




# Linear Algebra in Coding Theory

Applications and Analysis

Manas, Jahnavi, Vasana



# Introduction

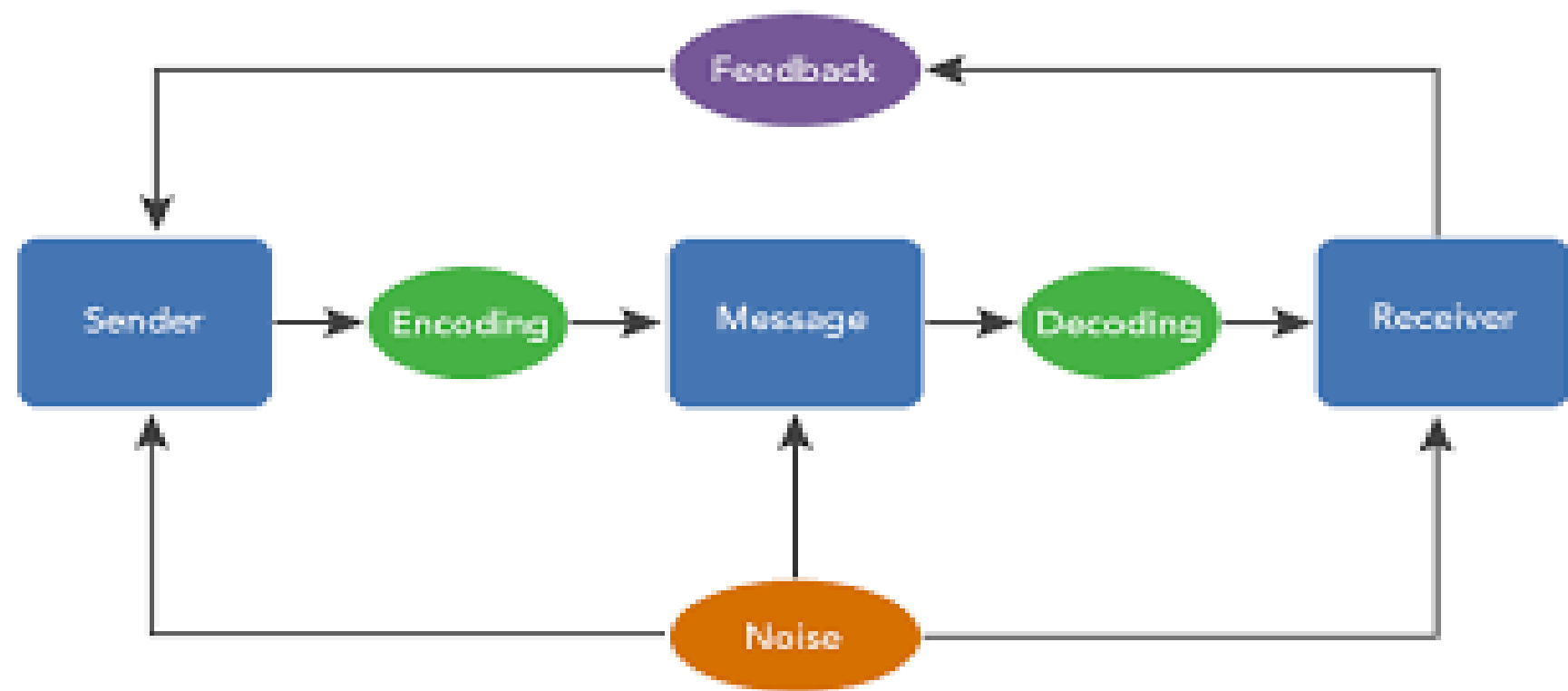
- Coding theory is the study of the properties of codes and their respective fitness for specific applications.
- Codes are used for data compression, cryptography, error detection and correction, data transmission and data storage. This field is extremely vast and has applications in ensuring reliable and secure communication. This presentation will focus on the following areas of study:
  1. Linear Codes
  2. Parity Check Matrices
  3. Syndromes
  4. Encoding and Decoding

# Linear Applications

- Method One : Cryptography
  - Invertible matrices
  - Matrix multiplication
- Method Two : Coding Theory
  - Modular arithmetic
  - Error correcting code
  - Matrix multiplication
  - Binary code

# Cryptography

- Study of encoding and decoding secret messages
- Useful in sending sensitive information so that only the intended receivers can understand the message
- A common use of cryptography is to send government secrets.



A	1
B	2
C	3
D	4
E	5
F	6
G	7
H	8
I	9
J	10
K	11
L	12
M	13
N	14

# Encoding

- First we will assign numbers to represent each letter of the alphabet.
- We then create a “plaintext” matrix that holds the message in terms of numbers.
- Then we pick an invertible square matrix, which can be multiplied with the “plaintext matrix”.

O	15
P	16
Q	17
R	18
S	19
T	20
U	21
V	22
W	23
X	24
Y	25
Z	26
_	27
'	28

# Encrypting the Message

$$\begin{bmatrix} 1 & 2 & -1 \\ 2 & 2 & 4 \\ 1 & 3 & -3 \end{bmatrix} \times \begin{bmatrix} 6 & 18 & 5 & 5 & 27 & 12 & 1 & 21 & 14 & 4 & 18 & 25 & 27 \\ 13 & 15 & 14 & 5 & 25 & 27 & 21 & 14 & 4 & 5 & 18 & 27 & 19 \\ 15 & 13 & 5 & 15 & 14 & 5 & 28 & 19 & 27 & 4 & 5 & 19 & 11 \end{bmatrix} =$$

Encoding matrix

↑  
Plaintext

$$\begin{bmatrix} 17 & 35 & 28 & 0 & 63 & 61 & 15 & 30 & -5 & 10 & 49 & 60 & 54 \\ 98 & 118 & 58 & 80 & 160 & 98 & 156 & 146 & 144 & 34 & 92 & 180 & 136 \\ 0 & 24 & 32 & -25 & 60 & 78 & -20 & 6 & -55 & 7 & 57 & 49 & 51 \end{bmatrix}$$

Ciphertext

# Deciphering the message

- In order to decode the message, we would have to take the inverse of the encoding matrix to obtain the decoding matrix.
- Multiplying the decoding matrix with the ciphertext would result in the plaintext version.
- Then the arbitrarily assigned number scheme can be used to retrieve the message.



A 1  
B 2  
C 3  
D 4  
E 5  
F 6  
G 7  
H 8  
I 9  
J 10  
K 11  
L 12  
M 13  
N 14

Decrypting the Message

$$\begin{bmatrix} 9 & -3/2 & -5 \\ -5 & 1 & 3 \\ -2 & 1/2 & 1 \end{bmatrix}$$
$$X$$
$$\begin{bmatrix} 17 & 35 & 28 & 0 & 63 & 61 & 15 & 30 & -5 & 10 & 49 & 60 & 54 \\ 98 & 118 & 58 & 80 & 160 & 98 & 156 & 146 & 144 & 34 & 92 & 180 & 136 \\ 0 & 24 & 32 & -25 & 60 & 78 & -20 & 6 & -55 & 7 & 57 & 49 & 51 \end{bmatrix}$$
$$=$$
$$\begin{bmatrix} 6 & 18 & 5 & 5 & 27 & 12 & 1 & 21 & 14 & 4 & 18 & 25 & 27 \\ 13 & 15 & 14 & 5 & 25 & 27 & 21 & 14 & 4 & 5 & 18 & 27 & 19 \\ 15 & 13 & 5 & 15 & 14 & 5 & 28 & 19 & 27 & 4 & 5 & 19 & 11 \end{bmatrix}$$

O 15  
P 16  
Q 17  
R 18  
S 19  
T 20  
U 21  
V 22  
W 23  
X 24  
Y 25  
Z 26  
\_ 27  
' 28

A 1

B 2

C 3

D 4

E 5

F 6

G 7

H 8

I 9

J 10

K 11

L 12

M 13

YOU MIGHT WANT TO READ THIS.



<i>F</i>	<i>R</i>	<i>E</i>	<i>E</i>	<i>_</i>	<i>L</i>	<i>A</i>	<i>U</i>	<i>N</i>	<i>D</i>	<i>R</i>	<i>Y</i>	<i>_</i>
6	18	5	5	27	12	1	21	14	4	18	25	27
<i>M</i>	<i>O</i>	<i>N</i>	<i>E</i>	<i>Y</i>	<i>_</i>	<i>U</i>	<i>N</i>	<i>D</i>	<i>E</i>	<i>R</i>	<i>_</i>	<i>S</i>
13	15	14	5	25	27	21	14	4	5	18	27	19
<i>O</i>	<i>M</i>	<i>E</i>	<i>O</i>	<i>N</i>	<i>E</i>	<i>'</i>	<i>S</i>	<i>_</i>	<i>D</i>	<i>E</i>	<i>S</i>	<i>K</i>
15	13	5	15	14	5	28	19	27	4	5	19	11

O 15

P 16

Q 17

R 18

S 19

T 20

U 21

V 22

W 23

X 24

Y 25

Z 26

\_ 27

' 28

# Coding Theory

- In order to decode the message, we would have to take the inverse of the encoding matrix to obtain the decoding matrix.
- Multiplying the decoding matrix with the ciphertext would result in the plaintext version.
- Then the arbitrarily assigned number scheme can be used to retrieve the message.

# Coding Theory

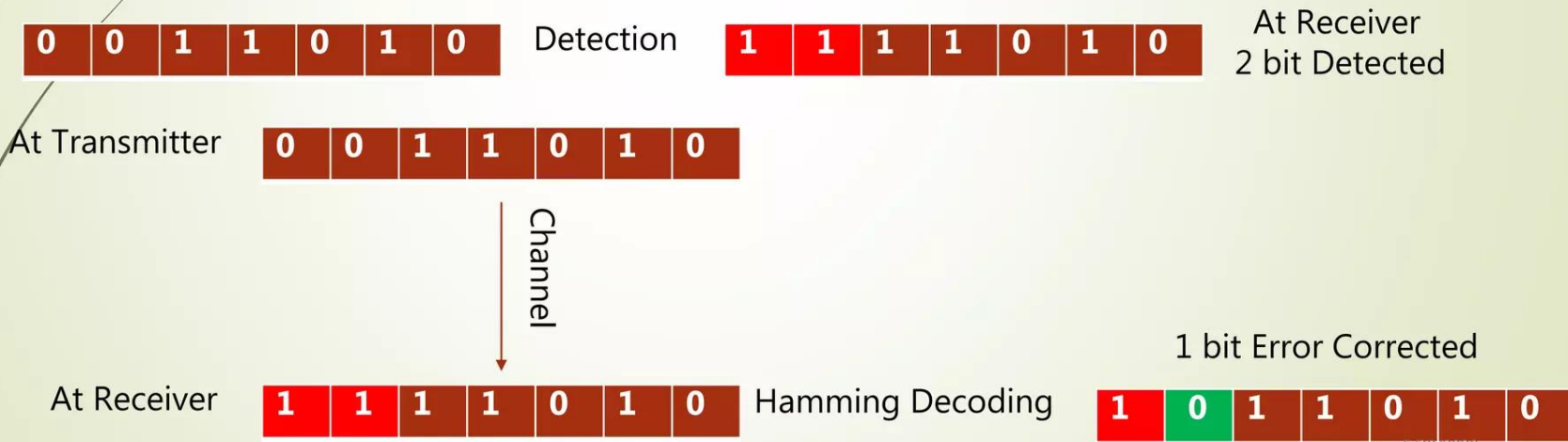


- Have the ability to detect the presence of errors, as well as reconstruct the original, error-free data
- Coding Theory applications exist in many forms of digital technology. An example would be sending a message across a noisy channel with perfect reception.

# Hamming code:

6

- Around 1947 Richard W. Hamming developed technique for detecting and correcting single bit errors in transmitted data. His technique requires that three parity bits (or check bits) be transmitted with every four data bits. The algorithm is called a (7, 4) code, because it requires seven bits to encoded four bits of data.
- In communication, **Hamming codes** are a family of **linear** error-correcting. Hamming codes can detect up to **two-bit** or correct **one-bit** errors without detection of uncorrected errors.



❑ Now Lets See what is these Hamming codes...

# Implementation of Hamming Code

8

- To implement Hamming code firstly we calculate Parity Bits
- **Generator Matrix-** It is matrix of  $[I,P]$  use to encode message bits
- For Example
- $p_1 = d_2 + d_3 + d_4$   
 $p_2 = d_1 + d_3 + d_4$   
 $p_3 = d_1 + d_2 + d_4$

$$G = \left[ \begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array} \right]$$

(a)

$$H = \left[ \begin{array}{ccc} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right]$$

(b)

# Continued...

- So for any Message bits (4 bits) we can get code directly using Generator matrix

$$C=[d]*[G]$$

- We send this **Codeword** through **Channel** and at receiver we decode this codeword using parity check matrix also we find error is there or not
- Let at Receiver we get a received codeword **"r"**

# Syndrome and Decoding

10

- **Syndrome**-The pattern of errors, called the error syndrome, identifies the bit in error.
- If all parity bits are correct, there is no error. Otherwise, the sum of the positions of the erroneous parity bits identifies the erroneous bit.

$$S = [r] \cdot [H^T]$$

If  $S = 0$  No Errors

$S \neq 0$  Errors are there and we got Error Vector **e**

- **S** also gives location of error and this location vector verified from  $[H^T]$



# Continued...

11

- At last after getting the error vector  $e$

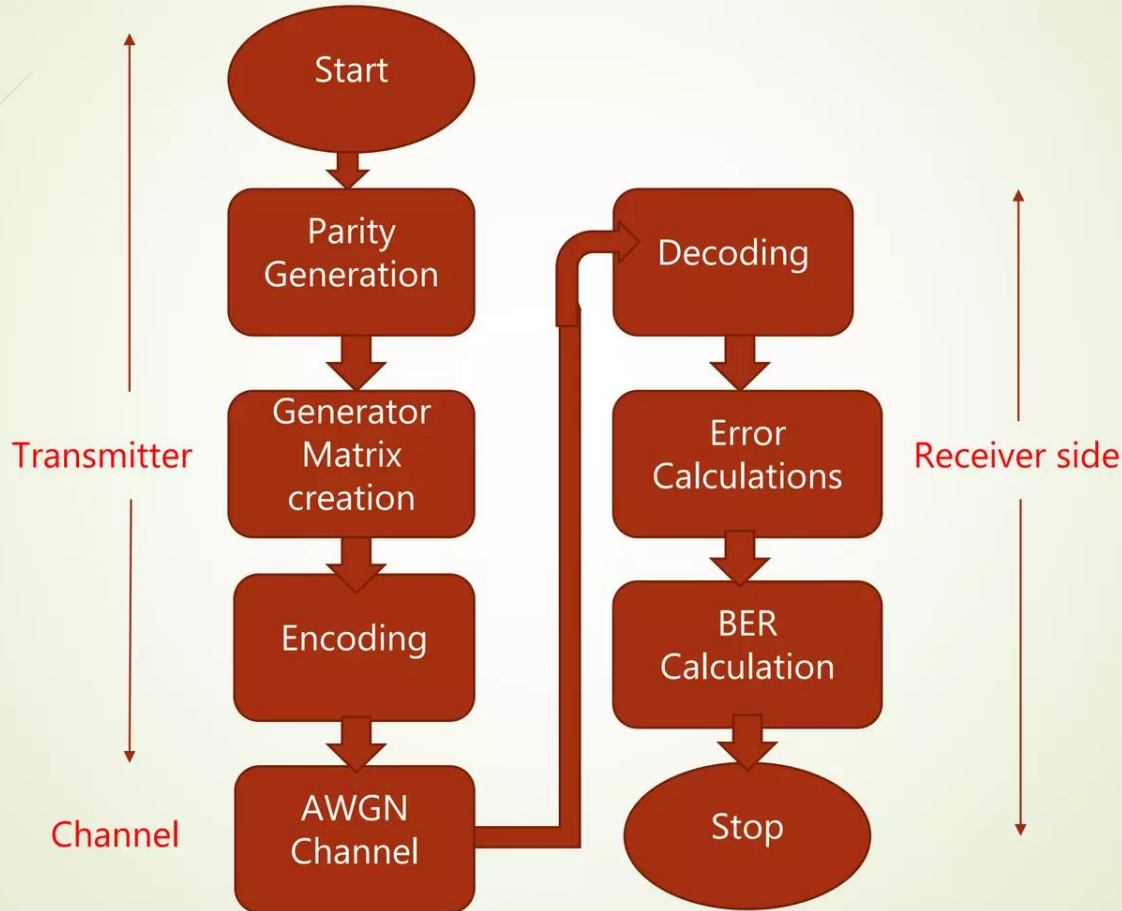
Correct codeword =  $r + e$

- ❑ This Syndrome Decoding for Hamming Code and error vector can't correct more than one error vector
- ❑ For correction more errors we have to add more redundancy

# MATLAB Implementation

12

➤ Flow Chart



# Conclusion

15

- Hamming Code system is very much important where we need only one error correction and two error detection system, it is very easy to implement and for large number of block of bits it is more efficient.

# Theorem 3.34 (Basic Ideas)

- Standard generator matrix  $G = \begin{bmatrix} I_k \\ A \end{bmatrix}$
- Standard parity (error-detecting matrix) check matrix  $P = \begin{bmatrix} B & I_{n-k} \end{bmatrix}$
- $P$  is the parity check matrix associated with  $G$  if and only if  $A = B$ .
- The corresponding  $(n, k)$  binary code is (single) error-correcting if and only if the columns of  $P$  are nonzero and distinct.
- Note: most digital messages are written in sequences of 0's and 1's in binary code

# (7, 4) Hamming Code

- We will use three parity check equations:

$$n - k = 3 \text{ and } k = n - 3$$

- $P = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$  ← Eq'n 1  
← Eq'n 2  
← Eq'n 3  
3 x 7  
matrix

Therefore, theorem 3.34 states  $G =$

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

7 x 4 matrix  
(7,4) Hamming  
Code

# Encoding

- We want to encode  $x = [1 \ 1 \ 0 \ 1]^T$

$$\text{So, } C = G \cdot x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 3 \\ 2 \\ 2 \end{bmatrix}$$

# Parity Check

- $P_c' = 0$  for this code to be a correct code

$$P_c' = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & \underline{\underline{0}} \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} \underline{\underline{4}} \\ 2 \\ 2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

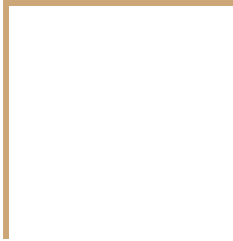
# Success!

Since  $Pc'=0$ , our code is correct, however had we made an error in multiplying our first batch of matrices to obtain  $\underline{c}$ ,  $Pc' \neq 0$ ...say we had the incorrect  $C$  vector (0 1 1 1 0 1 0) and did our parity check...




# What is encoding and decoding data in brief?

- Encoding refers to the process of converting information or data into a specific format that can be easily understood or processed by a computer or other devices. It involves transforming raw data into a structured form that can be transmitted, stored, or manipulated efficiently.
- One commonly used character encoding system is ASCII (American Standard Code for Information Interchange), which assigns numeric values to characters. For example, the letter 'A' is represented by the ASCII value 65, 'B' by 66, and so on. ASCII encoding is limited to the basic Latin alphabet and some common symbols.
- Decoding is the process of converting encoded data back into its original form or representation. It is the reverse operation of encoding and is necessary to interpret or understand data that has been previously encoded.



# Error Correcting Codes - Example

Reed Soloman Code



# Encoding

- Computing the codeword is a linear mapping, which satisfies the matrix equation  $C(x) = x^t A$  for a  $k \times n$  matrix  $A$  (which contains elements from  $F$ ). (where  $x^t$  is the transpose of the matrix  $x$ , the message polynomial.)

$$A = \begin{bmatrix} 1 & \dots & 1 & \dots & 1 \\ a_1 & \dots & a_k & \dots & a_n \\ a_1^2 & \dots & a_k^2 & \dots & a_n^2 \\ \vdots & & \vdots & & \vdots \\ a_1^{k-1} & \dots & a_k^{k-1} & \dots & a_n^{k-1} \end{bmatrix}$$

# Decoding

- Decoding and error correction can be done with the Berlekamp-Welch algorithm

$$b_i E(a_i) = E(a_i) F(a_i)$$

( $b_i$  is the received codeword at  $a_i$ ,  $E(a_i)$  is the error locator polynomial and  $F(a_i)$  is the original encoding polynomial.)

- $E(a_i) = 0$  for error cases when  $b_i \neq F(a_i)$
- $E(a_i) \neq 0$  when  $b_i = F(a_i)$
- $Q(a_i) = E(a_i) F(a_i)$
- $b_i E(a_i) - Q(a_i) = 0$

$$b_i(e_0 + e_1 a_i + e_2 a_i^2 + \dots + e_e a_i^e) - (q_0 + q_1 a_i + q_2 a_i^2 + \dots + q_q a_i^q) = 0$$

where  $q = n - e - 1$ .

Since  $e_e$  is constrained to be 1, the equations become:

$$b_i(e_0 + e_1 a_i + e_2 a_i^2 + \dots + e_{e-1} a_i^{e-1}) - (q_0 + q_1 a_i + q_2 a_i^2 + \dots + q_q a_i^q) = -b_i a_i^e$$

$$\begin{bmatrix} b_1 & b_1 a_1 & -1 & -a_1 & -a_1^2 & -a_1^3 & -a_1^4 \\ b_2 & b_2 a_2 & -1 & -a_2 & -a_2^2 & -a_2^3 & -a_2^4 \\ b_3 & b_3 a_3 & -1 & -a_3 & -a_3^2 & -a_3^3 & -a_3^4 \\ b_4 & b_4 a_4 & -1 & -a_4 & -a_4^2 & -a_4^3 & -a_4^4 \\ b_5 & b_5 a_5 & -1 & -a_5 & -a_5^2 & -a_5^3 & -a_5^4 \\ b_6 & b_6 a_6 & -1 & -a_6 & -a_6^2 & -a_6^3 & -a_6^4 \\ b_7 & b_7 a_7 & -1 & -a_7 & -a_7^2 & -a_7^3 & -a_7^4 \end{bmatrix} \begin{bmatrix} e_0 \\ e_1 \\ q0 \\ q1 \\ q2 \\ q3 \\ q4 \end{bmatrix} = \begin{bmatrix} -b_1 a_1^2 \\ -b_2 a_2^2 \\ -b_3 a_3^2 \\ -b_4 a_4^2 \\ -b_5 a_5^2 \\ -b_6 a_6^2 \\ -b_7 a_7^2 \end{bmatrix}$$

Using RS(7,3), GF(929), and the set of evaluation points  $a_i = i - 1$ :

$$a = \{0, 1, 2, 3, 4, 5, 6\}$$

If the message polynomial is  $p(x) = 003x^2 + 002x + 001$ , the codeword is obtained as follows:

Message vector  $\mathbf{x} = ((x_1), (x_2), \dots, (x_n)) \in F^k$

$$\text{Vandermonde matrix } \mathbf{A} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ a_1 & a_2 & \dots & a_n \\ a_1^2 & a_2^2 & \dots & a_n^2 \end{bmatrix}$$

To compute the codeword, we calculate  $\mathbf{C}(\mathbf{x}) = \mathbf{x}^T \cdot \mathbf{A}$ , where  $\mathbf{x}^T$  is the transpose of  $\mathbf{x}$ .

$$\mathbf{x} = ((x_1), (x_2), (x_3)) = ((003), (002), (001))$$

$$\mathbf{a} = \{0, 1, 2, 3, 4, 5, 6\}$$

# Working Exam

The Vandermonde matrix  $\mathbf{A}$  is:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0^2 & 1^2 & 2^2 & 3^2 & 4^2 & 5^2 & 6^2 \end{bmatrix}$$

Calculating  $\mathbf{x}^T \cdot \mathbf{A}$ :

$$\mathbf{x}^T = [003 \quad 002 \quad 001]$$

$$\mathbf{x}^T \cdot \mathbf{A} = [003 \quad 002 \quad 001] \cdot \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0^2 & 1^2 & 2^2 & 3^2 & 4^2 & 5^2 & 6^2 \end{bmatrix} = [001 \quad 006 \quad 017 \quad 034 \quad 057 \quad 086 \quad 121]$$

$$c = \{001, 006, 017, 034, 057, 086, 121\}$$

Errors in transmission might cause this to be received instead:

$$b = c + e = \{001, 006, 123, 456, 057, 086, 121\}$$

Errors in transmission might cause this to be received instead:

$$b = c + e = \{001, 006, 123, 456, 057, 086, 121\}$$

The key equations are:

$$b_i E(a_i) - Q(a_i) = 0$$

Assume maximum number of errors:  $e = 2$ . The key equations become:

$$b_i(e_0 + e_1 a_i) - (q_0 + q_1 a_i + q_2 a_i^2 + q_3 a_i^3 + q_4 a_i^4) = -b_i a_i^2$$

$$\begin{bmatrix} 001 & 000 & 928 & 000 & 000 & 000 & 000 \\ 006 & 006 & 928 & 928 & 928 & 928 & 928 \\ 123 & 246 & 928 & 927 & 925 & 921 & 913 \\ 456 & 439 & 928 & 926 & 920 & 902 & 848 \\ 057 & 228 & 928 & 925 & 913 & 865 & 673 \\ 086 & 430 & 928 & 924 & 904 & 804 & 304 \\ 121 & 726 & 928 & 923 & 893 & 713 & 562 \end{bmatrix} \begin{bmatrix} e_0 \\ e_1 \\ q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} 000 \\ 923 \\ 437 \\ 541 \\ 017 \\ 637 \\ 289 \end{bmatrix}$$

Using Gaussian elimination:

$$\begin{bmatrix} 001 & 000 & 000 & 000 & 000 & 000 & 000 \\ 000 & 001 & 000 & 000 & 000 & 000 & 000 \\ 000 & 000 & 001 & 000 & 000 & 000 & 000 \\ 000 & 000 & 000 & 001 & 000 & 000 & 000 \\ 000 & 000 & 000 & 000 & 001 & 000 & 000 \\ 000 & 000 & 000 & 000 & 000 & 001 & 000 \\ 000 & 000 & 000 & 000 & 000 & 000 & 001 \end{bmatrix} \begin{bmatrix} e_0 \\ e_1 \\ q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} 006 \\ 924 \\ 006 \\ 007 \\ 009 \\ 916 \\ 003 \end{bmatrix}$$

$$Q(x) = 003x^4 + 916x^3 + 009x^2 + 007x + 006$$

$$E(x) = 001x^2 + 924x + 006$$

$$Q(x)/E(x) = P(x) = 003x^2 + 002x + 001$$

Recalculate  $P(x)$  where  $E(x) = 0$ :  $\{2, 3\}$ .

Calculate  $F(a_2 = 1) = 017$  and  $F(a_3 = 4) = 034$  to produce corrected codeword:  $c = \{001, 006, 017, 034, 057, 086, 121\}$



# Python Code - Encoding Process

```
import numpy as np

p = np.array([3, 2, 1])

a = np.array([0, 1, 2, 3, 4, 5, 6])

A = np.vander(a, N=len(p), increasing=True)

x = p[::-1]

C = np.mod(np.dot(x, A.T), 929)

print("Codeword:", C)
```



# SOTA

The Cayley Complex



# Local Testability

Local testability ensures quick and efficient verification of message correctness where messages can be tested in a few spots to ascertain their accuracy.

Graphs have been used in coding since the invention of the Hamming code.

Lubotzky and Dinur perfected the use of graphs in coding in 2021.

The left-right Cayley complex is a complex and elegant graphing method.

It utilizes two-dimensional squares as fundamental units, assigning information bits and parity bits to nodes and edges.

# Local Testability

The graph's structure is crucial for achieving local testability.

It resembles the spine of a booklet with branching sections.

Errors in bits (squares) become visible from multiple nodes, facilitating error detection from distant nodes.

The left-right Cayley graph allows errors to be observed by nodes that are not directly connected by an edge.

This means that a single error can have a broader impact and be detected by multiple nodes in the graph.



**THANK  
YOU**