University of Glasgow | James Watt School of Engineering

1 Digital Communication 4: OFDM Communications Link

1.1 Introduction

This coding project will introduce you to Orthogonal Frequency Division Multiplexing, simulating a communications link with multipath interference and incorporating Reed-Solomon Channel Coding.

If you are using your own computer, make sure the Python libraries `scipy`, `numpy`, and `matplotlib` are installed. It is recommended that you use a suitable IDE for your project, such as Spyder, PyCharm or Visual Studio. Tip: if you are using Spyder, you can display, manipulate and save graphics in separate windows by setting

`Tools>Preferences>IPython console>Graphics>Backend`

from `Inline` to `Automatic`.

You will need to download and install, if required, the following programmes or libraries:

- `komm` Roberto Nobrega's library, which we used in the previous coding projects
  `pip install komm`
- `pyofdm` by Bernd Porr and David Hutchings `https://pypi.org/project/pyofdm/`
  `pip install pyofdm`
- `reedsolo` a python library to implement Reed-Solomon channel code and decode `https://pypi.org/project/reedsolo/` which can be installed using the command `pip install reedsolo`
- `Audacity` open source audio editor available for Windows, Mac or Linux at `https://audacityteam.org/download`

Each project will be scheduled over a two week period, within which there will be 2 scheduled consultation sessions where you will be able to ask teaching staff for guidance. The project should be written up as a short report describing what you've done and the results you have taken along with any conclusions that you draw. Include your python code(s) in the appendices. Make sure your name and student ID number is on the report. The report should be uploaded to the Moodle assignment by the stated deadline, either using Moodle's inbuilt html editor, or as a single PDF file.

The bespoke OFDM and standard modules to be imported are

```
from PIL import Image
import numpy as np
import scipy.io.wavfile as wav
import pyofdm.codec
import pyofdm.nyquistmodem
import matplotlib.pyplot as plt
```

University of Glasgow | 詹姆斯*瓦特工程学院

1 数字通信4：OFDM通信链路

1.1 Introduction

本编码项目将向您介绍正交频分复用，模拟具有多径干扰的通信链路，并结合里德-所罗门信道编码。

如果您使用的是自己的计算机，请确保安装了Python库scipy，numpy和matplotlib。建议您为项目使用合适的IDE，例如Spyder，PyCharm或VisualStudio。提示：如果您使用的是Spyder，则可以通过设置在单独的窗口中显示，操作和保存图形

`Tools>Preferences>IPython console>Graphics>Backend`

从内联到自动。

如果需要，您需要下载并安装以下程序或库:

*kommRobertoNobrega的库，我们在之前的编码项目中使用它来安装komm

*BerndPorr和DavidHutchings的pyofdmhttps：pypi.org项目pyofdmpip安装pyofdm

*reedsolo一个python库来实现Reed-Solomon通道代码和解码
https:pypi.org可以使用命令pip安装reedsolo安装的项目reedsolo

*Audacity开源音频编辑器适用于Windows，Mac或Linux，https：audacityteam.org下载

每个项目将安排在两个星期内，在这两个星期内将有两个预定的咨询会议，你将能够要求教学人员指导。这个项目应该写成一份简短的报告，描述你所做的事情和你所取得的结果以及你得出的任何结论。在附录中包含您的python代码。确保您的姓名和学生证号码在报告上。报告应该在规定的截止日期之前上传到Moodle分配，或者使用moodle的内置html编辑器，或者作为单个PDF文件。

要导入的定制OFDM和标准模块是

从PIL导入图像导入numpy作为np导入scipy.io.wavfile作为wav导入pyofdm。编解码器导入pyofdm。nyquistmodem导入matplotlib.pyplot作为plt

1.2 OFDM Transmitter

The arguments and default values (corresponding to 802.11g WiFi) for the OFDM encoder and decoder module are,

```
pyofdm.codec.OFDM(nFreqSamples=64, pilotIndices=[-21, -7, 7, 21],
    pilotAmplitude=1, nData=12, fracCyclic=0.25, mQAM=2)
```

Energy dispersal is done with a pre-seeded random number generator. Both pilot tones and the cyclic prefix are added so that the start of the symbol can be detected at the receiver. The complex time series after the inverse Fourier Transform is modulated into a real valued stream with a Nyquist quadrature modulator. On the receiver side the start of the symbol is detected by first doing a coarse search with the cyclic prefix and then a precision alignment with the pilots.

`nFreqSamples` sets the number of frequency coefficients of the IFFT (should be a power of 2). Pilot tones are injected at the values from the list `pilotIndices`. The real valued pilot amplitude is `pilotAmplitude` For transmission `nData` bytes are expected in an array. The relative length of the Cyclic prefix is `fracCyclic`; the default cyclic prefix is a $\frac{1}{4}$ of the length of the symbol. `mQAM` is the QAM order and the default `mQAM=2` is identical to QPSK.

Using the following code to initialise the parameters of the OFDM module, commensurate with the DVB-T 2k standard. The supplied routine `pyofdm.codec.setpilotindex()` provides the list of equally spaced pilot tones.

```
# Number of total frequency samples
totalFreqSamples = 2048

# Number of useful data carriers / frequency samples
sym_slots = 1512

# QAM Order
QAMorder = 2

# Total number of bytes per OFDM symbol
nbytes = sym_slots*QAMorder//8

# Distance of the evenly spaced pilots
distanceOfPilots = 12
pilotlist = pyofdm.codec.setpilotindex(nbytes,QAMorder,distanceOfPilots)

ofdm = pyofdm.codec.OFDM(pilotAmplitude = 16/9,
                         nData=nbytes,
                         pilotIndices = pilotlist,
                         mQAM = QAMorder,
                         nFreqSamples = totalFreqSamples)
```

First, encode a single (complex) symbol using random data bytes,

```
row = np.random.randint(256,size=nbytes,dtype='uint8')
complex_signal = ofdm.encode(row)
```

Inspect the complex valued time series `complex_signal` in the variable explorer, and by plotting the real and imag components as shown below. Can you account for its length? Can you identify any feature?

1.2 OFDM Transmitter

OFDM编码器和解码器模块的参数和默认值（对应于802.11gWiFi）为

```
pyofdm.codec.OFDM(nFreqSamples=64, pilotIndices=[-21, -7, 7, 21],
    pilotAmplitude=1, nData=12, fracCyclic=0.25, mQAM=2)
```

能量分散是用预种子随机数发生器完成的.导频音调和循环前缀两者都被添加，使得码元的开始可以在接收机处被检测到。傅里叶逆变换后的复数时间序列用奈奎斯特正交调制器调制成实值流。在接收机端，通过首先使用循环前缀进行粗搜索，然后与导频进行精确对齐来检测符号的开始。nFreqSamples设置IFFT的频率系数的数量（应该是2的幂）。从列表pilotIndices的值注入导频音调。实值导频幅度是pilotAmplitude用于传输nData字节预期在阵列中。循环前缀的相对长度为fracCyclic；默认循环前缀为1

4的符号的长度。mQAM是QAM顺序，默认mQAM=2与QPSK相同。

使用下面的代码初始化OFDM模块的参数，与DVB-T2K标准相称。提供的例程pyofdm。编解码器。setpilotindex()提供等间距导频音调的列表。

```
#总频率样本数totalFreqSamples=2048
#有用数据载波频率样本数sym_slots=1512

# QAM Order
QAMorder = 2

#每个OFDM符号的总字节数nbytes=sym_slots*QA
Morder8

#平均间隔飞行员的距离distanceOfPilots=12pilotlist=pyofdm。编解码器。setpilotind
ex（nbytes，QAMorder，distanceOfPilots）

ofdm = pyofdm.codec.OFDM(pilotAmplitude = 16/9,
                         nData=nbytes,
                         pilotIndices = pilotlist,
                         mQAM = QAMorder,
                         nFreqSamples = totalFreqSamples)
```

首先，使用随机数据字节对单个（复杂）符号进行编码

```
row = np.random.randint(256,size=nbytes,dtype='uint8')
complex_signal = ofdm.encode(row)
```

检查变量资源管理器中的复值时间序列complex_signal，并通过绘制实分量和实分量，如下所示。你能解释它的长度吗?你能识别任何特征吗?

```
plt.figure()
plt.title('OFDM Symbol')
plt.plot(complex_signal.real)
plt.plot(complex_signal.imag)
plt.show()
```

Now inspect the abs value of its discrete Fourier transform of the symbol without the prefix by plotting the following.

```
plt.figure()
plt.title("OFDM complex spectrum")
plt.xlabel("Normalised frequencies")
plt.ylabel("Frequency amplitudes")
plt.plot(np.abs(np.fft.fft(complex_signal[-totalFreqSamples:])/totalFreqSamples))
plt.show()
```

Note that in the pyofdm code, we have yet to incorporate an appropriate shift to the sub-carrier assignments so that the OBW (Occupied BandWidth) is optimally located in the centre of the complex spectrum. However, the code as-is provides the functionality sufficient for the investigations in the current assignment.

A number of 8 bit depth grayscale (pgm) images of various sizes have been provided for you to use in this coding project. You are also free to try with your own images but make sure you know how they can be represented as arrays of 8 bit integers (bytes). Replace the random byte stream with one of the provided (or your own) images which should be read into a `numpy` array of type `uint8` using the `PIL.Image` module. In this assignment (unlike previous), ensure you leave the data in **bytes**, which is the appropriate datatype for `pyofdm` (and `reedsolo` later). It is advised to recast your data array into a one-dimensional array of a length (taken to be `tx_byte` in the code snippet below) which is a whole multiple of `nbytes` prior to OFDM encoding, appending an appropriate number of dummy bytes. The OFDM encoding can be performed and appended to `complex_signal`

```
complex_signal = np.array([ofdm.encode(tx_byte[i:i+nbytes])
    for i in range(0,tx_byte.size,nbytes)]).ravel()
```

The stream can be modulated at the Nyquist frequency to give a sampled output using the supplied routine. Add some random length zero data to the start of the double-sampled, real-valued baseband signal in order to demonstrate the symbol start search later. Save the result as a *wav* file, noting that 44.1 kHz is the standard consumer digital audio sampling rate.

```
base_signal = pyofdm.nyquistmodem.mod(complex_signal)
# add some random length dummy zero data to the start of the signal here

# save it as a wav file
wav.write('ofdm44100.wav',44100,base_signal)
```

1.3 OFDM Receiver

Create a separate file for the OFDM receiver code. The transmitted data in the form of a *wav* file is read in using. You append additional zeros to the end of the data so the start search algorithm does not reach the end-of-data.


```
plt.figure()
plt.title('OFDM Symbol')
plt.plot(complex_signal.real)
plt.plot(complex_signal.imag)
plt.show()
```


现在通过绘制以下内容来检查没有前缀的符号的离散傅立叶变换的abs值。


```
plt.figure()
plt.title("OFDM complex spectrum")
plt.xlabel("Normalised frequencies")
plt.ylabel("Frequency amplitudes")
plt.plot(np.abs(np.fft.fft(complex_signal[-totalFreqSamples:])/totalFreqSamples))
plt.show()
```


请注意，在pyofdm代码中，我们还没有将适当的移位纳入子载波分配，以便OBW（占用带宽）最佳地位于复杂频谱的中心。但是，代码原样提供的功能足以用于当前分配中的调查。

提供了许多不同大小的8位深度灰度（pgm）图像供您在这个编码项目中使用。您也可以自由尝试自己的图像，但请确保您知道如何将它们表示为8位整数（字节）的数组。将随机字节流替换为提供的（或您自己的）图像之一，该图像应该使用PIL读取到uint8类型的numpy数组中。像模块。在此分配中（与以前不同），请确保将数据保留为字节，这是pyofdm（以及稍后的reedsolo）的适当数据类型。

建议在OFDM编码之前将数据数组重塑成一个长度为tx_byte的一维数组（在下面的代码片段中取为tx_byte），该数组是nbytes的整数倍，并附加适当数量的虚拟字节。可以执行OFDM编码并将其附加到complex_signal

complex_signal=np。阵（[ofdm.encode(tx_byte[i:i+nbytes])foriinrange(0 tx
_byte.大小，nbytes)])。拉威尔()

该流可以在奈奎斯特频率调制，以提供一个采样输出使用提供的例程。在双采样、实值基带信号的开始处添加一些随机长度零数据，以便稍后演示符号开始搜索。将结果保存为wav文件，注意到44.1kHz是标准的消费者数字音频采样率。

base_signal=pyofdm。奈奎斯莫德姆。mod(complex_signal)#在这里的信号开始处添加一些随机长度的伪零数据

#保存为wav文件wav。写('ofdm44100.wav'，44100，b
ase_signal)

1.3 OFDM Receiver

为OFDM接收机代码创建单独的文件。在使用中读取wav文件形式的传输数据。您将附加零附加到数据的末尾，以便开始搜索算法不会到达数据的末尾。

```
samp_rate, base_signal = wav.read('ofdm44100.wav')
# append some extra zeros to the base_signal here

complex_signal = pyofdm.nyquistmodem.demod(base_signal)
```

As described in lectures, it is necessary to accurately finding the start of the OFDM symbol. Here we use a two-step process. First, make use of the Cyclic Prefix by examining the cross-correlation at `totalFreqSamples`, and then second using the Pilot Tones by examining the sum of the squares of the imaginary component of the expected pilot tones. The `ofdm.findSymbolStartIndex` module performs both of these methods sequentially.

```
searchRangeForPilotPeak = 8
cc, sumofimag, offset = ofdm.findSymbolStartIndex(complex_signal,
    searchrangefine = searchRangeForPilotPeak)
print('Symbol start sample index =',offset)
```

Inspect the value of `offset`. Does it correspond to the previously inserted dummy data in your transmitter code? Plot the quantity `cc` to show the cross-correlation and identify the peak corresponding to this offset. Plot the quantity `sumofimag`, given for the range `[-search_range:search_range]` and confirm that it is substantially lower than the others for one unique delay value.

Initialise the OFDM decoder and decode one symbol at a time as follows. You will need to determine the expected total number of OFDM symbols `Nsig_sym`

```
ofdm.initDecode(complex_signal,offset)
rx_byte = np.uint8([ofdm.decode()[0] for i in range(Nsig_sym)]).ravel()
```

Display the received data as an image and confirm that it matches your original image as used in the transmitter code. Determine the bit error ratio (you will need to read in the original image data), which should be zero for this back-to-back simulation if the offset is correctly determined, as we haven't yet included distortion or noise.

1.4 Distortion and Noise in the Communications Channel

The `Audacity` open source audio editor will be used for adding noise and distortion. Open the *wav* file created by your OFDM transmitter code. Play a few seconds of it and comment on how it sounds.

Multipath Interference can be simulated using the *reverb* effect. Select `Effect>Reverb` and apply the default. Export the result as a 32-bit floating point *wav* file and apply your OFDM receiver code to it (I suggest using a different file name so that you avoid overwriting your original signal). Include an example of the resulting image in your report and note the bit error ratio value. You can vary the *reverb* options to explore the degradation of the signal and note the `ber` (remember to load the original signal into Audacity each time). Try at least 2 more of the *reverb* factory settings available under the *Manage* selection.

You can also explore the effects of additive white noise. If necessary, under the `Analyze` menu enable the `Measure RMS` plugin. Create a white noise track using `Generate>Noise`. Use `Analyze>Measure RMS` to obtain values for both the signal and noise track and hence note the value (in dB) for the signal-to-noise ratio. Mix the tracks using `Tracks>Mix` and export the result. Again, apply your OFDM receiver code and note the `ber`. Repeat for an additional couple of different signal-to-noise ratios by adjusting the noise amplitude.

samp_rate，base_signal=wav。读('ofdm44100.wav')#在这里的
base_signal上附加一些额外的零

```
complex_signal = pyofdm.nyquistmodem.demod(base_signal)
```

如上所述，有必要准确地找到OFDMsymbol的开始。
在这里，我们使用两个步骤的过程。首先，通过检查totalFreqSamples处的互相关来使用循环前缀，然后通过检查预期导频音调的虚部分量的平方和来使用导频音调。的ofdm。findSymbolStartIndex模块按顺序执行这两种方法。

searchRangeForPilotPeak=8cc，sumofimag，offset=ofdm。findSymbolStartIndex(complex_signal searchrangefine=searchRangeForPilotPeak)print('符号开始样本索引=' 偏移量)

检查offset的值。 它是否对应于先前插入的虚拟数据
在你的发射机代码里？绘制数量cc以示出互相关并识别对应于此偏移的峰值。
绘制针对范围[-search_range:search_range]给出的数量sumofimag，并确认对于一个唯一延迟值，它大大低于其他值。

初始化OFDM解码器并且如下每次解码一个符号。您将需要确定OFDM符号的预期总数Nsig_sym

```
ofdm.initDecode(complex_signal,offset)
```

rx_byte=np。uint8([ofdm.decode()[0]foriinrange(Nsig_sym)])。拉威尔()
将接收到的数据显示为图像，并确认它与发射机代码中使用的原始图像匹配。确定误码率（您将需要在原始图像数据中读取），如果偏移正确确定，则该误码率应为零，因为我们尚未包含失真或噪声。

1.4 通信信道中的失真和噪声

Audacity开源音频编辑器将用于添加噪音和失真。打开由OFDM发射机代码创建的wav文件。播放几秒钟，并评论它的声音。

可以使用混响效果来模拟多径干扰。选择效果>混响并应用默认值。将结果导出为32位浮点wav文件，并将您的OFDM接收器代码应用于它（我建议使用不同的文件名，以便避免复盖原始信号）。在报告中包含生成图像的示例，并记下误码率值。您可以改变混响选项来探索信号的退化并注意ber（记住每次将原始信号加载到Audacity中）。尝试至少2更多的混响出厂设置管理选择下可用。

您还可以探索加性白噪声的影响。如有必要，在"分析"菜单下启用"测量RMS"插件。使用生成>噪声创建白噪声轨道。使用Analyze>MeasureRMS获取信号和噪声轨迹的值，从而记下信噪比的值（以dB为单位）。混合使用轨道>混合和导出结果的轨道。再次，应用您的OFDM接收器代码并注意ber。通过调整噪声幅度，重复额外的几个不同的信噪比。

1.5 Reed-Solomon Channel Coding

In this final section forward error encoding will be incorporated using the Reed-Solomon method. We will use the popular ($N = 255, K = 223$) code which can correct up to 16 symbol (byte) errors in the code word. The RSC import and initialisation is done as follows,

```
from reedsolo import RSCodec
from reedsolo import ReedSolomonError #only required in receiver

N, K = 255, 223
rsc = RSCodec(N-K, nsize=N)
```

Prior to RSC encoding, we need to ensure that the number of bytes is a multiple of K, and therefore additional zero bytes should be added if required.

```
tx_byte = np.append(np.array(tx_im, dtype='uint8').flatten(),
    np.zeros(K-tx_im.size[1]*tx_im.size[0]%K, dtype='uint8'))
tx_enc = np.empty(0,'uint8')
for i in range(0,tx_im.size[1]*tx_im.size[0],K):
    tx_enc = np.append(tx_enc,np.uint8(rsc.encode(tx_byte[i:i+K])))
```

Now treat `tx_enc` as the data input stream to the OFDM encoder.

For the OFDM receiver code, first the OFDM start is identified and the OFDM decoding performed. Make sure you decode the appropriate number of OFDM symbols. The Reed-Solomon decoding is then done using the following. If the decoding is unsuccessful due to the number of byte errors, an exception will be raised. In that case, since the code is implemented in a systematic fashion, we can ignore the parity bytes and use the received data bytes which will undoubtedly contain errors.

```
rx_byte = np.empty(0, dtype='uint8')
for i in range(0,tx_im.size[1]*tx_im.size[0]*N//K,N):
    try:
        rx_byte = np.append(rx_byte,np.uint8(rsc.decode(rx_enc[i:i+N])[0]))
    except ReedSolomonError:
        rx_byte = np.append(rx_byte,rx_enc[i:i+K])
```

Display your recovered data as an image and compare to your original image as used in the transmitter code. Determine the bit error ratio, identifying the change in bit error ratios due to addition of the Reed-Solomon channel coding. Additionally, in the cases of white noise, compare your ber dependence on snr to the RSC(255,223) QPSK plot in the lecture notes.

1.6 Documentation

python 3 `https://docs.python.org/3/`

numpy and scipy `https://docs.scipy.org/doc/`

matplotlib `https://matplotlib.org/contents.html`

spyder `https://docs.spyder-ide.org/`

1.5 Reed–Solomon信道编码

在最后一节中，将使用Reed–Solomon方法合并前向错误编码。我们将使用流行的（ =255， =223）代码，它可以纠正代码字中最多16个符号（字节）错误。RSC导入和初始化操作如下

从reedsolo导入RSCodec从reedsolo导入ReedSolomonError＃仅在接收器中需要

```
N, K = 255, 223
rsc = RSCodec(N-K, nsize=N)
```

在RSC编码之前，我们需要确保字节数是 的倍数，因此如果需要，应该添加额外的零字节。

tx_byte=np。追加(np.数组（tx_im，dtype='uint8'）。flatten（），np。零（K–tx_im.尺寸[1]*tx_im。size[0]%K dtype='uint8'))tx_enc=np。empty(0 'uint8')foriinrange(0 tx_im.尺寸[1]*tx_im。尺寸[0] K):

```
    tx_enc = np.append(tx_enc,np.uint8(rsc.encode(tx_byte[i:i+K])))
```

现在将tx_enc视为到OFDM编码器的数据输入流。
对于OFDM接收机码，首先识别OFDM开始并且执行OFDM解码。确保解码适当数量的OFDM符号。里德所罗门解码然后使用以下完成。如果由于字节错误数导致解码不成功，则会引发异常。在这种情况下，由于代码以系统的方式实现，我们可以忽略奇偶校验字节并使用接收到的数据字节，这些字节无疑会包含错误。

rx_byte=np。empty(0 dtype='uint8')foriinrange(0 tx_im.尺寸[1]*tx_im。size[0]*NK N):尝试:

```
        rx_byte = np.append(rx_byte,np.uint8(rsc.decode(rx_enc[i:i+N])[0]))
    except ReedSolomonError:
        rx_byte = np.append(rx_byte,rx_enc[i:i+K])
```

将恢复的数据显示为图像，并与发射机代码中使用的原始图像进行比较。确定误码比，识别由于添加了里德–所罗门信道编码而导致的误码比的变化。此外，在白噪声的情况下，将您对snr的ber依赖性与讲义中的RSC(255 223)QPSK图进行比较。

1.6 Documentation

python 3 `https://docs.python.org/3/`

numpy and scipy `https://docs.scipy.org/doc/`

matplotlib `https://matplotlib.org/contents.html`

spyder `https://docs.spyder-ide.org/`

Getting the python libraries

If you are using your own computer, make sure the Python libraries `scipy`, `numpy` and `matplotlib` are installed. These libraries are installed by default with the Anaconda python distribution. It is recommended that you use a suitable IDE for your project, such as Spyder.

获取python库

如果您使用自己的计算机，请确保安装了Python库scipy，numpy和matplotlib。这些库默认与Anacondapython发行版一起安装。建议您为项目使用合适的IDE，例如Spyder。