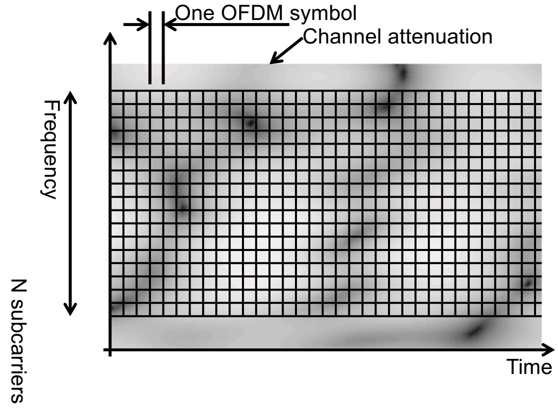
PROBLEM:

- Only one fading tap per subchannel => NO DIVERSITY => POOR PERFORMANCE
- The diversity is in there ... but additional techniques are needed to exploit it!

SOLUTION:

- Spreading the information (data) across several subcarriers or OFDM symbols
- This can be done using interleaving and coding => **Coded OFDM (CODFM)**

Channel correlation

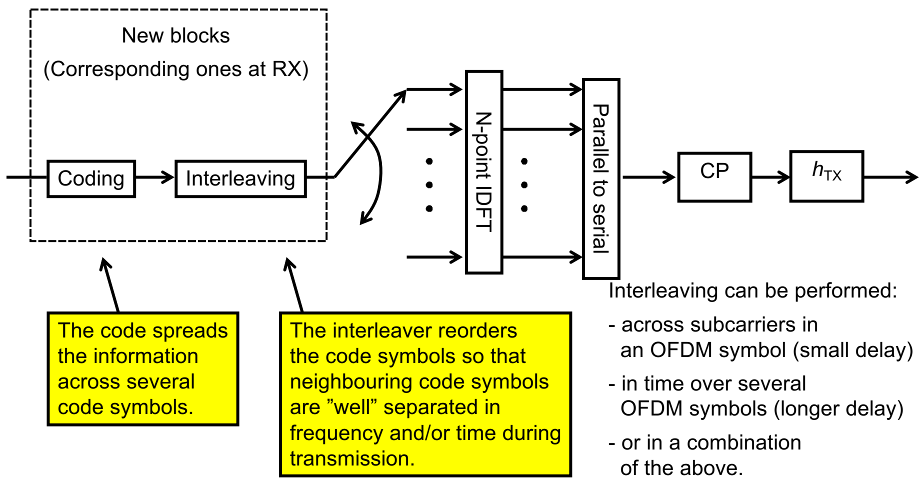


Channel attenuations are correlated in the time/frequency grid.

If we spread each bit of information over several well separated points in the OFDM time/frequency grid, the same "bit" is is received over several "one tap" fading channels.

Combining these in the receiver, we obtain diversity.

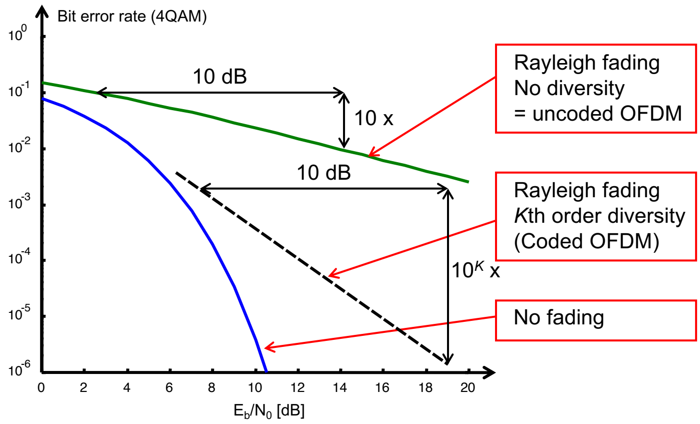
Coding and interleaving



Interleaving can be performed:

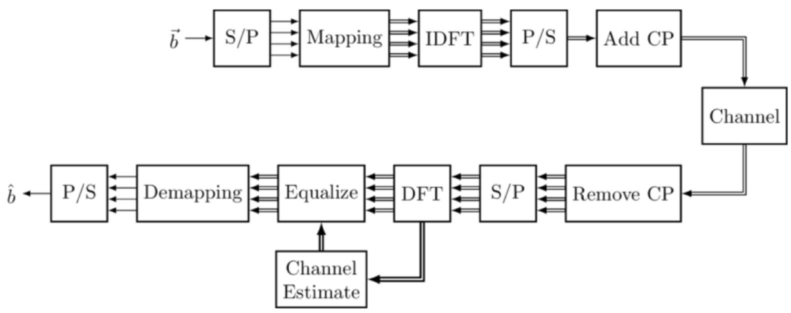
- across subcarriers in an OFDM symbol (small delay)
- in time over several OFDM symbols (longer delay)
- or in a combination of the above.

The better the coding and interleaving scheme, the larger the obtained diversity order.



Python-example

- http://dspillustrations.com/pages/posts/misc/python-ofdm-example.html



Initialisation

```
K = 64 # number of OFDM subcarriers
CP = K//4 # length of the cyclic prefix: 25% of the block
P = 8 # number of pilot carriers per OFDM block
pilotValue = 3+3j # The known value each pilot transmits
```

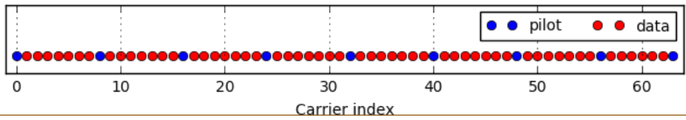
LUND UNIVERSITY

Indexes for carriers

```
allCarriers = np.arange(K) # indices of all subcarriers ([0, 1, ... K-1])
pilotCarriers = allCarriers[::K//P] # Pilots is every (K/P)th carrier.

# For convenience of channel estimation, let's make the last carriers also be a pilot
pilotCarriers = np.hstack([pilotCarriers, np.array([allCarriers[-1]])])
P = P+1

# data carriers are all remaining carriers
dataCarriers = np.delete(allCarriers, pilotCarriers)
```

LUND UNIVERSITY

Check:



print ("allCarriers: **%s**" % allCarriers)
print ("pilotCarriers: **%s**" % pilotCarriers)
print ("dataCarriers: **%s**" % dataCarriers)
plt.plot(pilotCarriers, np.zeros_like(pilotCarriers), 'bo', label='pilot')
plt.plot(dataCarriers, np.zeros_like(dataCarriers), 'ro', label='data')

allCarriers: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63]

pilotCarriers: [0 8 16 24 32 40 48 56 63]

dataCarriers: [1 2 3 4 5 6 7 9 10 11 12 13 14 15 17 18 19 20 21 22 23 25 26 27 28 29 30 31 33 34 35 36 37 38 39 41 42 43 44 45 46 47 49 50 51 52 53 54 55 57 58 59 60 61 62]

LUND UNIVERSITY

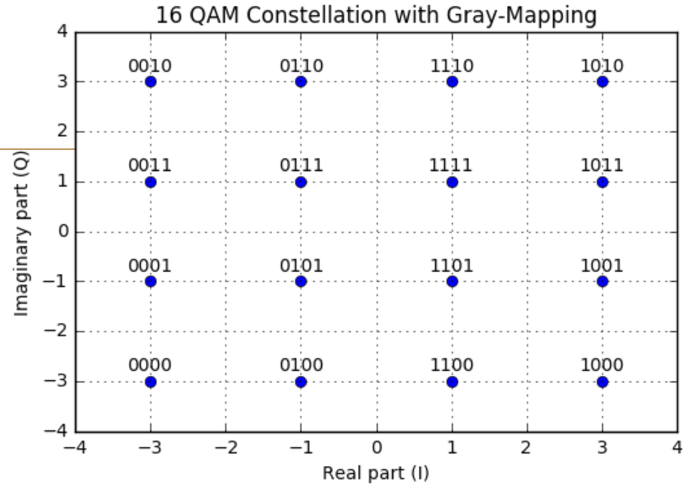
Modulation index and mapping

mu = 4 *# bits per symbol (i.e. 16QAM)*
payloadBits_per_OFDM = len(dataCarriers)*mu
 # number of payload bits per OFDM symbol

mapping_table = { (0,0,0,0) : -3-3j, (0,0,0,1) : -3-1j, (0,0,1,0) : -3+3j, (0,0,1,1) : -3+1j, (0,1,0,0) : -1-3j, (0,1,0,1) : -1-1j, (0,1,1,0) : -1+3j, (0,1,1,1) : -1+1j, (1,0,0,0) : 3-3j, (1,0,0,1) : 3-1j, (1,0,1,0) : 3+3j, (1,0,1,1) : 3+1j, (1,1,0,0) : 1-3j, (1,1,0,1) : 1-1j, (1,1,1,0) : 1+3j, (1,1,1,1) : 1+1j }

LUND UNIVERSITY

Plot of mapping table

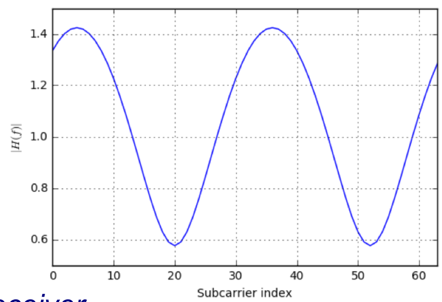



```
for b3 in [0, 1]:
    for b2 in [0, 1]:
        for b1 in [0, 1]:
            for b0 in [0, 1]:
                B = (b3, b2, b1, b0)
                Q = mapping_table[B]
                plt.plot(Q.real, Q.imag, 'bo')
                plt.text(Q.real, Q.imag+0.2, "".join(str(x) for x in B), ha='center')
```

LUND UNIVERSITY

Channel model

```
demapping_table = {v : k for k, v in mapping_table.items()}
channelResponse = np.array([1, 0, 0.3+0.3j])
# the impulse response of the wireless channel

H_exact = np.fft.fft(channelResponse, K)

plt.plot(allCarriers, abs(H_exact))

SNRdb = 25 # signal to noise-ratio in dB at the receiver
```



LUND UNIVERSITY

Test of link

```
bits = np.random.binomial(n=1, p=0.5, size=(payloadBits_per_OFDM, ))
print ("Bits count: ", len(bits))
print ("First 20 bits: ", bits[:20])
print ("Mean of bits (should be around 0.5): ", np.mean(bits))
```

```
Bits count: 220
First 20 bits: [1 0 1 1 0 1 1 1 1 0 1 0 1 0 1 0 0 0 0 0]
Mean of bits (should be around 0.5): 0.554545454545
```

Serial-to-parallell

```
def SP(bits):
    return bits.reshape((len(dataCarriers), mu))
bits_SP = SP(bits)
print ("First 5 bit groups")
print (bits_SP[:5,:])
```

```
First 5 bit groups
[[1 0 1 1]
[0 1 1 1]
[1 0 1 0]
[1 0 1 0]
[0 0 0 0]]
```

Mapping to constellation

```
def Mapping(bits):
    return np.array([mapping_table[tuple(b)] for b in bits])
QAM = Mapping(bits_SP)
print ("First 5 QAM symbols and bits:")
print (bits_SP[:5,:])
print (QAM[:5])
```

```
First 5 QAM symbols and bits:
[[1 0 1 1]
[0 1 1 1]
[1 0 1 0]
[1 0 1 0]
[0 0 0 0]]
[ 3.+1.j -1.+1.j 3.+3.j 3.+3.j -3.-3.j]
```

Allocation of subcarriers with data and pilot

```
def OFDM_symbol(QAM_payload):
    symbol = np.zeros(K, dtype=complex) # the overall K subcarriers
    symbol[pilotCarriers] = pilotValue # allocate the pilot subcarriers
    symbol[dataCarriers] = QAM_payload # allocate the payload subcarriers
    return symbol
OFDM_data = OFDM_symbol(QAM)
print ("Number of OFDM carriers in frequency domain: ", len(OFDM_data))
```

```
Number of OFDM carriers in frequency domain: 64
```

Transform into time-domain

```
def IDFT(OFDM_data):
    return np.fft.ifft(OFDM_data)
OFDM_time = IDFT(OFDM_data)
print ("Number of OFDM samples in time-domain before CP: ", len(OFDM_time))
```

Number of OFDM samples in time-domain before CP: 64

LUND UNIVERSITY

Add cyclic prefix

```
def addCP(OFDM_time):
    cp = OFDM_time[-CP:] # take the last CP samples ...
    return np.hstack([cp, OFDM_time]) # ... and add them to the beginning
OFDM_withCP = addCP(OFDM_time)
print ("Number of OFDM samples in time domain with CP: ", len(OFDM_withCP))
```

Number of OFDM samples in time domain with CP: 80

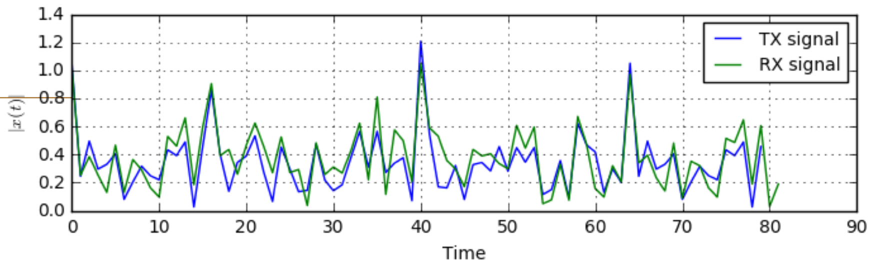
LUND UNIVERSITY

Transmitt over the air

```
def channel(signal):
    convolved = np.convolve(signal, channelResponse)
    signal_power = np.mean(abs(convolved**2))
    sigma2 = signal_power * 10**(-SNRdb/10) # calculate noise power based on signal power and SNR
    print ("RX Signal power: %.4f. Noise power: %.4f" % (signal_power, sigma2))

    # Generate complex noise with given variance
    noise = np.sqrt(sigma2/2) * (np.random.randn(*convolved.shape)+1j*np.random.randn(*convolved.shape))
    return convolved + noise
OFDM_TX = OFDM_withCP
OFDM_RX = channel(OFDM_TX)
```

RX Signal power: 0.1926. Noise power: 0.0006

Check




```
plt.figure(figsize=(8,2))
plt.plot(abs(OFDM_TX), label='TX signal')
plt.plot(abs(OFDM_RX), label='RX signal')
plt.legend(fontsize=10) plt.xlabel('Time');
plt.ylabel('$|x(t)|$');
plt.grid(True);
```

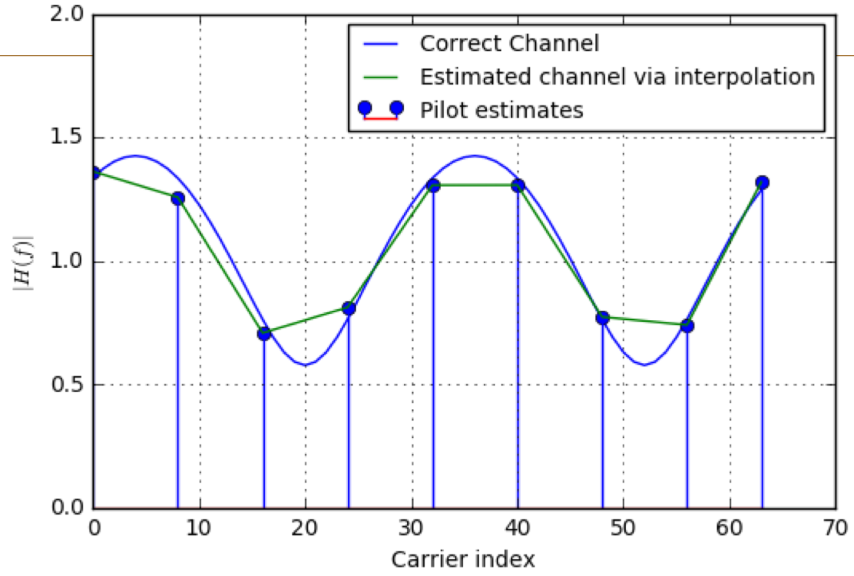
Remove cyclic prefix, transform to frequency domain

```
def removeCP(signal):
    return signal[CP:(CP+K)]
OFDM_RX_noCP = removeCP(OFDM_RX)

def DFT(OFDM_RX):
    return np.fft.fft(OFDM_RX)
OFDM_demod = DFT(OFDM_RX_noCP)
```

Estimate channel (Zero-forcing in this example)

```
def channelEstimate(OFDM_demod):
    pilots = OFDM_demod[pilotCarriers] # extract the pilot values from the RX signal
    Hest_at_pilots = pilots / pilotValue # divide by the transmitted pilot values
    # Perform interpolation between the pilot carriers to get an estimate
    # of the channel in the data carriers. Here, we interpolate absolute value and phase
    # separately
    Hest_abs = scipy.interpolate.interp1d(pilotCarriers, abs(Hest_at_pilots), kind='linear')(allCarriers)
    Hest_phase = scipy.interpolate.interp1d(pilotCarriers, np.angle(Hest_at_pilots),
kind='linear')(allCarriers)
    Hest = Hest_abs * np.exp(1j*Hest_phase)
    plt.plot(allCarriers, abs(H_exact), label='Correct Channel')
    plt.stem(pilotCarriers, abs(Hest_at_pilots), label='Pilot estimates')
    plt.plot(allCarriers, abs(Hest), label='Estimated channel via interpolation')
    plt.grid(True); plt.xlabel('Carrier index'); plt.ylabel('$|H(f)|$');
    plt.legend(fontsize=10) plt.ylim(0,2)
    return Hest
Hest = channelEstimate(OFDM_demod)
```

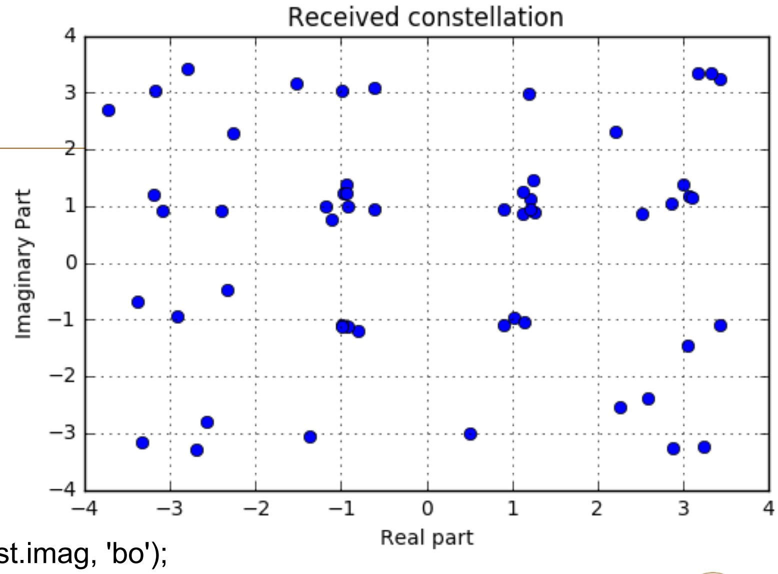
Result



Equalisation, extract data carriers

```
def equalize(OFDM_demod, Hest):
    return OFDM_demod / Hest
equalized_Hest = equalize(OFDM_demod, Hest)

def get_payload(equalized):
    return equalized[dataCarriers]
QAM_est = get_payload(equalized_Hest)
```

Check




```
plt.plot(QAM_est.real, QAM_est.imag, 'bo');
```

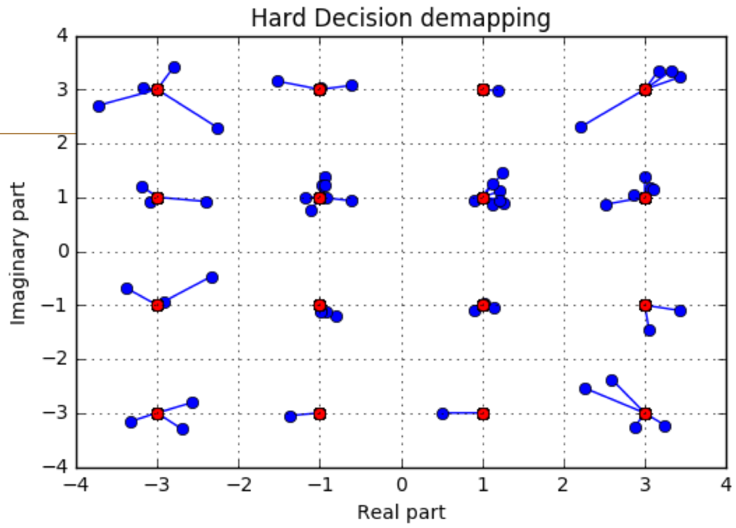
LUND UNIVERSITY

Detection

```
def Demapping(QAM):
    # array of possible constellation points
    constellation = np.array([x for x in demapping_table.keys()])
    # calculate distance of each RX point to each possible
    point dists = abs(QAM.reshape((-1,1)) - constellation.reshape((1,-1)))
    # for each element in QAM, choose the index in constellation
    # that belongs to the nearest constellation point
    const_index = dists.argmin(axis=1)
    # get back the real constellation point
    hardDecision = constellation[const_index]
    # transform the constellation point into the bit groups
    return np.vstack([demapping_table[C] for C in hardDecision]), hardDecision
PS_est, hardDecision = Demapping(QAM_est)
```

LUND UNIVERSITY

Check




```
for qam, hard in zip(QAM_est, hardDecision):
    plt.plot([qam.real, hard.real], [qam.imag, hard.imag], 'b-o');
    plt.plot(hardDecision.real, hardDecision.imag, 'ro')
```

LUND UNIVERSITY

Parallell to serial

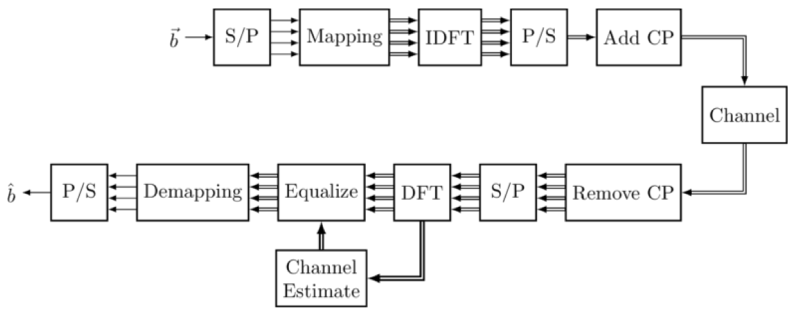
```
def PS(bits):
    return bits.reshape((-1,))
bits_est = PS(PS_est)

print ("Obtained Bit error rate: ", np.sum(abs(bits-bits_est))/len(bits))
```

Obtained Bit error rate: 0.0

LUND UNIVERSITY

Python-example

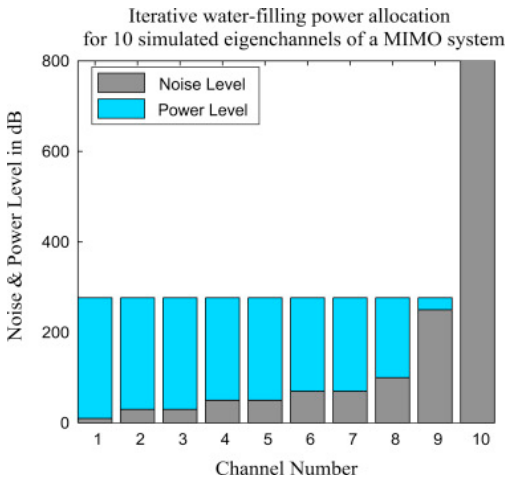
- http://dspillustrations.com/pages/posts/misc/python-ofdm-example.html



LUND UNIVERSITY

Waterfilling

- $P_n = max\left(0, \varepsilon - \frac{\sigma_n^2}{|\alpha_n|^2}\right)$ (19.31)
- α_n *is the gain of the n:th channel*
- σ_n^2 *is the noise variance*
- ε *is a threshold set to hold* $P = \sum_{n=1}^{N} P_n$



https://doi.org/10.1016/j.icte.2018.01.011

LUND UNIVERSITY

LUND UNIVERSITY