Correcting errors: putting elementary topology to work

Juan Medina
juan.medina@upct.es
Departmento de Matemáticas y Estadística
Universidad Politécnica de Cartagena
Paseo Alfonso XIII 52 - 30203 Cartagena
Spain

José A Vallejo
jvallejo@fc.uaslp.mx
Facultad de Ciencias
Universidad Autónoma de San Luis Potosí
Lat. Av. Salvador Nava s/n - 78290 San Luis Potosí
México

Abstract

Nowadays, virtually any field of Mathematics has some interesting application to technology, and topology is no exception. We show an example of a math lab session illustrating the use of basic notions of metric spaces (and a little bit of probability) in the context error correction codes, through the use of Hamming's distance. The CAS Maxima is very well suited to this task, and we also show how to write a simple set of commands for solving some basic exercises. Finally, we also present some online digital resources complementing the contents presented here, so the combined materials can be used in a blended learning program.

1 Introduction

Our purpose in this work is not to present new examples of applications, neither of Mathematics nor of technological tools, but rather to show how to design and implement a set of materials for a mathematics course. Our approach is based on the following route:

1. Once a topic is programmed for teaching, some interesting application of it is selected (this is probably the hardest part), and information about this particular technological application is provided to the students, with no reference to Mathematics, so they can better appreciate its importance.

2. Some time is devoted to expose the mathematical model of the application and to explore possible ways of solving the problem it poses.

3. After the student is convinced that the problem is not trivial, and that we should acknowledge and thank whoever solves it, the corresponding mathematical technique is studied in the classroom.

4. When doing a homework assignment, it is desirable to try to implement the algorithms in a computer; not only to reinforce programming skills, but to be able to do real-life, complicated calculations without being deterred by silly numerical errors (very often, the students commit a little numerical error deep inside a long computation and, when comparing with the provided answer, thinks that the whole approach is wrong, wasting a precious time trying to find an alternative to a correct idea.)

5. Finally, if the problem at hand involves some convoluted notions, mixes terminology from different areas, or it is not easy to find references for it in textbooks, then it is essential to complement the classroom exposition with some alternative material that the student can consult at will. We have found that the blended learning approach is one of the most appreciated by students.

The rest of the paper offers an example of this flowchart, taking as the starting point a point-set topology course, supposing that the bare essentials of metric spaces have been seen in the classroom, and then presenting some interesting (we think) application which is explored by using the CAS Maxima for the computations: the construction of error-correcting codes, as used, for instance, in the GPS. A complementary video lesson has been made available to implement the blended learning program, thus giving a more or less complete panorama of the use of Mathematics in everyday life through a concrete case study.

2 Motivating the problem: the GPS

Everyday, millions of people make use of the Global Positioning System (GPS) resources, from the pizza delivery boy to the mountain rescue team assisting a climber. As suggested by its name, the GPS is a system of satellites sharing a common communication protocol that provides, among other things, a positioning service (the Standard Positioning Service or SPS) that is used by any smartphone in the world in conjunction with a software layer to locate the gas station nearest to you.

The GPS consists of a basic constellation of 24 satellites (plus other 6 spare ones) orbiting around the Earth at an altitude of $20\,200\,\mathrm{m}$ above the mean sea level, distributed among six orbital planes (neither orbit is geostationary). The (circular) orbits have been calculated so at each point on the Earth's surface at least five satellites are visible at any moment (theoretically, only four satellites are needed to determine the position, but this seldom occurs in practice). At regular intervals of time, each satellite broadcasts through a radio signal its position and the time registered in its on-board clock.

The receiver can determine its own position by comparing the time at which the signal was emitted with the time at which it was received, thus determining (through Relativity-based formulas [1])

the distance to the satellite. Once that distance is known for several satellites, intersecting the corresponding spheres fixes a point, a process called trilateration.

Now, both the satellite and the GPS receiver *store* their data in digital format, as a string of 0 and 1. However, as stated, the satellites *broadcast* their positions through a radio signal. These are just electromagnetic waves that can be altered by several physical phenomena, the most important of these being atmospheric dispersion and the interference caused by the reception of reflected signals in the receiver's antenna. The net result is that some data will be corrupted: it is possible that the satellite sends a signal with the location data

$$110011001110111000110011\,,$$

but the receiver decodes it to be

$$110001001110111000100010\,,$$

The problem then is the following: how to detect if the communication has been corrupted and, if this is the case, how to correct the data? The GPS solves this by using Hamming codes [2], and we will see in what follows the basics of this technique, assuming a knowledge of the most elementary notion of metric topology and probability.

Besides the main literature on error-correcting codes on which this work is based [3, 4], there are many excellent resources on these topics that can be found in the Internet. We have found particularly useful those of the Mathematics Enhancement Programme (MEP) created by the Centre for Innovation in Mathematics Teaching [5]. We consider of the utmost importance to work with open access, public documents, so the bibliography consists (with only one exception) in freely available documents.

3 Hamming distance and error correction

Let $\mathcal{A}$ be a set of symbols, that we will call an *alphabet*. For instance, the binary alphabet is $\mathcal{A} = \{0, 1\}$, usually denoted in Mathematics by $\mathbb{Z}_2$. Another example is given by the set of symbols $\mathcal{B} = \{A, C, G, T\}$ (this is suited to the study of the DNA).

An $n-$length *word* in the alphabet $\mathcal{A}$ is simply an element $w \in \mathcal{A}^n$. As an example, a binary word of length 7 is $0110010 \in \mathbb{Z}_2^7$, while in the alphabet $\mathcal{B}$ such a word could be $GATTACA$. A *code* in the alphabet $\mathcal{A}$ is a series of words, all of them with the same length, that is, a subset $C \subset \mathcal{A}^n$ for some $n \in \mathbb{N}$. The elements of a code are called *codewords*. Intuitively, one can think of codewords as the words contained in a dictionary: given the ordinary alphabet $\{a, b, \ldots, z\}$, not any word can be used in ordinary communication (words such as $asdfg$ have no sense), and codewords are the admissible words out of which you can construct grammatically correct sentences.

Hamming distance between two same length words is defined as the number of symbols in which they differ. For instance, the distance between the two words 0110010 and 0010111 in $\mathbb{Z}_2^7$ is 3, while in $\mathcal{B}^7$, the distance between $GATTACA$ and $CATGACT$ is 4.

In the particular case of $\mathbb{Z}_2^n$, which will be our main concern from now on, it is possible to give a numerical definition of Hamming distance: given two words $w = (w_1, \ldots, w_n)$ y $v = (v_1, \ldots, v_n)$,

their distance is defined to be

$$\delta(w, v) = \sum_{i=1}^{n} |(w_i - v_i) \bmod 2| \,.$$

Notice that the difference is understood *modulo* 2.

The basics of Hamming's technique can be presented in the classroom in the form of a sequence of exercises (all of them are really easy, see [4])

(a) Prove that Hamming 'distance' in $\mathbb{Z}_2^n$ is actually a distance.

Having a distance, we can introduce a notion of 'proximity'. Given a code $C \subset \mathbb{Z}_2^n$ and a word $w \in \mathbb{Z}_2^n$, we say that the codeword $x \in C$ is the closest to w whenever

$$\delta(w, x) = \min\{\delta(w, y) : y \in C\} \,.$$

Notice that 'the' closest codeword is not necessarily unique, as the following exercise shows.

(b) In $\mathbb{Z}_2^3$ consider the code $C = \{110, 101, 011\}$ and the word $w_0 = 000$. Determine the codeword of C closest to w_0.

Given a word $w \in \mathbb{Z}_2^n$, its *weight* $\|w\|$ is defined as its distance to the null word $w_0 = 0 \cdots 0$, that is,

$$\|w\| = \delta(w, w_0) \,.$$

In plain English, the weight of a word in $\mathbb{Z}_2^n$ is simply the number of 1's it contains.

(c) Prove that, for any two arbitrary words $w, v \in \mathbb{Z}_2^n$, it holds

$$\delta(w, v) = \|w - v\| \,.$$

It is time to see how to apply these notions. In our terminology, a digital message is a code, and each word in the message is a codeword. We assume that the communication channel is a noisy one, therefore some of the codewords will get corrupted during the transmission; thus, if the number 5 was transmitted in its binary representation 101, maybe what was received is the number 7, represented by 111. A basic problem in communications engineering (and a crucial one in the case of the GPS, as already mentioned) is to be able of detecting and correcting these errors.

The detection part is easy. Suppose, for the sake of simplicity, that we have at our disposal 16 codewords to form codes (messages), all of them with length 4:

$$\begin{array}{cccc} 0000 & 0001 & 0010 & 0100 \\ 1000 & 1001 & 1010 & 1100 \\ 0101 & 0110 & 0011 & 1011 \\ 1100 & 1101 & 1110 & 1111 \,. \end{array}$$

If the message we intend to send is 110011100011 and the received one is slightly different, there is no way we can tell whether the received code is the same that the emitted code or not. A possible way out is to introduce *redundancies*, for instance, sending each codeword twice (an

alternative technique consists in using control digits at the end of the code), so the emitted message would be

$$110011001110111000110011\,.$$

In this way, errors in the transmission can be easily detected: if we receive, for example,

$$110001001110111000100010\,,$$

we would know that there is an error in the forst codeword, as the set of the first four digits do not coincide with the next set of four. Thus, redundancy solves the detection problem.

The problem of correcting the errors remains. Which is the correct word, 1100 or 0100? There are two popular solutions to this problem. In the first case, we apply a probabilistic principle and choose the most likely word. This is particularly done when an error in only one digit of the original word corresponds to more than one admissible word, that is, when 'the' closest word to the received one is not unique (recall item (b)).

In order to implement this solution algorithmically, we must make some assumptions on the characteristics of the noise introduced by the communication channel:

- Errors introduced in different digits of a word, are independent.
- Each digit has the same probability $r \ll 0.5$ of being erroneously transmitted.

(d) Explain why the probability that a word of length n contains i transmitted errors is given by

$$P(n,i) = \binom{n}{i}(1-r)^{n-i}r^i\,.$$

(e) Taking into account the preceding result, prove that the condition on r

$$r < \frac{1}{1+n}\,,$$

guarantees that the greatest probability corresponds to an error-free transmission, the next most probable situation is that of only one transmitted error, and so on.

Under these assumptions, if a code like

$$0100110011011101\,,$$

is received, we could not only detect that there is an error, but we could correct it because the higher probability corresponds to a single error, and it is located in the first four digits.

We mentioned two possibilities for correcting errors. The second one starts from the idea of avoiding the situation in which a single error in a digit originates a valid codeword. To this end, the codewords are selected in such a way that they are as far away among them as possible in the Hamming distance. To give an example, suppose that the codes can be formed only from the codewords

$$000000 \quad 010101 \quad 101010 \quad 111111\,,$$

and we send 000000, which is received as 000001. The detection of an error is immediate in this case because 000001 is not a valid codeword. The correction is achieved by noticing that the most likely case is that in which the correct codeword is the closest to 000001. Here the only possibility is 000000, as the others are at a Hamming distance from 000001 greater than 2, while the distance between 000000 and 000001 is just 1. The essential property making this work is that the valid codewords are very distant among them in the Hamming distance, so a given codeword is at most near only another one, as an elementary application of the triangle inequality.

These intuitive ideas are captured by the notion of minimum distance of a code. Given a code C, containing more than one codeword, we define its *minimum distance* $\delta(C)$ as

$$\delta(C) = \min\{\delta(x, y) : x, y \in C, x \neq y\}\,.$$

It is obvious that $\delta(C) \geq 1$. Geometrically, $\delta(C)$ is the *diameter* of the set C.

(f) Prove that, if $\delta(C) \geq s + 1$, then the code C can *detect* s errors.

(g) Prove that, if $\delta(C) \geq 2t + 1$, then the code C can *correct* t errors.

4 Maxima code

Here we develop several useful Maxima functions to work with Hamming codes. Only a very basic knowledge of Maxima is required, and if the reader has a previous experience with Mathematica, Maple or any other CAS, this will suffice.

4.1 Conversion from base 10 to base b

The function `ldecimal2base` takes as argument an integer n and the base b in which we want to express the number, that is, its syntax is `ldecimal2base(n,b)` and the output is a list containing the digits of n in base n. If the base b is not specified, the function simply returns a list with the digits of n in base 10.

```
(%i1) ldecimal2base([arg]) := block(
         [n:first(arg),v:[ ],q,
         b:if length(arg)>1 then second(arg) else 10],
         do ([n, q]: divide(n, b),
           v: cons(q, v),
           if n=0 then return(v)))$

(%i2) ldecimal2base(12345);
```

(%o2) $[1, 2, 3, 4, 5]$

```
(%i3) ldecimal2base(-123,2);
```

(%o3) $[-1, -1, -1, -1, 0, -1, -1]$

The command `simplode` converts the list of digits so obtained into a 'number' (actually, a string) by juxtaposition:

```
(%i4)  simplode(ldecimal2base(184937,2));
```

(%o6) 101101001001101001

Thus, we can define a function to convert numbers written in base 10 to any other base as follows:

```
(%i5)  decimal2base([arg]) := block(
        [n:first(arg),
         b:if length(arg)>1 then second(arg) else 10,
         v:[ ],q],
          do (
            [n, q]: divide(n, b),
            v: cons(q, v),
            if n=0 then return(simplode(v)))
       )$

(%i6)  decimal2base(23,6);
```

(%o6) 35

```
(%i7)  decimal2base(184937,2);
```

(%o7) 101101001001101001

4.2 Hamming distance in $\mathbb{Z}$

In order to compute Hamming distances, we must compare the binary lists corresponding to two numbers and count the number of positions in which they differ (padding the shortest list with zeros on the left if necessary). As these lists are composed of 0 and 1s, we can apply the command `bit_xor` (included in the package `bitwise`), which automatically pads the shorter list, applies the xor operation and returns the results in decimal notation. We can then convert it to binary, where it is easy to count the number of 1s:

```
(%i8)  load(bitwise)$

(%i9)  bit_xor(93,73);
```

(%o9) 20

```
(%i10) ldecimal2base(20,2);
```

(%o10) $[1, 0, 1, 0, 0]$

Thus, an efficient and compact way to compute the Hamming distance between m and n is given by `length(sublist(ldecimal2base(bit_xor(m,n),2),oddp))`. Here is an example:

```
(%i11) length(sublist(ldecimal2base(bit_xor(2367,8531),2),oddp));
```

(%o11) 6

We bundle it up into a function:

```
(%i12) hammingZ(m,n) := block(
           load(bitwise),
           length(sublist(ldecimal2base(bit_xor(m,n),2),oddp))
       )$

(%i13) hammingZ(73,93);
```

(%o13) 2

```
(%i14) hammingZ(2367,8531);
```

(%o14) 6

4.3 Hamming distance in code space

Things are easier when we are directly dealing with binary codes. A condition to be imposed is that all the words to be compared must have the same length n. As we know, each word is represented as a list, here is an example with $n = 14$:

```
c1:[1,0,0,0,1,1,0,1,0,1,0,0,1,0]
c2:[1,1,1,1,0,0,0,1,0,0,0,1,1,1]
```

As before, we apply xor and count the number of 1s:

```
(%i15) c1:[1,0,0,0,1,1,0,1,0,1,0,0,1,0]$

(%i16) c2:[1,1,1,1,0,0,0,1,0,0,0,1,1,1]$

(%i17) map(bit_xor,c1,c2);
```

(%o17) $[0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1]$

```
(%i18) length(sublist(%,oddp));
```

(%o18) 8

Thus, the corresponding function is simply:

```
(%i19) hammingC(c,d) := block(
       load(bitwise),
       length(sublist(map(bit_xor,c,d),oddp))
       )$
```

```
(%i20) hammingC(c1,c2);
```

(%o20) 8

The weight of a word is defined as its distance to $[0, 0, ..., 0]$. In other words, it is the number of 1s it contains. Then, the computation of the weight is straightforward:

```
(%i21) hamming_weight(c):=block(
       length(sublist(c,oddp))
       )$

(%i22) hamming_weight([1,1,0,1,1,0,1,1]);
```

(%o22) 6

4.4 Error correction

As explained before, if we have a set of messages that are transmitted through a noisy channel errors may occur, and we will see now how to detect and correct them. Suppose that our messages are:

```
m1:000000  m2:001111  m3:010011  m4:011100  m5:111010
```

Also suppose that one of them is transmitted (we do not know which one, yet), but what we receive is

```
mr:010100
```

Clearly, this is not one of the admissible messages. Is there any way we can tell which message, among the feasible ones, was transmitted? Hamming's answer is that, most likely, the messsage that was transmitted is the closest one (with respect to Hamming distance) to the one received, that is, the admissible message such that $d_H(mj, mr)$ is the least, for $1 \leq j \leq 5$.

Let us compute these distances for the example at hand. This is done through the following sequence of commands:

```
(%i23) m[1]:[0,0,0,0,0,0]$

(%i24) m[2]:[0,0,1,1,1,1]$

(%i25) m[3]:[0,1,0,0,1,1]$

(%i26) m[4]:[0,1,1,1,0,0]$

(%i27) m[5]:[1,1,1,0,1,0]$

(%i28) mr:[0,1,0,1,0,0]$
```

```
(%i29) for i from 1 thru 5 do l[i]:hammingC(m[i],mr);
```

(%o29) *done*

```
(%i30) makelist(l[j],j,makelist(k,k,1,5));
```

(%o30) $[2,4,3,1,4]$

```
(%i31) lmin(%);
```

(%o31) 1

The minimum distance is 1. With this information, we can detect the transmitted message and correct the received one. As the minimum distance is achieved for message `m4`, this is in all likelihood, the transmitted message. Because the Hamming distance between `mr` and `m4` is 1, the received message and the original `m4` differ only in one digit. To find out in which one, we compare:

```
m3:011100
mr:010100
```

We readily see that the difference lies in the third digit: the noisy channel changed an 1 in the original message by a 0.

A very good exercise, at this point, is to instruct the students to write a function in Maxima automating the process, that is, the input should be a pair of list, the first one containing a set of admissible messages (also given as lists), and the second one representing the received message. The output should consist of the message that was most likely emitted.

References

[1] N. Ashby: *Relativity in the Global Positioning System*. Living Rev. Relativ. (2003) 6: 1. `https://doi.org/10.12942/lrr-2003-1`.

[2] Global Positioning System Directorate: *Interface specification IS-GPS-200*. Publicly available at the URL `https://www.gps.gov/technical/icwg/IRN-IS-200H-001+002+003_rollup.pdf`.

[3] R. W. Hamming: *Error detecting and error correcting codes*. The Bell System Technical Journal XXIX 2 (1950) 147–160. Publicly available at the URL `https://archive.org/details/bstj29-2-147`.

[4] R. Hill: *A first course in coding theory*. Oxford University Press (1993).

[5] Mathematics Enhancement Programme, Centre for Innovation in Mathematics Teaching. Web page available at the URL `http://www.cimt.org.uk/projects/mep/index.htm`.