

# PROYECTO ÁLGEBRA LINEAL

 PYTHON  NUMPY  MATPLOTLIB



 GITHUB  JGARNIGAR

## Descripción

- Los datos fueron encriptados multiplicando una matriz 6x6 por una matriz 6x1. El resultado es una matriz 6x1 con los datos encriptados. Estos datos nuevos ocultan una imagen. La nueva matriz 6x1 nos da los valores (x,y).
- Para desencriptar estos datos, obtendremos la matriz inversa de la matriz 6x6 y multiplicaremos por la 6x1 con los datos encriptados.
- Finalmente usaremos matplotlib con el fin de visualizar los datos desencriptados.

## Índice

1.  [Descripción](#)
2.   [Instalación y requisitos](#)
3.  [Objetivo](#)
4.  [Estructura](#)
5.  [Operaciones matemáticas](#)
6.  [Métodos de Numpy](#)
7.  [Datos](#)
8.   [Ecuaciones](#)
  -  [1 Primera ecuación](#)
  -  [2 Segunda ecuación](#)
  -  [3 Tercera ecuación](#)
  -  [4 Cuarta ecuación](#)
  -  [Resultado de las ecuaciones](#)
9.  [Clases](#)
  -  [Librerías](#)
  -  [Cifrado\(\)](#)
    - [Funcionamiento de la clase Cifrado\(\)](#)
  -  [Crear\\_Array\(\)](#)
    - [Funcionamiento de la clase Crear\\_Array\(\)](#)
  -  [Ecuacion\(\)](#)
    - [Funcionamiento de la clase Ecuacion\(\)](#)
  -  [Desempaquetar\\_Array\(\)](#)

- [Funcionamiento de la clase Desempaquetar\\_Array\(\)](#)
- ◆ [Rotacion\(\)](#)
  - [Funcionamiento de la clase Rotacion\(\)](#)
- ◆ [Traslacion\(\)](#)
  - [Funcionamiento de la clase Traslacion\(\)](#)
- ◆ [GuardarDatos\(\)](#)
  - [Funcionamiento de la clase GuardarDatos\(\)](#)
- ◆ [App\(\)](#)
  - [Funcionamiento de la clase App\(\)](#)
- ◆ [Graficar\(\)](#)
  - [Funcionamiento de la clase Graficar\(\)](#)

10.  [Instancias](#)

11.  [Comprobación de datos](#)

- [Matriz inversa](#)
- [Desencriptar datos](#)
- [Rotación](#)
- [Traslación](#)

12.  \*[Desencriptar datos](#)\*

13.   [Gráficas - Datos Resueltos!](#)

- [Gráfico desencriptado](#)
- [Gráfico Matriz Rotación](#)
- [Gráfico Matriz Rotación y Traslado](#)

14.  [Guardar Datos](#)

15.  [Autor](#)

16.  [Anexo A:](#)

17.  [Anexo B:](#)

18.  [Anexo C:](#)

## Instalación y requisitos.

Utilizaremos **Python** 3.10 o superior, **Numpy** y **Matplotlib**. Para asegurarnos que todo funcione, por favor cree un entorno virtual.

**Instalar Python 3.10 o superior**

[Descargar Python](#)

 **Clone el repositorio:**

```
git clone https://github.com/jgarnigar/Proyecto-Algebra-Lineal.git
```

 **Crear el entorno virtual**

```
python -m venv venv
source venv/bin/activate      # Para Linux/Mac
venv\Scripts\activate         # Para Windows
```

 **Instale los requerimientos**

```
pip install -r requirements.txt
```

### Ejecución del programa

Una vez tengamos el repositorio clonado y todos los requisitos instalados, ejecute este código desde la carpeta principal:

```
`python main.py`
```

 **Nota:** asegúrese de ejecutar este comando desde la carpeta raíz del proyecto!

### Objetivo

- El objetivo es crear un algoritmo a través de Python y Numpy para poder desencriptar estos datos de forma automatizada y por último mostrar estos datos con Matplotlib.

### Estructura

```
└── Proyecto Álgebra Lineal
    ├── datos/ → Datos encriptados y desencriptados
    ├── funciones/ → Funciones lógicas
    ├── main.py → Archivo principal de ejecución
    ├── requirements.txt
    └── README.md
```

### Operaciones matemáticas

Estos son los temas utilizados para resolver el proyecto.

- ◆ Multiplicación de matrices.
- ◆ Matrices inversas.
- ◆ Determinantes.
- ◆ Sistema de ecuaciones.
- ◆ Producto escalar.
- ◆ Producto vectorial.

### Métodos de Numpy.

Expicaremos brevemente los métodos utilizados en Numpy para poder trabajar con nuestras clases.

- ◆ **`np.linalg.inv()`** => Inversa de una matriz.
- ◆ **`matriz_a @ matriz_b`** => Multiplicación de matrices.
- ◆ **`np.array()`** => Creación de arrays.
- ◆ **`np.linalg.solve(primer_termino, segundo_termino)`** => Resolución de sistema de ecuaciones.
- ◆ **`np.ravel()`** => convierte un array multidimensional en uno unidimensional
- ◆ **`np.deg2rad()`** => Convierte un ángulo en grados a radianes.
- ◆ **`np.cos()`** => Función Coseno.
- ◆ **`np.sen()`** => Función Seno.

◆ `np.reshape()` => Función para transformar la dimensión de arrays. Ejemplo, pasar de 1x6 a 6x1.

- Si el array es un 1x6, usamos `np.reshape(-1, 1)`: -1 significa que toma el máximo valor que en este caso es 6 que serán las filas, y : 1, que será el número total de columnas. Ahora el array es de 6x1.

## Datos

Todos los datos fueron almacenados dentro de la carpeta datos.

 Proyecto-algebra-linea

|-  datos > datos encriptados.txt  
| |- valores desencriptados.txt  
| |- valores rotados.txt  
| |- valores trasladados.txt

## Ecuaciones

Antes de empezar a desencriptar los datos, tenemos que tener todos los datos de nuestra matriz 6x6 la cual usaremos para resolver el proyecto.

Tenemos la siguiente matriz la cual tiene incógnitas que hay que encontrar:

$$A = \begin{matrix} a & 3 & b & 5 & c & 8 \\ 6 & d & 0 & e & 7 & f \\ g & 8 & h & 1 & i & 7 \\ 11 & j & 8 & k & 12 & m \\ n & -1 & p & -5 & r & 3 \\ 4 & t & 2 & w & 9 & z \end{matrix}$$

### ✓ 1 Primera Ecuación

Resolvemos nuestra ecuación de 10x10

$$6a + 7b - c - 12d + 14e + 5f - 12g - 3h + 9i - 5j = 48$$

$$2a - 15b + 8c + 6d - 7e + 9f - 9g + 5h - 8i - 6j = 64$$

$$-25a + 10b - 9d - 12e + 14f - 6g + 8h - 13i + 4j = -132$$

$$6a - 3b + 5c - 16d + e + 9f - 7g + 3h - 4i + 5j = -75$$

$$8a - 9b + 6c - d - e - 5f + 7g + 3i + 2j = -16$$

$$-5a + 6b + 9c - 2d + 10e - 14f + 3g + 5h - 12i + 6j = -408$$

$$-4a + 5b + 8c - 2d + 9e - 8f + 4g + h - 2j = -203$$

$$a + b + 2c - 3d + 4e - f - 4g - 7h + 2i - 4j = 59$$

$$10a + 5b - 9c + 6d + e + f + 7g - 8h + 3i + 11j = 126$$

$$-2a + 4b + 3c + 5d - 10e - f + 3g - h - 7i + j = 2$$

```

# Pasamos todos los datos del primer término a un array.
primer_termino = np.array([
    [6, 7, -1, -12, 14, 5, -12, -3, 9, -5],
    [2, -15, 8, 6, -7, 9, -9, 5, -8, -6],
    [-25, 10, 0, -9, -12, 14, -6, 8, -13, 4],
    [6, -3, 5, -16, 1, 9, -7, 3, -4, 5],
    [8, -9, 6, -1, -1, -5, 7, 0, 3, 2,],
    [-5, 6, 9, -2, 10, -14, 3, 5, -12, 6],
    [-4, 5, 8, -2, 9, -8, 4, 1, 0, -2],
    [1, 1, 2, -3, 4, -1, -4, -7, 2, -4],
    [10, 5, -9, 6, 1, 1, 7, -8, 3, 11],
    [-2, 4, 3, 5, -10, -1, 3, -1, -7, 1]
], dtype=int)

#Ahora pasamos los datos del segundo término a otro array.

segundo_termino = np.array([48, 64, -132, -75, -16, -408, -203, 59, 126, 2], dtype=int)

#Tenemos una clase Ecuacion() la cual hace uso de np.linalg.solve para resolver el sistema.
ecuacion = Ecuacion()

matriz = ecuacion.resolver(primer_termino, segundo_termino)

# la matriz da los resultados tales que.
# (2,-2,-3,5,-9,4,-7,-10,11,0)

print(matriz)

```

[ 2. -2. -3. 5. -9. 4. -7. -10. 11. 0.]

- ⚠ La clase `Ecuacion()` se verá más adelante.

💡 Para las incógnitas  $k, m, n, p, r, t, w, z$  debe resolver las siguientes operaciones entre vectores:

## 2 Segunda Ecuación

Se tienen los vectores  $U^\rightarrow = (3, 6, 7)$  y  $V^\rightarrow = (k, m, n)$ . El resultado de operar  $2U^\rightarrow \times 3V^\rightarrow$  es igual a  $612\hat{i} + 156\hat{j} - 396\hat{k}$  y el producto  $U^\rightarrow \cdot V^\rightarrow = 58$

Segunda Ecuación ecuación para encontrar k, m, n.

$$2UX3V = (612, 156, -396)$$

$$\frac{2UX3V}{6} = (102, 26, -66)$$

Despejamos k y m para poder dejar n como una variables independiente.

$$6n - 7m = 102$$

$$7k - 3n = 26 \quad \Rightarrow \quad m = \frac{6n - 102}{7}$$

$$3m - 6k = -66 \quad \Rightarrow \quad k = \frac{3n + 26}{7}$$

Ahora que (n) es una variable independiente, cambiamos a la ecuación de producto escalar despejando n, para esto reemplazamos k y m.

$$U * V = 58$$

Multiplicamos toda la expresión por 7, así removemos las divisiones.

$$3\left(\frac{3n+26}{7}\right) + 6\left(\frac{6n-102}{7}\right) + 7n = 58$$

$$9n + 78 + 36n - 612 + 49n = 406$$

Despejamos

$$94n = 490$$

$$n = 10$$

Conociendo el valor de n, podemos encontrar k y m con las ecuaciones anteriores:

$$m = \frac{6n - 102}{7}$$

$$k = \frac{3n + 26}{7}$$

Empecemos con (m) sustituyendo n por 10:

$$m = \frac{6(10) - 102}{7} \rightarrow m = -6$$

Ahora encontramos (k) sustituyendo n por 10:

$$k = \frac{3(10) + 26}{7} \rightarrow k = 8$$

Para finalizar tenemos los valores:

$$k = 8, m = -6, n = 10$$

### 3 Tercera Ecuación

Se tienen los vectores  $U^* = (6, p, r)$  y  $V^* = (t, 8, 9)$ . El resultado de operar  $3U^* - 10V^*$  es igual a  $-42\hat{i} - 68\hat{j} - 126\hat{k}$

$$U = (6, p, r) \quad V = (t, 8, 9)$$

$$3U - 10V = (-42, -68, -126)$$

$$3U = (18, 3p, 3r) \quad 10V = (10t, 80, 90)$$

$$(t) \quad 18 - 10t = -42$$

$$-10t = -60 \quad (t) = 6$$

$$(p) \quad 3p - 80 = -68$$

$$3p = 12 \quad (p) = 4$$

$$(r) \quad 3r - 90 = -126$$

$$3r = -36 \quad (r) = -12$$

$$(t) = 6, \quad (p) = 4, \quad (r) = -12$$


---

### 4 Cuarta Ecuación

Se tienen los vectores:

$$U = \left(\frac{-1}{2}, \frac{\sqrt{38}}{2}, \frac{5}{2}\right)$$

$$V = (11, \sqrt{342}, -21)$$

La magnitud de  $|U^*|$  es igual a  $w$ . Además, el producto escalar  $U^* \cdot V^*$  es igual a  $z$ .

**Buscamos solamente la variable w primero:**

$$U = \left(\frac{-1}{2}, \frac{\sqrt{38}}{2}, \frac{5}{2}\right)$$

$$|u| = -w$$

$$|U| = \sqrt{\left(\frac{1}{2}\right)^2 + \left(\frac{\sqrt{32}}{2}\right)^2 + \left(\frac{5}{2}\right)^2}$$

$$|U| = \sqrt{\frac{1}{4} + \frac{38}{4} + \frac{25}{4}}$$

$$|U| = \sqrt{\frac{64}{4}}$$

$$|U| = 4, \quad (W) = -4$$


---

**Ahora buscamos la variable z:**

El producto escalar  $U \cdot V$  es igual a  $z$ .

$$U * V = z$$

$$U = \left( \frac{-1}{2}, \frac{\sqrt{38}}{2}, \frac{5}{2} \right)$$

$$V = (11, \sqrt{342}, -21)$$

$$(U * V) = \left( \frac{-1}{2} * \frac{11}{1} \right) + \left( \frac{\sqrt{38}}{2} * \frac{3\sqrt{38}}{1} \right) + \left( \frac{5}{2} * \frac{-21}{1} \right)$$

$$(U * V) = \left( \frac{-11}{2} \right) + \left( \frac{3\sqrt{38}^2}{2} \right) + \left( \frac{-105}{2} \right)$$

$$(U * V) = \left( \frac{-11}{2} - \frac{-105}{2} \right) + \left( \frac{57}{1} \right)$$

$$(U * V) = -58 + 57$$

$$(U * V) = -1$$

$$(Z) = -1$$


---

## ✓ Resultado de las ecuaciones.

En total tenemos las siguientes variables encontradas con la ecuación también realizada anteriormente en la clase Ecuación.

$$\begin{aligned} a &= 2, & b &= -2, & c &= -3, & d &= 5, & e &= -9, & f &= 4, & g &= -7, & h &= -10, \\ i &= 11, & j &= 0, & k &= 8, & m &= -6, & n &= 10, & w &= -4, & z &= -1, \\ p &= 4, & t &= 6, & r &= -12 \end{aligned}$$

Al final la matriz para codificar los vectores es:

```
[ 2,  3, -2,  5, -3,  8]
[ 6,  5,  0, -9,  7,  4]
[-7,  8,-10,  1, 11,  7]
[11,  0,  8,  8, 12, -6]
[10, -1,  4, -5, -12,  3]
[ 4,  6,  2, -4,  9, -1]
```

#la matriz codificación sería:

```
matriz_codificacion = np.array([
    [ 2,  3, -2,  5, -3,  8],
    [ 6,  5,  0, -9,  7,  4],
    [-7,  8,-10,  1, 11,  7],
    [11,  0,  8,  8, 12, -6],
    [10, -1,  4, -5, -12,  3],
    [ 4,  6,  2, -4,  9, -1]
], dtype=int)
```

## ✗ Clases

Ahora que ya resolvimos nuestras ecuaciones y obtuvimos nuestra matriz completa, ya podemos comenzar a programar para poder desencriptar nuestros datos

## ✗ Librerías

Primero importamos las librerías que usaremos a lo largo de nuestro proyecto

```
import numpy as np
import matplotlib.pyplot as plt
```

## ✗ Cifrado()

Creamos nuestra clase Cifrado la cual Cifra, descifra datos y podemos obtener una matriz inversa para depurar.

### Funcionamiento de la clase Cifrado()

- ⚡ Primero tenemos la función `inversa()` la cual nos devuelve la inversa de la matriz ingresada.
- ⚡ Tenemos la función `cifrar()` la cual obtiene 2 matrices y las multiplica y nos devuelve la matriz multiplicada (cifrada).
- ⚡ Por último tenemos la función `descifrar()` la cual obtiene 2 matrices. La primer matriz es la matriz usada para descifrar y la segunda es la que tiene los datos cifrados. La función crear la matriz inversa y la multiplica por la matriz con los datos cifrados y por último devuelve una matriz con los datos descifrados.

 Nota: La función `inversa()` es utilizada para depurar, por eso la función `descifrar()` también resuelve la matriz inversa por sí sola.

```
class Cifrado():

    def inversa(self, matriz_codificar):
        self.matriz_codificar = matriz_codificar

        matriz_inversa = np.linalg.inv(self.matriz_codificar)

        return matriz_inversa

    def cifrar(self, matriz, matriz_codificar):
        self.matriz = matriz
        self.matriz_codificar = matriz_codificar
        matriz_cifrada = self.matriz_codificar @ self.matriz

        return matriz_cifrada

    def descifrar(self, matriz_codificacion, matriz_resolver):
        self.matriz_codificacion = matriz_codificacion
        self.matriz_resolver = matriz_resolver

        matriz_inversa = np.linalg.inv(self.matriz_codificacion)

        matriz_decifrada = matriz_inversa @ self.matriz_resolver

        return matriz_decifrada
```

## ▼ ◆ Crear\_Array()

La clase `crear_array()` fue creada con el objetivo de obtener un arreglo con valores alternados de  $(x, y)$  en un formato  $(x_1, y_1, x_2, y_2\dots)$  y los separará en dos arreglos independientes con las coordenadas  $(x)$  y  $(y)$ .

### Funcionamiento de la clase Crear\_Array()

#### Método `valores_x()`

- Recorre el arreglo original y toma los valores impares (1, 3, 5...). Cada uno de estos valores corresponde a  $(x)$ .
- Finalmente nos devuelve un arreglo con los valores de  $(x)$ .

#### Método `valores_y()`

- Recorre el arreglo original también, pero ahora solo toma los valores pares (2, 4, 6...) y cada uno de estos valores corresponderá a  $(y)$ .
- Agregamos todos esos valores en un nuevo arreglo y lo devolvemos.

```
class crear_array():

    def valores_x(self, array):
        self.array = array
        puntos_x = []
        contador = 1
        for x in self.array:
            if contador % 2 != 0:
                puntos_x.append(x)

            contador += 1

        return puntos_x

    def valores_y(self, array):
        self.array = array
        puntos_y = []
        contador = 1

        for y in self.array:
            if contador % 2 == 0:
                puntos_y.append(y)

            contador +=1

        return puntos_y
```

## ▼ ◆ Ecuacion()

Creamos una clase llamada `ecuación` con la finalidad de resolver la ecuación  $10 \times 10$  brindada en las instrucciones del proyecto.

### Funcionamiento de la clase Ecuacion()

- La clase nos pide los valores separados del primero término y segundo término de la ecuación.
- Hacemos uso de `np.linalg.solve()` para poder obtener los valores de las incógnitas.
- Por último nos devuelve un arreglo con las incógnitas descubiertas.

```
class Ecuacion():

    def resolver(self, primer_termino, segundo_termino):
        self.primer_termino = np.array(primer_termino)
        self.segundo_termino = np.array(segundo_termino)
```

```

solucion = np.linalg.solve(self.primer_termino, self.segundo_termino)
return solucion

```

## ❖ Desempaquetar()

Esta clase se encarga de transformar los arreglos de coordenadas  $(x, y)$  en listas unidimensionales. Debido a que la clase `App()` devuelve los datos en un formato anidado como:

```
[[array([x1, x2, x3])], [array([y1, y2, y3])]]
```

Sin embargo, para aplicar operaciones como *Traslación*, *Rotación* o para graficar los puntos usando `zip(x, y)`, es necesario trabajar con estructuras unidimensionales como:

```
[x1, x2, x3] y [y1, y2, y3]
```

## Funcionamiento de la clase Desempaquetar()

- 1 Pedimos los arreglos separados de  $(x, y)$ .
- 2 Creamos un for para cada valor de  $(x, y)$ .
- 3 Aplanamos los arreglos para dejarlos unidimensionales.
- 4 Agregamos los datos a un nuevo arreglo.
- 5 Devolvemos los arreglos ya aplanados.

```

class Desempaquetar_Array():
    def desempaquetar(self, valores_x, valores_y):
        self.valores_x = valores_x
        self.valores_y = valores_y

        new_values_x = []
        new_values_y = []

        for nueva_lista in self.valores_x:
            for x in np.ravel(nueva_lista):
                new_values_x.append(x)

        for nueva_lista in self.valores_y:
            for y in np.ravel(nueva_lista):
                new_values_y.append(y)

        return new_values_x, new_values_y

```

## ❖ Rotacion()

La clase *Rotacion* aplica una transformación geométrica de rotación a un conjunto de puntos en el plano cartesiano.

Recibe dos listas de coordenadas  $(x, y)$ , y devuelve las nuevas coordenadas resultantes después de ser rotadas un ángulo determinado.

La matriz de rotación en 2D está definida como:

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Y la transformación de un punto  $(x, y)$  se obtiene mediante:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R(\theta) \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

## ✓ Funcionamiento de la clase Rotacion()

- 1 Pedimos los valores de  $(x, y)$  es arreglos separados.
- 2 El ángulo (135 grados para nuestro ejemplo) es convertido a radianes.
- 3 Iteramos los valores de  $(x, y)$  y los agregamos a una matriz de 2x1
- 4 Multiplicamos la matriz de rotación por la matriz 2x1
- 5 Obtenemos los nuevos valores de  $(x, y)$  ya rotados y los agregamos a un nuevo arreglo.
- 6 Por último la función nos regresa dos arreglos con los datos  $(x, y)$  ya rotados.

```
class Rotacion():
    def rotar_matriz(self, valores_x, valores_y):
        self.valores_x = valores_x
        self.valores_y = valores_y
        angulo = 135
        angulo_rad = np.deg2rad(angulo)

        rotacion_x = []
        rotacion_y = []

        matriz_rotacion = np.array([
            [np.cos(angulo_rad), -np.sin(angulo_rad)],
            [np.sin(angulo_rad), np.cos(angulo_rad)]
        ])

        for x, y in zip(self.valores_x, self.valores_y):
            new_array = np.array([x, y]).reshape(2,1)
            array_rotado = matriz_rotacion @ new_array
            rotacion_x.append(array_rotado[0,0])
            rotacion_y.append(array_rotado[1,0])

    return rotacion_x, rotacion_y
```

## ✓ ◆ Traslacion()

La clase Traslacion aplica una transformación geométrica de traslación a un conjunto de puntos en el plano cartesiano.

Recibe coordenadas  $(x, y)$  y las desplaza una distancia  $a$  en el eje x y  $b$  en el eje y.

La matriz de traslación en coordenadas homogéneas se define como:

$$T(a, b) = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix}$$

Aplicada a un punto  $(x, y)$ , la transformación se realiza mediante el producto matricial:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = T(a, b) \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

## Funcionamiento de la clase Traslacion()

- 1 Pedimos los valores de  $(x, y)$  y  $(a, b)$  es arreglos separados.
- 2 Iteramos los valores de  $(x, y)$  y los agregamos a una matriz de 2x1.
- 3 Multiplicamos la matriz de traslación por la matriz 2x1.

4 Obtenemos los nuevos valores de  $(x, y)$  ya trasladados y los agregamos a un nuevo arreglo.

5 Por último la función nos regresa dos arreglos con los datos  $(x, y)$  ya trasladados.

```
class Traslacion():
    def trasladar_matriz(self, x, y, a, b):
        self.x = x
        self.y = y
        # a = movimiento en x
        self.a = a
        # b = movimiento en y
        self.b = b

        valores_x = []
        valores_y = []

        matriz_traslacion = np.array([
            [1, 0, self.a],
            [0, 1, self.b],
            [0, 0, 1]
        ])

        for punto_x, punto_y in zip(self.x, self.y):
            punto = np.array([punto_x, punto_y, 1]).reshape(3,1)

            resultado = matriz_traslacion @ punto

            valores_x.append(resultado[0,0])
            valores_y.append(resultado[1,0])

    return valores_x, valores_y
```

## ◆ GuardarDatos()

Esta clase permite almacenar las coordenadas generadas durante el proceso (rotación, translación, etc.) en un archivo de texto externo.

Los valores se guardan en formato de pares ordenados:

```
(x1, y1)
(x2, y2)
(x3, y3)
```

## Funcionamiento de la clase GuardarDatos()

- 1 Recibe los arreglos  $(x, y)$  y el nombre del archivo.
- 2 Creamos o reemplazamos los datos del nombre del archivo .txt
- 3 Recorremos los arreglos usando `zip(x, y)`.
- 4 Guardamos los pares  $(x, y)$  como una línea en nuestro archivo.
- 5 Cerramos el documento con `with open()`.

```
class GuardarDatos():
    def save(self, valores_x, valores_y, nombre_archivo):
        self.valores_x = valores_x
        self.valores_y = valores_y
        self.nombre_archivo = nombre_archivo

        with open(self.nombre_archivo, "w") as f:
            for x, y in zip(valores_x, valores_y):
                f.write(f"{{x}}, {{y}}\n")
```

## ❖ App()

La clase `App` se encarga de abrir un archivo que contiene datos encriptados o sin encriptar, procesarlos y ejecutar la lógica correspondiente para **cifrar** o **descifrar** los valores.

Los datos leídos se convierten en una matriz y posteriormente se desempaquetan para obtener los valores en los ejes `(x, y)`.

### Funcionamiento de la clase App()

- 1 Lee un archivo línea por línea.
- 2 Transforma los datos numéricos en una matriz columna.
- 3 Aplica el método de cifrado o descifrado según el parámetro `condicional`.
- 4 Finalmente, separa los valores resultantes en arreglos independientes para `(x, y)`.

```
class App():  
    def abrir_documento(self, matriz, archivo, condicional):  
        self.archivo = archivo  
        self.matriz = matriz  
        self.condicional = condicional  
  
        array = []  
        array_valores_x = []  
        array_valores_y = []  
  
        resultado = []  
  
        matriz_original = np.array(self.matriz)  
  
        with open(self.archivo, "r") as f:  
            for line in f:  
  
                valores_Array = [float(x) for x in line.strip().split()]  
  
                array = np.array(valores_Array).reshape(-1, 1)  
  
                cir = Cifrado()  
                create = crear_array()  
  
                if self.condicional == "cifrar":  
                    resultado = cir.cifrar(array, matriz_original)  
  
                elif self.condicional == "descifrar":  
                    resultado = cir.descifrar(matriz_original, array)  
  
                valores_x = create.valores_x(resultado)  
                valores_y = create.valores_y(resultado)  
  
                array_valores_x.append(valores_x)  
                array_valores_y.append(valores_y)  
  
        return array_valores_x, array_valores_y
```

## ❖ Graficar()

Esta clase nos permite graficar todos los puntos `(x, y)` con los datos Desencriptados, Rotados y Trasladados.

### Funcionamiento de la clase Graficar()

- 1 Obtiene dos arreglos con los valores `(x, y)`.

2 Utilizamos `Matplotlib` para graficar cada uno de estos datos.

3 Con el método `plt.show()` mostramos el gráfico.

```
class Graficar():
    def graficadora(self, x, y):
        self.x = x
        self.y = y
        plt.scatter(self.x, self.y, color='blue', marker='o', label='Puntos (x, y)')
        # Personalizar el gráfico
        plt.title("Gráfico de puntos")
        plt.xlabel("Eje X")
        plt.ylabel("Eje Y")
        plt.legend()
        plt.grid(True)
        plt.show()
```

## ▼ Instancias

Inicializamos nuestras *instancias de clases*:

```
aplicacion = App()
rotar = Rotacion()
desempaquetador = Desempaquetar_Array()
graficar = Graficar()
traslacion = Traslacion()
guardar_datos = GuardarDatos()
cir = Cifrado()
create = crear_array()
```

## ✓ Comprobación de Datos

### Matriz Inversa

Buscamos comprobar que la clase *Cifrado* con el método inversa funciona para poder obtener la inversa de nuestra matriz

```
inversa = cir.inversa(matriz_codificacion)
print(f"La matriz inversa es: \n{inversa}")

La matriz inversa es:
[[ -0.13562749 -0.03955583  0.16534674  0.097423   0.19435366 -0.08729311]
 [ 0.02119736 -0.18428661  0.04575739 -0.04399749  0.08185104  0.26227216]
 [ 0.28413995  0.12090928 -0.33697884 -0.11187932 -0.30448355  0.15573015]
 [ 0.03036874 -0.074251   0.03071444  0.03456945  0.01842657  0.00880997]
 [ 0.01988791  0.10211607 -0.03490467  0.02074167 -0.08727739 -0.06304735]
 [ 0.21047035  0.19392416 -0.17502619 -0.04965431 -0.19964907 -0.06674523]]
```

Multiplicaremos nuestra matriz inversa por la matriz con los datos codificados y de esta manera poder descifrarlos. Así podemos corroborar que los datos son correctos y que nuestra función sí nos brinda los resultados correctos.

La matriz con los datos encriptados es:

```
[230.3]
[263.5]
[238.8]
[814.8]
[-100]
[432.7]
```

```
#nuestra matriz de 6x1 y nuestra matriz de 6x6, usamos reshape para cambiar su dimensión a 6x1
dato_encriptado = np.array([230.3, 263.5, 238.8, 814.8, -100, 432.7]).reshape(-1, 1)
```

```
#guardamos los datos usando la clase decifrar.
datos_desencriptados = cir.descifrar(matriz_codificacion, dato_encriptado)
#mostramos los datos por pantalla
print(datos_desencriptados)
```

```
[[20.]
 [36.7]
 [23.5]
 [24.9]
 [21.5]
 [ 8.4]]
```

## ✓ Rotación de datos

Ya comprobamos que los datos son desencriptados correctamente usando nuestras clases y métodos. Ahora comprobaremos que los datos pueden ser rotados.

Teniendo nuestra variable `datos_desencriptados`, usaremos la clase `Rotacion` para rotar los datos brindados y darnos los nuevos resultados.

La clase `Rotacion()` nos pide `datos_x`, `datos_y`. Esto fue realizado para poder realizar nuestra gráfica con `matplotlib`. Tenemos la clase `crear_array()` la cual nos ayuda a obtener los valores `x`, `y` para utilizar este método.

```
#Primero obtenemos los resultados (x, y), así se los podemos pasar a nuestra clase.
valor_prueba_x = crear.create_valores_x(datos_desencriptados)
#mostramos los datos
print(f"Los valores de x son : {valor_prueba_x}")
#obtenemos los resultados para y
valor_prueba_y = crear.create_valores_y(datos_desencriptados)
#mostramos los datos de y
print(f"Los valores de y son: {valor_prueba_y}")
```

```
Los valores de x son : [array([20.]), array([23.5]), array([21.5])]
Los valores de y son: [array([36.7]), array([24.9]), array([8.4])]
```

⚠ Podemos notar que los valores de `(x,y)` están en arrays anidados, por lo cual será necesario desempaquetarlos a través de la clase `Desempaquetar_Array()`, así de esta manera por fin podremos rotar los datos.

```
valor_prueba_desempaquetar_x, valor_prueba_desempaquetar_y = desempaquetador.desempaquetar(valor_prueba_x,
                                                                                           valor_prueba_y)

# Mostramos los valores (x,y) desempaquetados
print(f'X: {valor_prueba_desempaquetar_x}')
print(f'Y: {valor_prueba_desempaquetar_y}')

# Ahora ya podemos usar la clase Rotacion()

valor_prueba_rotados_x, valor_prueba_rotados_y = rotar.rotar_matriz(valor_prueba_desempaquetar_x,
                                                                      valor_prueba_desempaquetar_y)

# Mostramos los valores de (x,y)
# Valores rotados de (x) y valores rotados de (y).
print(f'X rotado: {valor_prueba_rotados_x}')
print(f'Y rotado: {valor_prueba_rotados_y}')
```

```
X: [np.float64(20.000000000000007), np.float64(23.49999999999986), np.float64(21.50000000000001)]
Y: [np.float64(36.69999999999996), np.float64(24.90000000000002), np.float64(8.40000000000023)]
X rotado: [np.float64(-40.09295449327722), np.float64(-34.22396820942889), np.float64(-21.142492757477793)]
Y rotado: [np.float64(-11.808683245815308), np.float64(-0.9899494936611762), np.float64(9.263098833543765)]
```

Como pudimos comprobar, los valores fueron rotados exitosamente por lo que podemos comprobar que la clase `Rotacion()` funciona sin ningún problema, ahora pasaremos a trasladar los datos.

## ✓ Traslación

Ahora buscaremos comprobar que la clase `Traslacion()` funciona pasándole los datos anteriormente brindados.

La clase `Traslacion()` recibe los datos de `(x,y)` y `(a,b)`. Donde `(a,b)` son las distancias de traslación para los ejes `(x,y)`.

```
# Usaremos los datos anteriormente rotados para obtener los datos finales.  
valor_prueba_transladado_x, valor_prueba_trasladado_y = translacion.trasladar_matriz(valor_prueba_rotados_x,  
valor_prueba_rotados_y, 20, 30)  
  
# Pasamos los valores (a,b) como (20,30)  
# Mostramos los datos con los valores para (x) y (y).  
  
print(f"X: {valor_prueba_transladado_x}")  
print(f"Y: {valor_prueba_trasladado_y}")  
  
X: [np.float64(-20.09295449327722), np.float64(-14.22396820942889), np.float64(-1.1424927574777932)]  
Y: [np.float64(18.19131675418469), np.float64(29.010050506338825), np.float64(39.263098833543765)]
```

Con estas comprobaciones logramos obtener todos los datos y asegurarnos que las clases y métodos funcionan para ahora poder pasar un archivo.txt con todos los datos y así poder desencriptar los datos de forma automatizada.

Mostraremos los datos para corroborar las gráficas

## ▼ Gráficas de Comprobación

### > Datos Desencriptados

```
#Grafica para los puntos desencriptados únicamente.  
graficar.graficadora(valor_prueba_x, valor_prueba_y)
```

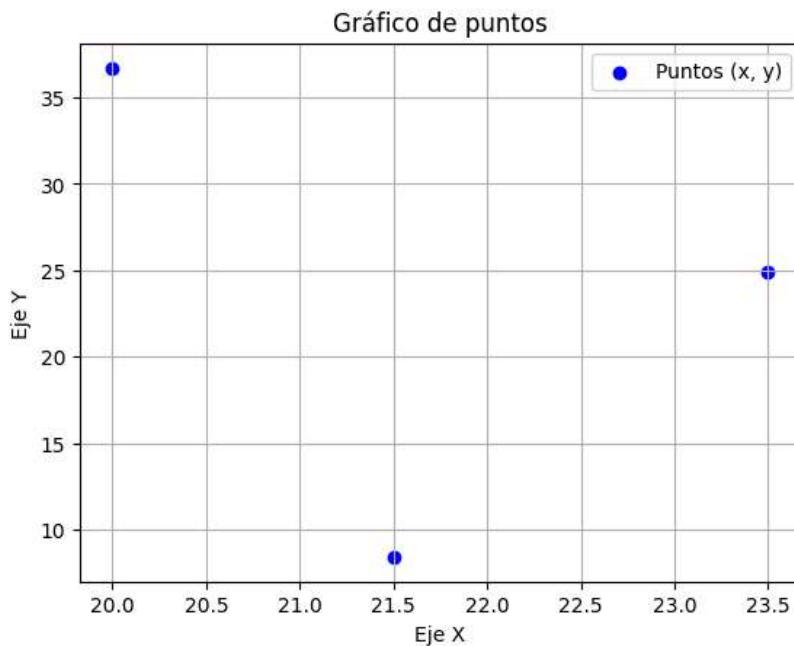


Gráfico con datos desencriptados

✓ > Datos Rotados

```
#Grafica para los datos rotados  
graficar.graficadora(valor_prueba_rotados_x, valor_prueba_rotados_y)
```

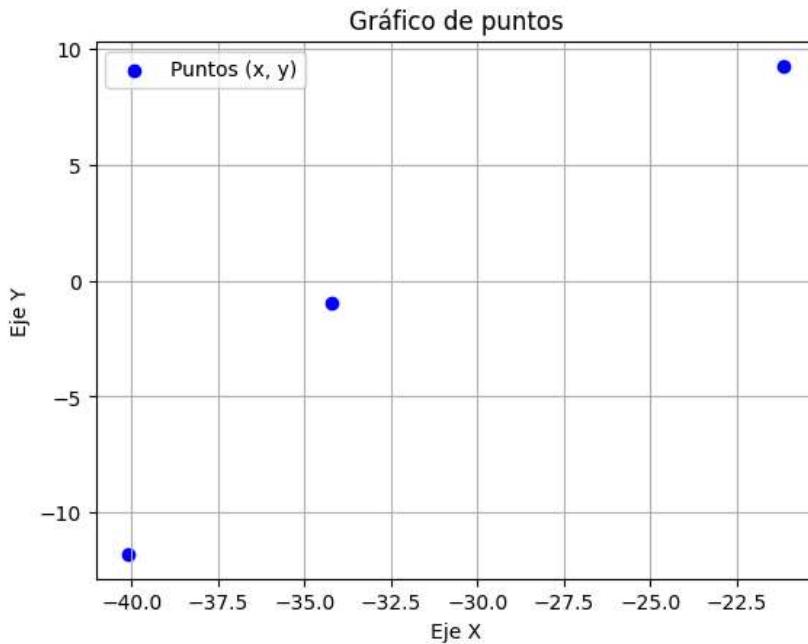


Gráfico con datos descriptados y rotados

✓ > Datos Trasladados

```
#Gráfica para los datos rotados y trasladados  
graficar.graficadora(valor_prueba_transladado_x, valor_prueba_transladado_y)
```

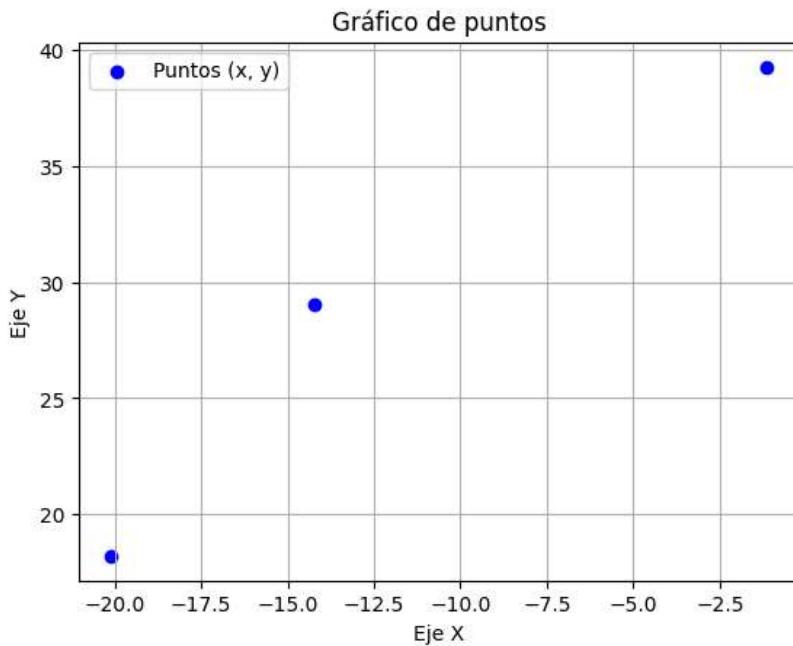


Gráfico con datos descriptados

## ▼ Desencriptar

Como ya comprobamos que las clases y métodos funcionan, ahora les pasaremos un archivo "datos encriptados.txt" el cual tiene todos los datos encriptados. Cada línea tiene 6 datos los cuales corresponderán a nuestras matrices 6x1 y resolveremos linea por linea de nuestro documento.

Obtenemos cada valor para nuestros datos:

```
#Desencriptamos todos los valores para (x, y)
valores_x, valores_y = aplicacion.abrir_documento(matriz_codificacion, "/content/drive/MyDrive/Colab Notebooks/datos"
#Los datos están anidados, así que los resolvemos con la clase Desempaquetar_Array()
desempaquetar_x, desempaquetar_y = desempaquetador.desempaquetar(valores_x, valores_y)

#Rotamos los datos ahora que están desempaquetados
valores_rotados_x, valores_rotados_y = rotar.rotar_matriz(desempaquetar_x, desempaquetar_y)

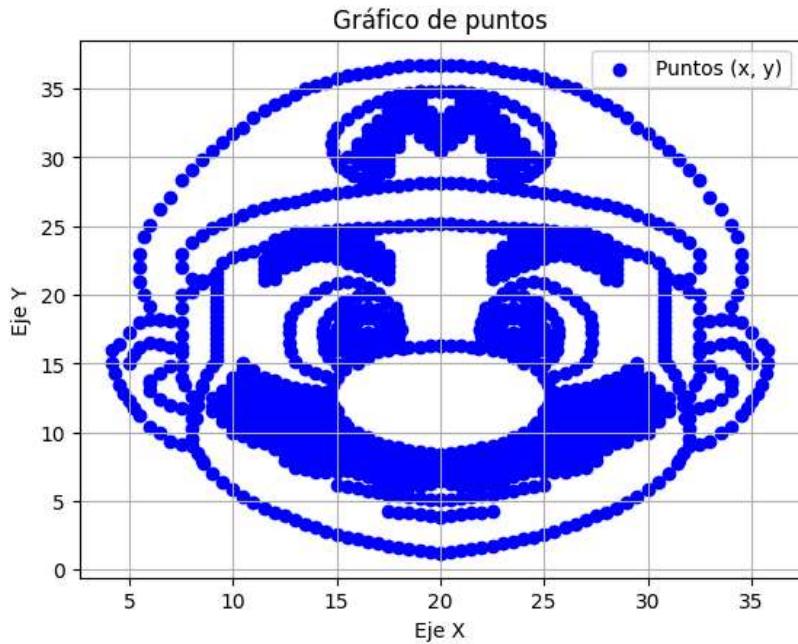
#Trasladamos los datos ya rotados para obtener la última gráfica.
valores_traslados_x, valores_traslados_y = translacion.trasladar_matriz(valores_rotados_x, valores_rotados_y, 20)
```

Como ya tenemos todos nuestros valores **Desencriptados**, **Rotados** y **Trasladados** ahora hace uso de nuestra clase **Graficar()** para visualizar los datos.

## ▼ > Graficas - Datos Resueltos!

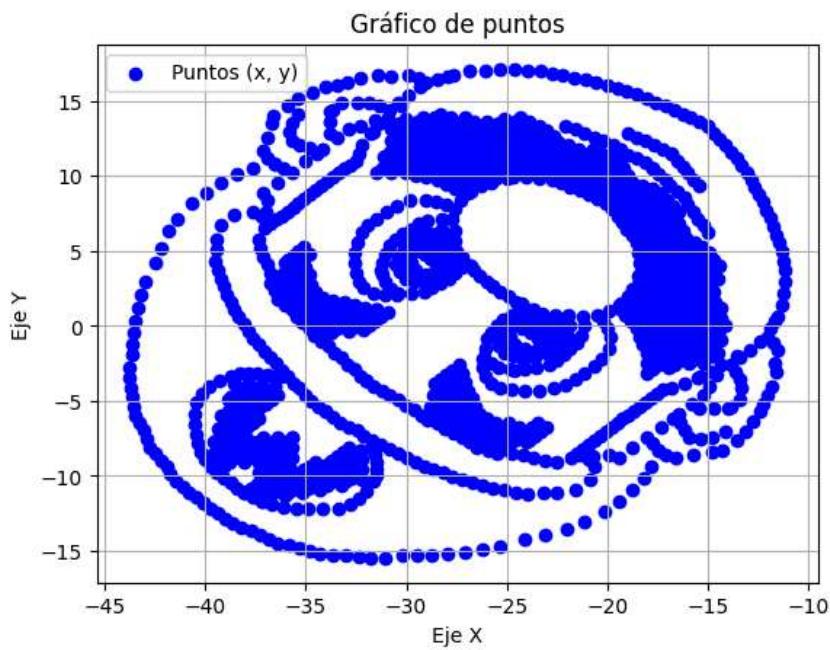
## ▼ > Gráfico desencriptado

```
graficar.graficadora(desempaquetar_x, desempaquetar_y)
```



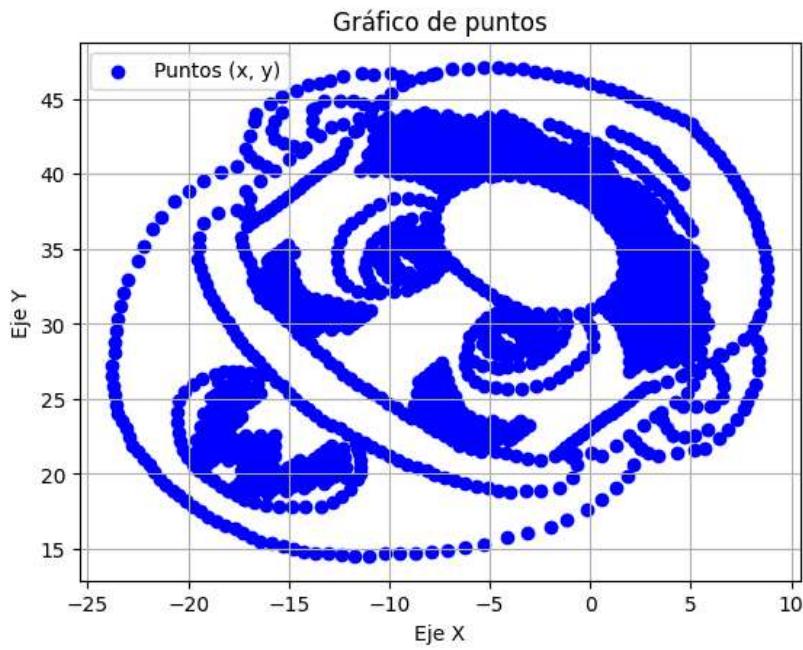
> Gráfico Matriz Rotación

```
graficar.graficadora(valores_rotados_x, valores_rotados_y)
```



> Gráfico Matriz Rotación y Traslado

```
graficar.graficadora(valores_trasladados_x, valores_trasladados_y)
```



## ✓ Guardar Datos

Guardamos los datos ya descriptados, rotados y trasladados en nuevos archivos.

```
guardar_datos.save(desempaquetar_x, desempaquetar_y,  
"datos\valores descriptados.txt")  
  
guardar_datos.save(valores_rotados_x, valores_rotados_y,  
"datos\valores rotados.txt")  
  
guardar_datos.save(valores_trasladados_x, valores_trasladados_y,  
"datos\valores trasladados.txt")
```

Los datos descriptados los puede encontrar en el siguiente enlace:

<https://github.com/jgarnigar/Proyecto-Algebra-Lineal/blob/master/datos/valores%20descriptados.txt>

Los datos descriptados y rotados los puede encontrar en el siguiente enlace:

<https://github.com/jgarnigar/Proyecto-Algebra-Lineal/blob/master/datos/valores%20rotados.txt>

Los datos descriptados, rotados y trasladados los puede encontrar en:

<https://github.com/jgarnigar/Proyecto-Algebra-Lineal/blob/master/datos/valores%20trasladados.txt>

## Conclusiones y Trabajo Futuro

El presente proyecto ha culminado con la **implementación y validación exitosa** de un sistema de descifrado y transformación de datos fundamentado en los principios del Álgebra Lineal. Mediante la correcta aplicación de la inversión de matrices y operaciones de multiplicación, se ha logrado **descifrar la totalidad de los datos** contenidos en el archivo fuente, confirmando la precisión de la clase **Cifrado()**.

Adicionalmente, se demostró la funcionalidad de las transformaciones geométricas. Las clases **Rotacion()** y **Traslacion()\*** ejecutaron las manipulaciones vectoriales deseadas, lo cual fue corroborado por las gráficas finales. La **integración** de todas las clases (descifrado, desempaquetado y transformaciones) permitió procesar el archivo de datos de forma **automatizada y eficiente**, cumpliendo a cabalidad con el objetivo principal del proyecto.

Para expandir el alcance de este trabajo, se proponen las siguientes líneas de mejora:

1. Implementar un algoritmo de cifrado más robusto que utilice métodos de Álgebra Lineal distintos al cifrado Hill básico.
2. Optimizar el manejo de la memoria con NumPy para procesar archivos de datos de mayor volumen de manera más eficiente.

## ✓ Autor

Hecho con  (quizá demasiado)!

*Junior Eduardo Garniga Rojas* 

## Anexo A:

### Matriz

Una matriz es un conjunto bidimensional de números o expresiones dispuesto en filas y columnas. Es la herramienta fundamental utilizada para codificar y descifrar datos en este proyecto. En nuestro caso, se emplea una matriz cuadrada de  $6 \times 6$  como clave de codificación.

### Matriz Inversa

La matriz inversa, denotada como  $A^{-1}$  para una matriz  $A$ , es crucial para el descifrado. Si  $A \cdot X = B$  (donde  $X$  es la matriz de datos encriptados y  $B$  es la matriz de datos originales), entonces la matriz original se recupera multiplicando la matriz inversa por la matriz encriptada:  $A^{-1} \cdot A \cdot X = A^{-1} \cdot B$ . Para que una matriz tenga inversa, su determinante debe ser distinto de cero.

### Cifrado Hill (Conceptual Simple)

El Cifrado Hill es un esquema de cifrado por sustitución polialfabética basado en el Álgebra Lineal. Consiste en dividir el mensaje de texto en bloques (vectores) y multiplicarlos por una **matriz clave** (nuestra matriz  $6 \times 6$ ) para obtener el texto cifrado. Para descifrar el mensaje, el texto cifrado se multiplica por la **matriz inversa** de la clave, que es el método que replicamos en este proyecto.

## Anexo B:

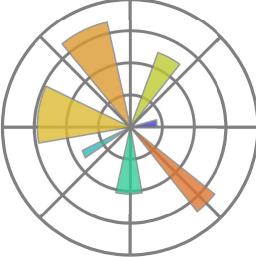
### Estructura completa

```
└── Proyecto-algebra-linea
    ├── datos/
    │   ├── valores_traslados.txt
    │   ├── datos_encriptados.txt
    │   ├── datos.txt
    │   ├── practica.txt
    │   ├── valores_desencriptados.txt
    │   └── valores_rotados.txt
    |
    └── funciones/
        ├── init.py
        ├── app.py
        ├── cifrado.py
        ├── crear_array.py
        ├── desempaquetar.py
        ├── ecuacion.py
        ├── graficar.py
        ├── guardar.py
        ├── rotacion.py
        └── traslacion.py
    |
    └── imagenes
```

```
| └─ 📜 datos_desencriptados_test.png  
| └─ 📜 datos_desencriptados.png  
| └─ 📜 datos_rotados_test.png  
| └─ 📜 datos_rotados.png  
| └─ 📜 datos_trasladados_test.png  
| └─ 📜 datos_trasladados.png  
|  
|  
└─ 📄 .gitignore  
└─ 📄 FICHA_TECNICA.md  
└─ 📄 INSTRUCCIONES_PROYECTO.pdf  
└─ 📄 PROYECTO_ALGEBRA_LINEA.pdf  
└─ 📄 README.md  
└─ 📄 clases.py  
└─ 📄 ecuacion.py  
└─ 📄 main.py  
└─ 📄 requirements.txt
```

## Anexo C:

### 📦 Tecnologías usadas

Tecnología	Función
Python	 Lenguaje principal
NumPy	 Operaciones numéricas
Matplotlib	 Gráficas y visualización