

## MA213 Second Year Essay

### Coding Theory

#### INTRODUCTION

Coding theory is “[t]he theory of the encryption of messages employed for security during the transmission of data, or for the recovery of information from corrupted data (e.g. in reconstructing messages sent over long distances by space probes. [...] Many error-correcting codes which identify and correct errors have been constructed using Galois fields.” [1]

In this essay we shall briefly consider some of the underlying principles of coding theory and its applications to communication in the two areas: error detection and correction; and data compression. At the end we shall see how the techniques discussed can be unified in the digital transmission of an analogue signal.

#### 1: THE PRINCIPLES OF CODING THEORY

**Definition 1:** Let  $S$  be some set and let  $S^n$  denote the usual Cartesian product of  $S$  with itself  $n$  times. Define

$$\begin{aligned} S^+ &= \bigcup_{n \in \mathbb{N}} S^n \\ &= \{x \mid x \in S^n \text{ for some } n \in \mathbb{N}\} \end{aligned}$$

For example, if  $S$  were the Roman alphabet then  $S^6$  would be the set of all six-lettered sequences of Roman characters and  $S^+$  the set of all sequences of finite length.

**Definition 2:** A *code* is a function  $c: X \subseteq \mathcal{X}^+ \rightarrow Y \subseteq \mathcal{Y}^+$ , where  $\mathcal{X}$  and  $\mathcal{Y}$  are sets of symbols with which we may communicate information.

At first sight this definition may seem to be very abstract: this is so that it can cover all the things we really mean to be codes. It would be no use, for example, to say that a code is a bijection  $c: \mathcal{X} \rightarrow \mathcal{Y}$  since the cardinalities of the sets might make such a map impossible to construct. In fact, we would encounter the same problem even if we required a bijection  $c: \mathcal{X}^m \rightarrow \mathcal{Y}^n$  for fixed  $m, n \in \mathbb{N}$ . We require finite-length sequences of elements of  $\mathcal{X}$  and  $\mathcal{Y}$ , and do not yet wish to require unique decodability, hence the above definition.

When we code a message we intend for it to be transmitted along some kind of channel on which there may be ‘noise’ or ‘eavesdroppers.’ The diagram below illustrates our standard setup:

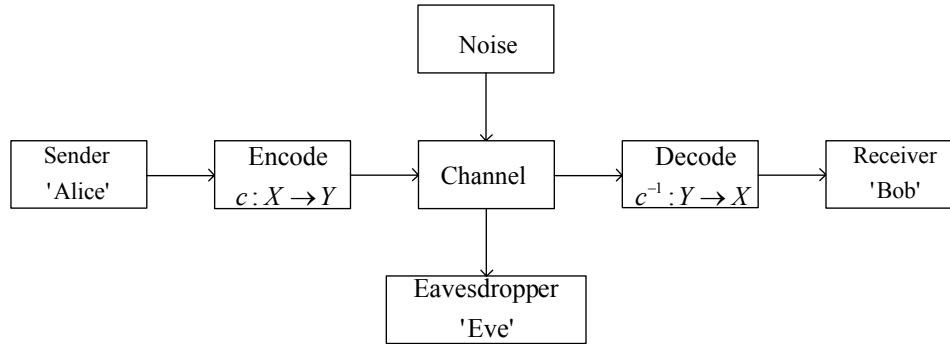


Fig. 1 – Standard setup for communication. (After [2])

Note that we tacitly assert the existence of  $c^{-1}: Y \rightarrow X$ . We shall shortly encounter a necessary and sufficient condition for this in the case that  $|\mathcal{X}|, |\mathcal{Y}| < \infty$ .

There are certain attributes that we may desire of our codes:

*Error-correcting codes:* Suppose that noise causes the message transmitted by Alice to be unequal to the one received by Bob. An error-correcting code allows Bob to detect and correct such errors.

*Compression codes:* These are codes for which the coded message is in some sense more efficient than the uncoded message in that it requires less time/bandwidth/power to transmit, or some similar quantity of which we wish to make as little use as possible.

*Cryptographic codes:* These are codes in which Alice codes the message in such a way that Eve will not be able to determine the meaning of the message but Bob will. We shall not study these in this essay.

**Definition 3:** A *codeword* is the image of an element of  $X$  under a code  $c: X \rightarrow Y$ .

As is suggested by the term *codeword* the image of a single element of  $X$  may well be a sequence of elements of  $\mathcal{Y}$ .

**Definitions 4:** Let  $(y_1, \dots, y_n) \in \mathcal{Y}^+$  be a codeword and suppose that for some  $k < n$   $(y_1, \dots, y_k)$  is also a codeword. Then  $(y_1, \dots, y_k)$  is called a *prefix* (for  $(y_1, \dots, y_n)$ ). A code for which no codeword is a prefix for any other is called *prefix-free*.

For example, it would appear that in the English language ‘THE’ is a prefix for ‘THEIR’. However, any word in English is followed either by a space or a punctuation mark like a full stop or comma, and ‘THE ’ is not a prefix for ‘THEIR’. The use of a ‘word space’ is an easy way to make a code prefix-free. Another frequently-used method is to make code words of uniform length: the ANSI (American National Standards Institute) encoding uses precisely 8 binary bits per character.

The importance of this notion of being prefix-free lies in the following fact, which the observant reader will already have guessed:

**Proposition 1:** [3: pp.118-119]. *Any prefix-free code is uniquely decodeable. That is, there is only one possible pre-image for any given code message.*

**Proof:** (In [3], paraphrased.) The proof uses contradiction. We assume that a code message in a prefix-free code has an ambiguity (i.e. at least two possible decodings) in some triple of codewords  $(d_{k-1}, d_k, d_{k+1})$ . It follows without much difficulty that  $d_{k-1}$  is a prefix. ■

There is a very strong theorem which tells us exactly when we may construct a prefix-free code, known as the Kraft-McMillan Inequality:

**Theorem 2:** (The Kraft-McMillan Inequality) [3: pp.119-120]. *Let  $|\mathcal{X}|, |\mathcal{Y}| \in \mathbb{N}$ . A prefix-free code  $c: \mathcal{X} \rightarrow Y \subseteq \mathcal{Y}^+$  with codewords of length  $l_i$ ,  $i=1, \dots, |\mathcal{X}|$ , exists if and only if*

$$\sum_{i=1}^{|\mathcal{X}|} |\mathcal{Y}|^{-l_i} \leq 1.$$

**Proof:** The proof is too long to give here, and so is omitted. ■

## 2: ERROR-CORRECTING CODES

We shall assume throughout this section that the alphabets in use are all the binary alphabet  $\mathcal{B} = \{0,1\}$  since binary digital transmission is the focus of much code-theoretic research.  $\mathcal{B}$  is a rich source of examples in any case, and has the desirable attribute that if we know a bit is in error we can immediately correct it as there is only one other symbol it could be. (Note that  $(\mathcal{B}, +, \cdot)$  forms the Galois field of order 2.)

**Definition 5:** We say that a code is  $(N, K, T)$  if it codes blocks (sequences of symbols) of length  $K$  into blocks of length  $N$  and is capable of detecting and correcting up to  $T$  errors in those  $N$ .

We can formalize the notion of “number of bits in error” by using the notion of *Hamming distance*. This is a metric on  $\mathcal{B}^n$  (the axioms are easy to check) defined by

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n [x_i \neq y_i],$$

for  $\mathbf{x} = (x_1, \dots, x_n), \mathbf{y} = (y_1, \dots, y_n)$ , and using Iverson’s notation for truth-valued statements:



Coded version: 1100110  
 Single-bit error: 1110110  
 Bob calculates:  $\alpha = 0, \beta = 1, \gamma = 1$   
 Bit in error:  $(011)_2 = 3$   
 Corrected message: 1100110  
 Decoded message: 0110

However, Alice is making quite a dangerous assumption about the nature of the noise source, namely that it can be trusted to produce at most one error in each block of length 7 that she sends. In fact it is far more likely that a burst of noise is produced, potentially corrupting many bits in succession. Do we now need to hunt for a code with much greater  $T$ ? The answer is no: the error-correcting capacity of the code can be extended by a simple rearrangement process known as *interleaving*.

Suppose that the message of 7-bit blocks Alice sends looks like this (braces added for clarity and emphasis):

$$\{x_{11}, x_{12}, \dots, x_{17}\}, \{x_{21}, x_{22}, \dots, x_{27}\}, \dots, \{x_{N1}, x_{N2}, \dots, x_{N7}\} \quad (1)$$

Now suppose that we reorder the bits like this (it may be helpful to compare this with the operation of matrix transposition):

$$\{x_{11}, x_{21}, \dots, x_{N1}\}, \{x_{12}, x_{22}, \dots, x_{N2}\}, \dots, \{x_{17}, x_{27}, \dots, x_{N7}\} \quad (2)$$

Now imagine a burst of noise that corrupts at most  $N$  successive bits in format (2) of the message. When we undo the interleaving process we find that the errors have been ‘redistributed’: there is at most one bit in error in each of the braced sections in format (1). We can now decode and error-correct in the original way.

This method of interleaving is used in the Cross-Interleaved Reed-Solomon Code (CIRC), which is used to encode data on an audio compact disc. CIRC is capable of correcting up to 4000 consecutive errors.

### 3: COMPRESSION CODES

Suppose that Alice has a message that she wished to transmit to Bob in as short a time as possible. Is it possible to find a code which will ‘compress’ her original message into a shorter, more efficient form? Obviously, we are looking for an algorithm which will do well at this most of the time: we cannot expect our code to be perfect, since each possible message will have different characteristics and we do not know in advance which message will be sent.

Suppose we have a source alphabet  $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$  and that the codeword  $c(x_i)$ , which is more likely to be a sequence of symbols in  $\mathcal{Y}$  than a single symbol, takes a time  $t_i$  to transmit. An easy way to maximize the amount of information sent per unit time, then, would be to choose the  $c(x_i)$  so that if  $x_i$  is the most (or least) frequently-

used symbol then  $t_i$  is least (or greatest, respectively). So, in English, we would typically give ‘E’ the shortest code word and ‘Q’ or ‘Z’ a long one.

In essence, any form of shorthand notation is a compression code: a student of history might use ‘C.’ to stand for ‘century,’ or a mathematical analyst ‘cts.’ for ‘continuous.’ It should be intuitively clear that the more often a word or concept is referred to the greater the need for a shorthand (and the greater the ‘profit’ when one is used) – the historian would probably derive no benefit from abbreviating the word ‘continuous.’

This idea underlies the construction of *Huffman codes*, which are optimal in this sense of making the most frequently-used symbols the quickest to transmit. To some extent, this principle is applied in Morse Code,<sup>†</sup> although Morse Code predates the development of coding theory and Huffman codes in particular.

Another type of compression is employed in the compression of waveforms such as music and other sounds. Suppose that Alice wishes to transmit to Bob a sound wave (speech, for example) the displacement of which from the equilibrium position at a time  $t$  is  $g(t)$  for  $0 \leq t \leq A$ :

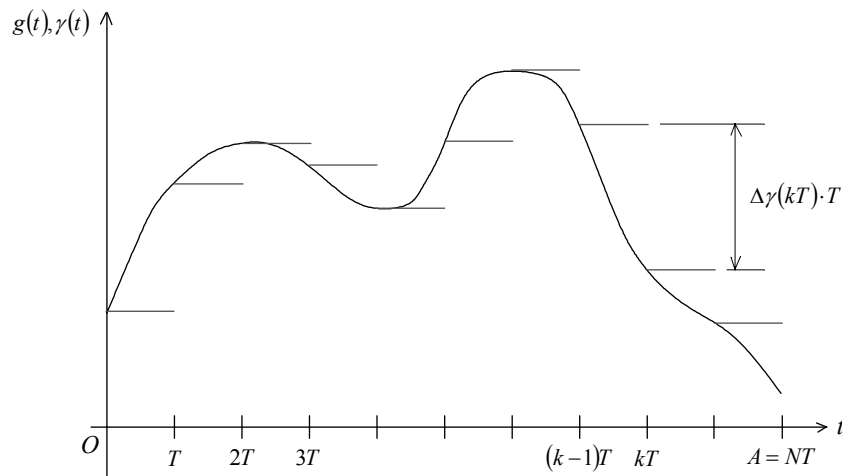


Fig. 2 – A wave and discrete approximation to it.

Now, since  $g : [0, A] \rightarrow \mathbb{R}$  is continuous it is certainly regulated, so we can approximate it to any order of accuracy we desire by some step function  $\gamma : [0, A] \cap \mathbb{F} \rightarrow \mathbb{F} \subset \mathbb{R}$  whose partition points and image values lie in our floating-point set  $\mathbb{F}$ <sup>‡</sup>. For simplicity we shall assume that we have taken the simplest uniform partition and so we have a sequence of points  $(\gamma(0), \gamma(T), \gamma(2T), \dots, \gamma(NT))$ , where  $A = NT$ .

Alice might now choose to send Bob the values taken by  $\gamma$ , measured to a given degree of accuracy. Certainly, if  $T$  is small enough,  $\gamma(t)$  will ‘sound’ roughly the same as  $g(t)$ . However, is it possible to completely reconstruct  $g(t)$  knowing only the

<sup>†</sup> See Appendix – Morse Code

<sup>‡</sup> See Appendix – Floating-Point Number Systems

finite set of values taken by  $\gamma$ ? The answer is affirmative, and is a famous theorem of information theory known as Nyquist's Sampling Theorem, which relies on a certain amount of Fourier theory.

Suppose we have a periodic function  $g : \mathbb{R} \rightarrow \mathbb{R}$ : the reader is probably familiar with the idea that subject to a few constraints on  $g$  we can construct a trigonometric series, known as a Fourier series, so that

$$g(t) = a_0 + 2 \sum_{n=1}^{\infty} \left( a_n \cos\left(\frac{2\pi n t}{\tau}\right) + b_n \sin\left(\frac{2\pi n t}{\tau}\right) \right) = \sum_{n=-\infty}^{\infty} c_n \exp\left(\frac{2\pi i n t}{\tau}\right),$$

where  $\tau$  is the period of  $g$  and  $c_n = a_n + i b_n$ ,  $c_{-n} = a_n - i b_n = c_n^*$ .

This series can be viewed as measuring the 'frequency components' of our function  $g$  for each integer multiple of some basic frequency. However, we can form a more general frequency decomposition (for which we do not need  $g$  to be periodic) by use of a Fourier transform:

$$\text{FT}\{g(t)\} = G(f) = \int_{-\infty}^{\infty} g(t) \exp(-2\pi i f t) dt.$$

As an example, if we take the Fourier transform of a light signal landing on a photodetector then  $G(f)$  is the spectrum of the light received, say, lots of 'blue' and little 'red.' The Fourier transform is a mathematical way of passing from analyzing the function in time,  $t$ , to frequency,  $f$ .

**Definition 6:** We define the *convolution*  $g_1 \otimes g_2$  of  $g_1, g_2 : \mathbb{R} \rightarrow \mathbb{R}$  by

$$(g_1 \otimes g_2)(t) = \int_{-\infty}^{\infty} g_1(s) g_2(t-s) ds.$$

For convenience we shall write  $g_1(t) \otimes g_2(t)$  for  $(g_1 \otimes g_2)(t)$

**Theorems 5:** (The Convolution Theorems) *Convolution and the Fourier transform satisfy the dual relations*

$$\begin{aligned} \text{FT}\{g_1(t) \otimes g_2(t)\} &= \text{FT}\{g_1(t)\} \text{FT}\{g_2(t)\}, \\ \text{FT}\{g_1(t) g_2(t)\} &= \text{FT}\{g_1(t)\} \otimes \text{FT}\{g_2(t)\}. \end{aligned}$$

*I.e. the Fourier transform takes products to convolutions and convolutions to products.*

**Proof:** Neither is particularly difficult, and so both are left to the reader. ■

**Theorem 6:** (Nyquist’s Sampling Theorem) [4: p.13]. *If a signal  $g(t)$  has bandwidth  $B$ , i.e.  $G(f)=0$  for  $f \notin [-B, B]$ , then the signal can be perfectly reconstructed from samples taken with period  $T$  if  $1/T > 2B$ .*

**Proof:** The proof makes use of the so-called *Dirac  $\delta$ -function*, which is ‘defined’ by

$$\delta(x) = 0 \text{ for } x \in \mathbb{R} \setminus \{0\},$$

$$\int_{-\infty}^{\infty} \delta(x) dx = 1,$$

and has the nice property that  $\int_{-\infty}^{\infty} h(x)\delta(x-a)dx = h(a)$ . We consider our sampled signal  $g_s(t)$  to be the product of the original signal with a sequence of  $\delta$ -functions centred on the times  $t = 0, T, 2T, \dots$  (known as a *Dirac comb*). We then show that the Fourier transform of a Dirac comb is another Dirac comb with spacing  $1/T$ . It follows that the spectra of the original and sampled signals are related by

$$G_s(f) = G(f) \otimes \frac{1}{T} \sum_{n=-\infty}^{\infty} \delta\left(f - \frac{n}{T}\right),$$

i.e. the sampled spectrum is the convolution of the original spectrum and the Dirac comb in the frequency domain. The result is that we have copies of  $G(f)$  centred on each  $\delta$ -function position.

If  $1/T > 2B$  then these copies of  $G(f)$  do not overlap. So, if one filters out from  $G_s(f)$  all frequencies outside the permitted bandwidth the result is  $G(f)$ . Now simply apply the inverse Fourier transform  $\text{FT}^{-1}\{G(f)\} = g(t) = \int_{-\infty}^{\infty} G(f)\exp(2\pi ift)df$  to obtain  $g(t)$ .

If  $1/T < 2B$  then spectral components from different parts of  $G(f)$  are ‘mixed up’ in  $G_s(f)$ . They can no longer be separated by filtering: there is unavoidable mixing of frequency components with  $f > 1/T$  down to lower frequencies in the reconstructed signal. This is known as *aliasing* and is the reason why cars’ wheels appear to rotate backwards or even stand still on television or film. ■

In information-theoretic terms, Nyquist’s Theorem shows that the original continuous function and the discrete sampled function have the same information content. Now that we know the signal can be reconstructed using the step function values the question remains: how can we more efficiently transmit that information?

While it takes a fixed number of bits to represent any number in our  $\mathbb{F}$  it is a fact that we could construct a different set  $\mathbb{F}'$  using fewer bits to describe smaller numbers (the cost is that  $\mathbb{F}'$  would be unable to describe large numbers). This would be of advantage if Alice were to record and transmit not the absolute displacement of the

wave at each sampling point but the change relative to the previous measurement. (Here we use ‘absolute’ in the sense of ‘not relative’ rather than ‘modulus.’) So, if we define  $\Delta\gamma$  by

$$\Delta\gamma(kT) = \frac{\gamma(kT) - \gamma((k-1)T)}{T}$$

then Alice would instead send a sequence  $(\gamma(0), \Delta\gamma(T), \Delta\gamma(2T), \dots, \Delta\gamma(NT))$ . For this to be ‘profitable’ we require that  $g$  be globally Lipschitz with a known Lipschitz constant  $L$ , i.e.  $L$  is such that for all  $x, y \in [0, A]$ ,  $|g(x) - g(y)| \leq L|x - y|$ . This condition is almost certain to be satisfied by any classical wave phenomenon such as a sound wave, and follows from the bandwidth limitation condition in Nyquist’s Theorem.

When we wish to reconstruct our approximate wave function  $\gamma$  we make use of a discrete version of the Fundamental Theorem of Calculus. Recall that if  $g: [a, b] \rightarrow \mathbb{R}$  is differentiable then

$$g(b) = g(a) + \int_a^b g'(x) dx.$$

In its discrete form as applied to this problem we get the summation statement

$$\gamma(nT) = \gamma(0) + \sum_{k=1}^n \Delta\gamma(kT)T.$$

From here we can use the Convolution Theorems and the inverse Fourier transform to undo the sampling process and regain the original function  $g$ .

In practice we would not apply this ‘method of differences’ to the whole of  $[0, A]$ ; we would instead subdivide  $[0, A]$  and apply the method to each of them in turn. The reason for this is that if we were to make a mistake in recording or retrieving an early  $\Delta\gamma$  all the remainder of the wave would be in error when reconstructed, offset by the error amount. This problem is known as *error propagation*. Subdividing our wave and repeatedly taking an ‘absolute reading’ helps prevent this.

These methods of compression are all employed in the very successful MP3 (Motion Picture Experts Group, Layer 3) sound format. The use of the absolute values  $(\gamma(0), \gamma(T), \gamma(2T), \dots, \gamma(NT))$  corresponds to a CD Audio or Wave format, which takes up approximately ten times as much space as the corresponding MP3. It is also interesting to note that the usual sampling rate for an MP3 is 44kHz, while the upper frequency for human hearing is about 20kHz – an example of Nyquist’s inequality in practice.

## CONCLUSION

We have seen what a code is and some of the constraints that exist on the construction of codes with the highly desirable attribute of unique decodability. In Section 2 we saw how one could construct codes capable of detecting and correcting errors introduced by random noise.

In Section 3 we briefly examined how one may compress data by judicious choice of codewords, and how a waveform may be approximated by a step function stored in a compressed form making use of the Lipschitz, differentiability and integrability properties of certain functions.

As a final, unifying example, observe that Alice can now in theory take an analogue signal (speech, say) and sample it at discrete points; compress this information by measuring relative differences; convert these numerical data to a binary form using a prefix-free code; apply an error-correcting code to that binary representation; transmit the encoded message to Bob, who will then detect and correct errors caused by channel noise; recover the numerical data describing the signal at discrete points; and finally use an inverse transform to reconstruct the original signal.

APPENDIX – MORSE CODE

Morse Code uses a simple binary channel to transmit information: the channel can either be on or off. A short ‘on’ pulse lasts for one time unit and is a dot (.). A long ‘on’ pulse lasts for three time units and is a dash (-). Between letters the line is left off for one time unit; between words there is a three unit break.

Below are the standard characters for International Morse Code, their codes, and the time it takes to transmit each of them.

*Letters*

<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>	<b>F</b>	<b>G</b>	<b>H</b>	<b>I</b>	<b>J</b>	<b>K</b>	<b>L</b>	<b>M</b>
.-	-...	-.-.	-..	.	..-.	--.	....	..	.---	-.-	.-..	--
5	9	11	7	1	9	9	7	3	13	9	9	7
<b>N</b>	<b>O</b>	<b>P</b>	<b>Q</b>	<b>R</b>	<b>S</b>	<b>T</b>	<b>U</b>	<b>V</b>	<b>W</b>	<b>X</b>	<b>Y</b>	<b>Z</b>
-.	---	.-.	--.-	.-.	...	-	..-	...-	.-.	-..-	-.-.	-..
5	11	11	13	7	5	3	7	9	9	11	13	7

*Numbers*

<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>
-----	.-----	..-----	...----	....-	.....	-.....	--....	---...	----.
19	17	15	13	11	9	11	13	15	17

*Punctuation*

<b>.</b>	<b>,</b>	<b>:</b>	<b>?</b>	<b>‘or’</b>	<b>-</b>	<b>/</b>	<b>( or )</b>	<b>“ or ”</b>
.-.-.-	--.-.-	---...	...-..	.-----	-....-	-.-.	-.-.-	.-.-.
17	19	17	15	19	15	14	19	15

There is also a correction character used to tell the receiver that a mistake has been made and that the last word should be ignored, ..... (eight dots), which takes 15 time units to transmit.

APPENDIX – FLOATING-POINT NUMBER SYSTEMS

A floating-point number is a number like  $6.67 \times 10^{-33}$  :

$$\underbrace{6.67}_{\text{mantissa}} \times \underbrace{10}_{\text{base}}^{\overbrace{-33}^{\text{exponent}}}$$

We have a base  $b$  ( $=10$ ), a mantissa  $m$  ( $=6.67$ ),  $0 \leq m < b$ , with a maximum number of digits after the ‘decimal’ point, and an integer exponent  $e$  ( $=-33$ ), the choice of which is also bounded above and below. The set of all such numbers is a floating-point set, often denoted by  $\mathbb{F}$ .

The first thing that should be clear is that a set of such numbers is finite. For example, if we take  $b = 10$ , allow up to 2 decimal places and let  $-99 \leq e \leq 99$  then we have  $9 \times 10 \times 10 \times (2 \times 99 + 1) = 179100$  possible numbers. More generally, if, for some base  $b$ , we allow  $n$  ‘decimal’ places and  $l \leq e \leq u$  then our  $\mathbb{F}$  has  $b^n(b-1)(l-u+1)$  elements.

Computers make use of floating-point number sets. There is no single standard  $\mathbb{F}$  – there are various ones allowing various degrees of accuracy. For example, a ‘long double’ floating-point number used in computing uses 10 bytes (i.e. 80 binary bits) to store a number in absolute value between  $3.4 \times 10^{-4932}$  and  $1.1 \times 10^{4932}$ , together with its sign. It is customary to denote  $\sup \mathbb{F}$  as ‘infinity’ and  $\inf \mathbb{F}$  as ‘-infinity’.

**The IEEE Standard 754 [5]**

IEEE (Institute of Electrical and Electronic Engineers) Standard 754 floating-point is the most common representation for real numbers on computers today. IEEE floating-point numbers have three basic components: the sign, the mantissa and the exponent. The base (2) is implicit and is not stored.

The following table shows the layout for single (32-bit) and double (64-bit) precision floating-point numbers. The number of bits for each field are shown (bit ranges are in square brackets):

	Sign	Exponent	Mantissa	Bias
Single Precision	1 [31]	8 [30-23]	23 [22-00]	127
Double Precision	1 [63]	11 [62-52]	52 [51-00]	1023

Table 1 – Single and double precision IEEE floating-point numbers.

*Sign:* The sign bit denotes the sign of the number: 0 for positive, 1 for negative.

*Exponent:* Since the exponent field needs to represent both positive and negative exponents a *bias* is added to the actual exponent in order to get the stored exponent.

So, for example, a stored value of 190 corresponds to an actual value of 63. Exponents of all zeros and all ones are reserved for special numbers.

*Mantissa:* The mantissa stores the precision bits in what is known as *normalized form*. This basically puts the radix point after the first non-zero digit, so 650 would be  $6.5 \times 10^2$ . (A nice little optimization is available to us with a base of 2: since the only possible non-zero digit is 1 we can toss away the 1 and just assume that it exists, giving us one extra bit of precision for free. Thus, the mantissa has effectively 24 bits of resolution in single precision, and 53 in double precision.)

BIBLIOGRAPHY

- [1] *Penguin Dictionary of Mathematics*, 2<sup>nd</sup> Edition, ed. D. Nelson, Penguin (1998)
- [2] *A Mathematical Theory of Communication*, C.E. Shannon in *The Bell System Technical Journal*, Vol. 27 (1948)
- [3] *Probability and Information: An Integrated Approach*, D. Appelbaum, Cambridge University Press (1996)
- [4] *PX214 Information Theory and Noise*, R.F. Pettifer, University of Warwick (2002)
- [5] *IEEE Standard for Binary Floating-Point Arithmetic, IEEE Std. 754-1985*, Institute of Electrical and Electronic Engineers Computer Society (1985).