# ENR145 Computational Methods:

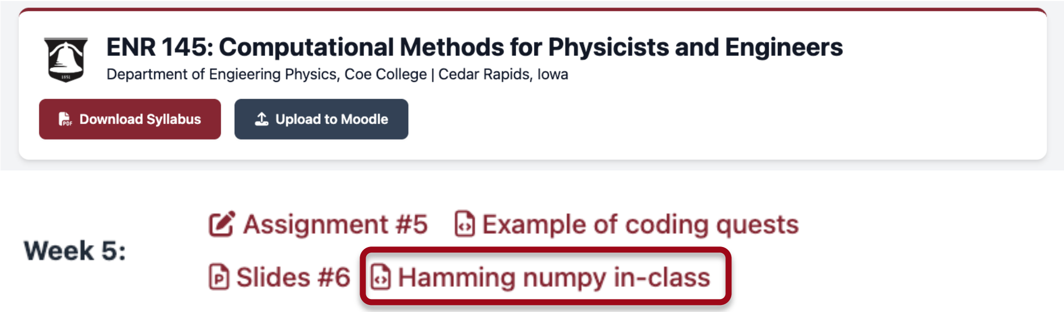
## Hamming Numpy 301: a detour from traffic control so we can do one step operation for many things

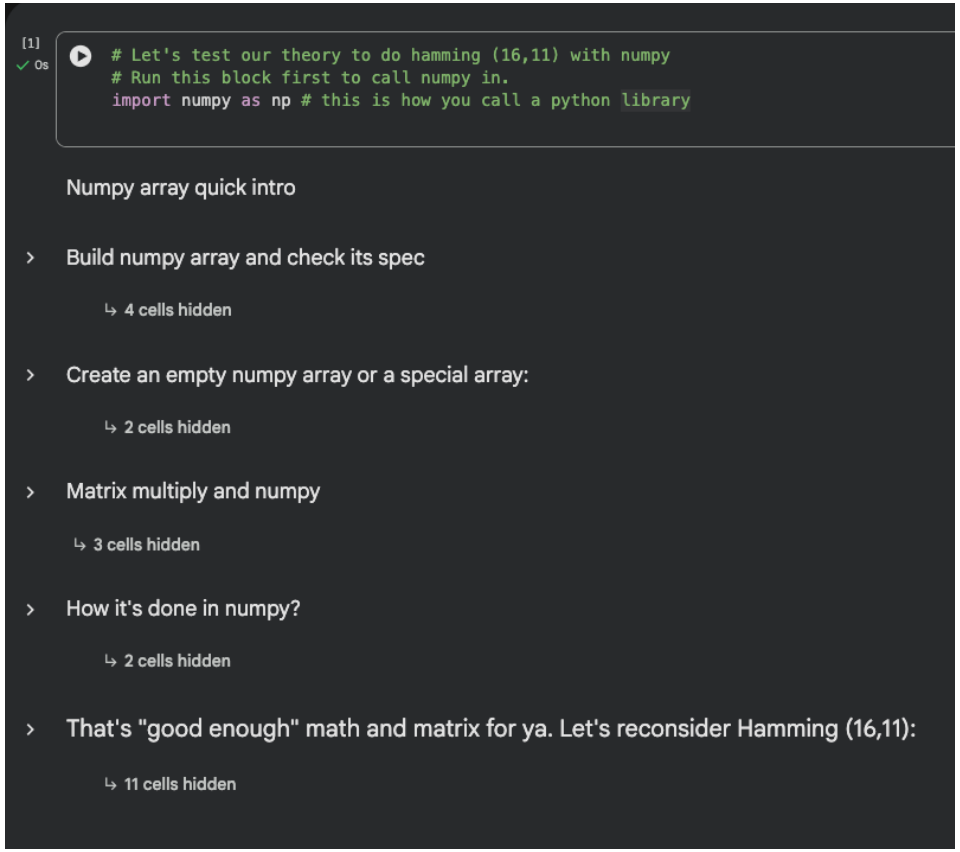
Xiang Li

Spring 2026

COE COLLEGE

# Download the in-class notebook here:



# Two things about hamming (1/2):

**1. Hamming code doesn't't' need to be a 4x4 matrix**

| This can work: |
|---|

| Data Here |
|---|
| 1 |
| 0 |
| 0 |
| 1 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 1 |
| 1 |

→

| Hamming Code Matrix | | | |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |

| Can this work? |
|---|

| Data Here |
|---|
| 1 |
| 0 |
| 0 |
| 1 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 1 |
| 1 |

→

| Hamming Code Array | |
|---|---|
| 1 | D1 |
| 0 | D2 |
| 0 | D3 |
| 1 | D4 |
| 0 | D5 |
| 0 | D6 |
| 0 | D7 |
| 0 | D8 |
| 0 | D9 |
| 1 | D10 |
| 1 | D11 |
| 1 | P1 |
| 0 | P2 |
| 1 | P3 |
| 0 | P4 |
| 0 | P0 |

# Two things about hamming (1/2):

**1. Hamming code doesn't't' need to be a 4x4 matrix**

This can work:

| Data Here |
|---|
| 1 |
| 0 |
| 0 |
| 1 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 1 |
| 1 |

| Hamming Code Matrix | | | |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |

Can this work?

| Data Here |
|---|
| 1 |
| 0 |
| 0 |
| 1 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 1 |
| 1 |

| Hamming Code Array | |
|---|---|
| 1 | D1 |
| 0 | D2 |
| 0 | D3 |
| 1 | D4 |
| 0 | D5 |
| 0 | D6 |
| 0 | D7 |
| 0 | D8 |
| 0 | D9 |
| 1 | D10 |
| 1 | D11 |
| 1 | P1 |
| 0 | P2 |
| 1 | P3 |
| 0 | P4 |
| 0 | P0 |

**Might have error code mapping issue, maybe?**

# Two things about hamming (2/2):

**2. Array to array conversion will be so easy:**

This is math:

$$P_{i,j} = \sum_{k=1}^{n} Q_{i,k} \times R_{k,j}$$

This is math application:

[2,3] x [3,4] = [2,4]

This is Engineering application:

11 bits in

16 bits out

So instead of double for loops, we can do matrix operation to get it in one go?

[1,11] x [?] = [1,16]?

# Would it mathematically work?

Can this work?

| Data Here |
| --- |
| 1 |
| 0 |
| 0 |
| 1 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 1 |
| 1 |

| Hamming Code Array | |
| --- | --- |
| 1 | D1 |
| 0 | D2 |
| 0 | D3 |
| 1 | D4 |
| 0 | D5 |
| 0 | D6 |
| 0 | D7 |
| 0 | D8 |
| 0 | D9 |
| 1 | D10 |
| 1 | D11 |
| 1 | P1 |
| 0 | P2 |
| 1 | P3 |
| 0 | P4 |
| 0 | P0 |

**I can show it could theoretically work in 3 steps.**

COE COLLEGE

# Step 1: how does matrix multiply works:

When "math" gives an example:

$$P_{i,j} = \sum_{k=1}^{n} Q_{i,k} \times R_{k,j}$$

$$P = \begin{bmatrix} Q_{11}R_{11} + Q_{12}R_{21} + \cdots + Q_{1n}R_{n1} & Q_{11}R_{12} + Q_{12}R_{22} + \cdots + Q_{1n}R_{n2} & \cdots & Q_{11}R_{1q} + Q_{12}R_{2q} + \cdots + Q_{1n}R_{nq} \\ Q_{21}R_{11} + Q_{22}R_{21} + \cdots + Q_{2n}R_{n1} & Q_{21}R_{12} + Q_{22}R_{22} + \cdots + Q_{2n}R_{n2} & \cdots & Q_{21}R_{1q} + Q_{22}R_{2q} + \cdots + Q_{2n}R_{nq} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{m1}R_{11} + Q_{m2}R_{21} + \cdots + Q_{mn}R_{n1} & Q_{m1}R_{12} + Q_{m2}R_{22} + \cdots + Q_{mn}R_{n2} & \cdots & Q_{m1}R_{1q} + Q_{m2}R_{2q} + \cdots + Q_{mn}R_{nq} \end{bmatrix}$$

When "CS" gives an example:

$$E = AD = \begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \end{bmatrix} \times \begin{bmatrix} 2 & 3 & 5 & 7 \\ 11 & 13 & 17 & 19 \\ 23 & 29 & 31 & 37 \end{bmatrix} = \begin{bmatrix} 930 & 1160 & 1320 & 1560 \\ 2010 & 2510 & 2910 & 3450 \end{bmatrix}$$

COE COLLEGE

# Step 1: how does matrix multiply works:

When “engineering” gives an example:

**Indexing of M3**

$$M3 = M1 \times M2 = \begin{bmatrix} \boxed{a_1 \quad b_1} \\ c_1 \quad d_1 \end{bmatrix} \times \begin{bmatrix} \boxed{\begin{matrix} a_2 \\ c_2 \end{matrix}} & \begin{matrix} b_2 \\ d_2 \end{matrix} \end{bmatrix} = \begin{bmatrix} \boxed{a_1 \times a_2 + b_1 \times c_2} & a_1 \times b_2 + b_1 \times d_2 \\ c_1 \times a_2 + d_1 \times c_2 & c_1 \times b_2 + d_1 \times d_2 \end{bmatrix}$$

[1,] x [,1] [1,1]

$$M3 = M1 \times M2 = \begin{bmatrix} \boxed{a_1 \quad b_1} \\ c_1 \quad d_1 \end{bmatrix} \times \begin{bmatrix} \begin{matrix} a_2 \\ c_2 \end{matrix} & \boxed{\begin{matrix} b_2 \\ d_2 \end{matrix}} \end{bmatrix} = \begin{bmatrix} a_1 \times a_2 + b_1 \times c_2 & \boxed{a_1 \times b_2 + b_1 \times d_2} \\ c_1 \times a_2 + d_1 \times c_2 & c_1 \times b_2 + d_1 \times d_2 \end{bmatrix}$$

[1,] x [,2] [1,2]

$$M3 = M1 \times M2 = \begin{bmatrix} a_1 \quad b_1 \\ \boxed{c_1 \quad d_1} \end{bmatrix} \times \begin{bmatrix} \boxed{\begin{matrix} a_2 \\ c_2 \end{matrix}} & \begin{matrix} b_2 \\ d_2 \end{matrix} \end{bmatrix} = \begin{bmatrix} a_1 \times a_2 + b_1 \times c_2 & a_1 \times b_2 + b_1 \times d_2 \\ \boxed{c_1 \times a_2 + d_1 \times c_2} & c_1 \times b_2 + d_1 \times d_2 \end{bmatrix}$$

[2,] x [,1] [2,1]

$$M3 = M1 \times M2 = \begin{bmatrix} a_1 \quad b_1 \\ \boxed{c_1 \quad d_1} \end{bmatrix} \times \begin{bmatrix} \begin{matrix} a_2 \\ c_2 \end{matrix} & \boxed{\begin{matrix} b_2 \\ d_2 \end{matrix}} \end{bmatrix} = \begin{bmatrix} a_1 \times a_2 + b_1 \times c_2 & a_1 \times b_2 + b_1 \times d_2 \\ c_1 \times a_2 + d_1 \times c_2 & \boxed{c_1 \times b_2 + d_1 \times d_2} \end{bmatrix}$$

[2,] x [,2] [2,2]

Matrix indexing: [ i , j ]

Row # Column #

COE COLLEGE

# Step 2: how to do parity checks

**Recall:**
**Two ways to do parity check:**

**Logical operation**

**Math operation**

# Step 2: how to do parity checks

**Recall:**
**Two ways to do parity check:**

Logical operation

**i) XOR (the superior way, imo)**

**"exclusive OR"**

**XOR gate truth table**

| Input | | Output |
|---|---|---|
| A | B | A XOR B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Math operation

**ii) Modulo 2 (mod 2, %2)**

0 modulo 2 = 0

1 modulo 2 = 1

2 modulo 2 = 0

3 modulo 2 = 1

4 modulo 2 = 0

5 modulo 2 = 1

COE COLLEGE

Truth table from: https://en.wikipedia.org/wiki/~_gate

# Step 3: how to preserve and operate numbers in matrix

```
## TODO: understand this matrix operation
input = np.array([
    [1, 2, 3, 4]])
Operation_I = np.array ([
    [1,0,0,0,0],
    [0,1,0,0,1],
    [0,0,1,0,1],
    [0,0,0,1,1],
])

output = input @ Operation_I
```

```
shape check of input:
(1, 4)
shape check of operation matrix:
(4, 5)
shape check of output matrix:
(1, 5)
this is the output:
[[1 2 3 4 9]]
```

**Since we can do SUM, let's to %2 this time for parity check.**
**Then we just need to "SUM" the right bits for P1, P2,… P4 and P0.**

# Step 3: how to preserve and operate numbers in matrix

**We already had a road map for P1-P4 and P0 from “classic hamming”.**

| | | | |
|---|---|---|---|
| 0: 1 (0000) | 1: 1 (0001) | 2: 0 (0010) | 3: 0 (0011) |
| 4: 0 (0100) | 5: 0 (0101) | 6: 1 (0110) | 7: 1 (0111) |
| 8: 1 (1000) | 9: 1 (1001) | 10: 1 (1010) | 11: 1 (1011) |
| 12: 0 (1100) | 13: 0 (1101) | 14: 0 (1110) | 15: 0 (1111) |

https://harryli0088.github.io/hamming-code/

COE COLLEGE

# Step 3: how to preserve and operate numbers in matrix

This **can** work…

```
## TODO: understand this matrix operation
input = np.array([
    [1, 2, 3, 4]])
Operation_I = np.array ([
    [1,0,0,0,0],
    [0,1,0,0,1],
    [0,0,1,0,1],
    [0,0,0,1,1],
])

output = input @ Operation_I
```

| Data Here |
|---|
| 1 |
| 0 |
| 0 |
| 1 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 1 |
| 1 |

| Hamming Code Array | |
|---|---|
| 1 | D1 |
| 0 | D2 |
| 0 | D3 |
| 1 | D4 |
| 0 | D5 |
| 0 | D6 |
| 0 | D7 |
| 0 | D8 |
| 0 | D9 |
| 1 | D10 |
| 1 | D11 |
| 1 | P1 |
| 0 | P2 |
| 1 | P3 |
| 0 | P4 |
| 0 | P0 |

**As it becomes the combination of two simple matrix operation problem:**

1.) To preserve D1-D13: an "Identity matrix"

(11,11)

```
[[1 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 1]]
```

2.) To generate sum of the some: a "SoS matrix "

(11,5)

```
[[1 1 1 0 1]
 [1 1 0 1 1]
 [1 0 1 1 1]
 [0 1 1 1 1]
 [1 1 0 0 1]
 [1 0 1 0 1]
 [1 0 0 1 1]
 [0 1 1 0 1]
 [0 1 0 1 1]
 [0 0 1 1 1]
 [1 1 1 1 1]]
```

COE COLLEGE

# Step 3: how to preserve and operate numbers in matrix

This **can** work…

```
## TODO: understand this matrix operation
input = np.array([
    [1, 2, 3, 4]])
Operation_I = np.array ([
    [1,0,0,0,0],
    [0,1,0,0,1],
    [0,0,1,0,1],
    [0,0,0,1,1],
])

output = input @ Operation_I
```

| Data Here |
|---|
| 1 |
| 0 |
| 0 |
| 1 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 1 |
| 1 |

→

| Hamming Code Array | |
|---|---|
| 1 | D1 |
| 0 | D2 |
| 0 | D3 |
| 1 | D4 |
| 0 | D5 |
| 0 | D6 |
| 0 | D7 |
| 0 | D8 |
| 0 | D9 |
| 1 | D10 |
| 1 | D11 |
| 1 | P1 |
| 0 | P2 |
| 1 | P3 |
| 0 | P4 |
| 0 | P0 |

**As it becomes the combination of two simple matrix operation problem:**

3.1) AKA a generator matrix: (11,11+5)

```
[[1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1]
 [0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 1]
 [0 0 1 0 0 0 0 0 0 0 0 1 0 1 1 1]
 [0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1]
 [0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 1]
 [0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1]
 [0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1]
 [0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1]
 [0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1]]
```

3.2) Then do a %2 of the output, duh

**Time to test our theory out in Google Colab!**