Final work in Wireless Communication Systems

Vili Raunola

This document covers parts 1 to 3 of the final work in the wireless communication course.

Part 1: Report from Ch. 5-9 from Popovski, Petar

Here are summarised chapters 5,6,7,8 and 9 from the book “Wireless Connectivity : An Intuitive and Fundamental Guide”. This means that all of the facts and information are based on the book. Due to using only one source in the chapter, each reference is not marked individually. The reader thus can assume that everything is based on the previously mentioned source.

Chapter 5

This chapter focuses on the different modulation methods that can be used, the noise the signals are experiencing and the errors that are caused by the noise. The most basic idea in wireless communication is the expression of y=hz+n. This expression describes the receiving of a signal given the channel coefficient (h), symbol (z) and noise (n). As from the expression we can see that all of these factors have an impact on the received signal. In the simplest scenario h and n equal zero meaning that y=z which gives the sent bit to the receiver. To increase the data rate of the signal we can map the sent bits to different amounts of complex values. This would mean that we could send multiple bits at once. In the real world, we can see that the n variable has an impact on the received signal which will introduce errors to our transmission. A way to combat the noise is to use the coefficient h that needs to be learned before the transmission using pilots. Pilots are sequences of several bits that are sent before the transmission to help the receiver and sender calculate the needed h.

To send the signal we can use different constellation points. They describe all of the symbols that the broadcasting system can send in a real-imaginary axis as points. The hard questions to answer when implementing the baseband signal is to decide on the constellation points and which bit should be mapped to which constellation point. To decide on these one could use Euclidean distance which tells the distance of the two constellation points. We want to maximise this distance since the noise creates a randomness in the signal thus it also creates uncertainty around the constellation point. This randomness tells us that the received bit can land anywhere within the radius of the noise. The probability of the received bit landing inside the noise cloud is presented as p_c . If multiple constellation points are overlapping it becomes impossible for the receiver to know at which contellation point the original bit is supposed to be. So if we can select the constellation points as far away from each other we reduce the possibility of getting the received signals mixed up. There are ways to add more constellation points whose noise clouds don't overlap but this means increasing the magnitude of the transmitted symbols which then affects the transmission cost.

This leads us to the Binary Phase Shift Keying (BPSK) and Quadrature Phase Shift Keying (QPSK). In BPSK the bits are mapped one to one to the constellation points but in QPSK two bits can be mapped to one constellation point. QPSK is also the best method for separating four constellation points. It separates the points in a circle so that they are equally apart from each other and thus minimizes the symbol error. This also means that the QPSK sends the information only in the phase of the signal. The way the bits are mapped in QPSK is called Gray mapping. It uses Hamming distance that tells us how far apart the sequences are from each other using the amount of how much the sequences differ from each other. In Gray mapping, the Hamming distance is one to minimize the bit error caused by the symbol error. Another way to send the data is to use Pulse Amplitude Modulation (PAM) and Quadrature Amplitude Modulation (QAM). All of these different modulation methods can affect the data rate. It tells how much data can be sent at once. The actual data rate does depend on the packet error probability.

The last topic from this chapter that I want to include here is the adaptive modulation that was mentioned in the book. The idea behind it is that it should select the modulation method that provides the best goodput in a given situation. The goodput in this case means how many bits are delivered that are useful to the receiver. The modulation method should thus be changed based on the current SNR value. This can be done if the system can detect the channel information. Usually, the worse the SNR is, the more suitable the BPSK is, if the SNR is good we can use 16-QAM or higher to achieve the best possible goodput. If the SNR is bad it is better to use a more noise-resistant method so we can receive the bits more reliably.

Chapter 6

The chapter starts by introducing Shannon’s general model of communication systems. It consists of the information source that gives the message to the transmitter which creates the signal. The signal is sent to the receiver through a channel and is affected by a noise. The receiver receives the noisy signal and decodes it after which it can be used. This model is the basis all of the communication systems since it depicts the most basic principles of communication. It is also mentioned that the channel of the communication system is the part which is unable to change. This means that the sender and receiver need to use the channel smartly in order to maximise the channel use. Channel use is defined to mean the amount of bits that can be sent per channel use.

The next important topic that the book goes over is the Additive White Gaussian Noise (AWGN) which has also been discussed in the course. This means that the sent signal will experience noise which is defined by n which is an independent Gaussian random variable. This noise affects the signal and makes it harder to determine on the receiver's side what the original message was. Another feature of AWGN is that it is memoryless meaning that the previous sent signal doesn’t affect the current signal in any way. This is an important feature to know since each channel usage is then only affected by the noise and the current input. To use the AWGN channel optimally we don’t want to restrict the power to a constant in the signal transmission method. This is because by altering the power in the transmission we can send more information per channel use.

The book also mentions different ways to use the channel. The strategy which to use depends heavily on the probability of receiving the correct input and also on the parties knowing the Channel State Information (CSI). CSI means the channel coefficient h which has been defined previously. If both parties know the CSI beforehand it allows them to use time-multiplexing. If the CSI is not known by either then deciding the received input relies heavily on guessing unless different power levels are used for transmission. The situation in which the sender knows the CSI but the receiver doesn’t is quite tricky but there still is a strategy that can be used. It relies on the sender to monitor the channel and send the signal at the correct time in order to manipulate the receiver's distribution. The last situation is when the sender doesn’t know the CSI and the receiver does. In this case, the sender must use a fixed transmission scheme and the receiver can deduct what the sender originally wanted to send.

The book also covers the Binary Symmetric Channel (BSC) which was covered in the course. It shows the sending and receiving a bit with the probabilities of the transmission succeeding or failing. This also relates to the Grey mapping since we know from BSC that transmission errors have a probability of happening so when it happens we want to keep it contained as a single-bit error rather than a multiple-bit error. The probability of error in symbol transmission is not the same for all of the constellation methods due to geometry. For example, in QPSK the bits in a symbol are affected by real and imaginary values so the error is independent of each other. But in higher-order constellations, the outermost points don’t have as many neighbours so they are less susceptible to errors than the points in the middle of the constellation.

The problem with BSC is that the sender cannot be fully certain that the receiver got the bit that was sent. To get a positive throughput we need to use Binary Erasure Channel (BEC). This allows the receiver to know certainly if the received bit is correct or not and if an error occurred it can ask for re-transmission from the sender. This leads to a positive throughput in the channel.

Chapter 7

This chapter discusses coding and error correction in communication. The idea behind error correction in the simplest terms is to add for example extra bits to the sent signal which carry information about the actual payload. These error-correcting bits are formed from the original payload and thus can be used at the receiver's side to verify the message or to identify that the message was received with an error. If the error was recognized the receiver could also try to fix the error by themselves. If the error correction is not possible on the receiver's end they can ask for a retransmission of the packet. Many different assumptions are included in this such as that the error correction doesn't suffer from the the same noise as the actual payload.

One way to add transmission reliability is to send multiple bits in a row that represent one original message bit. For example, if we want to send 1 we could use 4 repetitions which would lead to a signal of 1111. This improves the reliability since each bit has the same probability of suffering from the noise that can distort the signal but if we send the same bit multiple times a row the probability that multiple bits suffer enough noise in a row is much smaller. On the receiver's end, they can decide on what was sent by the majority of vote. This does come at a price though which is the channel goodput. Since we need to send a bit multiple times we decrease the amount of data we can send per second.

Repetition is not the best option always. Sometimes it is better to use coding. Coding aims to improve channel redundancy and give better performance than repetition. The idea of how the coding works is to use the channel multiple times to send the data by selecting a group of inputs. In theory, this should lead to a more reliable channel. This does have a problem if the sender has as many codewords as the receiver. The receiver might get the signal with noise that moves the codeword to the neighbouring code word with a quite high change. To combat this problem the sender should use codewords that are not overlapping on the receivers side. This way the receiver can still know with quite high certainty that the received codeword belongs to the specified group.

The previous concept is related to the Maximum Likelihood decoding. The idea is to calculate the areas to which the signal is most likely to end up thus giving the receiver the best probability to decode the message correctly. This is quite computationally heavy if there are many channel uses.

One way to add the error coding as previously mentioned is to use encoding. In encoding the sent bits are mapped to a longer bit sequence in which the extra bit carries the error check. One way to do this is to use linear block codes. The downside of it is that if more than two errors happen the error will go undetected which is undesirable.

To further help with channel reliability, we can use retransmission. The very basic idea of retransmission is for the sender to acknowledge the packets it has received and if an error occurs or the sender doesn't receive an acknowledgement they resend the packet. This can be improved in many ways for example the receiver can store the received packets which were exposed to the error and compare them to the newly received packets to help with the error correction if the new packets also suffer from errors. This is also the standard in wireless communication to detect errors. It is called the ACK/NACK feedback in which ACK

stands for the acknowledgement packet that is transmitted after each correctly received packet and NACK stands for the request to retransmit the packet due to an error on the side of the receiver.

Chapter 8

This chapter focuses on the information-theoretic view of wireless channel capacity. I try to summarize the chapter by not involving mathematics but rather focusing on the core concepts. The channel capacity of a wireless channel means the maximum value of bits sent per channel use. Using this variable one can understand and study the channel by its communication capability. An important notion to the channel capacity is that the ratio expects one hundred per cent reliability as the amount of sent packets grows. This brings an important difference to the throughput, since in throughput measurement some loss can be accepted whereas in channel capacity there is none.

One of the important parts of information theory is the concept of the Law of Large Numbers (LLN). In short, LLN means that if a large enough test is conducted with a random variable the outcome of the test should correspond very closely to the expected value. The more trials are conducted to more likely it is that the average of the test moves towards the expected value. In tests in which the trial amount is low, there is a higher chance that the outcome differs more from the expected value.

Another important part of the information theory is the concept of lossless and lossy source coding. In lossless source coding the decoder is able to decode the packet completely but in lossy source coding the decoder is able to decode it partly. This brings us a new value called distortion which defines the amount of distortion which is allowed in the lossy source coding.

The book also covers the topic of mutual information which was covered during the course. The basic idea in mutual information is that a random variable can provide useful information about another random variable. By manipulating the other variable’s probability that is sent and used to determine the other we could maximize the mutual information which on the other hand would lead to the mutual information being equal to the capacity of the channel.

A case where mutual information maximisation could be used is in the Z-channel. Z-channel means that sending a bit can be done by not sending anything and another bit is sent by sending the signal. This leads to a situation in which the error of the other bit is zero. By maximising the mutual information in this case one could find the probabilities of the bits to which to use for decoding the message.

The book also goes over a concept called data processing inequality. This concept means that the receiver of a signal cannot get more information from the signal than the original sender's higher layer had put on the signal. This is an important theory since we now know that the receiver cannot increase the amount of information from the received signal but can in an optimal case extract the same amount of information that was put into the signal.

Lastly, I will mention the actual protocol that could be used between two parties in wireless communication that is defined in the book. It consists of three phases which are channel estimation, channel reporting and the communication phases. In the first phase, the sender sends the pilot signal to the receiver who will use them to estimate the channel coefficient. After calculating the coefficient the receiver will send the signal-to-noise ratio to the sender. Now the sender can use the signal-to-noise variable to send the transmission in a way that gives the system an excellent goodput.

Chapter 9

The chapter introduces the topic of the physical side of signal transmission and the main concepts that are part of it. It goes over the Medium Access Control (MAC), how the frequency affects the transmitted signal and also gives a brief introduction to a few different transmission protocols.

MAC is defined as a way to control the space in which the transmission takes place. It utilizes time to do so. The main focus of it is to make sure that all of the parties that transmit get the opportunity to do so in a way that minimizes the interference of other transmissions. The transmission can also be described as a channel use which means an opportunity that a transmitter has to transmit its signal. This can also be called Degree Of Freedom (DOF).

When the signal is generated the time between sending each symbol can be controlled by the transmitter. In some cases, the time between can be set as the time of the transmitted signal. This way the signals are sent just as the previous signal ends. In this case there is no idle time between sending the symbols but there is also no interference. If this time is set to be lower than the time it takes to send the message there will be some interference but we get a better data rate since it takes less time to send the same amount of data. The interference of the sent data bits is called Intersymbol Interference (ISI). Using ISI makes the channel have a memory since the current transmission is affected by the previous and the next one. However, having ISI makes the decoding of the signal harder.

The chapter also covers the generation of the actual physical signal which was also taught in the lecture. In short, a continuous signal is created which is altered in a way so that it can carry data. The receiver can then extract the real and imaginary values from the signal since they are orthogonal to each other. This does assume that the receiver and the transmitter are synchronized in time in order for the orthogonality to take effect.

That brings us to the first transmission method called Orthogonal Frequency Multiplexing (OFDM). The basic idea of OFDM is that it sends the data in multiple channels that are separated from each other by frequency. It uses the continuous signal as previously mentioned to send the data and each of the signals that carry the information is called a subcarrier. This method allows the sending of more data at the same time frequency and is thus a widely adopted method.

Other transmission methods were also covered such as Frequency Division Multiple Access (FDMA) and Code Division Multiple Access (CDMA). FDMA's working principle is to use different channels for different users. These channels are separated by frequency. This ensures that different parties can communicate simultaneously since their transmission is on different frequencies thus not interfering with each other. This can be achieved by using a filter mask to contain the signal's frequency so it doesn't interfere with the channel close to it.

CDMA on the other hand uses spreading sequences to spread the transmission into multiple parts which are located at different frequencies. One of the interesting facts about CDMA and the spreading of the transmission into multiple channels is that the communication is hard to detect if an outsider is listening since they don't know the timings and frequencies of

the channel hopping. This is called covert communication. So using CDMA leads to a low probability of interception and is thus a quite secure way of communicating.

Part 2: Existing technologies

In this chapter are the video summarization and the explanation of how an OFDM system works. The source for each of the chapters is the corresponding video. The reason for not adding the sources after each part would feel silly since the course provided the sources to use and are thus known.

The first video explains extremely briefly the history of wireless communication. The first generation of mobile wireless communication had difficulties in providing smooth service. But in later iterations, technological leaps were made which made the services much more reliable and faster. Different countries' organizations also grouped up to form a global connectivity institution around the time of 1G. Lastly, the technologies used in 4G were mentioned such as the way the signal is modulated, how the spectrum is divided and the bandwidth is distributed.

The second video covers the full duplex methods that the LTE supports currently. A full duplex means that both parties communicating with each other can receive and send data simultaneously. The first method is Frequency Division Duplexing (FDD). The idea behind FDD is that there are two channels located at different frequencies between the parties communicating. The channels are separated in order for both parties to send data to each other simultaneously. The channels are separated by a guard band in order to make sure there is some room if noise affects the signal. The benefits of FDD are that the latency is small, it is not affected by distance too much and also the MAC layer is quite simple to implement. Currently, the majority of LTE is based on FDD.

The other duplexing method is called Time Division Duplexing (TDD). The basic idea in TDD is that the transmission time is separated into slots. These slots are for the parties to either receive or send data. It uses the same frequency all the time during the transmission. There is also a guard period between the transmission times to reduce confusion. The benefits of TDD are that the equipment to handle the communication are cheaper and less complex to make, the traffic is very well balanced and the spectral efficiency is good since it only uses one frequency.

The third video explains how multiple users can use the same channel at once. It is based on multiplexing in which messages are combined together using a multiplexor and sent through a shared channel. When multiplexing is put on a real world it can be used to create multiple access which allows users to communicate over a single common channel. There are many ways to implement multiplexing such as FDMA, AMPS, TDMA, CDMA and OFDMA.

One of the more basic ones that was used in 2G is the TDMA which uses time slots to give users the opportunity to send data on the channel. CDMA on the other hand was used in the 3G. It works by multiplying the given data signal with a random code and sending it on the channel. It can send multiple data signals at once. On the receivers end they can be decoded using the same random code. These technologies still lacked the data rates, peak hours traffic and bad spectral efficiency. Due to the bad performance, new ways of multiplexing were developed.

The fourth video introduces us to the FDMA. FDMA works by splitting the available carrier frequency into multiple smaller sub-carriers. By doing this, it can send more data in the same amount of time since the guard bands of sub-carriers will have a much smaller impact than giving a guard band to each symbol sent on a single carrier. This is called slow modulation in which the symbol duration is increased. The sent data signals are centred around the carrier frequencies that are responsible for transmitting the signal. They are also separated by a guard band. OFDMA on the other hand is a more refined implementation of the FDMA. In OFDMA the O stands for orthogonality which the technology is based on. Each subcarrier is orthogonal to each other which makes it possible to place them next to each other. This leads to significantly better data rates. For these channels to be able to send any data, they need to be mapped using the constellation methods.

The fifth and last video shows the basic principles from the start of the transmission to the receiver decoding and using the information in 4G. The first step in sending the data is to use map the bits that are to be sent to a constellation point using modulation. After the constellation points are used for the data bits the subcarrier is adjusted to by the constellation points that were created. To send the data using the subcarriers, IFFT is used to create the continuous signal that is transmitted. The receiver then listens to the incoming signal and takes samples at a high frequency and converts these points to the continuous signal using a DTOA converter. Once the receiver has this signal it reverses the previous operations that were done at the sender. This results in the receiver obtaining the original bits that were sent.

For a device that can only produce low-power signals OFDM(A) is not suitable. To combat this problem CSFDMA was developed. The idea of CSFDMA is to send the symbol using all of the available frequencies. This will result in a single carrier transmission which will a good power efficiency.

Next, I will go through how the OFDM system works based on all of the information that I have gathered on the course and the books and videos introduced by the course in my own words. The assumptions that I will make for the system are that I will not cover the actual implementations of the electronics but the basic concepts that are included between the sender getting the bits and the receiver having the bits on their end. The first step in the system is for the sender to receive a stream of bits that it needs to send over a wireless communication channel. Depending on the system a Forward Error Coding (FEC) can be done to the data. This means that the incoming data is fed through the FEC which will add redundancy to the bit stream which will help to detect errors in the bitstream at the receiver's end.

After the FEC comes the modulation of the bit stream. The system has decided beforehand which kind of modulation it uses. There are many different modulation methods such as PSK, 16-QAM or 64-QAM all of which have their benefits and downsides. Using the predefined modulation method constellation points are formed to which the incoming bit stream is mapped. Depending on the modulation method 1 to many bits are combined at a single constellation point called a symbol. This symbol is used in the actual transmission.

After the modulation of the signal, the symbols need to be sent using a physical signal. In OFDM this is achieved by using the OFDM's subcarriers. Since the given frequency in OFMD is divided into multiple sections in which there are their own signals that will carry the symbols. The subcarriers are orthogonal to each other meaning that they will not create interference to each other. Each symbol will be combined with the subcarrier's signal in order to have the signal carry the information. After the subcarrier signals are combined with the symbols the subcarriers are fed through an Inverse Fast Fourier Transformation (IFFT). The job of IFFT is to create a continuous signal that can be transmitted. IFFT therefore transforms the signal from the frequency domain to the time domain. Now the signal can be sent.

On the receiver's side, the signal will be listened to at certain time frames and the data is saved. Based on the observations the incoming transmission is recreated as a continuous signal. This operation is done using DTOA. The signal is then fed through FTT to get the frequency spectrum. From the frequency spectrum, the symbols can be identified and be demodulated. After the demodulation, the bit stream is checked using the FEC decoder if it was used. If the bitstream had small errors due to the transmission they are spotted and even possibly fixed. Depending on the higher layer implementation the receiver can ask for a re-transmission of the signal in hopes that it comes error-free this time. After the error checks are complete and the receiver is sure that the message is what it is supposed to be it can be passed to the upper layers for them to use the received bit stream.

Part 3 of the assigment.

1. Selecting and plotting a signal in it's time and frequency domain. The first signal is time-limited and the second one is band-limited.

In [ ]:
```
#install missing libraries
! pip install --upgrade numpy
! pip install --upgrade matplotlib
! pip install scikit-dsp-comm
! pip install komm
```

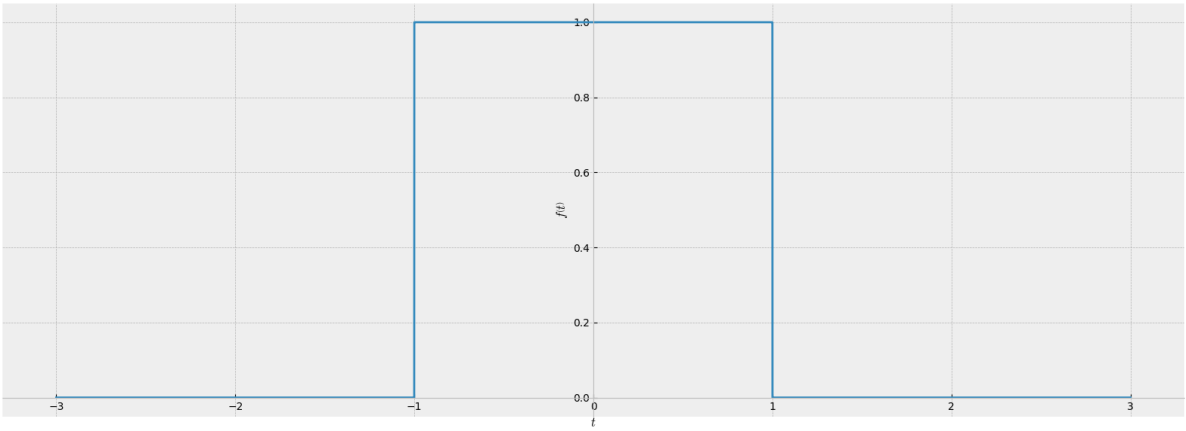
```
Requirement already satisfied: numpy in g:\anaconda\lib\site-packages (1.26.2)
Requirement already satisfied: matplotlib in g:\anaconda\lib\site-packages (3.8.2)
Requirement already satisfied: contourpy>=1.0.1 in g:\anaconda\lib\site-packages
(from matplotlib) (1.0.5)
Requirement already satisfied: cycler>=0.10 in g:\anaconda\lib\site-packages (from
matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in g:\anaconda\lib\site-packages
(from matplotlib) (4.25.0)
Requirement already satisfied: kiwisolver>=1.3.1 in g:\anaconda\lib\site-packages
(from matplotlib) (1.4.4)
Requirement already satisfied: numpy<2,>=1.21 in g:\anaconda\lib\site-packages (fr
om matplotlib) (1.26.2)
Requirement already satisfied: packaging>=20.0 in g:\anaconda\lib\site-packages (f
rom matplotlib) (23.1)
Requirement already satisfied: pillow>=8 in g:\anaconda\lib\site-packages (from ma
tplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in g:\anaconda\lib\site-packages
(from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in g:\anaconda\lib\site-packag
es (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in g:\anaconda\lib\site-packages (from pyt
hon-dateutil>=2.7->matplotlib) (1.16.0)
Requirement already satisfied: scikit-dsp-comm in g:\anaconda\lib\site-packages
(2.0.3)
Requirement already satisfied: numpy>=1.20.0 in g:\anaconda\lib\site-packages (fro
m scikit-dsp-comm) (1.26.2)
Requirement already satisfied: matplotlib>=3.0.0 in g:\anaconda\lib\site-packages
(from scikit-dsp-comm) (3.8.2)
Requirement already satisfied: scipy>=1.1.0 in g:\anaconda\lib\site-packages (from
scikit-dsp-comm) (1.11.1)
Requirement already satisfied: contourpy>=1.0.1 in g:\anaconda\lib\site-packages
(from matplotlib>=3.0.0->scikit-dsp-comm) (1.0.5)
Requirement already satisfied: cycler>=0.10 in g:\anaconda\lib\site-packages (from
matplotlib>=3.0.0->scikit-dsp-comm) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in g:\anaconda\lib\site-packages
(from matplotlib>=3.0.0->scikit-dsp-comm) (4.25.0)
Requirement already satisfied: kiwisolver>=1.3.1 in g:\anaconda\lib\site-packages
(from matplotlib>=3.0.0->scikit-dsp-comm) (1.4.4)
Requirement already satisfied: packaging>=20.0 in g:\anaconda\lib\site-packages (f
rom matplotlib>=3.0.0->scikit-dsp-comm) (23.1)
Requirement already satisfied: pillow>=8 in g:\anaconda\lib\site-packages (from ma
tplotlib>=3.0.0->scikit-dsp-comm) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in g:\anaconda\lib\site-packages
(from matplotlib>=3.0.0->scikit-dsp-comm) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in g:\anaconda\lib\site-packag
es (from matplotlib>=3.0.0->scikit-dsp-comm) (2.8.2)
Requirement already satisfied: six>=1.5 in g:\anaconda\lib\site-packages (from pyt
hon-dateutil>=2.7->matplotlib>=3.0.0->scikit-dsp-comm) (1.16.0)
Requirement already satisfied: komm in g:\anaconda\lib\site-packages (0.8.2)
Requirement already satisfied: numpy in g:\anaconda\lib\site-packages (from komm)
(1.26.2)
Requirement already satisfied: scipy in g:\anaconda\lib\site-packages (from komm)
(1.11.1)
```

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
style.use('bmh')
import warnings
warnings.filterwarnings('ignore')
import sympy as sp
sp.init_printing()
from matplotlib import style
style.use('bmh')
```

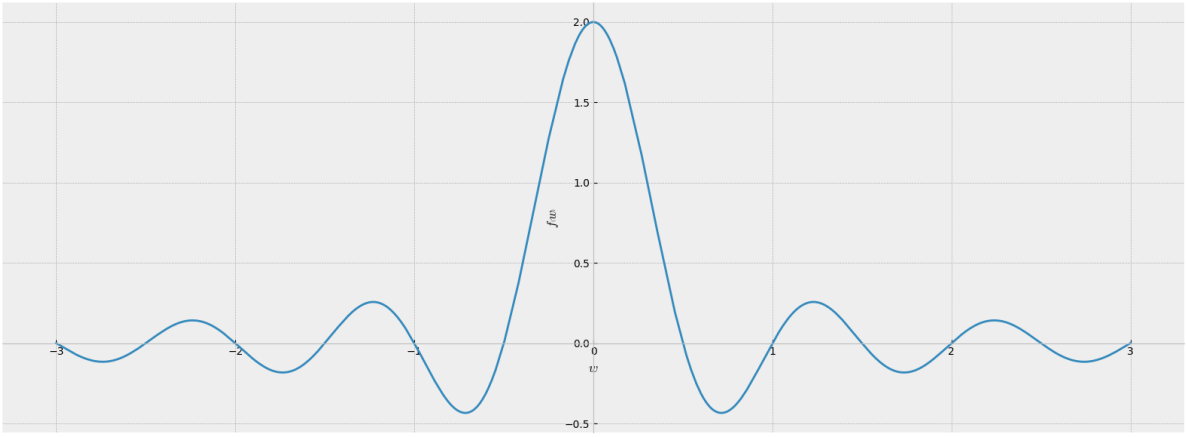
```
plt.rcParams['figure.figsize'] = 16, 6
import komm
import scipy.fftpack as fftpack
import scipy.special
```

In [ ]:
```
# a) Timelimited pulse in time domain constructed by two heaviside functions
t = sp.symbols('t')
sp.plot(sp.Heaviside(t+1) - sp.Heaviside(t-1), (t, -3, 3))
```



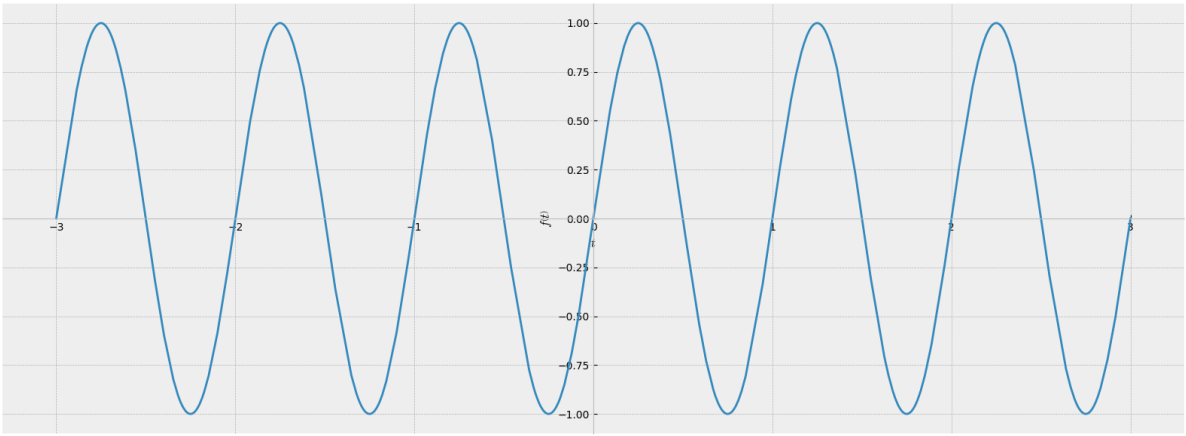
Out[ ]:
```
<sympy.plotting.plot.Plot at 0x16cf685e990>
```

In [ ]:
```
# a) Timelimited pulse in freaquency domain constructed by two heaviside functions

w = sp.symbols('w')
sp.plot(sp.fourier_transform(sp.Heaviside(t+1) - sp.Heaviside(t-1), t, w),(w,-3,3))
```



Out[ ]:
```
<sympy.plotting.plot.Plot at 0x16cde100050>
```

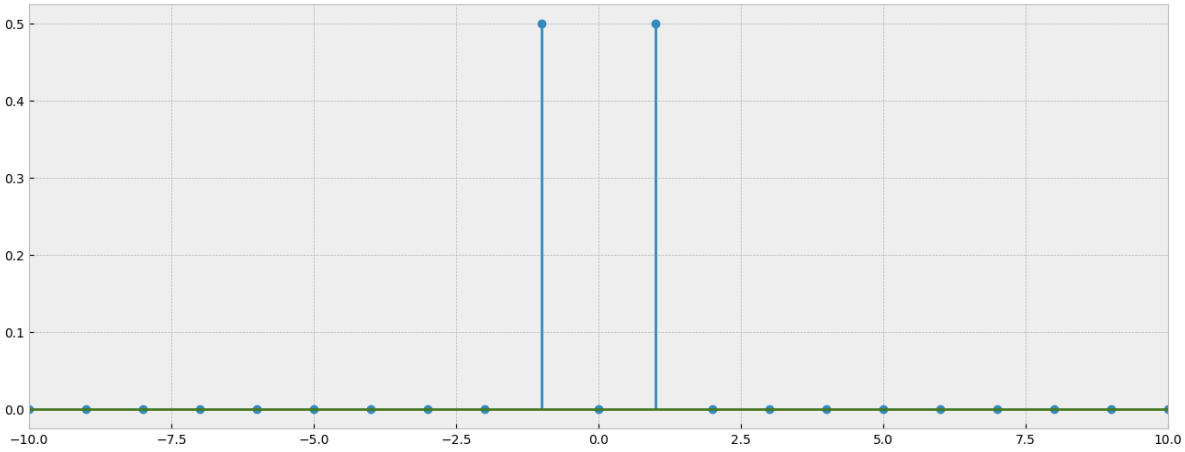
In [ ]:
```
# b) band limited signal in time domain constructed of sin function

sp.plot(sp.sin(2*t*sp.pi), (t, -3, 3))
```



Out[ ]: <sympy.plotting.plot.Plot at 0x16cde6ba9d0>

In [ ]:
```
# b) band limited signal in frequency domain constructed of sin function
# Could not use the same sp.fourier_transform function as previously since with the
# Code is heavily inspired by the second week's notebook on the course.
m = sp.sin(2*t*sp.pi)
m_converted = sp.lambdify(t, m, "numpy")
fig, ax = plt.subplots()
tp = np.linspace(0, 1, 10_000)
x = m_converted(tp)
X = fftpack.fft(x) / 10_000

freqs = fftpack.fftfreq(len(x)) * 10_000
ax.stem(freqs, np.abs(X))
ax.set_xlim(-10, 10)
```

Out[ ]: $(-10.0,\ 10.0)$



1. Comparing the 2-PAM, 4-PAM, 4-QAM, 16-QAM, 64-QAM, 4-PSK, 8-PSK and 16-PSK in an AWGN channel.

Need to calculate the symbol error function for PAM.

1. Pe ~= Ndmin*Q*(sqrt(np*Eb/2N0)), where np = (dmin)^2/Eb

- We know that dmin = sqrt(12*Es/(M^2-1)) for PAM
- M = 2^k
- Ndmin = 2
- Es = 1
- Eb = Es/log2(M) = 1/log2(M) = 1/log2(2^k)= 1/k

- np = sqrt(12*Es/(M^2-1))^2/Eb = 12/((M^2-1)/log2(M)) = 12k/(M^2-1)
- Pe = 2Q(sqrt(np*Eb/2*N0)) = 2Q(*sqrt(12kEb/2*(M^2-1)N0)) = 2Q(sqrt(6kEb/(M^2-1)N0)) = 2Q(sqrt(6/(M^2-1)*N0))

1. The same calculations for QAM

- dmin = sqrt(6Es/(M-1))
- M = 2^k
- Ndmin = 4
- np = sqrt(6Es/(M-1))^2/(1/k)
- Pe = 4Q(sqrt(np*Eb/2N0)) = *4Q(sqrt(6k*1/k)/2*(M-1)N0) = 4Q(sqrt(3/(M-1N0)))

1. The same calculations for PSK

- dmin = sqrt(Es)(*2*pi/M)
- Ndmin = 2
- np = sqrt(2Es(1-cos(2*pi/M)))^2/Eb = 2(1-cos(*2*pi/M))k
- Pe = 2Q(sqrt((1-cos(2*pi/m))/N0))

In [ ]:
```
# Creating the AWGN
awgn = komm.AWGNChannel()
noise_power_db = np.arange(-40, 4, 0.05)
noise_power = 10**(noise_power_db/10)
```

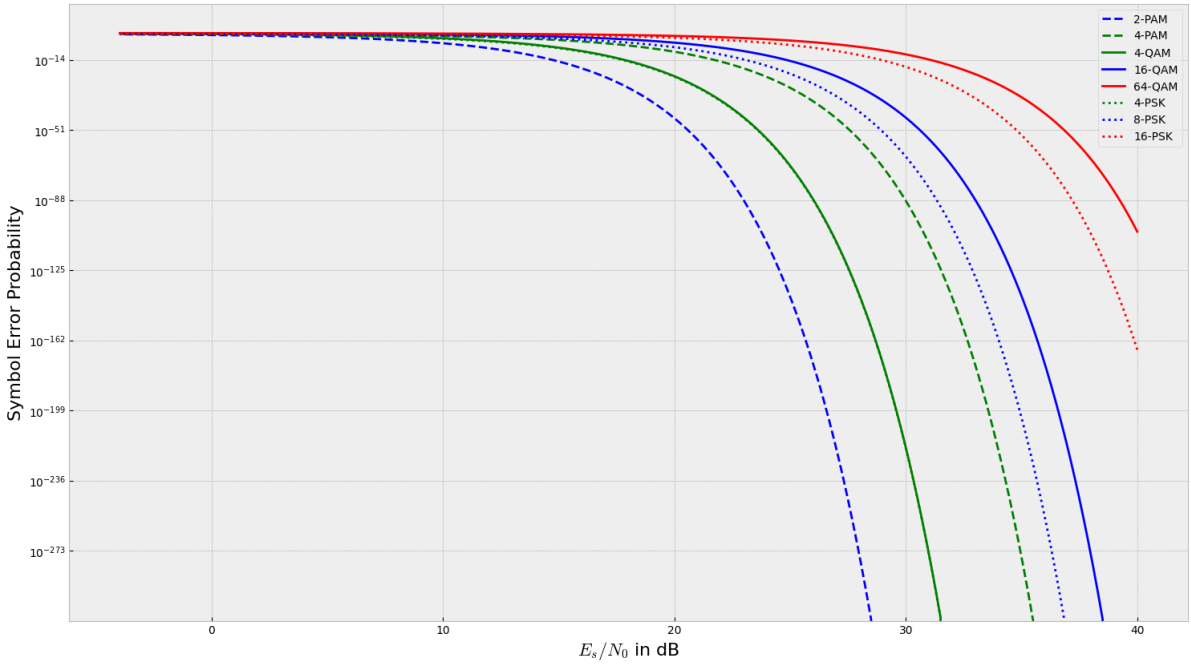
In [ ]:
```
def Q(x):
    return 1/2*scipy.special.erfc(x/np.sqrt(2))
```

In [ ]:
```
# 2-PAM
M = 2
ep_2pam = 2*Q(np.sqrt(6/((M**2-1)*noise_power)))
# 4-Pam
M = 4
ep_4pam = 2*Q(np.sqrt(6/((M**2-1)*noise_power)))
##
# 4-QAM
M = 4
ep_4qam = 4*Q(np.sqrt(3/((M-1)*noise_power)))
# 16-QAM
M = 16
ep_16qam = 4*Q(np.sqrt(3/((M-1)*noise_power)))
# 64-QAM
M = 64
ep_64qam = 4*Q(np.sqrt(3/((M-1)*noise_power)))
# 4-PSK
M = 4
ep_4psk = 2*Q(np.sqrt((1 - np.cos(2 * np.pi / M)) / noise_power ))
# 8-PSK
M = 8
ep_8psk = 2*Q(np.sqrt((1 - np.cos(2 * np.pi / M)) / noise_power ))
# 16-PSK
M = 16
ep_16psk = 2*Q(np.sqrt((1 - np.cos(2 * np.pi / M)) / noise_power ))
```

In [ ]:
```
plt.figure(1,figsize=(18,10))
plt.semilogy(10 * np.log10(1/noise_power), ep_2pam,'--b',label='2-PAM')
plt.semilogy(10 * np.log10(1/noise_power), ep_4pam,'--g',label='4-PAM')
#
plt.semilogy(10 * np.log10(1/noise_power), ep_4qam,'g',label='4-QAM')
```

```
plt.semilogy(10 * np.log10(1/noise_power), ep_16qam,'b',label='16-QAM')
plt.semilogy(10 * np.log10(1/noise_power), ep_64qam,'r',label='64-QAM')
#
plt.semilogy(10 * np.log10(1/noise_power), ep_4psk,':g',label='4-PSK')
plt.semilogy(10 * np.log10(1/noise_power), ep_8psk,':b',label='8-PSK')
plt.semilogy(10 * np.log10(1/noise_power), ep_16psk,':r',label='16-PSK')
#
plt.xlabel('$E_s/N_0$ in dB',fontsize=16)
plt.ylabel('Symbol Error Probability', color='k',fontsize=16)
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x16cdd45f050>
```



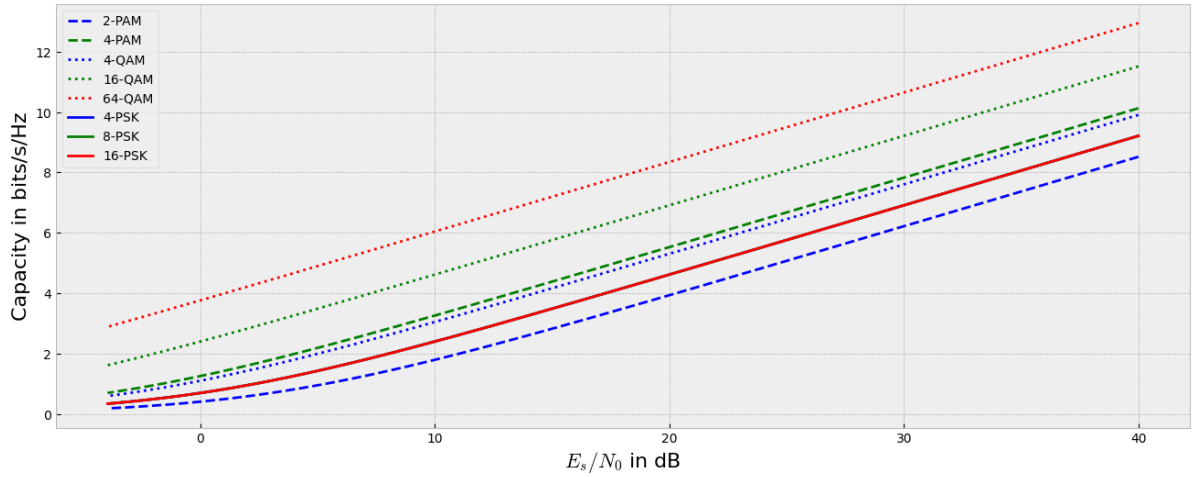
b) Comparing these modulation methods in relation to spectral efficiency and bit error probability.

In [ ]:
```
# Spectral efficiency
# 2-PAM
order = 2
base_amplitude = 1
modulation = komm.PAModulation(order, base_amplitude)
awgn.signal_power_2pam  = modulation.energy_per_symbol
awgn.snr_2pam  = awgn.signal_power_2pam  / noise_power/2
# 4-PAM
order = 4
modulation = komm.PAModulation(order, base_amplitude)
awgn.signal_power_4pam  = modulation.energy_per_symbol
awgn.snr_4pam  = awgn.signal_power_4pam  / noise_power/2
# 4-QAM
order = 4
modulation = komm.QAModulation(order, base_amplitude)
awgn.signal_power_4qam  = modulation.energy_per_symbol
awgn.snr_4qam  = awgn.signal_power_4qam  / noise_power
# 16-QAM
order = 16
modulation = komm.QAModulation(order, base_amplitude)
awgn.signal_power_16qam  = modulation.energy_per_symbol
awgn.snr_16qam  = awgn.signal_power_16qam  / noise_power
# 64-QAM
order = 64
modulation = komm.QAModulation(order, base_amplitude)
awgn.signal_power_64qam  = modulation.energy_per_symbol
```

```
awgn.snr_64qam  = awgn.signal_power_64qam  / noise_power
# 4-PSK
order = 4
modulation = komm.PSKModulation(order, base_amplitude)
awgn.signal_power_4psk  = modulation.energy_per_symbol
awgn.snr_4psk  = awgn.signal_power_4psk  / noise_power
# 8-PSK
order = 8
modulation = komm.PSKModulation(order, base_amplitude)
awgn.signal_power_8psk  = modulation.energy_per_symbol
awgn.snr_8psk  = awgn.signal_power_8psk  / noise_power
# 16-PSK
order = 16
modulation = komm.PSKModulation(order, base_amplitude)
awgn.signal_power_16psk  = modulation.energy_per_symbol
awgn.snr_16psk  = awgn.signal_power_16psk  / noise_power
```

In [ ]:
```
plt.plot(10 * np.log10(1/noise_power), np.log(1+awgn.snr_2pam),'--b',label='2-PAM')
plt.plot(10 * np.log10(1/noise_power), np.log(1+awgn.snr_4pam),'--g',label='4-PAM')
plt.plot(10 * np.log10(1/noise_power), np.log(1+awgn.snr_4qam),':b',label='4-QAM')
plt.plot(10 * np.log10(1/noise_power), np.log(1+awgn.snr_16qam),':g',label='16-QAM')
plt.plot(10 * np.log10(1/noise_power), np.log(1+awgn.snr_64qam),':r',label='64-QAM')
plt.plot(10 * np.log10(1/noise_power), np.log(1+awgn.snr_4psk),'b',label='4-PSK')
plt.plot(10 * np.log10(1/noise_power), np.log(1+awgn.snr_8psk),'g',label='8-PSK')
plt.plot(10 * np.log10(1/noise_power), np.log(1+awgn.snr_16psk),'r',label='16-PSK')
plt.xlabel('$E_s/N_0$ in dB',fontsize=16)
plt.ylabel('Capacity in bits/s/Hz', color='k',fontsize=16)
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x16cf6591450>
```



The bit error probabilities

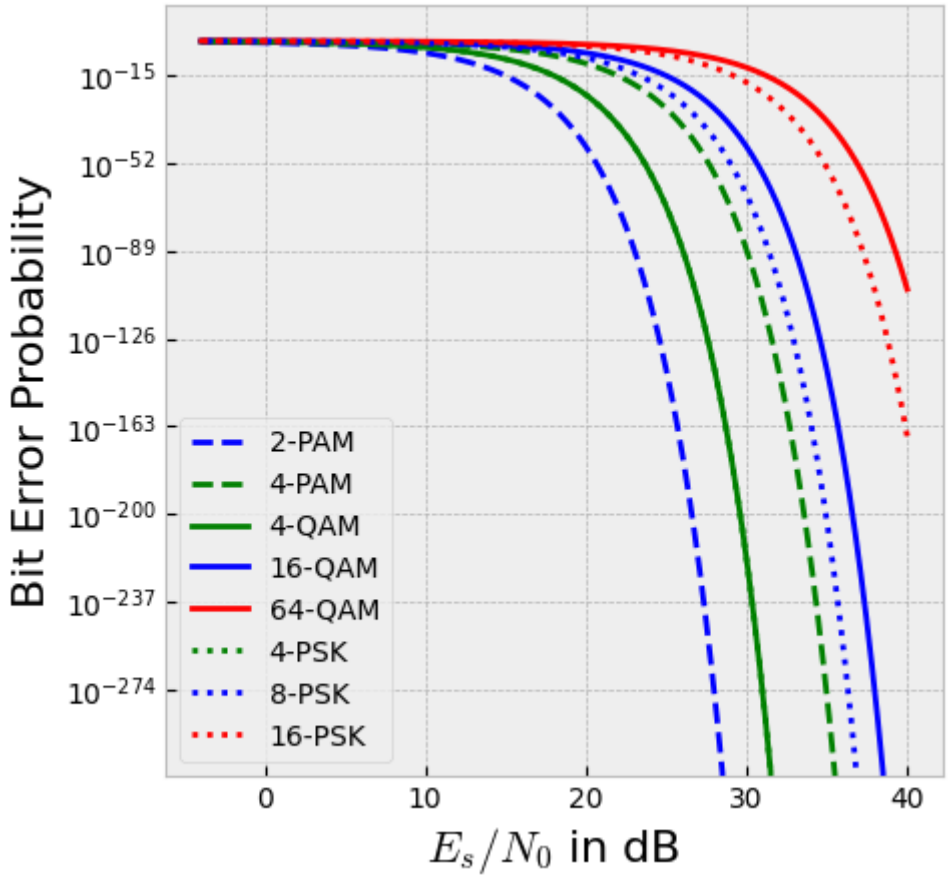
- Pb = Q(sqrt(np*Eb/2N0)), so we can use the functions defined earlier

In [ ]:
```
# Bit error probability
# 2-PAM
M = 2
eb_2pam = Q(np.sqrt(6/((M**2-1)*noise_power)))
# 4-Pam
M = 4
eb_4pam = Q(np.sqrt(6/((M**2-1)*noise_power)))
##
# 4-QAM
M = 4
eb_4qam = Q(np.sqrt(3/((M-1)*noise_power)))
# 16-QAM
```

```
M = 16
eb_16qam = Q(np.sqrt(3/((M-1)*noise_power)))
# 64-QAM
M = 64
eb_64qam = Q(np.sqrt(3/((M-1)*noise_power)))
# 4-PSK
M = 4
eb_4psk = Q(np.sqrt((1 - np.cos(2 * np.pi / M)) / noise_power ))
# 8-PSK
M = 8
eb_8psk = Q(np.sqrt((1 - np.cos(2 * np.pi / M)) / noise_power ))
# 16-PSK
M = 16
eb_16psk = Q(np.sqrt((1 - np.cos(2 * np.pi / M)) / noise_power ))
```

In [ ]:
```
plt.figure(1,figsize=(5,5))
plt.semilogy(10 * np.log10(1/noise_power), eb_2pam,'--b',label='2-PAM')
plt.semilogy(10 * np.log10(1/noise_power), eb_4pam,'--g',label='4-PAM')
#
plt.semilogy(10 * np.log10(1/noise_power), eb_4qam,'g',label='4-QAM')
plt.semilogy(10 * np.log10(1/noise_power), eb_16qam,'b',label='16-QAM')
plt.semilogy(10 * np.log10(1/noise_power), eb_64qam,'r',label='64-QAM')
#
plt.semilogy(10 * np.log10(1/noise_power), eb_4psk,':g',label='4-PSK')
plt.semilogy(10 * np.log10(1/noise_power), eb_8psk,':b',label='8-PSK')
plt.semilogy(10 * np.log10(1/noise_power), eb_16psk,':r',label='16-PSK')
#
plt.xlabel('$E_s/N_0$ in dB',fontsize=16)
plt.ylabel('Bit Error Probability', color='k',fontsize=16)
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x16cde69eb10>
```

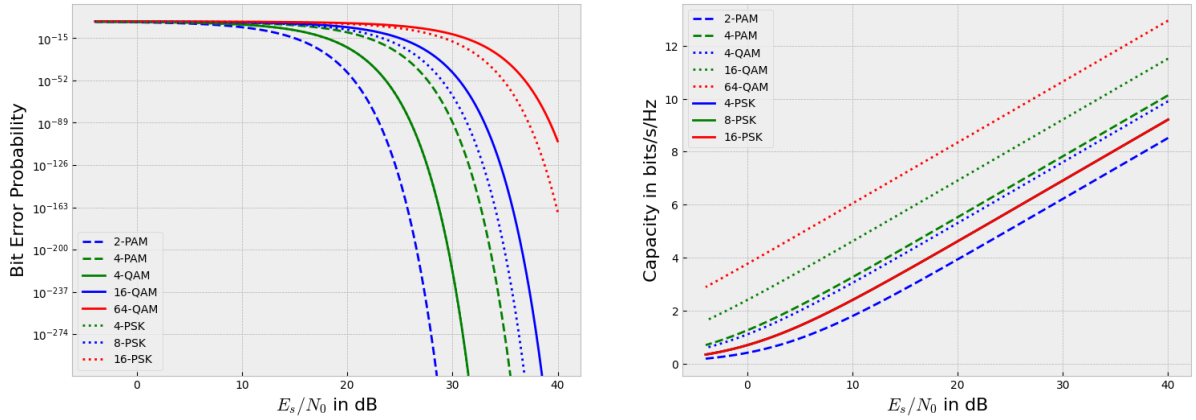


In [ ]:
```
#Plotting the figures next to each other
plt.figure(1,figsize=(18,6))
```

```
plt.subplot(121)
plt.semilogy(10 * np.log10(1/noise_power), eb_2pam,'--b',label='2-PAM')
plt.semilogy(10 * np.log10(1/noise_power), eb_4pam,'--g',label='4-PAM')
#
plt.semilogy(10 * np.log10(1/noise_power), eb_4qam,'g',label='4-QAM')
plt.semilogy(10 * np.log10(1/noise_power), eb_16qam,'b',label='16-QAM')
plt.semilogy(10 * np.log10(1/noise_power), eb_64qam,'r',label='64-QAM')
#
plt.semilogy(10 * np.log10(1/noise_power), eb_4psk,':g',label='4-PSK')
plt.semilogy(10 * np.log10(1/noise_power), eb_8psk,':b',label='8-PSK')
plt.semilogy(10 * np.log10(1/noise_power), eb_16psk,':r',label='16-PSK')
#
plt.xlabel('$E_s/N_0$ in dB',fontsize=16)
plt.ylabel('Bit Error Probability', color='k',fontsize=16)
plt.legend()

plt.subplot(122)
plt.plot(10 * np.log10(1/noise_power), np.log(1+awgn.snr_2pam),'--b',label='2-PAM')
plt.plot(10 * np.log10(1/noise_power), np.log(1+awgn.snr_4pam),'--g',label='4-PAM')
plt.plot(10 * np.log10(1/noise_power), np.log(1+awgn.snr_4qam),':b',label='4-QAM')
plt.plot(10 * np.log10(1/noise_power), np.log(1+awgn.snr_16qam),':g',label='16-QAM')
plt.plot(10 * np.log10(1/noise_power), np.log(1+awgn.snr_64qam),':r',label='64-QAM')
plt.plot(10 * np.log10(1/noise_power), np.log(1+awgn.snr_4psk),'b',label='4-PSK')
plt.plot(10 * np.log10(1/noise_power), np.log(1+awgn.snr_8psk),'g',label='8-PSK')
plt.plot(10 * np.log10(1/noise_power), np.log(1+awgn.snr_16psk),'r',label='16-PSK')
plt.xlabel('$E_s/N_0$ in dB',fontsize=16)
plt.ylabel('Capacity in bits/s/Hz', color='k',fontsize=16)
plt.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x16cfb455a90>
```

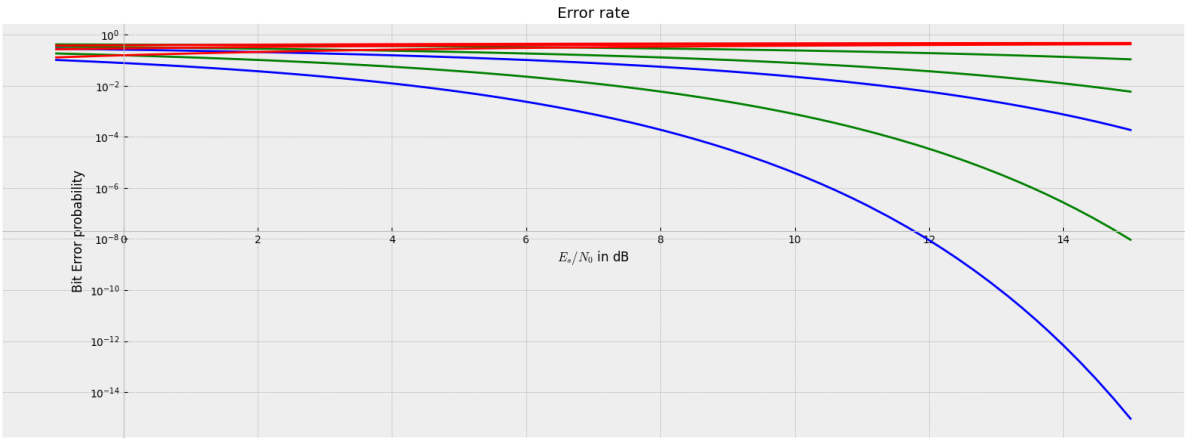


As from the plots can be seen generally the better the modulation's capacity is the worse the bit error probability is.

In [ ]:

```
# Or using the different way of doing it that gives closer look at the smaller sign

def Q(x):
    return sp.simplify(1/2 * sp.erfc(x/sp.sqrt(2)))

y = sp.Symbol("y")

plot_aux = sp.plot(
    sp.N(Q(sp.sqrt((6/((2**2-1))*10**(y/10))))),
    sp.N(Q(sp.sqrt((6/((4**2-1))*10**(y/10))))),
    sp.N(Q(sp.sqrt((3/((4-1))*10**(y/10))))),
    sp.N(Q(sp.sqrt((3/((16-1))*10**(y/10))))),
    sp.N(Q(sp.sqrt((3/((64-1))*10**(y/10))))),
    sp.N(Q(sp.sqrt((1 - sp.cos(2 * sp.pi / 4)) / 10**(y/10) ))),
    sp.N(Q(sp.sqrt((1 - sp.cos(2 * sp.pi / 8)) / 10**(y/10) ))),
```

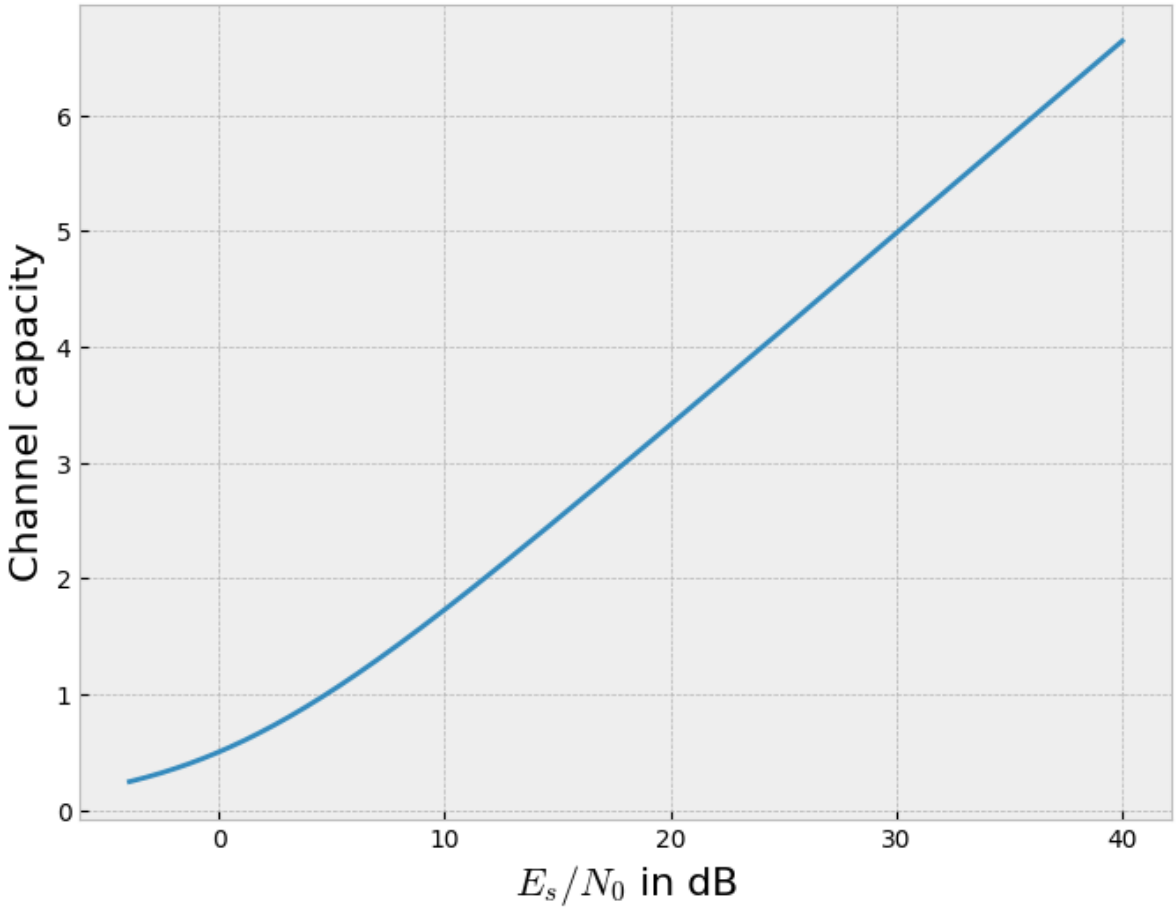
```
    sp.N(Q(sp.sqrt((1 - sp.cos(2 * sp.pi / 16)) / 10**(y/10) ))),
    (y,-1,15),
    xlabel='$E_s/N_0$ in dB',
    ylabel='Bit Error probability',
    yscale= 'log',
    title='Error rate',
    show=False)
plot_aux[0].line_color = 'b'
plot_aux[1].line_color = 'b'
plot_aux[2].line_color = 'g'
plot_aux[3].line_color = 'g'
plot_aux[4].line_color = 'g'
plot_aux[5].line_color = 'r'
plot_aux[6].line_color = 'r'
plot_aux[7].line_color = 'r'
plot_aux.show()
```



c) AWGN channel capacity as a function of SNR

In [ ]:
```
capacity_bsc = np.zeros(len(noise_power))
for i in range(len(noise_power)):
    capacity_bsc[i] = komm.AWGNChannel(snr=1/noise_power[i]).capacity()
plt.rcParams['figure.figsize'] = 8, 6
plt.plot(10 * np.log10(1/noise_power),capacity_bsc)

plt.xlabel('$E_s/N_0$ in dB',fontsize=16)
plt.ylabel('Channel capacity',fontsize=16)
```

Out[ ]:
```
Text(0, 0.5, 'Channel capacity')
```



d) The performance of the modulations could be improved by changing the signal's power. This would result in a lower Signal to Noise Ratio which would help the transmission go through without the probability of the noise affecting it too much. The best modulation method that mimizes the bit and symbol error's is the 2-PAM which can be seen in the figures performing the best regarding the errors. Unfortanetly it has the worst capacity of the modulation methods meaning that the transmission would be quite slow compared to other modulation methods. (The question was quite shady so I wasn't sure if the overall performance meaning all of the modulation methods were supposed to be considered or picked the best modulation method for the lowest error probability. So I provided the answer to both scenarios.)

To know how good my solution is one can look at the bit error and symbol error graphs and note that 2-PAM provides the best performance in regards of error probability both in bit error and symbol error. And to know how much the power of the signal affects the error probabilities, one can select a modulation method and see how the signal power has influence over the error probability.

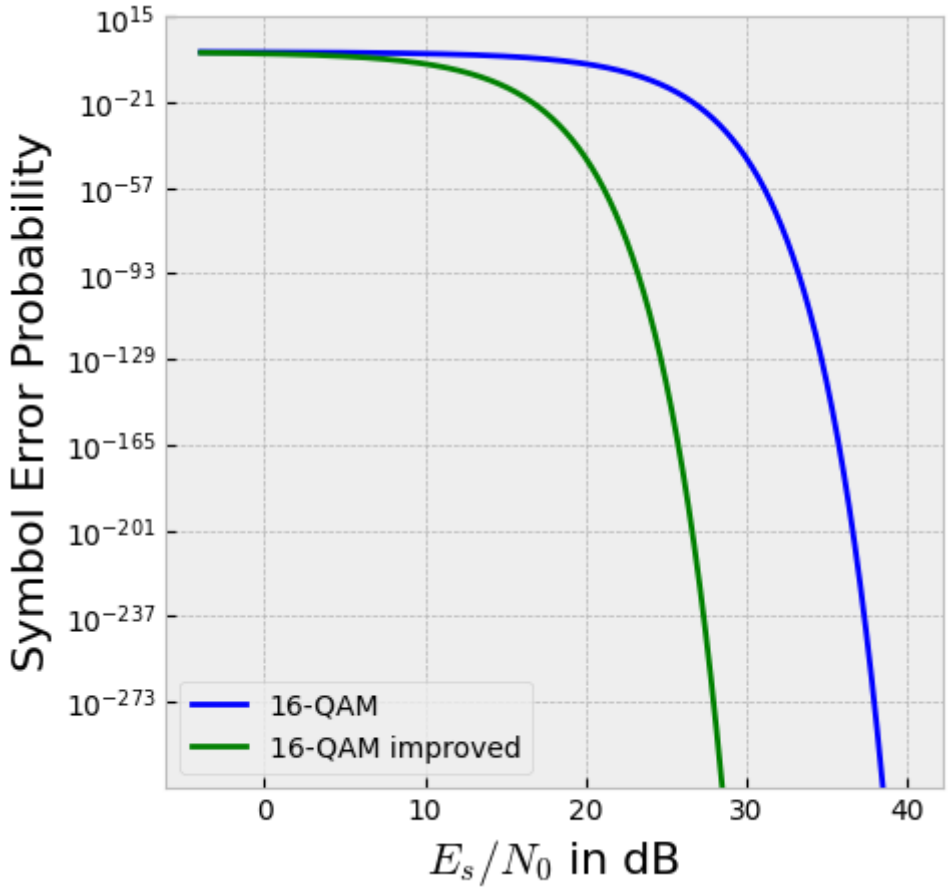
In [ ]:
```
def Q(x):
    return 1/2*scipy.special.erfc(x/np.sqrt(2))
```

In [ ]:
```
# 16-QAM
M = 16
ep_16qam = 4*Q(np.sqrt(3/((M-1)*noise_power)))
eb_16qam_improved = 4*Q(np.sqrt(3*10/((M-1)*noise_power)))
```

In [ ]:
```
plt.figure(1,figsize=(5,5))

plt.semilogy(10 * np.log10(1/noise_power), ep_16qam,'b',label='16-QAM')
```

```
plt.semilogy(10 * np.log10(1/noise_power), eb_16qam_improved,'g',label='16-QAM impr

plt.xlabel('$E_s/N_0$ in dB',fontsize=16)
plt.ylabel('Symbol Error Probability', color='k',fontsize=16)
plt.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x16cff0001d0>



In the example above the energy per bit is increased 10x. The result can be seen in the figure that the improved version does infact get much lower error probability compared to the original modulation method.