FINAL PROJECT

DEEP DIVE

CS113: Linear Algebra

Minerva University

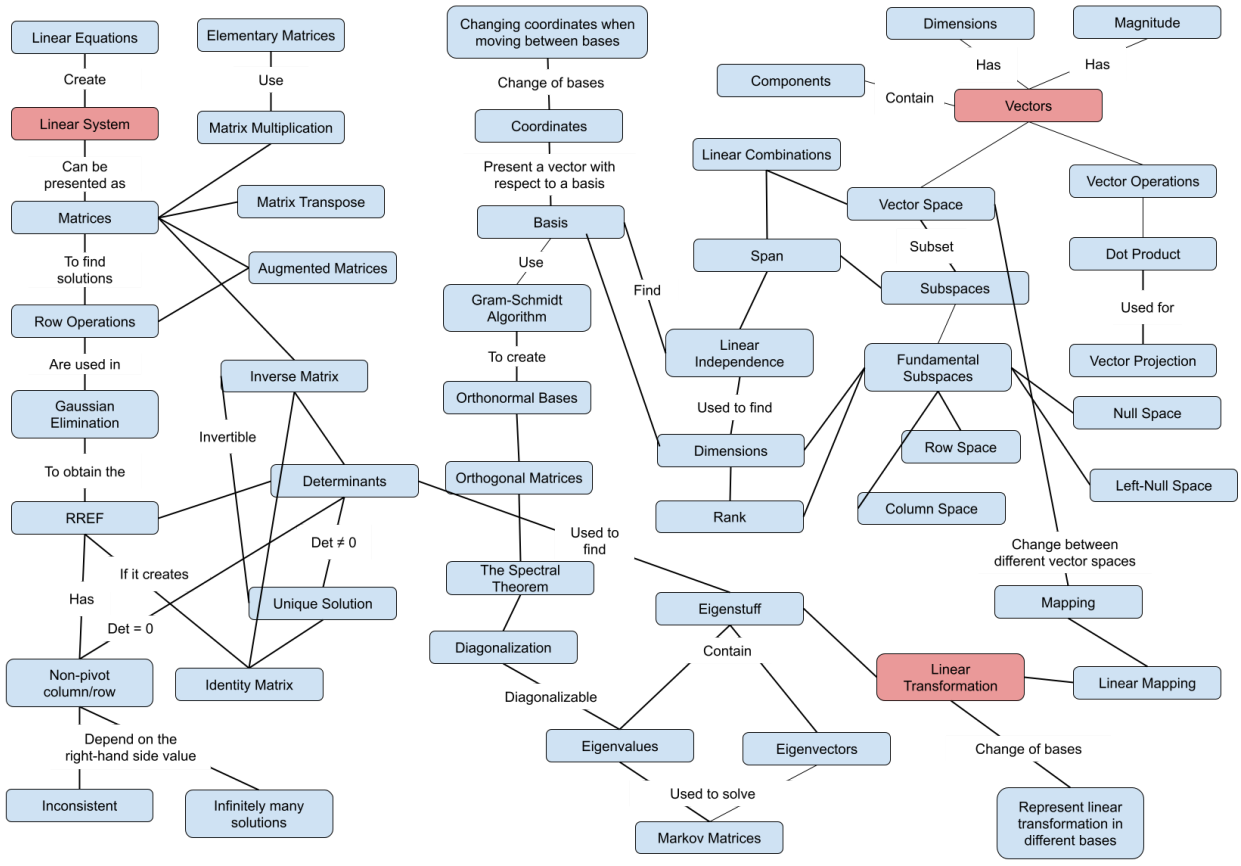
Prof. Rena Levitt

December 17, 2021

TABLE CONTENT

Concept Exploration

(This concept map is also available via this link)



1. A Bit of Algebra

In this course you have learned tools to analyze real vector spaces. These concepts generalize to vector spaces over other fields. Computational scientists often work with other fields better suited to a given application. For example, computer scientists will represent data as strings of numbers, or bits, in $F_2 = \{0, 1\}$, also known as GF(2). F_2 has two operations, addition (+) and multiplication (×), defined in Figure 1.

+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

Figure 1: Operations in $\mathbb{F}_2$

a. Algebra with bits can seem a bit odd. Find the additive inverse of both bits in F2. What is the additive identity in F2?

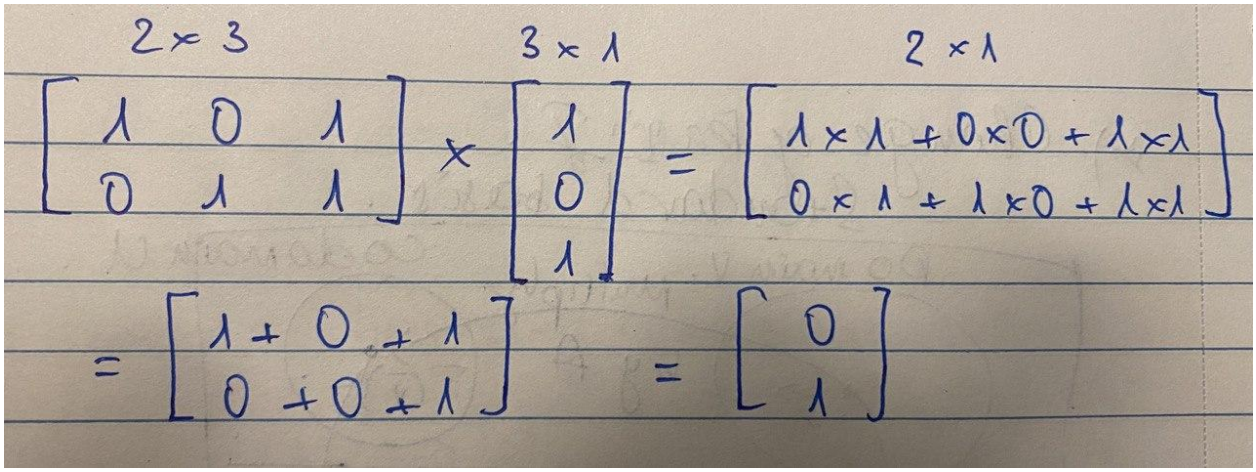
In Mathematics, the additive inverse of a number k is the number that, when added to k, yields zero. Based on the additive operation in F_2, the additive inverse of 0 is 0 because 0 + 0 = 0, while the additive inverse of 1 is 1 because 1 + 1 = 0.

In Mathematics, the additive identity of a set is an element that, when added to any element k in the set, yields k. In the set of F_2, we have two elements {0, 1}. Based on the additive operation in F_2, the additive identity of F_2 is 0 because 0 + 0 = 0 and 1 + 0 = 1, which means any element added to 0 stays the same in value.

b. Otherwise matrix arithmetic works the same way as before. Compute the following examples by hand to practice working in F_2.

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

By hand:



Check by SageMath:

Code Cell 1 of 3

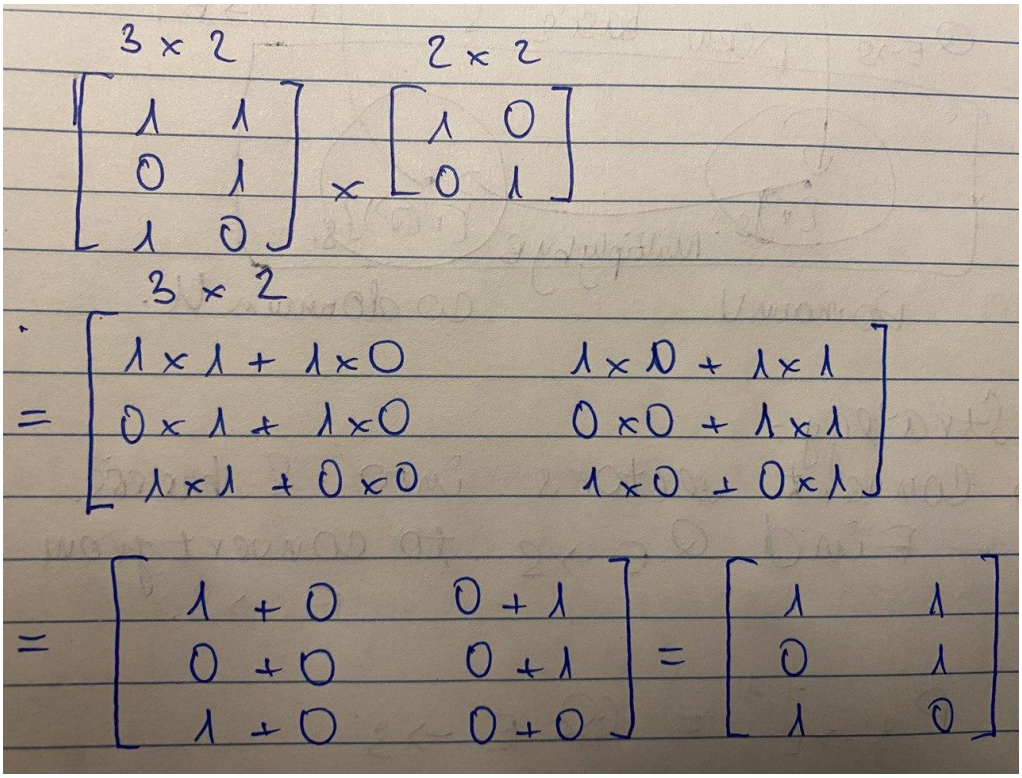
```
In [3]   1 M1 = matrix(GF(2),[[1,0,1],[0,1,1]])
         2 M2 = vector(GF(2),[1,0,1])
         3 M1*M2
```

Run Code

```
Out [3]
          (0, 1)
```

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

By hand:



Check by SageMath:

Code Cell 1 of 3

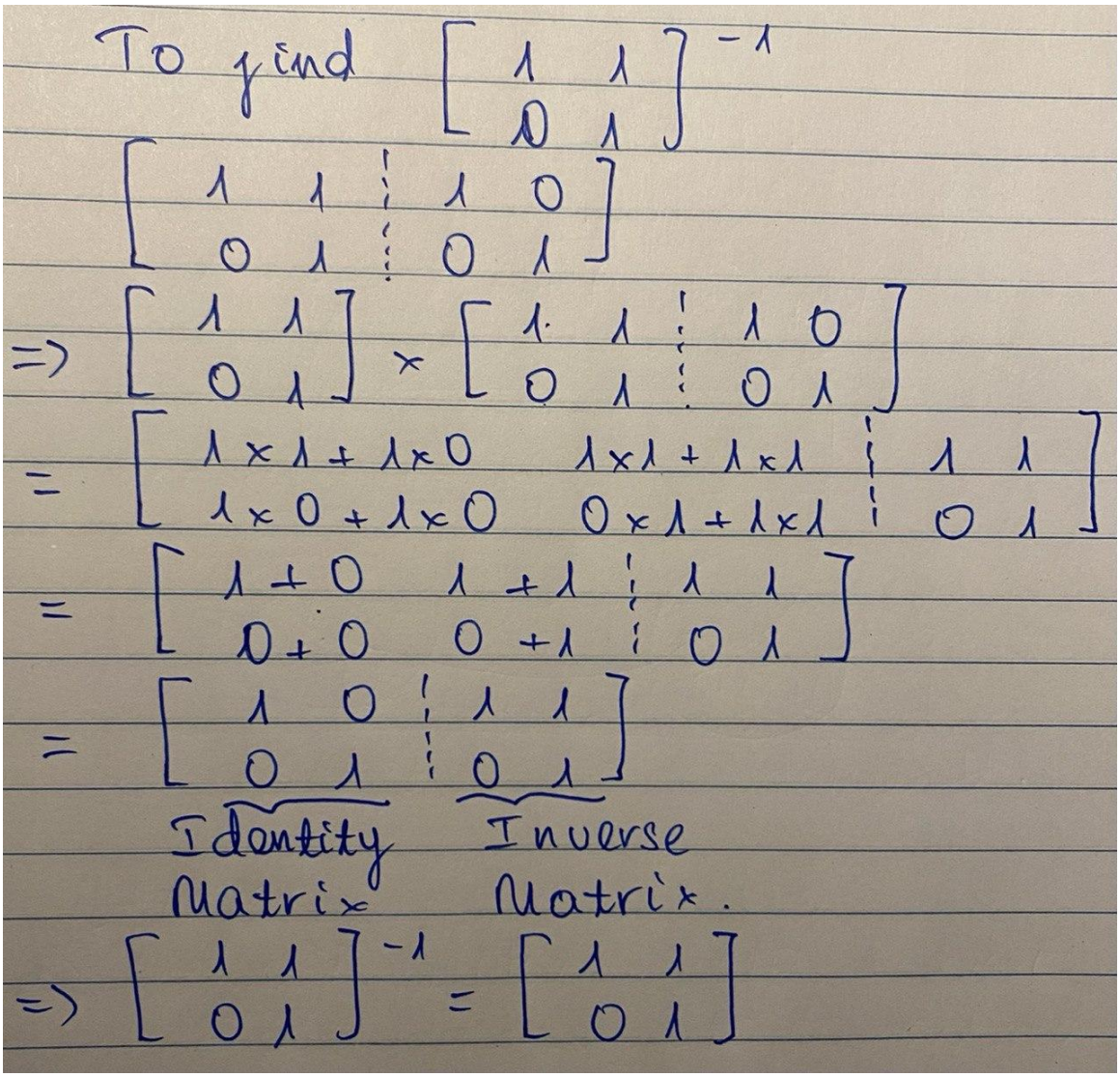
```
In [6]  1 M1 = matrix(GF(2),[[1,1],[0,1],[1,0]])
        2 M2 = matrix(GF(2),[[1,0],[0,1]])
        3 M1*M2
```

Run Code

```
Out [6]
          [1 1]
          [0 1]
          [1 0]
```

iii. $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}^{-1}$

By hand:



Check by SageMath:

Code Cell 1 of 3

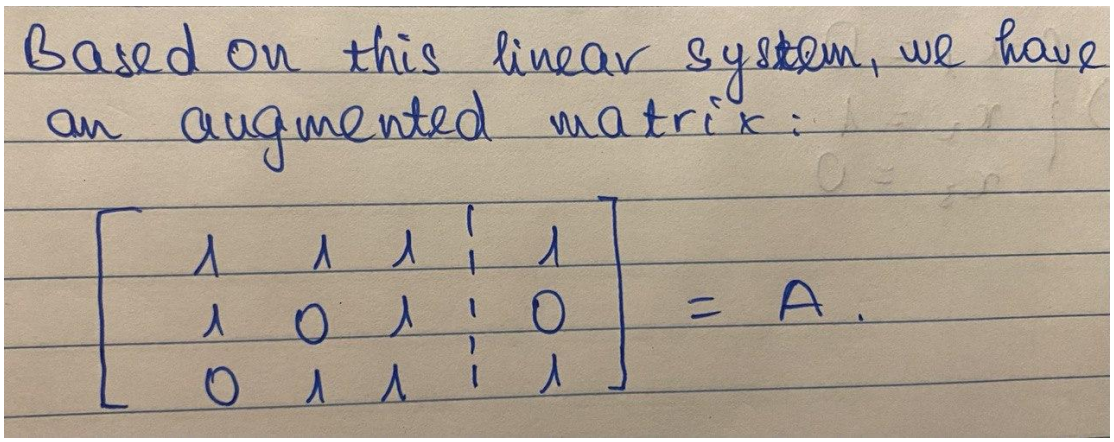
```
In [7]   1  M1 = matrix(GF(2),[[1,1],[0,1]])
         2  M1.inverse()
```

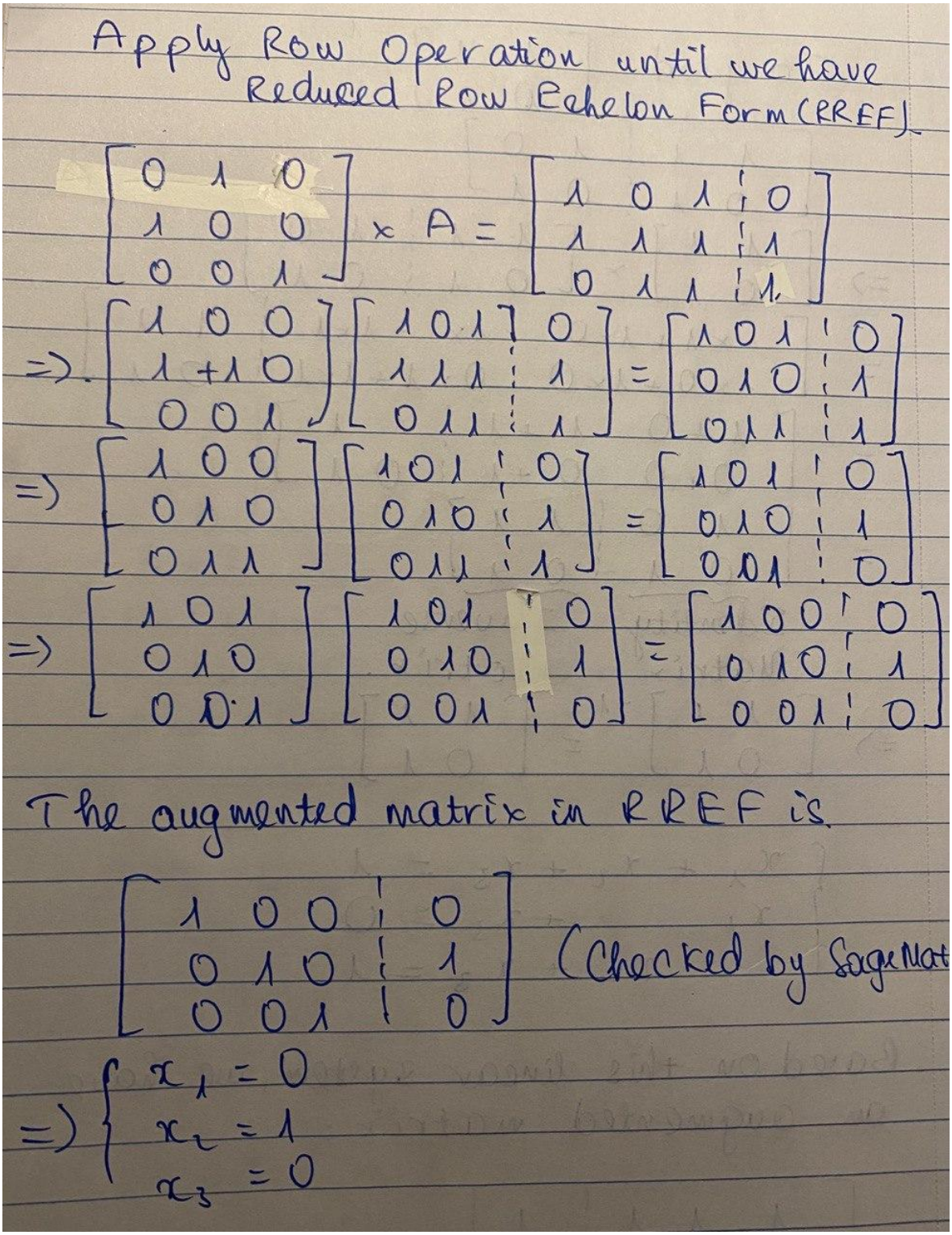
Run Code

```
Out [7]
           [1 1]
           [0 1]
```

c. Linear equations can also be defined over F_2. Solve the following linear system over F_2.

$$x_1 + x_2 + x_3 = 1$$
$$x_1 + x_3 = 0$$
$$x_2 + x_3 = 1$$





Check by SageMath

Code Cell 1 of 3

```
In [9]
1 M = matrix(GF(2),[[1,1,1],[1,0,1],[0,1,1]])
2 v = vector(GF(2),[1,0,1])
3 A = M.augment(v, subdivide = True)
4 A.rref()
```

Run Code

```
Out [9]
[1 0 0|0]
[0 1 0|1]
[0 0 1|0]
```

d. Let's look at a common application. When you send information over a channel as a string of bits there is a chance that a bit might flip, corrupting the message (i.e., a 0 becomes a 1 or 1 becomes a 0). There are a couple of ways to try to catch this.

   i. Repetition: Instead of sending x send the vector (x, x, x). How should the receiver interpret the following repeated bits?

   (i) (1, 1, 1); With this vector, there are no flipped bits, so that the sent bit is 1.

   (ii) (1, 0, 0); We assume that the major bit appearing in the vector is the sent bit. Therefore, in this vector, the sent bit is 0.

   (iii) (0, 1, 0); We also assume that the major bit appearing in the vector is the sent bit. Therefore, in this vector, the sent bit is 0 as well.

How many flips could this method detect?

In this method, we assume that the number of flipped bits is minimized and the major bit appearing in the vector is the one that we send. Therefore, this method can only detect one flipped bit in the vector, which may not be appropriate. For example, in the second vector, the intended bit is 1, so the initial vector should be (1, 1, 1). But, when it appears (1, 0, 0), we would think the vector is (0, 0, 0) and the sent bit is 0. This method is incorrect in some cases.

ii. Parity codes This method combines repetition and addition to detect flips when sending two bits. Instead of sending $\overline{x} = (x_1, x_2)$ send $\overline{b} = (x_1, x_1, x_2, x_2, x_1 + x_2)$. How should the receiver interpret the following vectors?

(i) (1, 1, 0, 1, 1);

- First scenario: Assuming that the last component is not flipped, we start with the last component because it is $x_1 + x_2$, so that we can minimize the number of possibilities. In this vector, the last component is 1 and 1 = 0 + 1. The first two components are 1, so x_1 is 1. Based on the last component, when $x_1 = 1$, x_2 has to be 0. Therefore, in this vector, the fourth component is flipped. In general, one component is flipped.
- Second scenario: Assuming that the last component is flipped, we start with the last component, which is 0 now. When $x_1 + x_2 = 0$, there are two

possibilities: (x_1, x_2) = {(1, 1), (0, 0)}. Because the first two components are 1, x_1 is 1, which means we only take the pair of (x_1, x_2) = (1, 1) and $x_2 = 1$. Therefore, in this vector, the third and last component is flipped. In general, two components are flipped.

(ii) (1, 1, 0, 0, 0);

- First scenario: Assuming that the last component is not flipped, we start with the last component, which is 0. In this case, there are 2 pairs of possibilities: (x_1, x_2) = {(1, 1), (0, 0)}. If the first two elements are not flipped, x_1 = 1 and x_2 = 1, which means the third and fourth components are flipped. If the third and fourth elements are not flipped, x_2 = 0 and x_1 = 0, which means that the first and second components are flipped. In general, two components are flipped.
- Second scenario: Assuming that the last component is flipped, it now becomes $1 = x_1 + x_2$. In this case, x_1 = 1 and x_2 = 0, the first four components are not flipped. In general, one component is flipped.

(iii) (1, 1, 0, 0, 1);

- First scenario: Assuming that the last component is not flipped, it is $1 = x_1 + x_2$. Based on the first four components, we have x_1 = 1 and x_2 = 0,

which satisfies the last component. Therefore, this vector does not have any error. In general, no component is flipped.

- Second scenario: Assuming that the last component is flipped, it now becomes 0. In this case, there are 2 pairs of possibilities: (x_1, x_2) = {(1, 1), (0, 0)}. If the first two elements are not flipped, x_1 = 1 and x_2 = 1, which means the third and fourth components are flipped. If the third and fourth element are not flipped, x_2 = 0 and x_1 = 0, which means the first and second components are flipped. In general, three components are flipped.

(iv) (0, 1, 0, 0, 1);

- First scenario: Assuming that the last component is not flipped, we have 1 = x_1+x_2. When the third and fourth components are 0, x_2 = 0. Then, x_1 has to be 1 to satisfy the last component, which means the first element is flipped. In general, one component is flipped.
- Second scenario: Assuming that the last component is flipped, it now becomes 0. In this case, there are two pairs of possibilities: (x_1, x_2) = {(1, 1), (0, 0)}. When the third and fourth components are 0, x_2 = 0. Then, x_1 has to be 0, which means the second component is flipped as well. In general, two components are flipped.

(v) (0, 1, 0, 0, 0).

- First scenario: Assuming that the last component is not flipped, we have 0 = x_1+x_2. When the third and fourth components are 0, x_2 = 0. Then, x_1 has to be 0 to satisfy the last component, which means the second element is flipped. In general, one component is flipped.
- Second scenario: Assuming that the last component is flipped, it now becomes 1. In this case, there are two pairs of possibilities: (x_1, x_2) = {(1, 0), (0, 1)}. When the third and fourth components are 0, x_2 = 0. Then, x_1 has to be 1, which means the first element is flipped. In general, two components are flipped.

How many flips could this method detect? (Hint: What is the maximum number of flips before a receiver would get the same output for two distinct inputs?)

Based on the interpretation and assumptions above, we can see that, in the worst-case scenario, there are a maximum of 3 components that are flipped when a receiver would get the same output for two distinct inputs. Specifically, in the third vector (1, 1, 0, 0, 1), when a receiver would get the same output for two distinct inputs, the inputs are (1, 0) but the outputs are (1, 1) or (0, 0). There are 3 components that are flipped.

e. These are examples of *codes*. A code is *linear* if it sends each n-bit block $\vec{x} \in \mathbb{F}_2^n$ of a message as an m-bit block $\vec{y} = A\vec{x} \in \mathbb{F}_2^m$ for A an $m \times n$ matrix with entries in $\mathbb{F}_2$. We call the matrix A the *generator* of the code and $\vec{y} \in Im(A)$ a *codeword*.[1] Find the generator and set of codewords for the repeition and parity methods described above.

With Repetition method, we have:

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

Because $\overline{y} \in Im(A)$, we firstly find the subspace Im(A) by checking the linear independence of A's column vectors. We use SageMath to find the RREF of A.

Code Cell 1 of 3

```
In [3]   1  A = matrix([[1,1,0],[1,0,1],[1,0,0]])
         2  A.rref()
```

Run Code

```
Out [3]
           [1 0 0]
           [0 1 0]
           [0 0 1]
```

Based on this result, all the column vectors are linearly independent, thus:

$$Im(A) = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Because $\overline{y} \in Im(A)$, so we can create a linear combination of $\overline{y}$ based on Im(A):

$$\vec{y} = c_1 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + c_2 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + c_3 \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

where $c_1,\ c_2,\ c_3 \in F_2 = \{0,\ 1\}$

With Parity Method:

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

Because $\overline{y} \in Im(A)$, we firstly find the subspace Im(A) by checking the linear independence of A's column vectors. We use SageMath to find the RREF of A.

Code Cell 1 of 3

```
In [5]  1 A = matrix([[1,1,1,0,0],[1,1,1,1,1],[0,0,0,0,0],[1,0,0,0,0],[1,0,1,1,0]])
        2 A.rref()
```

Run Code

```
Out [5]
[ 1  0  0  0  0]
[ 0  1  0  0  1]
[ 0  0  1  0 -1]
[ 0  0  0  1  1]
[ 0  0  0  0  0]
```

Based on this result, only first four column vectors are linearly independent, thus:

$$Im(A) = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Because $\bar{y} \in Im(A)$, so we can create a linear combination of $\bar{y}$ based on Im(A):

$$\vec{y} = c_1 \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} + c_2 \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + c_3 \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} + c_4 \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

where $c_1,\ c_2,\ c_3,\ c_4 \in F_2 = \{0,\ 1\}$

f. Consider the generating matrix

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

i. What is the encoding of the message $\vec{x}$ = (0, 1, 0, 1).

Because $\bar{y} = H\bar{x}$, we encode $\bar{x}$ by calculating $\bar{y}$. In this case,

$$\vec{y} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Code Cell 1 of 3

```
In [5]  1 H = matrix(GF(2),[[1,1,0,1,0,0,1],[0,1,0,1,0,1,0],[1,0,0,1,1,0,0],[1,1,1,0,0,0,0]]).T
        2 x = vector(GF(2),[0,1,0,1])
        3 H*x
```

Run Code

```
Out [5]  (1, 0, 1, 1, 0, 1, 0)
```

ii. Write pseudocode for a procedure that decodes a codeword $\overline{y}$. Use your procedure to decode $\overline{y_1}$ = (0, 0, 1, 1, 0, 0, 1).

Because $\overline{y} = H\overline{x}$, to decode a codeword $\overline{y}$, we only need to find vector $\overline{x}$.

Pseudocode to decode a codeword $\overline{y}$:

1. Create augmented matrix of H and $\overline{y}$
2. Use Row Operations or SageMath to obtain the RREF of the augmented matrix

3. Based on the RREF, we find $\bar{x}$ and check whether the codeword $\bar{y}$ is decodable

   - If there is an unique solution of $\bar{x}$, the codeword $\bar{y}$ is decodable.
   - If there is no solution of $\bar{x}$, the codeword $\bar{y}$ is not decodable

Decode $\overline{y_1}$ = (0, 0, 1, 1, 0, 0, 1).

Code Cell 1 of 3

```
In [17]  1 H = matrix(GF(2),[[1,1,0,1,0,0,1],[0,1,0,1,0,1,0],[1,0,0,1,1,0,0],[1,1,1,0,0,0,0]]).T
         2 y1 = vector(GF(2),[0, 0, 1, 1, 0, 0, 1])
         3 y2 = vector(GF(2),[0, 1, 1, 1, 0, 0, 1])
         4 print(H.augment(y1, subdivide = True).rref())
```

Run Code

```
Out [17]  [1 0 0 0|1]
          [0 1 0 0|0]
          [0 0 1 0|0]
          [0 0 0 1|1]
          [0 0 0 0|0]
          [0 0 0 0|0]
          [0 0 0 0|0]
```

Based on the result, we have one unique solution for $\bar{x}$

$$\vec{x} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

iii. Suppose that you receive the message $\overline{y_2}$ = (0, 1, 1, 1, 0, 0, 1) via a noisy channel. Can you decode $\overline{y_2}$? Why or why not?

To decode $\overline{y_2}$, we have to make sure that $\overline{y_2} \in Im(H)$, which means $\overline{y_2}$ has to be linearly dependent on Im(H)'s column vectors. Therefore, we put H and $\overline{y_2}$ in an augmented matrix and then obtain its RREF. Based on the result of RREF, all the column vectors of Im(H) and $\overline{y_2}$ are linearly independent on each other, which means $\overline{y_2} \notin Im(H)$. In this case, $\overline{y_2}$ cannot be decoded

Check by SageMath:

Code Cell 1 of 3

```
In [18]  1 H = matrix(GF(2),[[1,1,0,1,0,0,1],[0,1,0,1,0,1,0],[1,0,0,1,1,0,0],[1,1,1,0,0,0,0]]).T
         2 y1 = vector(GF(2),[0, 0, 1, 1, 0, 0, 1])
         3 y2 = vector(GF(2),[0, 1, 1, 1, 0, 0, 1])
         4 print(H.augment(y2, subdivide = True).rref())
```

Run Code

```
Out [18]  [1 0 0 0|0]
          [0 1 0 0|0]
          [0 0 1 0|0]
          [0 0 0 1|0]
          [0 0 0 0|1]
          [0 0 0 0|0]
          [0 0 0 0|0]
```

iv. To find the error let's assume at most one bit was flipped. Then $\vec{y}_2 = \vec{y}_1 + \vec{e}_i$ where $\vec{e}_i$ is the ith standard basis vector in $\mathbb{F}_2^7$. Why?

In the scenario that at most one bit was flipped, only one component of $\overline{y_1}$ is an error. Because we are operating in $F_2 = \{0, 1\}$, there are two possibilities of the mistaken component: 0 or 1. Because $\overline{e_i}$ is an i_{th} standard basis vector, the component of $\overline{e_i}$ at the

i_{th} position is always 1. Based on F_2 operations, if the mistaken component is 0, it would become 1 after adding 1 because 1 + 1 = 0. If the mistaken component is 1, it would become 0 after adding 1 because 0 + 1 = 1. Therefore, the statement that $\overline{y_2} = \overline{y_1} + \overline{e_i}$ where $\overline{e_i}$ is an i_{th} the standard basis vector is valid.

v. This implies that $\overline{y_2} = \overline{y_1} + \overline{e_i}$

$$\overline{y_2} = \overline{y_1} + \overline{e_i}$$

$$\Leftrightarrow \overline{y_2} + \overline{e_i} = \overline{y_1} + \overline{e_i} + \overline{e_i}$$

$$\Leftrightarrow \overline{y_2} + \overline{e_i} = \overline{y_1}$$

(because two $\overline{e_i}$ has the component of 1 located at the same i_{th} position and 1 + 1 = 0, $\overline{e_i} + \overline{e_i} = \overline{0}$)

vi. Putting the last two observations together gives us a method to decode a nosiy signal: Compute $\vec{y}_2 + \vec{e}_j$ for $j = 1 \ldots m$ to find j so that $\vec{y}_2 + \vec{e}_j$ is a codeword. Then the message is $\vec{x}$ such that $H\vec{x} = \vec{y}_2 + \vec{e}_j$. Use this method to decode $\vec{y}_2$.

By using SageMath with a for loop that runs through every $\overline{e_j}$, we can check which one is the correct codeword to decode from y_2.

Code Cell 1 of 3

```
H = matrix(GF(2),[[1,1,0,1,0,0,1],[0,1,0,1,0,1,0],[1,0,0,1,1,0,0],[1,1,1,0,0,0,0]]).T
y2 = vector(GF(2),[0, 1, 1, 1, 0, 0, 1])
for j in range(7):
    e0 = vector(GF(2),[0, 0, 0, 0, 0, 0, 0])
    e0[j] = 1
    yj = y2 + e0
    print(H.augment(yj, subdivide = True).rref())
    print('------------')
```

Run Code

```
[1 0 0 0|0]
[0 1 0 0|0]
[0 0 1 0|0]
[0 0 0 1|0]
[0 0 0 0|1]
[0 0 0 0|0]
[0 0 0 0|0]
------------
[1 0 0 0|1]
[0 1 0 0|0]
[0 0 1 0|0]
[0 0 0 1|1]
[0 0 0 0|0]
[0 0 0 0|0]
[0 0 0 0|0]
------------
[1 0 0 0|0]
[0 1 0 0|0]
[0 0 1 0|0]
[0 0 0 1|0]
[0 0 0 0|1]
[0 0 0 0|0]
[0 0 0 0|0]
------------
[1 0 0 0|0]
[0 1 0 0|0]
[0 0 1 0|0]
[0 0 0 1|0]
[0 0 0 0|1]
[0 0 0 0|0]
[0 0 0 0|0]
------------
[1 0 0 0|0]
[0 1 0 0|0]
[0 0 1 0|0]
[0 0 0 1|0]
[0 0 0 0|1]
[0 0 0 0|0]
[0 0 0 0|0]
------------
[1 0 0 0|0]
[0 1 0 0|0]
[0 0 1 0|0]
[0 0 0 1|0]
[0 0 0 0|1]
[0 0 0 0|0]
[0 0 0 0|0]
------------
[1 0 0 0|0]
[0 1 0 0|0]
[0 0 1 0|0]
[0 0 0 1|0]
[0 0 0 0|1]
[0 0 0 0|0]
[0 0 0 0|0]
```

Based on the results after the for loop, we can that there is only one unique result that satisfies the equation $\overline{y_2} + \overline{e_j} = H\overline{x}$. That result is the one we want to get from the codeword $\overline{y_2}$, which is $\overline{x} = (1,\ 0,\ 0,\ 1)$.

g. This suggests the following result:

Theorem *The linear code with generator G is an error-correcting code if and only if it satisfies the following three requirements: (1) The vector $\vec{e}_i$ is not a codeword for all i. (2) If $\vec{e}_i + \vec{e}_j$ is a codeword, then $\vec{e}_i + \vec{e}_j = 0$. (3) Every codeword has a unique input, i.e., $G\vec{x}_1 = G\vec{x}_2$ if and only if $\vec{x}_1 = \vec{x}_2$.*

Verify that H satisfies each of these requirements.

(1) The vector $\overline{e_i}$ is not a codeword for all i:

To prove this requirement, I run a for loop of different $\overline{e_i}$ with i = {1, 2,..., 7} and put $\overline{e_i}$ and H in an augmented matrix to obtain RREF. If any linear combination of $\overline{e_i}$ has a solution, that $\overline{e_i}$ is a codeword. Otherwise, $\overline{e_i}$ is not a codeword.

Code Cell 1 of 3

In [11]:
```
H = matrix(GF(2),[[1,1,0,1,0,0,1],[0,1,0,1,0,1,0],[1,0,0,1,1,0,0],[1,1,1,0,0,0,0]]).T
for j in range(7):
    e0 = vector(GF(2),[0, 0, 0, 0, 0, 0, 0])
    e0[j] = 1
    print(H.augment(e0, subdivide = True).rref())
    print('------------')
```

```
[1 0 0 0|0]   [1 0 0 0|0]    [1 0 0 0|0]    [1 0 0 0|0]
[0 1 0 0|0]   [0 1 0 0|0]    [0 1 0 0|0]    [0 1 0 0|0]
[0 0 1 0|0]   [0 0 1 0|0]    [0 0 1 0|0]    [0 0 1 0|0]
[0 0 0 1|0]   [0 0 0 1|0]    [0 0 0 1|0]    [0 0 0 1|0]
[0 0 0 0|1]   [0 0 0 0|1]    [0 0 0 0|1]    [0 0 0 0|1]
[0 0 0 0|0]   [0 0 0 0|0]    [0 0 0 0|0]    [0 0 0 0|0]
[0 0 0 0|0]   [0 0 0 0|0]    [0 0 0 0|0]    [0 0 0 0|0]
-----------   -----------    -----------    -----------
[1 0 0 0|0]   [1 0 0 0|0]    [1 0 0 0|0]
[0 1 0 0|0]   [0 1 0 0|0]    [0 1 0 0|0]
[0 0 1 0|0]   [0 0 1 0|0]    [0 0 1 0|0]
[0 0 0 1|0]   [0 0 0 1|0]    [0 0 0 1|0]
[0 0 0 0|1]   [0 0 0 0|1]    [0 0 0 0|1]
[0 0 0 0|0]   [0 0 0 0|0]    [0 0 0 0|0]
[0 0 0 0|0]   [0 0 0 0|0]    [0 0 0 0|0]
```

Based on the results, there is no vector $\overline{e_i}$ that can be a codeword. The first requirement is satisfied.

(2) If $\overline{e_i} + \overline{e_j}$ is a codeword, then $\overline{e_i} + \overline{e_j} = 0$

Because $\overline{e_i}\ and\ \overline{e_j}$ are standard basis vectors, there are only 2 possibilities of $\overline{e_i} + \overline{e_j}$ vector: (1) $\overline{e_i} + \overline{e_j} = \overline{0}$, which means they are the same; (2) $\overline{e_i} + \overline{e_j}$ has two components that have the value of 1 (e.g., $\overline{e_i} + \overline{e_j}$ = (1, 1, 0, 0, 0, 0, 0)). However, to be a codeword, the linear combination of $H\overline{x} = \overline{e_i} + \overline{e_j}$ has to have solution, meaning that $\overline{e_i} + \overline{e_j}$ has to be in the Column space of H - col(H).

With the first possibility when $\overline{e_i} + \overline{e_j}$ = 0, there always exists one unique solution that makes $H\overline{x} = \overline{e_i} + \overline{e_j} = 0$, which is $\overline{x} = \overline{0}$. Therefore, $\overline{e_i} + \overline{e_j} = \overline{0}$ is a codeword.

With the second possibility, when $\overline{e}_i + \overline{e}_j$ has two components with the values of 1, $\overline{e}_i + \overline{e}_j$ has two dimensions. However, looking at all the column vectors of H, we see that they all have at least three 1s, which means they all have at least three dimensions. Therefore, $\overline{e}_i + \overline{e}_j$ with two 1s cannot be in the column space of H - col(H). I also checked by SageMath with two for loops of i and j (Code below) and no case has a solution. Therefore, $\overline{e}_i + \overline{e}_j$ with two 1s is not a codeword.

In conclusion, the second requirement is satisfied.

Code Cell 1 of 3

```
In [14]
H = matrix(GF(2),[[1,1,0,1,0,0,1],[0,1,0,1,0,1,0],[1,0,0,1,1,0,0],[1,1,1,0,0,0,0]]).T
for i in range(6):
    for j in range(1,7,1):
        e0 = vector(GF(2),[0, 0, 0, 0, 0, 0, 0])
        e0[j] = 1
        e0[i] = 1
        print(e0)
        print(H.augment(e0, subdivide = True).rref())
        print('------------')
```

(3) Every codeword has a unique input ($\overline{x}$)

By looking at H's RREF, we can see that all columns are pivot, which means, with every codeword - $\overline{y}$ that is valid, we can always only obtain one unique solution of $\overline{x}$ - the input of words that we decode. Therefore, the third requirement is satisfied.

Code Cell 1 of 3

```
In [15]  1  H = matrix(GF(2),[[1,1,0,1,0,0,1],[0,1,0,1,0,1,0],[1,0,0,1,1,0,0],[1,1,1,0,0,0,0]]).T
         2  H.rref()
```

Run Code

```
Out [15]
            [1 0 0 0]
            [0 1 0 0]
            [0 0 1 0]
            [0 0 0 1]
            [0 0 0 0]
            [0 0 0 0]
            [0 0 0 0]
```

2. To infinity, and beyond

In class, we explored how we can apply the concepts of vectors and vector spaces. In this exercise, we know that $P(\mathbb{R})$ is the set of real-valued polynomials, where $(p + q)(x) = (p(x) + q(x))$, and $(cp)(x) = cp(\mathrm{x})$

a. To prove that $P(\mathbb{R})$ is a vector space, we need to satisfy the following conditions:

   i. The set is closed under Addition:

   Let's say that $P(1) = \{2x + 1,\ x + 3\}$ is in the basis of $P(\mathbb{R})$, then:

   $(2x + 1) + (x + 3) = 3x + 4$

   $(2x + x) + (1 + 3) = 3x + 4$

   ii. The set is closed under Scalar Multiplication:

   $(cp) = 2(2x + 1) = 4x + 2, c = 2$

   $\{4x + 2\} \in P(\mathbb{R})$

iii. Contains the **0**-vector:

Any general form of $P(\mathbb{R})$, would contain:

$P(1) = ax + b$

$P(2) = ax^2 + bx + c$

$P(3) = ax^3 + bx^2 + cx + d$, and so on, where every element $\{a_1, b_1, c_1, \ldots, z_n\} \in \mathbb{R}$, thus:

$for\ \{a, b\} = \{0, 0\} \Rightarrow P(1) = 0x + 0 = 0$

iv. Has an Additive Inverse:

For $u = 2x + 1$, $u \in P(\mathbb{R})$, then

$u + (-u) = 2x + 1 + (-2x - 1) = 0$

From (i), (ii), (iii), and (iv), we can say that $P(\mathbb{R})$ is a Vector Space, since it satisfies the necessary conditions of a set to be a vector space, because it is closed under additions, the requirements for a set to be a vector space, which include closure, communicativity, associativity, identity element, and inverse element under addition, which are associative, and we can just test for closed under addition and additive inverse. The multiplication operations include closure, distribution over a vector sum, scalar sum, scalar product, and scalar identity, for which we are using the associative heuristic to calculate.

b. If $P(\mathbb{R})$ is not to be a finite-dimensional Vector Space, then this means it is an infinite-dimensional Vector Space.

For $P(\mathbb{R})$, where $P(n)$ denoted the highest degree real-valued polynomial, then every one of these polynomials in the span of this list must have a degree at most n. Thus, our list cannot

span $R(\mathbb{R})$). The vector space $\mathbb{R}^n$, $n \to \infty$, consisting of all sequences of elements in $\mathbb{R}$ is infinite-dimensional, the highest degree of the polynomial is infinity.

c. A linearly independent spanning set of $P(\mathbb{R})$ is a basis for $P(\mathbb{R})$:

$$P(1) = ax + b$$

$$P(2) = ax^2 + bx + c$$

$$P(3) = ax^3 + bx^2 + cx + d$$

$$P(n) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x^1 + a_0$$

Therefore,

$$P(n-1) = a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_2 x^2 + a_1 x^1 + a_0$$

is an infinite basis for $P(\mathbb{R})$, since all the vectors in $P(n)$ are linearly independent.

d. A power series is a generalization of the concept of a polynomial. Formally described as:

$$\sum_{n=0}^{\infty} a_n x^n = a_0 + a_1 x^1 + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + a_n x^n$$

i. $$\left(\sum_{n=0}^{\infty} a_n x^n\right) + \left(\sum_{n=0}^{\infty} b_n x^n\right) = \sum_{n=0}^{\infty} (a_n + b_n) x^n$$

If we assume that $\{a, b\} \in \mathbb{R}$, then we can use a and b as coefficients to the polynomial in the form $a_n x^n$ and $b_n x^n$

1. Closed Under Addition:

$$a_1 x^1 + b_1 x^1 = (a_1 + b_1)(x^1)$$

2. Closed under Scalar Multiplication:

$$c \sum_{n=0}^{1} a_n x^n = c(a_1 x) = (ca_1)x$$

$$c \sum_{n=0}^{2} a_n x^n = c(a_1 x + a_2 x^2) = (ca_1)x + (ca_2)x^2$$

3. Contains the **0**-vector:

$$a_0 x^0 = 0(1) = 0$$

$$b_0 x^0 = 0(1) = 0$$

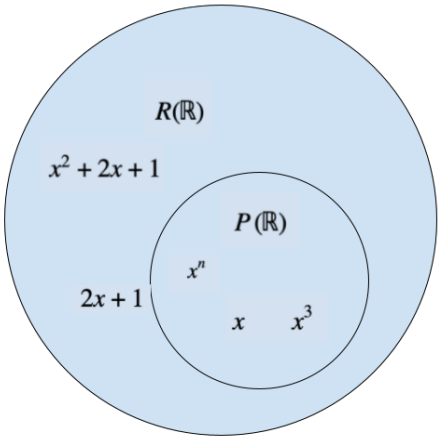
4. Additive Inverse:

$$w = a_1 x^1$$

$w + (-w) = a_1 x^1 + (-a_1 x^1) = 0$, so the set contains the additive inverse

From (1), (2), (3), and (4), we can say that $R(\mathbb{R})$ is a Vector Space, since it satisfies the necessary conditions of a set to be a vector space, because it is closed under additions, the requirements for a set to be a vector space, which include closure, communicativity, associativity, identity element, and inverse element under addition, which are associative, and we can just test for closed under addition and additive inverse. The multiplication operations include closure, distribution over a vector sum, scalar sum, scalar product, and scalar identity, for which we are using the associative heuristic to calculate.

ii. Let $P(\mathbb{R})$ and $R(\mathbb{R})$ are two sets.



From the diagram above we can see that every element of $R(\mathbb{R})$ is in $P(\mathbb{R})$, more specifically expressed in an example:

$v = x^2 + 1$

$w = x^2$, so we can say that v contains w in itself and can be re-written as:

$v = w + 1,\ w = v - 1$,thus $w = v + c$, where c is the constant in $P(\mathbb{R})$.

iii. To show that the basis $P(\mathbb{R})$ is not a basis for $R(\mathbb{R})$:

$$P(n) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x^1 + a_0$$

$$R(n) = \sum_{n=0}^{\infty} x^n = 1 + x + x^2 + \ldots + x^n$$

Thus, the set of $R(n)$ contains $P(n)$ in itself but $P(n)$ always has an element a_0, which is a constant.

3. Animation Studios

In class, we explored how linear transformations can be used in computer graphics to rotate, scale, reflect and project graphic objects. One operation that we still do not know how to perform is translation, that is, moving an object up/down, left/right.

a. Describe why linear transformations cannot perform translations. (Hint: Consider the zero-vector)

We thus introduce the idea of affine transformations. Those are transformations of the type $T(\overline{v}) = A\overline{v} + \overline{b}$

A translation is moving every point of a figure, shape, or space by the same distance in a given direction, while Linear Transformation is transforming from a domain to a co-domain in the same vector space. In a vector space, there always exists vector 0. In Linear Transformation, vector $\overline{0}$ would be transformed to vector $\overline{0}$. However, in Translation, because every point has to move by the same distance in the same direction, vector $\overline{0}$ would not stay at the original point after translation. For example, when translating to the right one unit, vector $\overline{0}$ moves to the position of $(1, 0)$. It is against the phenomenon in Linear Transformation. Therefore, linear transformations cannot perform translations.

b. What would the matrix A and the vector b be to perform a translation of one unit to up? Test your affine transformation on the vector $v = \langle 1, 3 \rangle$

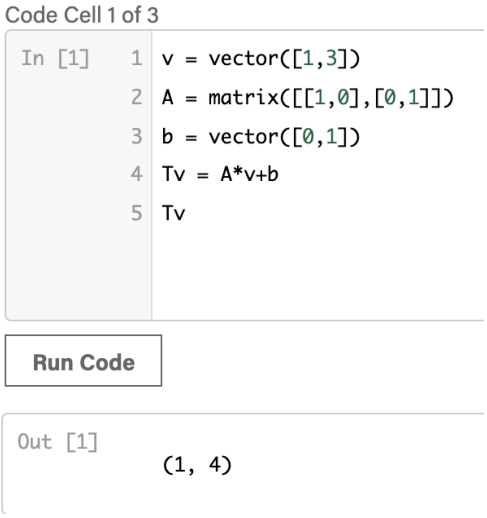
In translation, the magnitude and direction of the initial vector stay unchanged, so A is an identity matrix to maintain the magnitude and direction of the initial vector. However, the vector

moves up one unit, which means it moves to the (0, 1) direction with the distance of one unit. Therefore, we have:

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

$$\vec{b} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Based on the formula $T(\overline{v}) = A\overline{v} + \overline{b}$, we use SageMath to calculate $T(\overline{v}) = (1, 4)$



c. We may combine transformations, for example, by rotating a vector by 90 degrees counterclockwise, and then translating it to the right by one unit as a single affine transformation. What should A and vector b be to perform these two transformations? Again, test it with v = ⟨1, 3⟩.

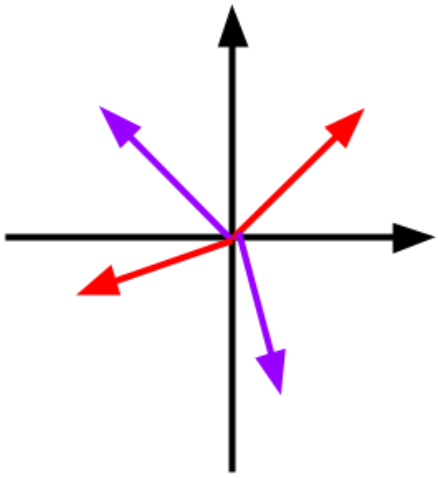


Figure 1: It shows some examples of random vectors rotated counterclockwise 90 degrees. The red vectors are the original vectors and the purple ones are the rotated vectors.

Based on the visualization, we can see that when rotating the vector counterclockwise 90 degree, the rotated vector's x-value is negative the original vector's y-value and the rotated vector's y-value is the original vector's x-value. On the other hand, when it translates to the right by one unit, it moves to the (1, 0) direction by the distance of one unit. Therefore, we have:

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

$$\vec{b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Based on the formula $T(\overline{v}) = A\overline{v} + \overline{b}$, we use SageMath to calculate $T(\overline{v})$ = (-2, 1)

Code Cell 1 of 3

```
In [2]
1 v = vector([1,3])
2 A = matrix([[0,-1],[1,0]])
3 b = vector([1,0])
4 Tv = A*v+b
5 Tv
```

Run Code

```
Out [2]
(-2, 1)
```

d. Order matters! What if we had instead translated the vector v one unit to the right first, and then rotated it by 90 degrees counterclockwise? What would be the matrix A and vector $\vec{b}$ that perform these operations, written in the standard affine form Av+b?

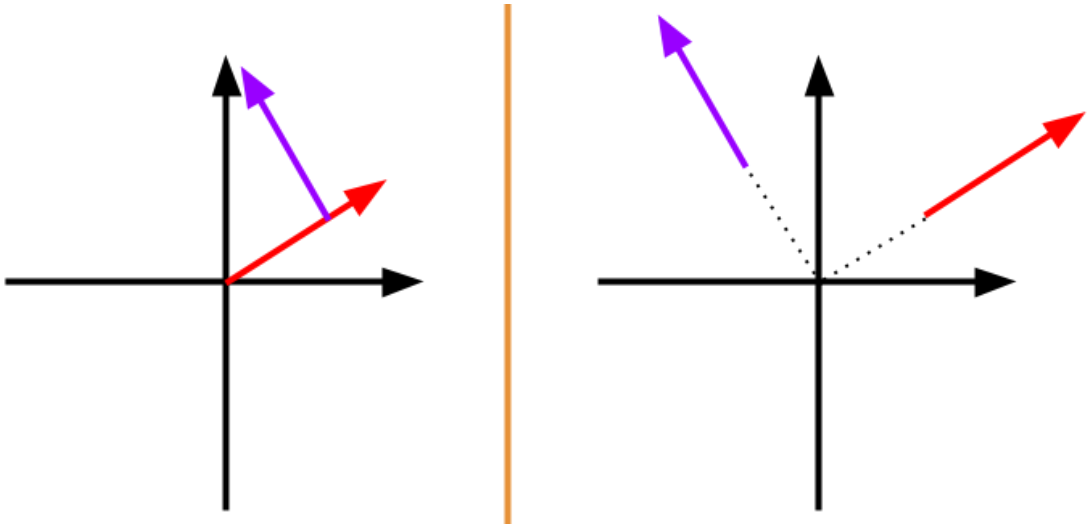


Figure 2: This figure shows the difference between the order of transformations: Rotation → Translation (Left) and Translation → Rotation (Right).

When rotating a vector, we rotate it around the original point (0, 0). If the order is Rotation → Translation, the original vector would be rotated around the original point before moving to a new position by translation (Figure 2 - Left). If the order is Translation → Rotation, the original vector would be moved to a new position by translation before rotating around the original point, where the distance between the original point and the vector has to be maintained (Figure 2 - Right). Therefore, after transformation in different orders, the position of each transformed vector is different from the other.

In the scenario of Translation → Rotation, when we translate the vector $\bar{v}$ one unit to the right first and then rotate it by 90 degrees counterclockwise, the rotation operation is the same when the rotated vector's x-value is negative the original vector's y-value and the rotated vector's y-value is the original vector's x-value. It means that A of Translation → Rotation is similar to A of Rotation → Translation. However, the translation between the two orders is different, which means vector $\bar{b}$ in two cases are different. When doing Translation → Rotation, the translated vector is rotated around the original point as well, So, to maintain the distance between the original point with the translated vector, $\bar{b}$ has to rotate with $\bar{v}$. In this case, $\bar{b}$ also has to rotate counterclock 90 degree. It turns out that, instead of moving to the right one unit, the transformed vector moves up one unit. Therefore, we have:

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

$$\vec{b} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Based on the formula $T(\bar{v}) = A\bar{v} + \bar{b}$, we use SageMath to calculate $T(\bar{v})$ = (-3, 2).

Code Cell 1 of 3

```
In [3]  1  v = vector([1,3])
        2  A = matrix([[0,-1],[1,0]])
        3  b = vector([0,1])
        4  Tv = A*v+b
        5  Tv
```

Run Code

```
Out [3]
           (-3, 2)
```

e. Compare your matrices A and vector b, and your final transformed vector Av+b from parts (c) and (d).

In part (c), matrix A, vector b and the final transformed vector are:

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

$$\vec{b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$T(\vec{v}) = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

In part (d), matrix A, vector b and the final transformed vector are:

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

$$\vec{b} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$T(\vec{v}) = \begin{bmatrix} -3 \\ 2 \end{bmatrix}$$

Comparison:

- Matrix A in two transformations is the same when it does the task of rotation with the same angle and counterclockwise.
- Vector b in two transformations are different when $\overline{b}$ in part (c) is for moving to the right one unit and $\overline{b}$ in part (d) is for moving up one unit.
- The final transformed vectors in two transformations are different, showing that they have different positions.

f. Finally, we can "cheat" to encode everything as a single matrix multiplication. We construct a combined vector V given as:

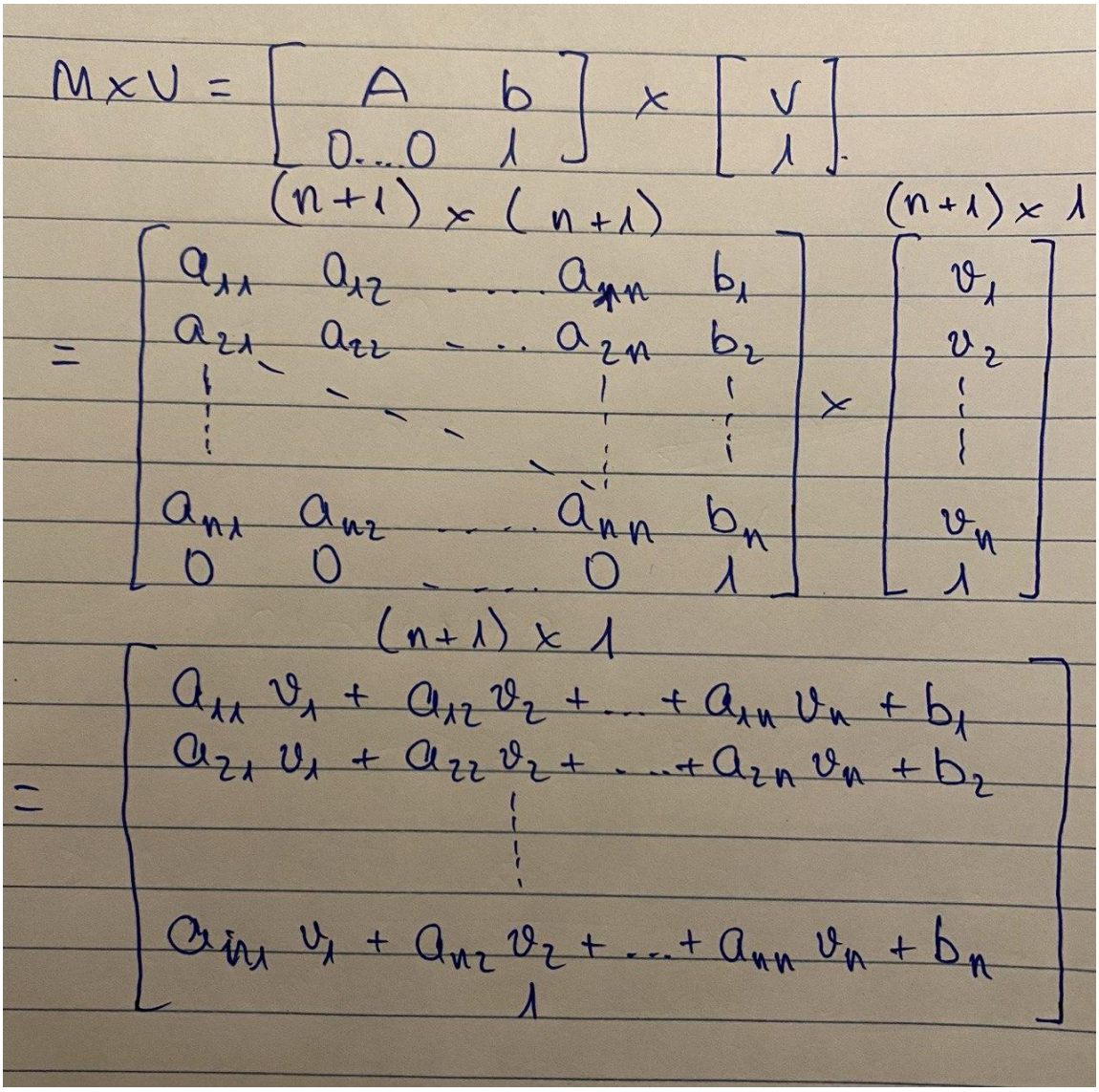
$$V = \begin{bmatrix} v \\ 1 \end{bmatrix}$$

and a combined matrix M:

$$M = \begin{bmatrix} A & \vec{b} \\ 0 \cdots 0 & 1 \end{bmatrix}$$

Describe the result of multiplying the matrix M by the vector V.

Supposed we have n x n square matrix A, vector $\overline{v}$ and vector $\overline{b}$ both have n components:



The final result vector is the vector that contains first n components of M*V's result vector.

g. Construct the combined vector V and matrix M that perform each of the affine transformations in parts (b) - (e) and verify your results.

In all part, we test with $\bar{v} = (1, 3)$, so matrix V is:

$$V = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

i. Part (b)

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

$$M * V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 1 \end{bmatrix}$$

Check by Sage:

Code Cell 1 of 3

```
In [7]  1  V = matrix([[1,3,1]]).T
        2  M = matrix([[1,0,0],[0,1,1],[0,0,1]])
        3  M*V
```

Run Code

```
Out [7]
           [1]
           [4]
           [1]
```

The final result vector is (1, 4), which is the same as the part (b)'s result.

ii. Part (c)

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

$$M * V = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}$$

Check by SageMath:

Code Cell 1 of 3

```
In [8]  1  V = matrix([[1,3,1]]).T
        2  M = matrix([[0,-1,1],[1,0,0],[0,0,1]])
        3  M*V
```

Run Code

```
Out [8]
           [-2]
           [ 1]
           [ 1]
```

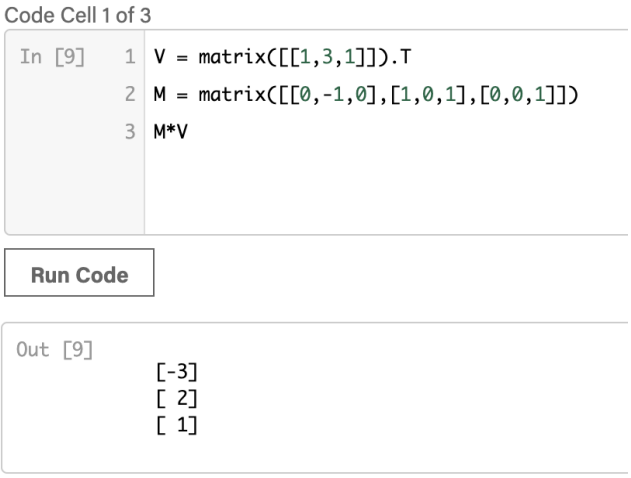
The final result vector is (-2, 1), which is the same as the part (c)'s result.

iii. Part (d)

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

$$M * V = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} -3 \\ 2 \\ 1 \end{bmatrix}$$

Check by SageMath:



The final result vector is (-3, 2), which is the same as the part (d)'s result.

4. A valuable factor

The last unit of this course focused on the concept of eigenvalues and eigenvectors. We saw that factoring a diagonalizable matrix n × n matrix A = SDS^{-1} is a powerful tool for understanding systems. What if A is not diagonalizable? Or not square? Can we do something similar? The answer turns out to be yes! .

a. Consider the 2 × 3 matrix

$$A = \begin{bmatrix} 1 & 1 & 0 \\ -2 & 1 & 3 \end{bmatrix}$$

i. A symmetric matrix is a square matrix that is equal to its transpose:

$$AA^T = \begin{bmatrix} 2 & -1 \\ -1 & 14 \end{bmatrix}$$

To test whether AA^T is symmetric, we need to evaluate the entries along the diagonal, which are equal to one another, which means the $\left(AA^T\right)^T = AA^T$.

Code Cell 1 of 3

```
In [89]  1  A = matrix([[1, 1, 0],[-2,1,3]])
         2  #A1=A*A^T
         3  A1 = A*A.T
         4  A1T = A1.T
         5  A1, A1T
```

Run Code

```
Out [89]
           (
           [ 2 -1]  [ 2 -1]
           [-1 14], [-1 14]
           )
```

A^TA is also symmetric since $A^TA = \left(A^TA\right)^T$.

$$A^TA = \begin{bmatrix} 5 & -1 & -6 \\ -1 & 2 & 3 \\ -6 & 3 & 9 \end{bmatrix}$$

Code Cell 1 of 3

```
In [90]  1  A = matrix([[1, 1, 0],[-2,1,3]])
         2  #A2=A^T*A
         3  A2 = A.T*A
         4  A2T = A2.T
         5  A2, A2T
```

Run Code

```
Out [90]
           (
           [ 5 -1 -6]  [ 5 -1 -6]
           [-1  2  3]  [-1  2  3]
           [-6  3  9], [-6  3  9]
           )
```

ii. Let M be a $m\ x\ n$ matrix, such that:

$$M = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}$$

$$M^T = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$$

$$MM^T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} = \begin{bmatrix} a^2+b^2+c^2 & ad+be+cf \\ ad+be+cf & d^2+e^2+f^2 \end{bmatrix}$$

Here, we can see how the matrix MM^T has elements that are symmetric along the diagonal of the matrix. If we take the elements of the matrix to go to infinity, the same can be observed:

$$\lim_{(i,j) \to (\infty,\infty)} MM^T{}_{ij} = \begin{bmatrix} (a_{11})^2 + \ldots + (a_{ij})^2 & \ldots & a_{11}a_{12} + \ldots + a_{1j-1}a_{1j} \\ \vdots & \vdots & \vdots \\ \cdot & (a_{ij})^2 + \ldots + (a_{ij-1})^2 & \cdot \\ \vdots & \vdots & \vdots \\ a_{11}a_{12} + \ldots + a_{1j-1}a_{1j} & \ldots & (a_{ij})^2 + \ldots + (a_{i-1j-1})^2 \end{bmatrix}$$

This means that MM^T will always be a symmetric matrix, no matter what the dimensions of m and n are, as long as $m \neq n$.

iii. Find U, V, D_1 *and* D_2

With AA^T ***matrix:***

Starting from forming a matrix from the equation $AA^T - \lambda I$:

$$\begin{bmatrix} 2-\lambda & -1 \\ -1 & 14-\lambda \end{bmatrix}$$

Then finding the $det(A - \lambda I)$:

$\lambda^2 - 16\lambda + 27 = 0 \Rightarrow \lambda_1 = 8 - \sqrt{37}$ *and* $\lambda_2 = 8 + \sqrt{37}$

So, D_1 is the diagonal matrix with the value of eigenvalues:

$$D_1 = \begin{bmatrix} 8-\sqrt{37} & 0 \\ 0 & 8+\sqrt{37} \end{bmatrix}$$

When $\lambda_1 = 8 - \sqrt{37}$, the eigenvector is:

$$\vec{u_1} = \begin{bmatrix} -6 + \sqrt{37} \\ 1 \end{bmatrix}$$

When $\lambda_2 = 8 + \sqrt{37}$, the eigenvector is:

$$\vec{u_2} = \begin{bmatrix} -6 - \sqrt{37} \\ 1 \end{bmatrix}$$

Check by SageMath:

Code Cell 1 of 3

```
In [15]  1 A = matrix(QQ, [[1,1,0],[-2, 1, 3]])
         2 B = A*A.T
         3 C = A.T*A
         4 u_1 = vector(RR, [1, -6+sqrt(37)])
         5 u_2 = vector(RR, [1, -6-sqrt(37)])
         6 print(B.eigenvectors_right())
         7 print('u_1 =', u_1)
         8 print('u_2 =', u_2)
```

Run Code

```
Out [15]  [(1.9172374697017817, [(1, 0.08276253029821969?)], 1), (14.082762530298222, [(1,
          -12.082762530298222)], 1)]
          u_1 = (1.0000000000000000, 0.08276253029821933)
          u_2 = (1.0000000000000000, -12.082762530298200)
```

Because U is an orthogonal matrix of eigenvectors of AA^T, we have to normalize eigenvectors before creating U.

Code Cell 1 of 3

```
In [13]
1 A = matrix(QQ, [[1,1,0],[-2, 1, 3]])
2 B = A*A.T
3 C = A.T*A
4 u_1 = vector(RR, [1, -6+sqrt(37)])
5 u_2 = vector(RR, [1, -6-sqrt(37)])
6 u_1_norm = u_1/u_1.norm()
7 u_2_norm = u_2/u_2.norm()
8 print('u_1_norm = ',u_1_norm)
9 print('u_1_norm = ',u_2_norm)
```

Run Code

```
Out [13]  u_1_norm =  (0.996592676029717, 0.0824805315448929)
          u_1_norm =  (0.0824805315448933, -0.996592676029717)
```

Therefore, we have D_1 and U:

```
8 D1 = matrix(RR, [[8-sqrt(37),0],[0,8+sqrt(37)]])
9 U = matrix([u_1_norm,u_2_norm]).T
10 print('D_1 =\n',D1)
11 print('U =\n',U)
```

Run Code

```
Out [24]  D_1 =
           [ 1.91723746970178 0.000000000000000]
          [0.000000000000000  14.0827625302982]
          U =
           [ 0.996592676029717 0.0824805315448933]
          [0.0824805315448929 -0.996592676029717]
```

Check by the equation $AA^T = UD_1U^T$, which is valid.

```
10 print('U*D_1*U^T=\n',U*D1*U.T)
```

Run Code

```
Out [26]    U*D_1*U^T=
             [ 2.00000000000000 -1.00000000000000]
            [-1.00000000000000  14.0000000000000]
```

With A^TA matrix:

When we repeat the process for matrix $A^TA - \lambda I$, we get:

$$\begin{bmatrix} 5-\lambda & -1 & -6 \\ -1 & 2-\lambda & 3 \\ -6 & 3 & 9-\lambda \end{bmatrix}$$

Then we find the $det(A^TA - \lambda I)$:

$$\begin{vmatrix} 0 & -1 & 0 \\ \lambda^2 - 7\lambda + 9 & 2-\lambda & 6\lambda - 9 \\ 9 - 3\lambda & 3 & -\lambda - 9 \end{vmatrix} = (0)(-1)^{1+1}\begin{vmatrix} 2-\lambda & 6\lambda - 9 \\ 3 & -\lambda - 9 \end{vmatrix}$$
$$+ (-1)(-1)^{1+2}\begin{vmatrix} \lambda^2 - 7\lambda + 9 & 6\lambda - 9 \\ 9 - 3\lambda & -\lambda - 9 \end{vmatrix} + (0)(-1)^{1+3}\begin{vmatrix} \lambda^2 - 7\lambda + 9 & 2-\lambda \\ 9 - 3\lambda & 3 \end{vmatrix}$$
$$= \begin{vmatrix} \lambda^2 - 7\lambda + 9 & 6\lambda - 9 \\ 9 - 3\lambda & -\lambda - 9 \end{vmatrix}$$

Then, finding the determinant of a 2x2 matrix will be

$det(A^TA - \lambda I) = ad - bc$:

$$\begin{vmatrix} \lambda^2 - 7\lambda + 9 & 6\lambda - 9 \\ 9 - 3\lambda & -\lambda - 9 \end{vmatrix} = (\lambda^2 - 7\lambda + 9) \cdot (-\lambda - 9) - (6\lambda - 9) \cdot (9 - 3\lambda) = -\lambda^3 + 16\lambda^2 - 27\lambda$$

Our next step will be to solve the equation:

$$\lambda \left(-\lambda^2 + 16\lambda - 27\right) = 0$$

$$\lambda_1 = 8 - \sqrt{37}, \lambda_2 = \sqrt{37} + 8, \lambda_3 = 0$$

So, D_2 is the diagonal matrix with the value of eigenvalues:

$$D_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 8 - \sqrt{37} & 0 \\ 0 & 0 & 8 + \sqrt{37} \end{bmatrix}$$

To find V, we use SageMath to find eigenvectors with respect to each eigenvalue, and then normalize each eigenvector. Finally, we create V:

Code Cell 1 of 3

```
In [45]
A = matrix(QQ, [[1,1,0],[-2, 1, 3]])
C = A.T*A
#Find eigenvectors
v_1 = C.eigenvectors_right()[0][1][0]
v_2 = C.eigenvectors_right()[1][1][0]
v_3 = C.eigenvectors_right()[2][1][0]
#Normalize eigenvector
v_1_norm = v_1/v_1.norm()
v_2_norm = v_2/v_2.norm()
v_3_norm = v_3/v_3.norm()
#Create D2 and V
D2 = matrix(RR, [[0,0,0],[0,8-sqrt(37),0],[0,0,8+sqrt(37)]])
V = matrix([v_1_norm,v_2_norm,v_3_norm]).T
print('D_2 =\n', D2)
print('D_2 =\n', D2)
```

```
D_2 =
 [0.000000000000000 0.000000000000000 0.000000000000000]
[0.000000000000000  1.91723746970178 0.000000000000000]
[0.000000000000000 0.000000000000000  14.0827625302982]
V =
 [         1/3*sqrt(3)  0.6006106597046816?  0.5531125582698098?]
[        -1/3*sqrt(3)   0.779314856466197? -0.2435878099530801?]
[         1/3*sqrt(3)  0.1787041967615151? -0.7967003682228899?]
```

Check by the equation $A^T A = V D_2 V^T$, which is valid.

```
14 V*D2*V.T
```

Run Code

```
Out [49]
          [  5.00000000000000 -0.999999999999999  -6.00000000000000]
          [-0.999999999999999   2.00000000000000   3.00000000000000]
          [ -6.00000000000000   3.00000000000000   9.00000000000000]
```

iv. (Optional) We skip this part

v. This suggests an approach. Try factoring with U and V instead. Compute $\Sigma = U^T AV$. What form does Σ take? How is Σ related to D_1 and D_2 above?

```
21 Sigma = U.T*A*V
22 Sigma
```

Run Code

```
Out [55]
          [-(2.77555756156289e-17)*sqrt(3)              1.38464344497122          1.22124532708767e-15]
          [-(1.11022302462516e-16)*sqrt(3)             0.000000000000000             3.75270069820366]
```

Sigma is a 2x3 matrix. We can see that Sigma only has 2 entries that are non-zero, which are: 1.384643445 and 3.752700698. D_1 *and* D_2 also have only 2 entries that are non-zero, which are $8 - \sqrt{37}$ *and* $8 + \sqrt{37}$. By calculation, we see that $\sqrt{8 - \sqrt{37}} = 1.384643445$, which is one non-zero entry of Sigma; $\sqrt{8 + \sqrt{37}} = 3.752700698$, which is one non-zero entry of Sigma.

This is an example of a *singular valueable decomposition (SVD)*: Any $m \times n$ matrix A can be factored as $A = U\Sigma V^T$ where the columns of U are the eigenvectors of AA^T and the columns of V are the eigenvectors of A^TA. The *singular values* on the diagonal of Σ are the square roots of the nonzero eigenvalues of AA^T and A^TA.

b. Find singular value compositions for $B = \begin{bmatrix} 4 & 0 & 3 \end{bmatrix}$ and $C = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$.

Code Cell 1 of 3

In [80]:
```
B = matrix([[4, 0, 3]])
#B1=B*B^T
B1 = B*B.T
#B2=B^T*B
B2 = B.T*B
#Find and normalize eigenvectors of B1 and create U
u_1 = B1.eigenvectors_right()[0][1][0]
u_1_norm = u_1/u_1.norm()
U = matrix([u_1_norm]).T
#Find and normalize eigenvectors of B2 and create V
v_1 = B2.eigenvectors_right()[0][1][0]
v_2 = B2.eigenvectors_right()[1][1][0]
v_3 = B2.eigenvectors_right()[1][1][1]
v_1_norm = v_1/v_1.norm()
v_2_norm = v_2/v_2.norm()
v_3_norm = v_3/v_3.norm()
V = matrix([v_1_norm,v_2_norm,v_3_norm]).T
#Find Sigma_B
Sigma_B = U.T*B*V
Sigma_B
```

```
Sigma_B = [5 0 0]
```

Code Cell 1 of 3

In [88]:

```
C = matrix([[0, 1, 0],[1,0,0]])
#B1=B*B^T
C1 = C*C.T
#B2=B^T*B
C2 = C.T*C
#Find and normalize eigenvectors of B1 and create U
u_1 = C1.eigenvectors_right()[0][1][0]
u_2 = C1.eigenvectors_right()[0][1][1]
u_1_norm = u_1/u_1.norm()
u_2_norm = u_2/u_2.norm()
U = matrix([u_1_norm,u_2_norm]).T
#Find and normalize eigenvectors of B2 and create V
v_1 = C2.eigenvectors_right()[0][1][0]
v_2 = C2.eigenvectors_right()[1][1][0]
v_3 = C2.eigenvectors_right()[1][1][1]
v_1_norm = v_1/v_1.norm()
v_2_norm = v_2/v_2.norm()
v_3_norm = v_3/v_3.norm()
V = matrix([v_1_norm,v_2_norm,v_3_norm]).T
#Find Sigma_B
Sigma_C = U.T*C*V
print('Sigma_C =\n',Sigma_C)
```

```
Sigma_C =
 [0 0 1]
[0 1 0]
```

c. SVD has many applications. In fact we used SVD when we derived least-squares by finding the *pseudoinverse* $A^+ = (A^T A)^{-1} A^T$. Research other applications of SVD. Write a brief synopsis of the application you find most intriguing. (300 words)

Besides computing the pseudoinverse of a matrix $A^+ = \left(A^T A\right)^{-1} A^T$, we can use SVD to compress images, after we decompose them to matrices with values from 1 to 255 (Similar to what we did in the Change of Basis problem in Deep Dive 2).We can store a picture in a network. Each picture comprises a bunch of pixels which are the structure squares of that picture. Every pixel addresses the shading or on the other hand the power of light in a particular area in the picture. In a grayscale picture with PNG design, similarly to what the previous Deep Dive was about - image compression in the grayscale - every pixel has a worth among O and 1, where zero compares to dark and 1 relates to white. So a grayscale picture with mXn pixels can be put away in a mXn matrix grid. Here we use the read capacity to stack a greyscale and we use SVD to disintegrate the network, utilizing the initial solitary qualities.

Reflection

1. Give two examples of applying HCs to solve a specific problem. How did you apply to the HC? Was the application successful? If yes, why? If not, how could you improve your application?

   #heuristics: We applied this HC to the problem with finding vector spaces in question 2, where we used the tallying heuristic to cut down the amount of work, knowing that some of the operations under multiplication and addition are associative. To critique this

application, there is a chance that this heuristic might lead to saving time but expunging accuracy, which makes its use risky.

#constraints: We used this HC to identify the constraint of the matrix multiplication in question 4, where some matrices in the equation expression did not satisfy the condition of the number of rows equaling the number of columns. We applied constraint satisfaction to this constraint by adding a row of zeros for the calculator to be possible.

2. Give two examples of applying HCs to complete this assignment as a group. How did you apply to the HC? Was the application successful? If yes, why? If not, how could you improve your application?

   #differences: In our first meeting, we spent 20 minutes talking about the CS113 course, Linear Algebra in general, and which parts we like the most in Linear Algebra, especially the topics covered during the course. With Duc, he loves doing linear systems (e.g., playing with different matrices, creating row operations, finding solutions) and working with problems that are geometrically driven. He knows that he is not very good at Transformation and Vector. Based on those skills and interests, he is in charge of the first problem related to linear systems, concept exploration, and the third problem related to transformation because he also wants to challenge himself with transformation. With Tino, he loves doing complicated problems, working with vectors, eigenstuff, and new theorems in Math because he wants to follow Math concentration in Computer Science college. Therefore, he is in charge of the second problem related to vectors and the last problem related to eigenstuff and new theorems (e.g., Singular Valuable Decomposition).

About working style, both of us are comfortable with solving problems on paper before presenting them on the document. Therefore, our working process was (1) Solving the problem on paper; (2) Sharing and revising solutions together; (3) Going to Office Hours of Prof. Levitt to ask questions; (4) Finalizing and presenting on the document. By clearly dividing work based on members' interests and good skills, and creating a transparent working process, we can obtain the differences and utilize them to get the best results in this assignment.

#purpose: Before starting working on this assignment, in the conversation at the first meeting, we mentioned what goals we want to achieve via this teamwork project. Fortunately, for both of us, our highest goal is to do great in this assignment and get an A from this course. On the other hand, we also want to challenge ourselves in different ways: With Duc, he wants to challenge himself with transformation, while Tino wants to do more complicated problems in Linear Algebra. Based on those purposes, we identify and follow different actions to pursue goals intentionally. For example, to challenge himself in Transformation, Duc volunteered to take the third problem and went to Office Hours many times to ask questions and receive help from the professor to ensure that everything was going well. With Tino, his computer was broken during the final week, which made him unable to work on the computer for many days. However, because of our purposes in this course, we actively changed to working completely on paper to keep up with the progress and deadline with the constraint of time and technology.