THE MCELIECE CRYPTOSYSTEM

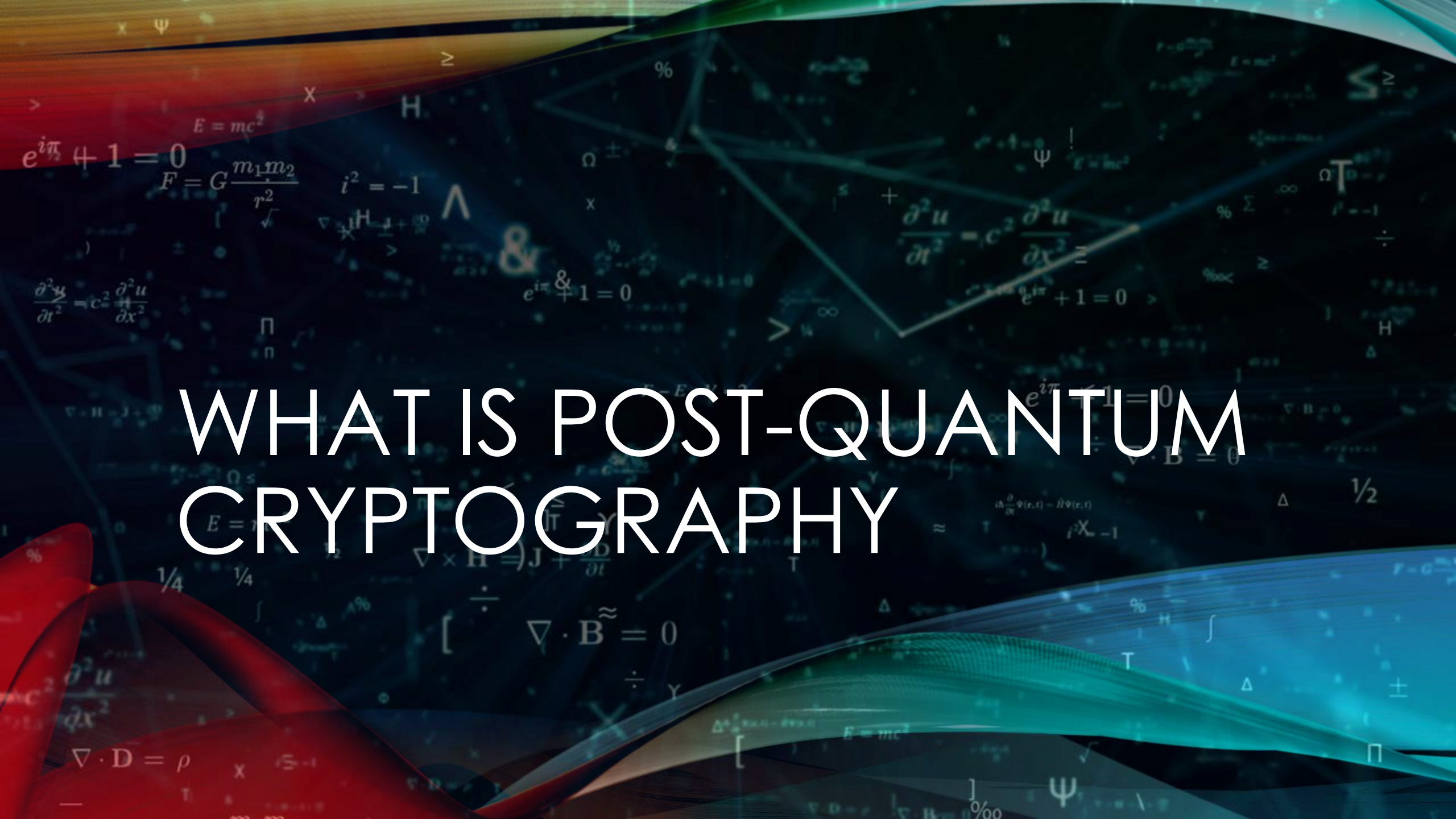
A Case of Post-Quantum Cryptography

SETUP

- To download the repository, run the following command:

  git clone https://github.com/Mr-JLD01/McEliece-Assignment.git

- Includes workshop assignment details



WHAT IS POST-QUANTUM CRYPTOGRAPHY

WHY THE DISTINCTION?

Classical

Helped to define the fundamentals

Pushed the limits of what could be done by hand and mechanical machines

Modern

Use of computers allows us to make the weakest point the human element

Shifted the usual alphabets to binary

Easier for everyone to have access to privacy

Post-Quantum

Cryptosystems made in response to Shor's algorithm

Push is to get ahead of the curve

Many organizations are taking a Save Now Decrypt Later approach to data

GENERAL PROCESS OF SHOR'S ALGORITHM

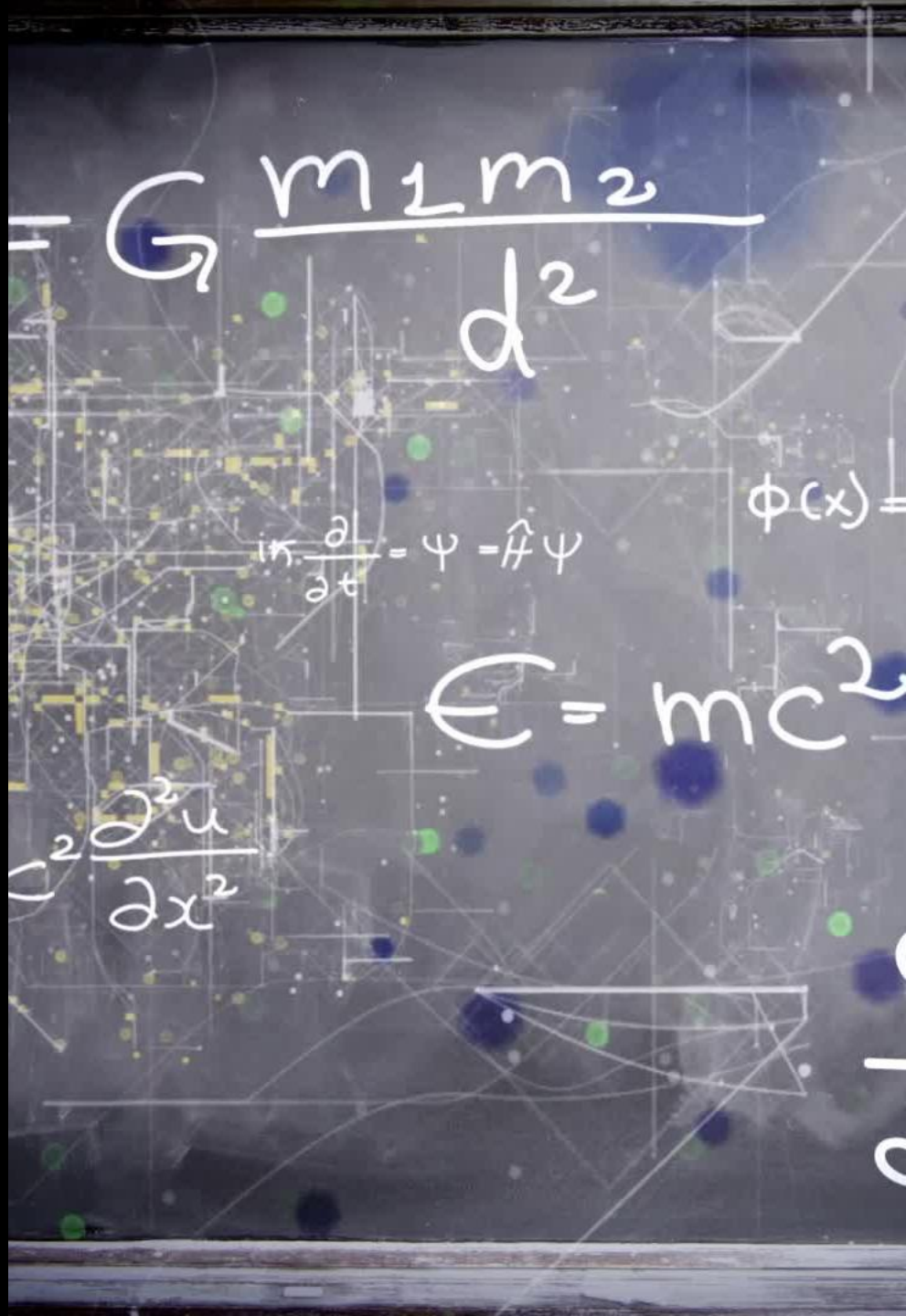
1. Make a guess of a factor, g

2. Use g to do quantum magic and wave collapses to get a power, p

3. Make a better guess, $g^{p/2} \pm 1$

SHOR'S ALGORITHM

- Dr. Peter Shor's Quantum Class: https://math.mit.edu/~shor/435-LN/
- Minute Physics explains Shor's algorithm: https://www.youtube.com/watch?v=IvTqbM5Dq4Q
- Minute Physics uses Shor's algorithm to factor 314191: https://www.youtube.com/watch?v=FRZQ-efABeQ



GOOD NEWS

- These are still technically theoretical at best
- Threatens only Public Key Cryptosystems due to difficulty in efficiently finding larger keys
- Symmetric Key Cryptosystems can still be increased arbitrarily

NEED A NEW BASIS

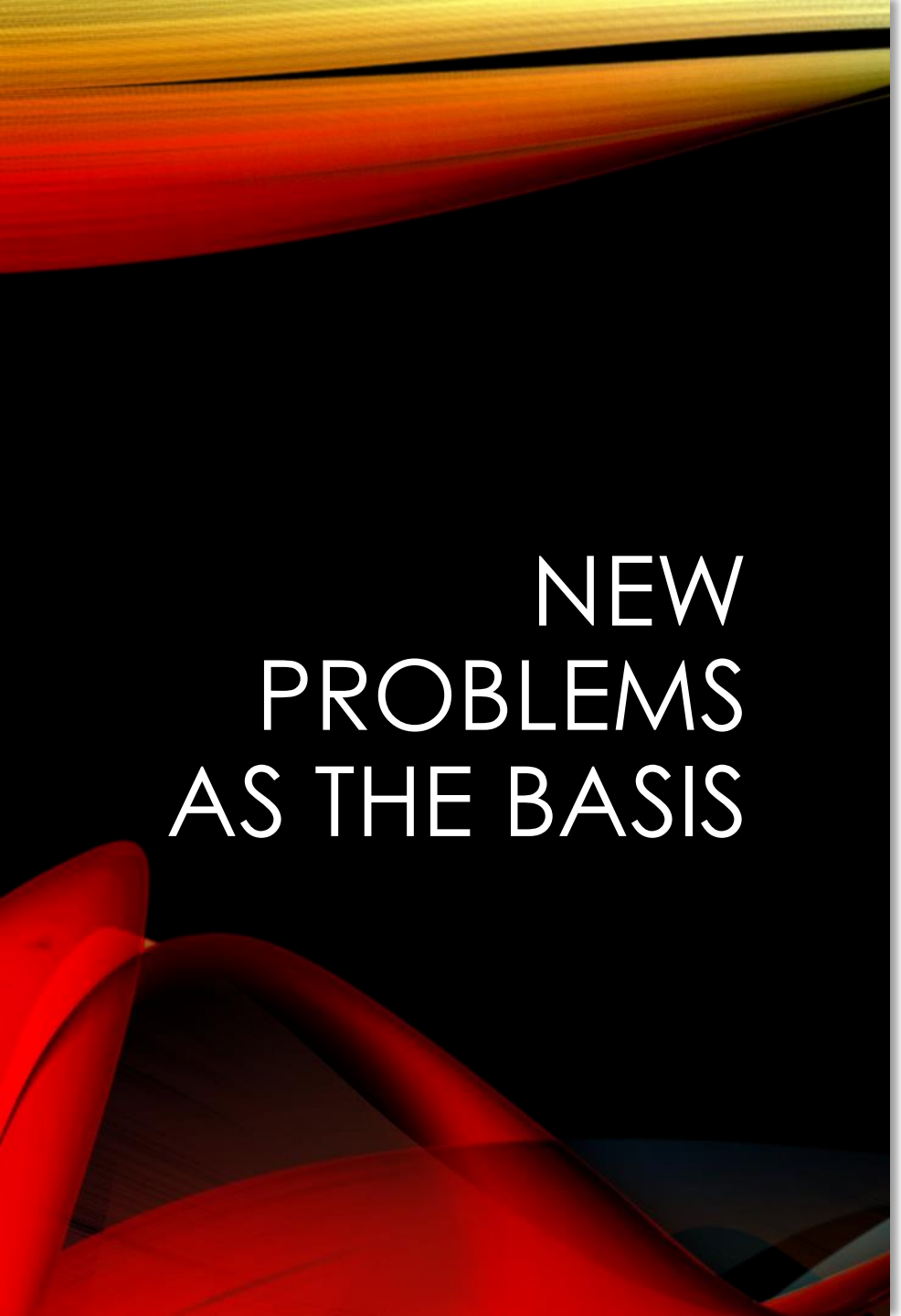
The problems of Public Key Cryptosystems are **theoretically** compromised

Still, why wait to improve

There are centuries of Mathematics for us to use!

WHAT IS IN COMMON BETWEEN CLASSIC AND QUANTUM COMPUTERS?

- They are bad at Linear Algebra!!
- Well, there aren't any inherent optimizations to be made

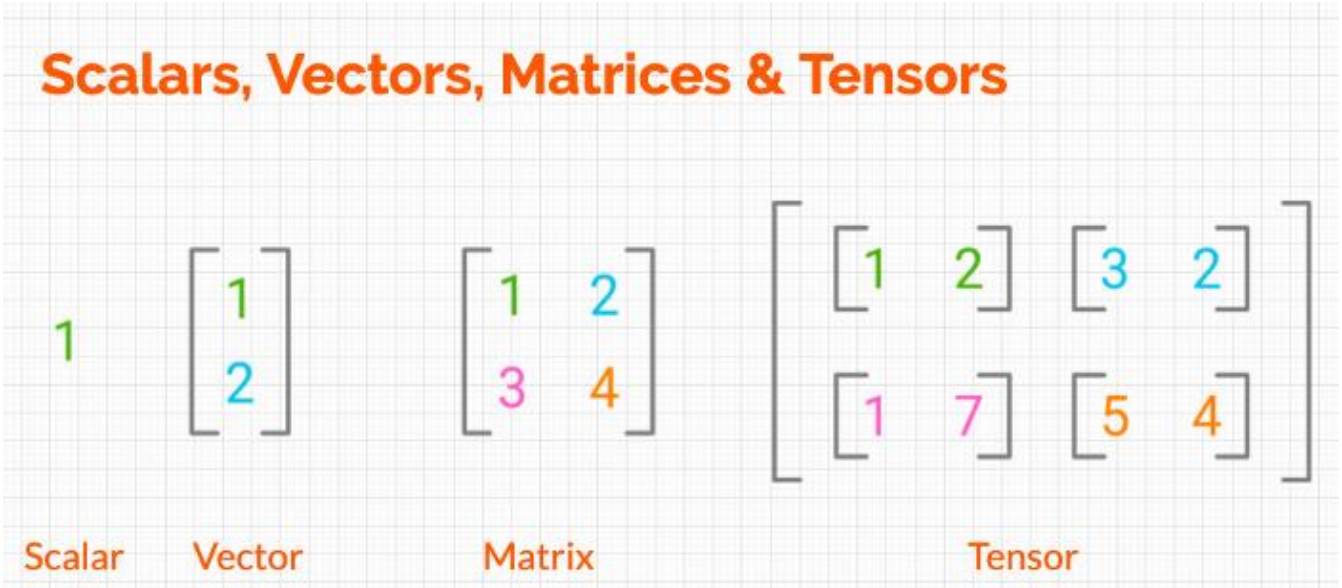


NEW PROBLEMS AS THE BASIS

- Lattice-based
- Hash-based
- Multivariate
- Code-based

- We will be looking at an example of this today!

LINEAR ALGEBRA REFRESHER



https://substackcdn.com/image/fetch/f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fbucketeer-e05bbc84-baa3-437e-9518-adb32be77984.s3.amazonaws.com%2Fpublic%2Fimages%2F360f8be5-a78d-479a-89ec-b8f49b5ad245_700x381.png

DATA TYPES

In Python with Numpy:
A.shape to find dimensions

Addition of Matrices

cuemath
THE MATH EXPERT

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & b_{13} & \cdots & b_{1n} \\ b_{21} & b_{22} & b_{23} & \cdots & b_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ b_{m1} & b_{m2} & b_{m3} & \cdots & b_{mn} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2n}+b_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mn}+b_{mn} \end{bmatrix}$$

$$[a_{ij}]_{m \times n} \qquad [b_{ij}]_{m \times n} \qquad [a_{ij} + b_{ij}]_{m \times n}$$

https://d138zd1ktt9iqe.cloudfront.net/media/seo_landing_files/addition-of-matrices-1625854788.png

ADDITION

In Python with Numpy: A + B

$$2 \cdot \begin{bmatrix} 10 & 6 \\ 4 & 3 \end{bmatrix} = \begin{bmatrix} 2 \cdot 10 & 2 \cdot 6 \\ 2 \cdot 4 & 2 \cdot 3 \end{bmatrix}$$

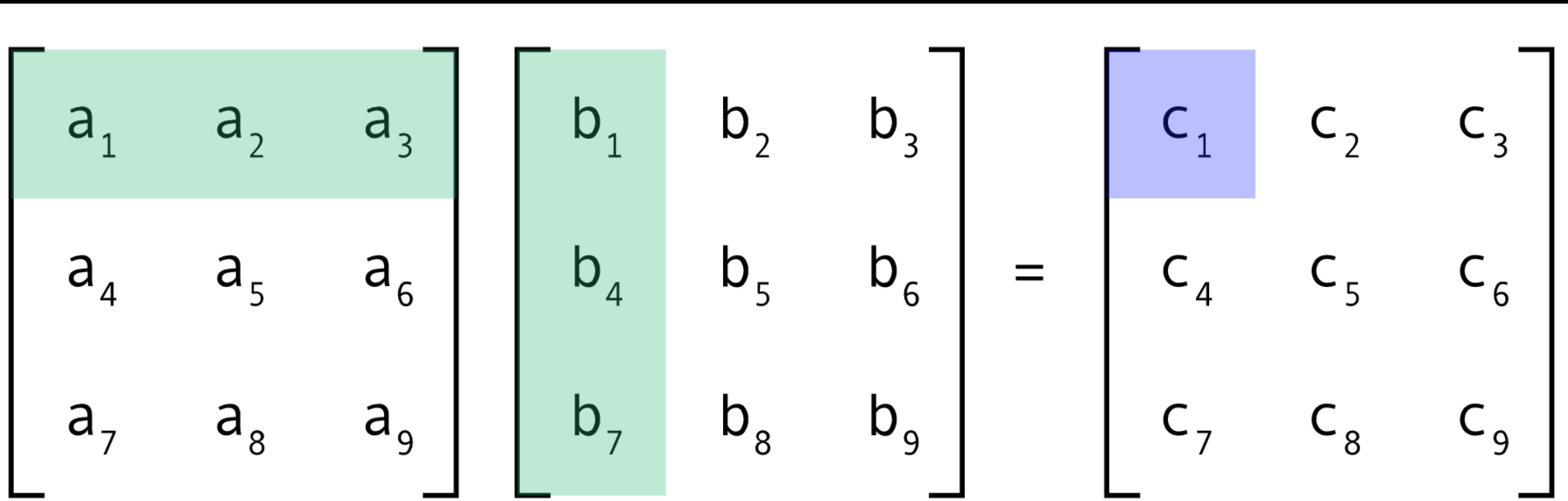
https://cdn.kastatic.org/googleusercontent/xj8YqV88KB29MEKR5Iq68oUo1h2kFAIAewMsHeWS9-7l0KaB6Bl3sOmpfGSCzsVU8z5Evq6QIrwbEAqBnZ5W06g0CQ

SCALAR MULTIPLICATION

In Python with Numpy: c * A

MATRIX MULTIPLICATION

In Python with Numpy: `A @ B`



https://miro.medium.com/v2/resize:fit:2000/1*HjcZkViYtPKg-Wm2o7DFDg.png

TRANSPOSE

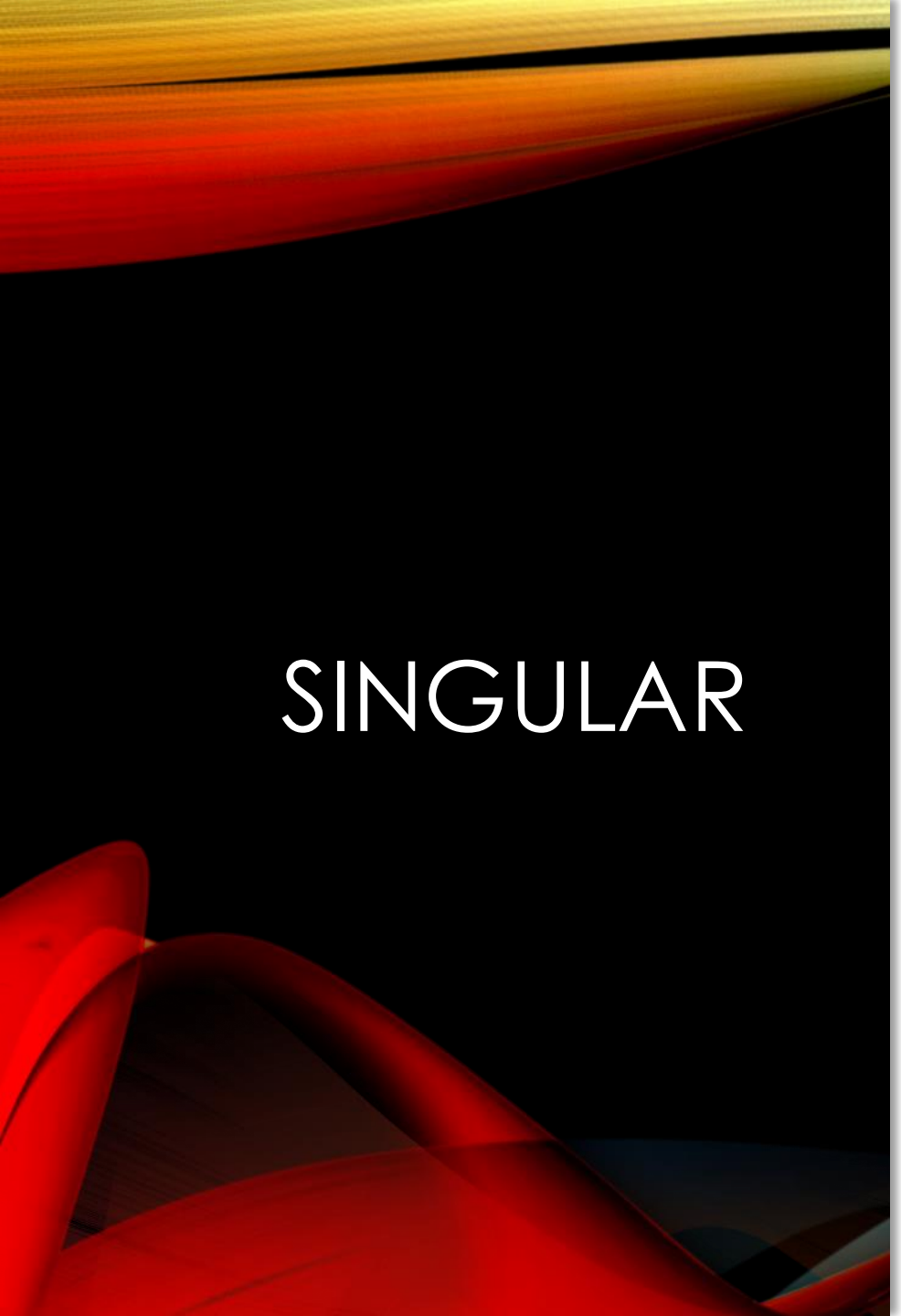
Denoted with the superscript T, like A^T

In Python with Numpy: A.T

2	4	-1
-10	5	11
18	-7	6

2	-10	18
4	5	-7
-1	11	6

https://assets.leetcode.com/uploads/2021/02/10/hint_transpose.png

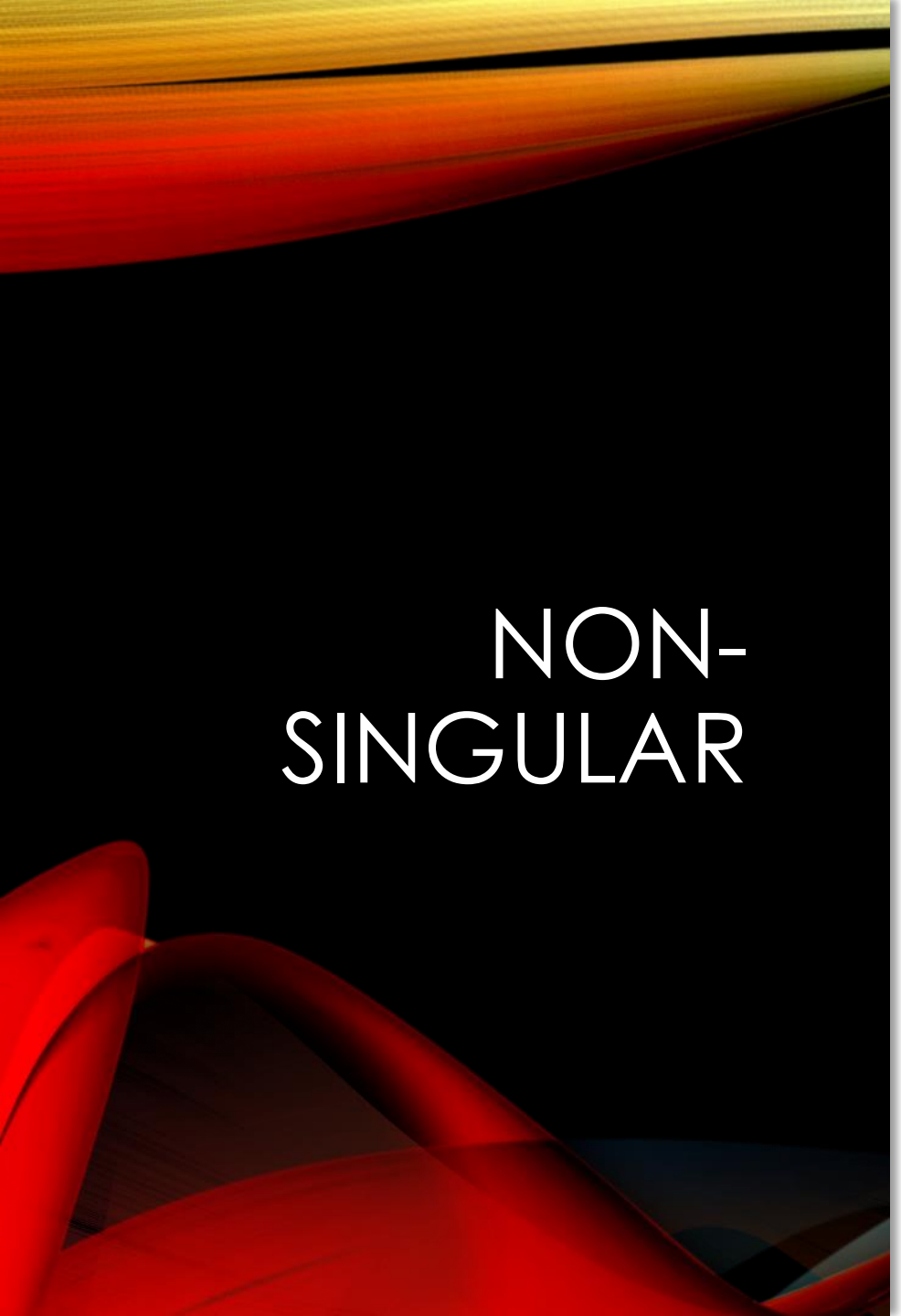


SINGULAR

Two definitions

The one we care about is if a matrix A is singular, there exists a matrix B such that their product is the Zero Matrix

i.e. $A * B = 0$



Two Definitions again!

We care about => A matrix A is non-singular if there exists a matrix B such that A * B = I, where I is the Identity matrix

In other words, $B = A^{-1}$

MUTUALLY EXCLUSIVE

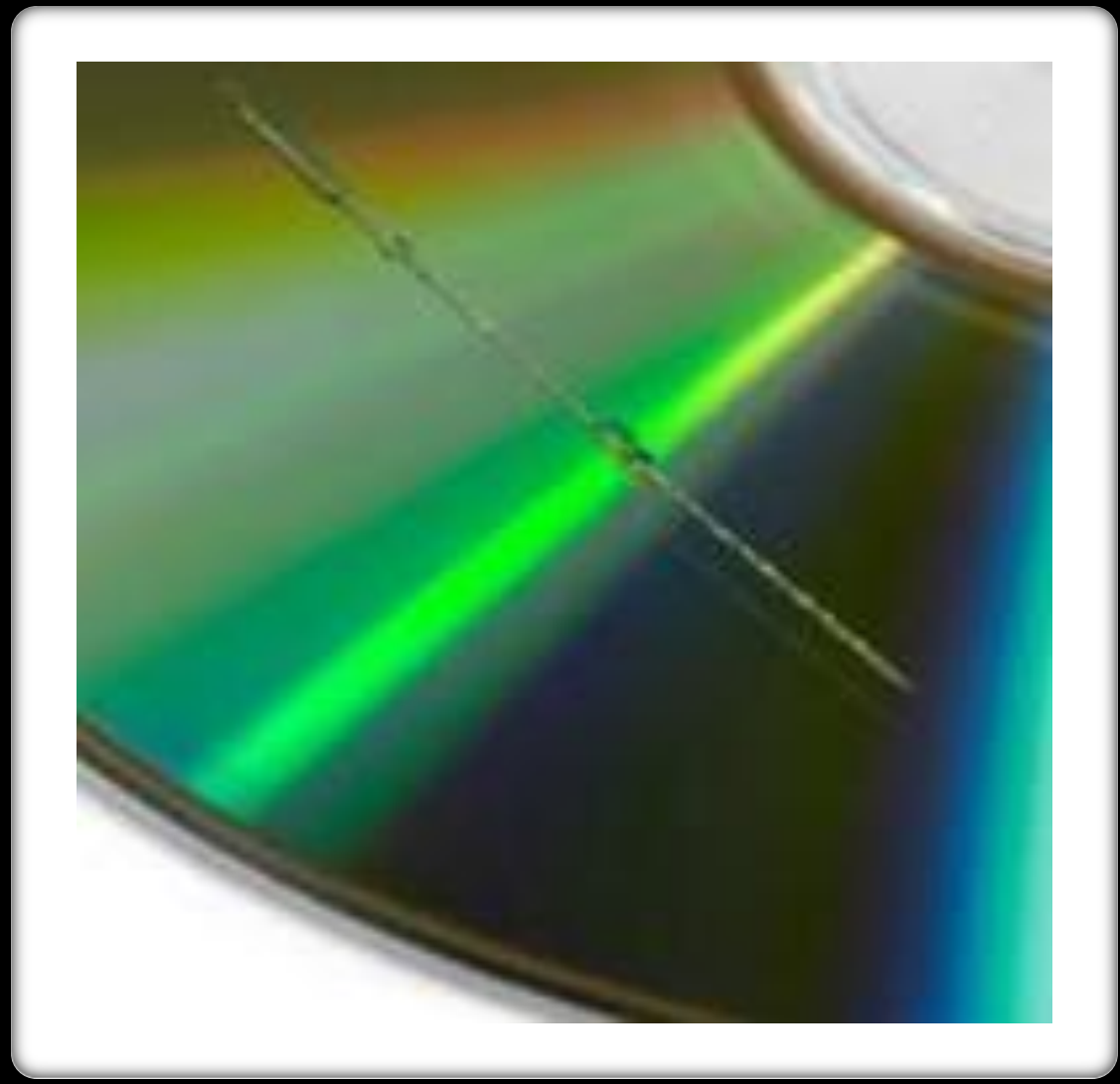
A matrix cannot be both singular and non-singular, why?

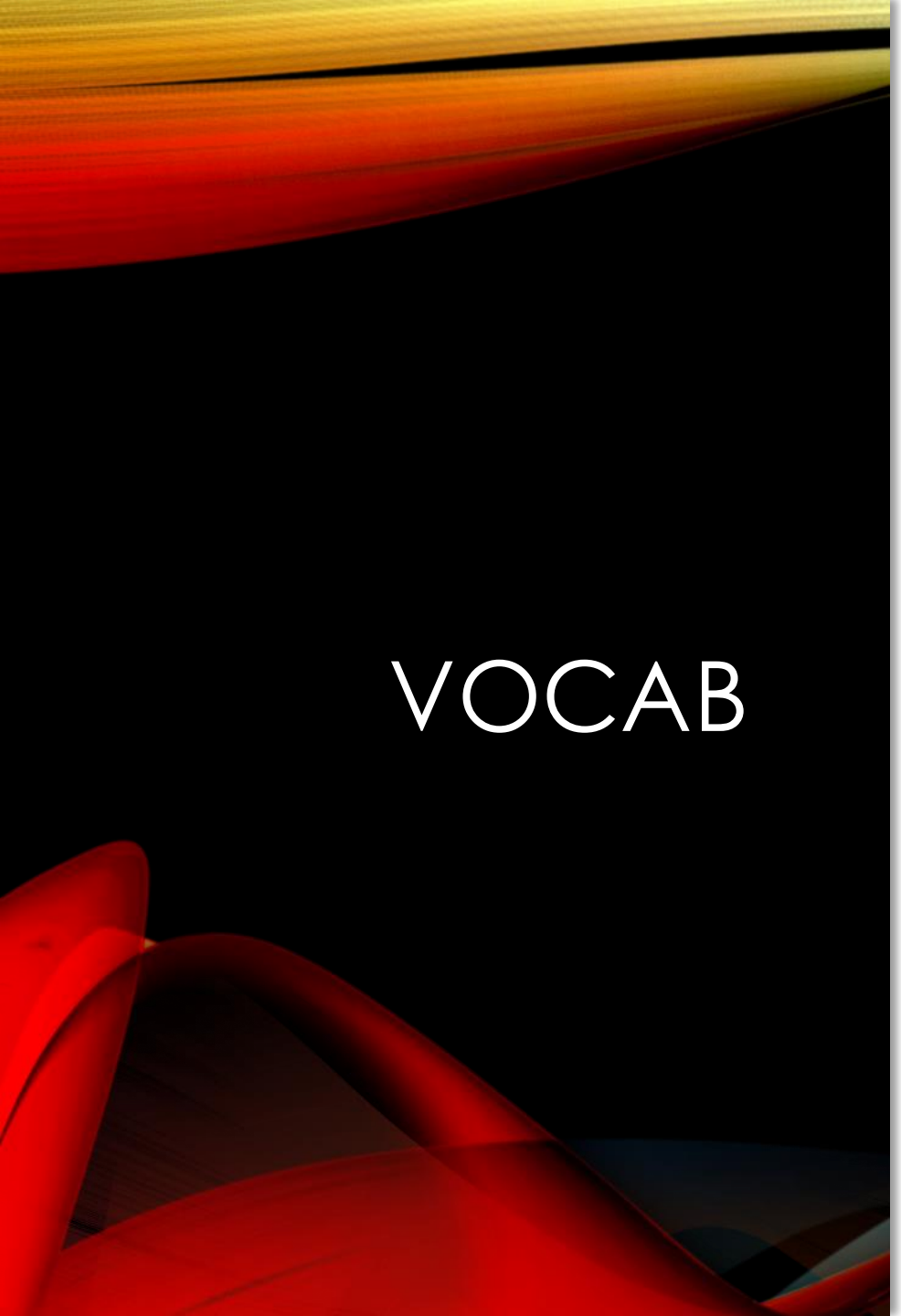
Hint: it can be shown similarly to why $1 \neq 0$


ERROR CORRECTING
CODES

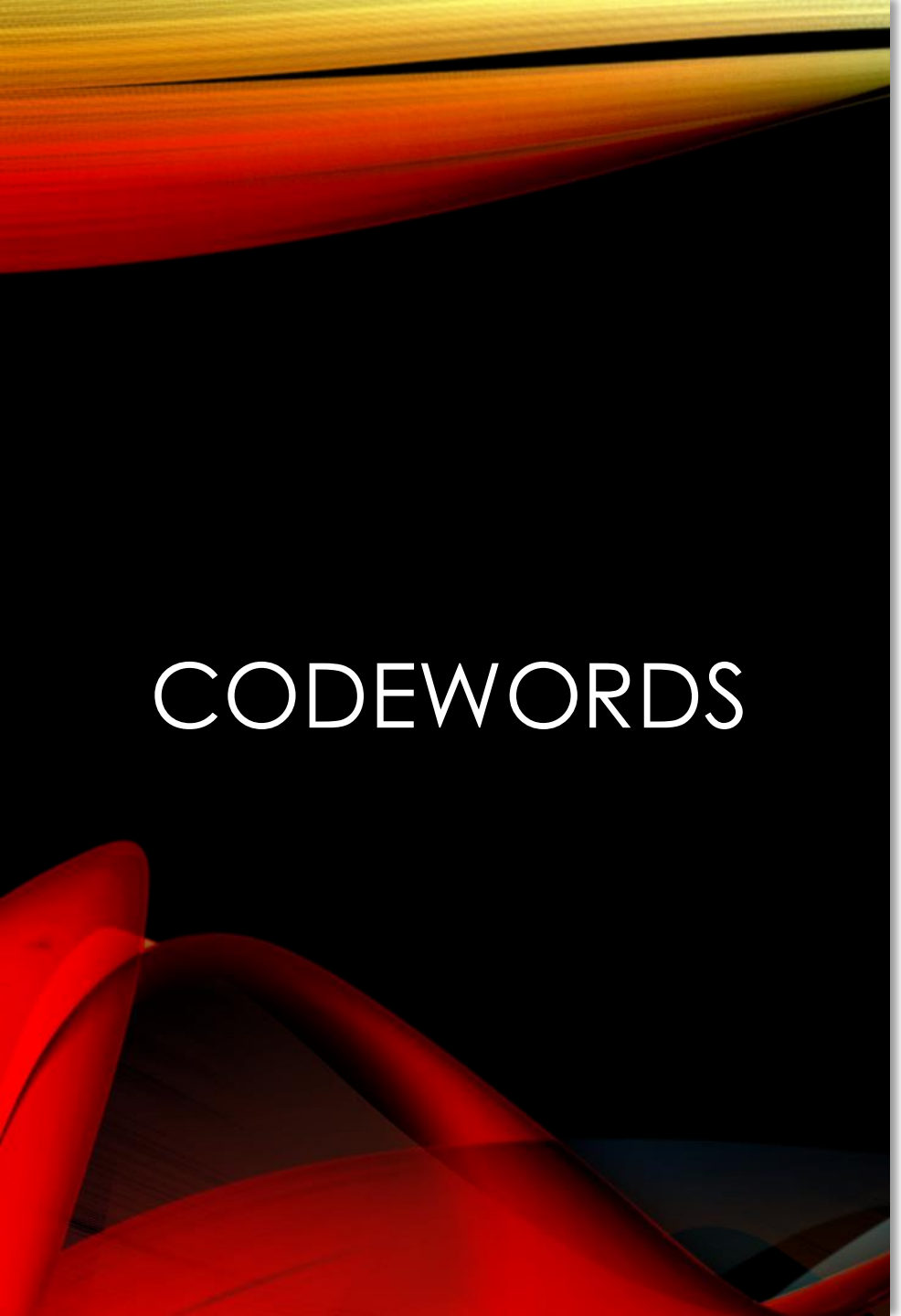
NOISY NETWORKS

- Communicating bits can come with errors
- How do we know what the message was if there was interference?
- Scratched CD?





- Generator Matrix
- Parity Check Matrix
- Length, n
- Dimension, k
- (n, k)-code
- Distance
- Hamming weight
- Errors Detected, s
- Errors Corrected, t
- Syndrome Decoding



CODEWORDS

These are strings of numbers that are a valid message in our Code

In a (n,k)-code, there are 2^k code words

- We will see why when we look at generator matrices

GENERATOR MATRIX

- The Generator is a k by n matrix that determines how we generate codewords

- Lengthy process of finding it is make a matrix of ALL codewords, and perform Gaussian Elimination
  - There will be at most n steps of this algorithm
  - This is to determine the minimum number of linearly independent codewords
    - i.e. it cannot be made up of the other codewords we are keeping in the generator matrix
  - *Will show an example of this in a bit

PARITY CHECK MATRIX

The Generator Matrix is a singular matrix, thus there is a matrix we can multiply to it to get the Zero Matrix

This matrix is called the Parity Check Matrix

It can be used to verify if a received message is a valid codeword

LENGTH N AND DIMENSION K

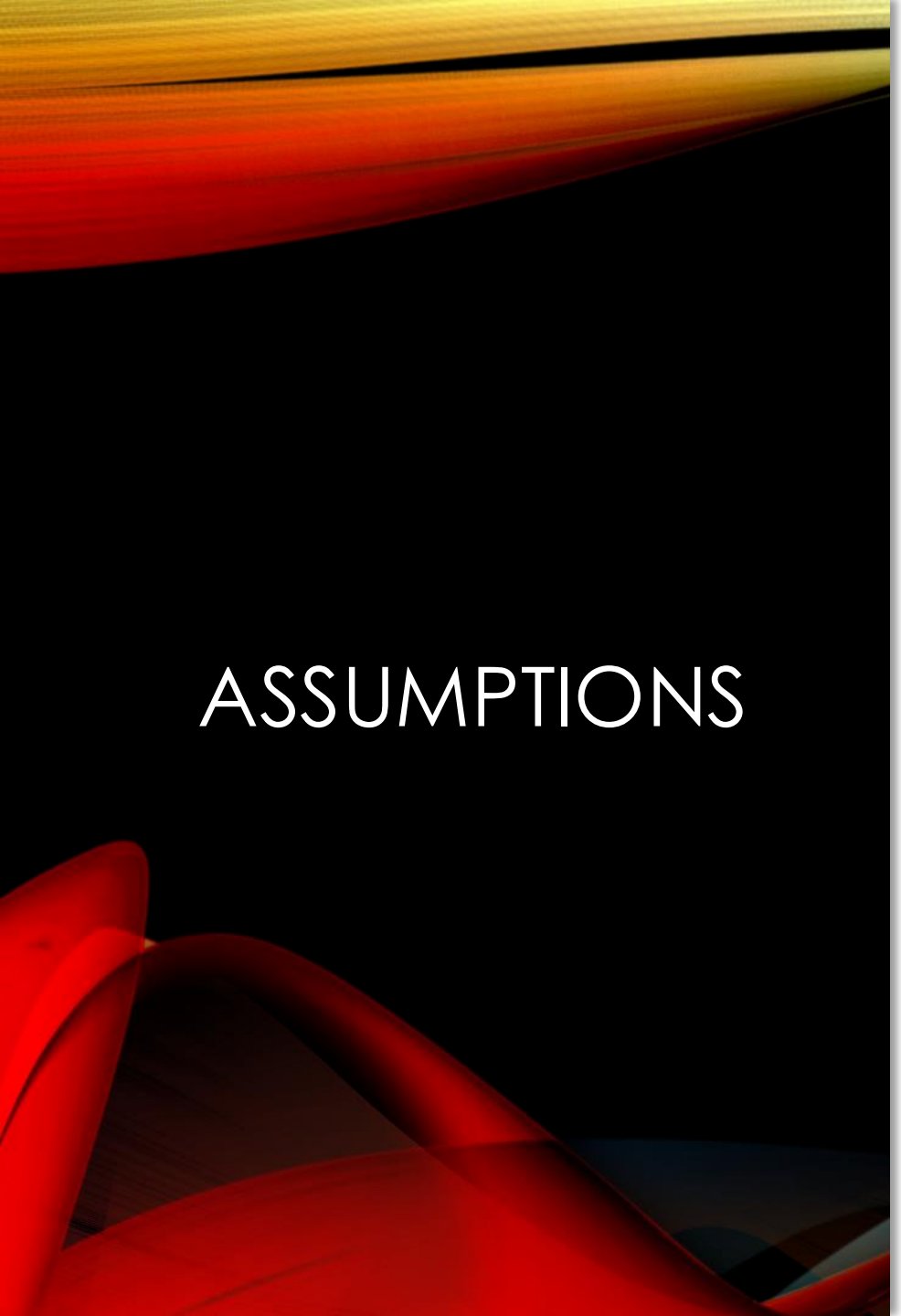
- The length, n, of a Code is the length of the message the Generator matrix produces
  - This is also equal to the number of columns
- The dimension, k, is the minimum number of codewords needed to generate all of the others
- So why is the number of codewords equal to 2^k?
  - Answer: Combinatorics!
  - Since we are dealing with binary any combination will use coefficients of 0 or 1, the sum of all choices becomes 2^k
  - Again, we will see this in the first example

WEIGHT AND DISTANCE

- Given a string of binary, say 010101101, we have a notion of **weight**
- The Hamming Weight of a codeword is the number of non-zero elements
  - i.e. counting the 1's
  - So the weight of 010101101 is 5
- The Hamming Distance between two codewords is equal to the number of differences between them
  - So D(010101101, 101001000) = 6
- Extending the to a Code, the Distance d of a Code is the minimum distance between all non-zero codewords and the zero vector

ERRORS DETECTED AND CORRECTED

- Now that we know the Distance, d, of a code, we can make some observations
- We can detect d-1 errors, notated as s
- We can correct $\mathbf{ceiling}(\frac{d}{2}) - 1$, notated as t



Closest Neighbor

- We assume that if any errors occur, they are minimal
- i.e. the actual message is the one that takes the fewest changes to get to

What if two codewords are equally close?


Secant Lines
Tangent Line
x+h
$f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$
$f(x) = \lim_{h\to 0} \frac{(x+h)^2 - x^2}{h}$
$= \lim_{h\to 0} \frac{x^2+2xh+h^2-x^2}{h}$
$= \lim_{h\to 0} \frac{2xh+h^2}{h}$
$= \lim_{h\to 0} \frac{h}{h}$
$= 1$
$2\sqrt{x}$

EXAMPLES

EXAMPLE 1: REPETITION CODE

- This will be a (4,2)-code
- $Let\ G = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$ => [I_k | P]
- $Let\ H = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$ => [$-P^{\dagger}$ | I_{n-k}]

EXAMPLE MESSAGES

- $[0 \quad 1] * \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} = [0 \quad 1 \quad 0 \quad 1]$
- $[1 \quad 1] * \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} = [1 \quad 1 \quad 1 \quad 1]$
- $[1 \quad 0] * \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} = [1 \quad 0 \quad 1 \quad 0]$
- $[0 \quad 0] * \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} = [0 \quad 0 \quad 0 \quad 0]$

REPETITION CODES CAN'T CORRECT ERRORS

- Receive [0 1 1 1]
- What was meant to be sent?
  - [0 1 **0** 1]

  Or
  - [**1** 1 1 1]

Example : The linear block code with the following generator matrix and parity check matrix is a $[7, 4, 3]_2$ Hamming code.

$$\boldsymbol{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}, \boldsymbol{H} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

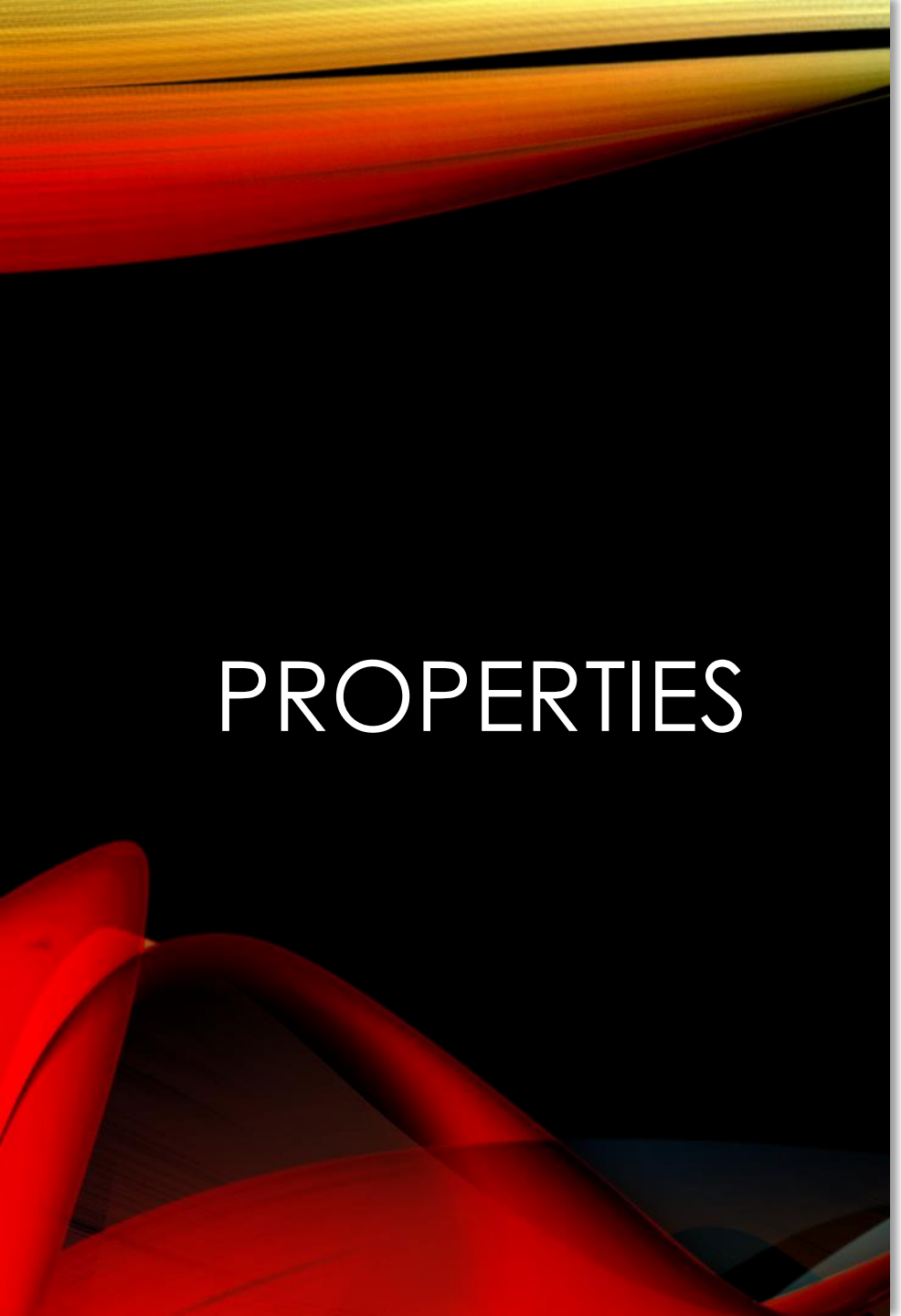
https://en.wikipedia.org/wiki/Linear_code#Example:_Hamming_codes

EXAMPLE 2: HAMMING CODE

*THIS WILL BE THE CODE FAMILY WE USE FOR TODAY


QUESTIONS?

10 MINUTE INTERMISSION



PROPERTIES

Uses the properties of binary (n,k)-linear codes

- What we were just talking about

This means we can correct t errors

We have a decoding algorithm A

- This is based on our choice of Code and relates to the Parity Check Matrix

KEYGEN

- We need to compute 2 non-singular matrices
  - This is so we can invert them later
  - Since our Generator matrix, G, is k by n the two matrices will look like
    - k by k called S (Substitution)
    - n by n called P (Shift Rows)
  - NOTE: all values in the matrices are only 1 or 0

- Calculate $G' = S * G * P$

- Public Key: (G', t)
- Private Key: (S, P, A)

GENERATOR AND PARITY CHECK MATRIX

- Easier to start with Parity Check for Hamming Codes

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

HOW TO GET THE PARITY CHECK FROM A GENERATOR MATRIX (AND VICE VERSA)

- Find the nullspace of the Generator matrix
- We can do so by putting the Generator matrix into standard form
  - $G = [I \quad P]$
  - $H = [-P^T \quad I]$
- Using the SymPy library: sympy.Matrix(g).nullspace()

GENERATOR MATRIX

- By using the previous methods,

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

MAKING A PERMUTATION MATRIX

- Start with an identity matrix and shuffle rows/columns
- These are always invertible

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} = P$$

MAKING A SUBSTITUTION MATRIX

- Must **ALWAYS** be invertible
- Easiest way: make random upper triangular then shuffle rows/columns

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} = S$$

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

PUTTING THEM TOGETHER

- $G' = S * G * P$

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

CODING THE KEY GENERATION

*30 minutes

ENCRYPTION

- We have a message of length k
- Compute the codeword c' = m ∗ G'
- We make a random error vector, z, of weight t(the number of errors we can correct)
- The ciphertext becomes c = c' + z

WHAT IS A MESSAGE?

- Since G' is a 4x7 matrix, our messages we can send must be 4 bits

Let $\mathbf{m} = [1 \quad 0 \quad 1 \quad 1]$

MAKING A CODE WORD

- m * G'

$$[1 \quad 0 \quad 1 \quad 1] * \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} = [1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0]$$

ADDING IN RANDOM ERROR(S)

- A Hamming code can correct t=1 errors
- Make a random error vector to add to our codeword
- Let $z = [0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]$
- Our ciphertext is c = m' + z
- $c = [1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0] + [0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0] = [1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0]$

CODING THE ENCRYPTION FUNCTION

*15 minutes

DECRYPTION

- Start by calculating P^{-1} and S^{-1}
- We will compute $c'' = c * P^{-1}$
  - $c'' = m * G' * P^{-1} + z * P^{-1}$
  - $c'' = m * S * G * P * P^{-1} + z * P^{-1}$
  - $c'' = m * S * G * I + z * P^{-1}$
  - $c'' = m * S * G + z * P^{-1}$
- We can use our decoding algorithm with our Parity Check Matrix to discover $m * S$
  - And finally, $m * S * S^{-1} = m * I = m$

START WITH INVERTED PERMUTATION

- Can find the inverse of a matrix using np.linalg.inv()

$$P^{-1} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- $C*P^{-1}$

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

SYNDROME DECODING

- Because we have c = m' + z, $c*H^T = m'*H^T + z*H^T = 0 + z*H^T$
- $z*H^T$ is called the Syndrome
- H^T: each row is a possible Syndrome

$$[1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1] * \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= [0 \quad 0 \quad 1]$$

SYNDROME TABLE

Syndrome	Error Location Based on H^T
$[0 \quad 0 \quad 0]$	No Error
$[0 \quad 0 \quad 1]$	$[0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1]$
$[0 \quad 1 \quad 0]$	$[0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0]$
$[0 \quad 1 \quad 1]$	$[0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$
$[1 \quad 0 \quad 0]$	$[0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0]$
$[1 \quad 0 \quad 1]$	$[0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0]$
$[1 \quad 1 \quad 0]$	$[1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$
$[1 \quad 1 \quad 1]$	$[0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]$

FOUND ERROR LOCATION

- Remove error from c’

$$[1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1] + [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1] = [1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0]$$

FOR SANITY

- Check that removed error now has a Syndrome of the 0 vector

$$[1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0] * \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= [0 \quad 0 \quad 0]$$

NOW, UNDO THE GENERATION OF THE CODE WORD

- From our H, we have:

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

So, m*S is

$$[\mathbf{1} \quad \mathbf{0} \quad \mathbf{0} \quad \mathbf{0} \quad 1 \quad 1 \quad 0]$$

NEED S^{-1}

- Can find the inverse of a matrix using np.linalg.inv()

$$S^{-1} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

FINAL STEP

- Take $m*S*S^{-1}$

$$[1 \quad 0 \quad 0 \quad 0] * \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} = [1 \quad 0 \quad 1 \quad 1]$$

- We have successfully retrieved m!

CODING THE DECRYPTION FUNCTION

*Remainder of the time