Preface

These notes are being updated to better reflect the contents of MATH32032 which is taught in the second semester of the 2017/18 academic year. An effort has been made to separate the examinable material from background and historical notes. The rules, however, are as follows: everything which was taught in class (lectures, tutorials) is examinable. You should therefore be guided by your own notes, supplemented by slides and podcasts where available and provided by the lecturer.

Questions and comments on these lecture notes should be directed to Yuri.Bazlov@manchester.ac.uk.
Chapter 1

Introduction (version 2019-02-10)

This chapter is devoted to general remarks and motivation. The definitions contained here are examinable.

What is information?

It is fair to say that our age is the age of information. Huge quantities of information and data literally flow around us and are stored in various forms.

Information processing gives rise to many mathematical questions. Information needs to be processed because we may need, for example, to:

- store the information;
- encrypt the information;
- transmit the information.

For practical purposes, information needs to be stored efficiently, which leads to problems such as compacting or compressing the information. For the purposes of data protection and security, information may need to be encrypted. We will NOT consider these problems here.

In this course, we will address problems that arise in connection with information transmission.

We do not attempt to give an exhaustive definition of information. Whereas some mathematical models for space, time, motion were developed hundreds of years ago, the mathematical theory of information was only born in 1948 in the paper A Mathematical Theory of Communication by Claude Shannon (1916–2001). The following will be enough for our purposes:
Definition (information, alphabet, symbol). Fix a set $F$ and call it \textit{the alphabet}. Elements of $F$ are called \textit{symbols}.

By \textit{information}, we mean a stream (a sequence) of symbols.

Remark. The alphabet should contain at least two symbols, otherwise any sequence of symbols will contain zero information. The philosophical explanation for this is: if there is only one possible we know each symbol in the sequence, hence reading the sequence does not give us any new knowledge.

What does it mean to transmit information?

It means that symbols are sent by one party (the sender) and are received by another party (receiver). The symbols are transmitted via some medium, which we will in general refer to as \textit{the channel}. More precisely, the channel is a mathematical abstraction of various real-life media such as a telephone line, a satellite communication link, a voice (in a face to face conversation between individuals), a CD (the sender writes information into it — the user reads the information from it), etc.

In this course we will assume that when a symbol is fed into the channel (the input symbol), the same or another symbol is read from the other end of the channel (the output symbol). Thus, we will only consider channels where neither erasures (when the output symbol is unreadable) nor deletions (when some symbols fed into the channel simply disappear) occur. Working with those more general channels requires more advanced mathematical apparatus which is beyond this course.

Importantly, we assume that there is \textit{noise} in the channel, which means that the symbols are randomly changed by the channel. Our \textit{simple model of information transmission} is thus as follows:

```
| sender | stream of symbols | channel | stream of symbols with random changes | receiver |
```

When a single symbol $x \in F$ is being transmitted, there are two possible outcomes:

- The received symbol is $x$. We say that no error occurred in this symbol.

- The received symbol is $y \neq x$. An error occurred in this symbol.

We will now give the most basic example of a channel.

Definition (binary alphabet, bit). The \textit{binary alphabet} is the set $\{0, 1\}$. A \textit{bit} is the same as \textit{binary symbol}, an element of the binary alphabet.
Definition \((BSC(p))\). The **binary symmetric channel with bit error rate** \(p\), denoted \(BSC(p)\), is a channel which transmits binary symbols according to the following rule. A bit (0 or 1), transmitted via the channel, arrives unchanged with probability \(1 - p\), and gets flipped with probability \(p\):

\[
\begin{array}{c}
0 \quad 1 - p \\
\downarrow \quad p \\
\downarrow \\
1 \quad 1 - p \\
\end{array}
\]

The error in any given bit is an event which is independent of all the previous bits. We thus say that this channel is **memoryless**.

**How to detect and/or correct transmission errors?**

Coding Theory is motivated by the need to protect information from transmission errors, thus making information transmission more robust. In this Introduction, we will consider a simple case (which predates mathematical coding theory) to illustrate the idea.

**Example.** A stream of binary symbols needs to be transmitted via a noisy channel. Assume the channel to be \(BSC(0.01)\), i.e., there is a one in a hundred chance that the transmitted bit is inverted by the channel. (In practical terms, this is a very high error rate — probably at the top end of possible error rate of ancient copper wire modems.)

**Approach # 1.** Send every bit as is. An error occurs if 0 is changed to 1 or vice versa. But looking at the received bit, which is 0 or 1, we have no way to determine whether an error has occurred.

**Approach # 2.** Repeat each bit three times. So, instead of 0 send 000, and instead of 1 send 111. (In the next chapter, we will say that we are sending one of two *codewords* of length 3.)

Suppose that the received word is 101. What was transmitted?

Intuitively, the likelihood is that 111 was sent. Indeed, we can observe that it takes only one error to change 111 to 101, but it takes two errors to change 000 into 101. We work under the following reasonable assumption: *a smaller number of errors in a codeword is more likely to occur that a larger number of errors*. We therefore **decode** the received word 101 as 111, as this most likely is the codeword which was transmitted.
It is easy to see that the original codeword will be correctly decoded if no more than 1 error occurs in transmission of a codeword. But this comes at a price of multiplying the amount of information transmitted by 3.

Also, if one or two bits get flipped in a codeword of length 3, the receiver will know that errors have occurred — i.e., will detect an error. This is because the received word will not be 000 nor 111, hence could not have been transmitted.

Let us check whether tripling each bit really improves detection of errors. Under Approach # 1, there is a 1% likelihood of an error in a given bit; there is no way to detect the error. Under Approach # 2, the only way for an error to go undetected is 3 consecutive bit errors. The probability of that is $0.01^3 = 0.0001\%$. Hence the error detection has been improved by a factor of 10,000.

We will formalise the process of transmitting information using an error-correcting code in the next chapter.

**The Hamming distance**

Codes have been used for error correction for thousands of years: a natural language is essentially a code! If we “receive” a corrupted English word such as PHEOEEM, we will assume that it has most likely been THEOREM, because this would involve fewest mistakes.

The number of errors, or more generally the number of symbol differences between two given words is the most basic notion in Coding Theory. The following formal definition is credited to Richard Hamming (1915–1998), whose first construction of more efficient codes was the beginning of the modern coding theory.

**Definition** (word, Hamming distance). A word of length $n$ in the alphabet $F$ is an element of $F^n$. Note that $F^n$ is the set of all $n$-tuples of symbols:

$$F^n = \{ \underline{v} = (v_1, v_2, \ldots, v_n) \mid v_i \in F, \ 1 \leq i \leq n \}.$$

The **Hamming distance** between two words $\underline{x}, \underline{y} \in F^n$ is the number of positions where the symbol in $\underline{x}$ differs from the symbol in $\underline{y}$:

$$d(\underline{x}, \underline{y}) = \# \{ i \in \{1, \ldots, n\} : x_i \neq y_i \}.$$ 

We may write a word $(x_1, x_2, \ldots, x_n) \in F^n$ simply as $x_1 x_2 \ldots x_n$ if this is unambiguous. So, for example, the binary words $000$, $101$ and $111$ belong to $\{0, 1\}^3$, and we have $d(101, 111) = 1$ and $d(101, 000) = 2$. Of course, $d(101, 101) = 0$. 
Further examples

The following examples are part of historical background to Coding Theory and are not covered in lectures.

Example 1

Here is a real-world example of how Coding Theory is used in scientific research.

Voyager 1 is an unmanned spacecraft launched by NASA in 1977. Its primary mission was to explore Jupiter, Saturn, Uranus and Neptune. Voyager 1 sent a lot of precious photographs and data back to Earth. It has recently been in the news because the NASA scientists had concluded that it reached the interstellar space. See for example a BBC News item dated 12 September 2013.

The messages from Voyager 1 have to travel through the vast expanses of interplanetary space. Given that the spacecraft is equipped with a mere 23 Watt radio transmitter (powered by a plutonium-238 nuclear battery), it is inevitable that noise, such as cosmic rays, interferes with its transmissions. In order to protect the data from distortion, it is encoded with the error-correcting code called extended binary Golay code. We will look at this code later in the course. More modern space missions employ more efficient and
Figure 1.2: A punch card. Image taken from http://www.columbia.edu/cu/computinghistory

more sophisticated codes.

**Example 2**

Here is a more down-to-earth example of the use of error-correcting codes. A CD can hold up to 80 minutes of music, represented by an array of zeros and ones. The data on the CD is encoded using a *Reed-Solomon code*. This way, even if a small scratch, a particle of dust or a fingerprint happens to be on the surface of the CD, it will still play perfectly well — all due to error correction.

However, every method has its limits, and larger scratches or stains may lead to something like a thunderclap during playback!

**Example 3**

To finish this historical excursion, let us recall one of the very first uses of error-correcting codes.

In 1948, Richard Hamming was working at the famous *Bell Laboratories*. Back then, the data for “computers” was stored on *punch cards*: pieces of thick paper where holes represented ones and absences of holes represented zeros. Punchers who had to perforate punch cards sometimes made mistakes, which frustrated Hamming.

Hamming was able to come up with a code with the following properties: each codeword is 7 bits long, and if one error is made in a codeword (i.e., one bit is changed from 0 to
1 or vice versa), one can still recover the original codeword. This made the punch card technology more robust, as a punch card with a few mistakes would still be usable. The trade-off, however, was that the length of data was increased by 75%: there are only 16 different codewords, therefore, they can be used to convey messages which have the length of 4 bits.

The original *Hamming code* will be introduced in the course very soon!