Simulations of linear and Hamming codes using SageMath

Tahta D. Timur¹, Dieky Adzkiya² and Soleha³

¹,²,³Department of Mathematics, Institut Teknologi Sepuluh Nopember, Kampus ITS
Sukolilo-Surabaya 60111, Indonesia
E-mail: timur.17071@mhs.its.ac.id¹, dieky@matematika.its.ac.id², seleha.07@matematika.its.ac.id³

Abstract. Digital data transmission over a noisy channel could distort the message being transmitted. The goal of coding theory is to ensure data integrity, that is, to find out if and where this noise has distorted the message and what the original message was. Data transmission consists of three stages: encoding, transmission, and decoding. Linear and Hamming codes are codes that we discussed in this work, where encoding algorithms are parity check and generator matrix, and decoding algorithms are nearest neighbor and syndrome. We aim to show that we can simulate these processes using SageMath software, which has built-in class of coding theory in general and linear codes in particular. First we consider the message as a binary vector of size k. This message then will be encoded to a vector with size n using given algorithms. And then a noisy channel with particular value of error probability will be created where the transmission will took place. The last task would be decoding, which will correct and revert the received message back to the original message whenever possible, that is, if the number of error occurred is smaller or equal to the correcting radius of the code. In this paper we will use two types of data for simulations, namely vector and text data.

1. Introduction

Digital communication technologies such as the internet are now considered as primary telecommunication tools by most people around the world. The internet consists of large network of networks, which are interconnected and constantly sending and receiving huge amount of data [1]. Operations of these networking devices are governed by a set of rules i.e. protocols on how to send, receive and process digital signals [1]. These signals are actually binary digits (bits) transmitted through dial-up modems, wires, radio signals, and optic cables. During transmission, these bits could easily be changed due to hardware or software errors. How to detect these errors and to obtain the original message, are the main goals of coding theory [2, 3].

There are two kinds of codes, namely error-detecting and error-correcting codes. The ISBN-10 (International Standard Book Number) code is an error-detecting code. The tenth digit d₁₀ can be obtained by d₁₀ = 11 − (10d₁ + 9d₂ + ... + 2d₉ (mod11)), such that any changes made to the other digits could be detected, although we could not know exactly which digit has changed [2]. The tenth digit on ISBN-10 code is called check digit, that is, an additional digit to help us detect whether a code has changed due to errors. This idea leads to error-correcting codes, which has the abilities of detecting and correcting errors, provided they are less or equal to correcting radius of the code [4]. Some applications of coding theory was shown by [5] on secure
video steganography using Hamming code, [6] on lossless data compression, [7] on development of Hamming codes as error-reducing codes, and [8] on biomedical engineering’s medical image authentication using Hamming code.

In this paper we will use two types of data for simulations, namely vector and text data. These data then will be represented as binary vectors which will be denoted as bold letters such as \( \mathbf{x}, \mathbf{y}, \mathbf{u} \) or \( \mathbf{v} \) and written as transposed row matrices such as \( [1 \ 1 \ 0 \ 1]^T \) or simply as 1101. Matrices will be denoted as italic capital letters such as \( A \) or \( H \). Simulations will be performed using SageMath mathematical software [9]. In this work we use sagetex package [10] to incorporate SageMath’s computations and literate programming onto Latex. Inline SageMath’s commands and source codes will be written as typewriter font style such as for \( i \) in range(3): print \( i \), while its display commands will be written as

\[
\text{sage: } \text{vector}(\text{GF}(2), [1,0,0,0])
\]

\[
(1, 0, 0, 0)
\]

aiming for clarity of writing. Sections will be organised as follows. Section 2 will shortly discuss some basic theories on linear and Hamming codes, including error-correcting codes, parity check and generator matrix encoders, and nearest neighbor and syndrome decoders. In Section 3, we will perform some simulations on data encoding, transmission, and decoding, with given data and methods using SageMath. We aim to show how message could be changed by errors, how often errors change messages, how to construct codes, how to encode, transmit and decode messages, and investigate properties of codes using SageMath. Finally in Section 4 we will discuss some conclusions based on works we did in preceding sections.

2. Preliminaries

In this section we will discuss some definitions and theories regarding encoding and decoding algorithms used in linear and Hamming codes.

2.1. Parity check encoder

In Section 1 we briefly discussed error-detecting codes such as the ISBN-10 code. In this section we will introduce error-correcting codes and parity check encoder. Digital data transmission consists of three stages: encoding, transmission, and decoding. These processes can be modeled as Figure 1 [4].

![Figure 1. Digital Data Transmission Model.](image)

Part (1) in Figure 1. is the information source. This informations then encoded in (2), and the encoded message is transmitted through channel (3). If noise (4) exists, then (3) is called noisy channel and there is a probability of errors which could change the encoded message. After encoded message is received, it will be decoded and corrected if possible in (5), and finally the decoded message will arrive at sink (6). Like on error-detecting codes, we need to add some additional check digits to the message such that any changes made to the other digits will be detected and corrected properly.
Let $F_q$ be a finite field with characteristic $q$. The set $M \subseteq F^k_q$ is called message space, which its elements are all possible vector messages of length $k$. The set $C \subseteq F^n_q$ is called code of length $n$ over $F_q$ which its elements then referred to as codewords. The error vector $e$ is the difference $x - y$ between the transmitted codeword $x$ and the received codeword $y$, where $x, y \in C$. Another concept we need to discuss is the Hamming distance. Hamming distance $d(x, y)$ between two vectors is the number of different coordinates in $x$ and $y$ [2, 4]. We need Hamming distance to define the minimum distance of a code $C$ with respect to Hamming distance. In [2], the minimum distance of code $C$ is defined by

$$d_{\text{min}}(C) = \min d(x, y)$$  

where $x \neq y$ for all $x, y \in C$.

**Definition 2.1 (Linear Codes)**

Let $H$ be a full rank $(n - k) \times n$ check matrix over $F_q$. The set

$$C = \{ x : Hx^T = 0, x \in F^n_q \}$$

is called a linear code $C$ over $F_q$ of length $n$, where $k$ and $n - k$ is the length of message and check digits, respectively [2]. Transmission rate is defined as $k/n$, and if $q = 2$, then $C$ is called binary codes.

From Definition 2.1 we see that linear code $C$ actually is the set of all solutions of matrix equation $Hx^T = 0$. For instance let vector $[x_1 \ x_2 \ x_3]$ be a message $a$ and let $x_4, x_5, x_6$ be check digits. We can obtain these check digits by

$$x_1 + x_2 + x_4 = 0$$
$$x_1 + x_3 + x_5 = 0$$
$$x_2 + x_3 + x_6 = 0$$

or equally written as a check matrix

$$H = \begin{bmatrix}
1 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 1
\end{bmatrix}.$$ 

By using Equation (2) or $H$, message $a = [0 \ 1 \ 0]$ can be encoded to $[0 \ 1 \ 0 \ 1 \ 0 \ 1]$. Such method is called parity check encoder.

### 2.2 Generator matrix encoder

Suppose we have $C$ an $(n, k)$ linear code over $F_q$ with check matrix $H$. If $H = (A | I_{n-k})$, then we can define Generator Matrix of $C$ as follows [2].

**Definition 2.2.** Let $C$ be a linear $(n, k)$ code over $F_q$ with check matrix $H = (A | I_{n-k})$. Matrix $G = (I_k | -A^T)$ is called Generator Matrix of $C$, and codewords $x \in C$ of length $n$ can be obtained by $x = aG$, for all vector messages $a$ in message space.

Continuing the discussion in previous subsection, with given $H$ we can obtain generator matrix

$$G = \begin{bmatrix}
1 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 1
\end{bmatrix}$$

and we can obtain codeword $x = aG = [0 \ 1 \ 0] G = [0 \ 1 \ 0 \ 1 \ 0 \ 1]$. 

3
2.3. Nearest neighbor decoder

We have discussed two encoding algorithms in previous subsections. Now we will continue on one of decoding algorithms called nearest neighbor decoder. Suppose we have a linear code $C \subseteq F_q^n$. The set $S_r(x)$ is called sphere of radius $r$ with respect to Hamming distance, centered at $x$. Or we can write

$$S_r(x) = \{ y \in F_q^n : d(x, y) \leq r, x \in C \}. \quad (3)$$

This process is also referred to as sphere packing [2, 4], that is, constructing spheres with codewords as its center and contain all vectors within its radius. If vector $y$ is received and it belongs to precisely one of the spheres, then it will be decoded to its center, namely vector $x$.

We have to make the radius of these spheres as large as possible to be able to decode codewords as much as possible. On the other hand, the radius have to be small enough such that there are no two spheres intersect. If such event occurred, then the codeword that belongs to the intersection cannot be decoded.

**Theorem 2.1.** A linear code $C$ with $d_{\min}(C) = d$ can detect up to $d - 1$ errors and correct up to $\lfloor \frac{1}{2}d \rfloor$ errors.

Suppose we transmit a codeword $x$ and vector $y$ received. If $y$ belongs to one of the spheres, say $S_r(x_1)$, then $y$ will be decoded to $x_1$, which is the center of the sphere. This method is called nearest neighbor decoding.

2.4. Syndrome decoder

Let $C$ be a linear $(n, k)$ code over $F_q$ and consider $C$ as subspace of $F_q^n$. The set $F_q^n/C$ contains all cosets $a + C = \{ a + x | x \in C \}$ where $a \in F_q^n$. If a vector $y$ is received, then it belongs to an exactly one of these cosets [2].

**Definition 2.3.** Let $C$ be a linear $(n, k)$ code over $F_q$ with check matrix $H$. Vector $S(y) = Hy^T$ of length $n - k$ is the syndrome of $y$, for any $y \in F_q^n$.

Let us consider a linear code $C$ and a codeword $x \in C$ is transmitted. Suppose a vector $y$ belongs to a particular coset $e + C$, where $e$ is a vector of minimum weight in the coset. If $y$ is received, then its syndrome $S(y)$ must be equal to $S(e)$ because $Hy^T = H(x + e)^T = He^T$. Thus, $y$ is decoded to $x = y - e$.

2.5. Hamming code

Hamming code is a special class of linear codes. It is an error-correcting codes with 2 detection radius and 1 correcting radius with high transmission rate [2, 7].

**Definition 2.4.** Let $C_m$ be a binary code of length $n = 2^m - 1$ where $m \geq 2$. If $H$ is the $m \times (2^m - 1)$ parity check matrix which columns consist of all nonzero binary vectors of length $m$, then $C_m$ is called the Hamming code.

All encoder and decoder algorithms we discussed earlier also hold for Hamming code because it is inherently a linear code.

3. Simulations of vector and text data

SageMath provides a broad range of class (or object) in coding theory [11], including linear and Hamming codes. These classes can be accessed in SageMath under codes class. In this work we will use codes.LinearCode() and codes.HammingCode() for simulations. Simulations implemented in SageMath using Jupyter Notebook [12].
3.1. Data definition and representation

Data exact definition and representation is needed for simulation programs to work properly. For text data, we will consider a list $T = \{, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, .\}$ of both lower and upper case letters, numbers, full stop mark (point) and whitespace with total of 64 elements. Each element of the list will be represented by a binary vector of its index digit, ranging from 0 to 63. For instance, 'c' which has index of 3, has binary vector representation of 000011.

3.2. Simulation workflow

First we will briefly define and discuss a general workflow of simulation processes we about to perform.

**Require:** C, input, channel, encoder, decoder

1. data ← make binary vector representations of input
2. **procedure** Transmission(C, channel, encoder, decoder)
   3. for all $j \in$ data do
      4. encoded ← ENCODE(C, $j$, encoder)
      5. transmitted ← TRANSMIT(channel, encoded)
      6. decoded ← DECODE(C, encoded, decoder)
   7. end for
3. output ← revert binary vectors back to original values

First we need to represent the input to its binary vectors form. Then, encoding, transmission and decoding will be performed element-wise. If the output is equal to the input, then it is successfully transmitted. Otherwise, it has errors greater than correcting radius of the code, hence incorrectly decoded.

3.3. Codes and channels construction

First we construct linear code $C$ by using parity check and generator matrices which will be defined as follows:

$$H = \begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}, \quad G = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0
\end{bmatrix},$$

where $H$ is the parity check matrix and $G$ is the generator matrix. After defining these two matrices, we can then construct linear code $C$ by running

```sage
C = codes.LinearCode(generator=G)
sage: C
\[12, 6\] linear code over GF(2)
sage: C.random_element()
(0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0)
```

In SageMath. In the third command above, we can take a random element of $C$ i.e. a codeword, by running $C.random_element()$ which should give an output of random vector of length 12.
Channel construction is provided by `channels.StaticErrorRateChannel` class which can construct channel that creates static errors or randomly choose errors between non-negative integers \( a \) and \( b \) to our codewords. If otherwise we want a randomly chosen errors instead of static ones, then we need only to replace \( E \) with a tuple \((a,b)\), where \( a \) and \( b \) is the lower and upper bound, respectively. In this section, we use 0, 1 and 3 as values for \( E \).

3.4. Encoding and transmission processes
Suppose we have a message vector \( msg1 = [1 \ 0 \ 0 \ 1 \ 1 \ 0] \) and \( msg2 = [1 \ 0 \ 1 \ 0] \). We can encode the first message using parity check encoder (which is named ‘Systematic’ in SageMath) using `encode()` method. We can use generator matrix encoder on both codes by changing the string value of `encoder_name` to ‘GeneratorMatrix’ which will yield the same results.

As we have encoded messages or codewords, now we can transmit them. Transmission in SageMath is conveniently provided by `transmit` method. We can see that given two errors, `encoded1` has changed after transmission.

3.5. Correcting and decoding processes
Suppose the transmitted codewords arrive. Then we can decode these transmitted codewords using \( C\).decode() or \( C\).decode_to_message() methods. We will consider three cases, which in each case the number of errors is 0, 1 and 3, respectively.

a) For 0-error, it is obvious that \( \text{encoded1} \) will be exactly the same as \( \text{transmitted1} \), for any given codewords. Hence the value of \( \text{decoded1} \) and \( \text{decoded2} \) will be equal to \( \text{msg1} \) and \( \text{msg2} \), respectively. Suppose that we use nearest neighbor decoder. There is a sphere \( S_1(1010101) \) which contains all vectors \( y \) such that \( d(1010101, y) \leq 1 \). Because 1010101 is the center of \( S_1(1010101) \), then it will be decoded back correctly to 1010. If we use syndrome decoder, then we must compute the syndrome \( S(1010101) = H \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}^T = 0 = S(0000000) \). Hence, 1010101 will be decoded back correctly to 1010. In other words, if an encoded codeword is transmitted without being changed, then it will be decoded back correctly to its original message.

b) As discussed earlier that \( C \) and \( D \) has minimum distance of 3. Thus both of them can detect up to 2 errors and correct up to 1 error. vectors \( \text{transmitted1} \) and \( \text{transmitted2} \) will be decoded back correctly as expected. Suppose that we use syndrome decoder. Then we must compute the syndrome \( S(0010101) = H \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}^T = 100 = S(1000000) \). Hence 0010101 will be corrected as \( y - e = 0010101 - 1000000 = 1010101 \) and decoded back correctly to 1010.

c) Consider transmitted vectors which has 3 errors. As discussed earlier that both codes are unable to detect nor to correct them. Hence it will be incorrectly decoded. Hamming code \( D \) has a minimum distance of 3. This means that if both of \( x_1 \) and \( x_2 \) are codewords in \( D \), then the minimum distance \( d(x_1, x_2) \) between them would be 3. Because \( d(1010101, 1000011) = 3 \), then 1000011 actually is another codeword in \( D \). If we compute its syndrome \( S(1000011) = 0 \), then it will be decoded back *incorrectly* to 1000.

Simulation results using vector data we performed up to this point is presented in Table 1.

3.6. Simulations using text data
In this subsection we will perform two simulations, namely linear code \( C \) and Hamming code \( D \) using text data. Unlike in previous subsection, we will not provide any literate programming nor inline computations using sagetex. Instead, we will present only the simulation results as
Table 1. Vector simulation result using Hamming code $D$.

| $n$-errors | MESSAGE | ENCODED | TRANSMITTED | DECODED |
|------------|---------|---------|-------------|---------|
| 0          | 1010    | 1010101 | 1010101     | 1010    |
| 1          | 1010    | 1010101 | 0010101     | 1010    |
| 3          | 1010    | 1010101 | 1000011     | 1000    |

Table 2. Text Data simulation result using linear code $C$.

| $n$-errors | DECODED MESSAGE                   |
|------------|----------------------------------|
| 0          | This is a dummy text             |
| 1          | This is a dummy text             |
| 2          | This is a dummy text             |
| 3          | NHCKbhKxefZe9g5zoh4l              |
| 4          | 0yIqq7Mntt3YnK7msLV5              |
| 5          | IHQETccHfBep86XuUU .             |
| 6          | sRY6DhAA7GnQfkiSJ5               |

Table 3. Text Data simulation result using Hamming code $D$.

| $n$-errors | DECODED MESSAGE                   |
|------------|----------------------------------|
| 0          | This is a dummy text             |
| 1          | This is a dummy text             |
| 2          | i T0hadi.0 s.0tu9xdT              |
| 3          | ehetteyiT0hsTm9uumy               |
| 4          | .mmu0e.tymxy.dex xea             |
| 5          | s0eTyt9y TiaeTsadhyi             |
| 6          | eyxt.xt.0.9muuh.sTis             |

From Table 2 and Table 3 we can see that once the error exceeds the correcting radius of the code, the resulting decoded message changed considerably. This shows the importance of coding theory in the development of reliability and integrity of digital communication.

4. Conclusion

In this work we show that we can use SageMath for linear and Hamming code simulations. First we define the codes by defining either parity check or generator matrices, then create a channel which randomly choose errors between given interval. We can encode our messages using encode() methods to obtain our codewords. These codewords then can be transmitted through our previously created channel by using transmit() method. Depending on the number of tables with text message "This is a dummy text". Full implementation of the simulations can be obtained in our GitHub repository in https://github.com/tdtimur/sage3rdicompac2017.
errors given to codewords, we can decode these transmitted codewords by using either `decode()` or `decode_to_message()` methods. If the number of errors is less than or equal to the code’s correcting radius, then the transmitted codeword will be corrected and decoded back correctly to its original message. Otherwise, it will be incorrectly decoded to another codeword or message. Simulation result on text data shows that once the error exceeds the correcting radius of the code, the resulting decoded message could be changed significantly. This work can be extended by using different codes or by using different types of data such as image and audio.

References

[1] Kurose J F and Ross K W 2013 *Computer Networking: A Top-Down Approach* (Addison-Wesley)
[2] Lidl R and Pilz G 2012 *Applied abstract algebra* (Springer Science & Business Media)
[3] Berlekamp E R 2015 *Algebraic coding theory* (World Scientific Publishing Co)
[4] Hamming R W 1980 *Coding and Theory* (Prentice-Hall)
[5] Mstafa R J and Elleithy K M 2014 *Systems, Applications and Technology Conference (LISAT), 2014 IEEE Long Island* (IEEE) pp 1–6
[6] Al-Bahadili H 2008 *Computers & Mathematics with Applications* 56 143–150
[7] Rurik W and Mazumdar A 2016 *Information Theory Workshop (ITW), 2016 IEEE* (IEEE) pp 404–408
[8] Abadi M A M, Danyali H and Dehnavi M N 2010 *Biomedical Engineering (ICBME), 2010 17th Iranian Conference of* (IEEE) pp 1–4
[9] Developers T 2017 Sagemath, the sage mathematics software system (version 8.0)
[10] Drake D et al. 2009 *Available from CTAN/,macros/latex/contrib/sagetex*
[11] Joyner D and Miller R 2008 *ACM Communications in Computer Algebra* 42 74–78
[12] Perez F and Granger B E 2015 Project jupyter: Computational narratives as the engine of collaborative data science Tech. rep. Technical Report. Technical report, Project Jupyter