Proyecto Final | Desencriptar datos.

Junior Eduardo Garniga Rojas | Álgebra Lineal

[GitHub] (https://github.com/jgarnigar/Proyecto-Algebra-Lineal)

Descripción

Ejercicio de desencriptar datos. Los datos fueron encriptados a través de una matriz dada tal que:

```
[ 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]
```

Los vectores (x,y) en una matriz de 6x1. Se multilplica la matriz 6x6 X 6x1.

Nos dan los datos en el archivo "datos encriptados.txt". Para poder desencriptar los datos, tomamos los datos de 6 en 6 creando una matriz 6x1 la cual será multiplicada por la inversa de la matriz 6x6 dada anteriormente. El resultado serán los datos desencriptados, después tenemos que multiplicar cada punto (x,y) por una matriz de rotación con 135 grados. Por último tomamos los datos multiplicados por la matriz de rotación y multiplicamos esos datos (x,y) por una matriz de traslación.

Mostramos un gráfico por los datos desencriptados, con rotación y traslación.

Requisitos e instalación

Requisitos

- Python 3.10 o más nuevo
- Numpy
- Matplotlib

Instalación

- Para instalar el proyecto por favor clone el repositorio:
- Proyecto Algebra Lineal

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

Instalar dependencias:

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

> *Clases*

Cargamos nuestras librerías

```
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.

```
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 decifrar(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*

Creamos una clase crear_array para poder obtener los datos separados y así poder graficarlos, también nos servirá para poder operar los valores para rotar y trasladar nuestros vectores.

```
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
```

> *Ecuación*

Creamos una clase llamada ecuación la cual utilizaremos para poder resolver un sistema de ecuaciones de 10x10 y así obtener los datos de nuestra matriz utilizada para codificar y decodificar

```
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*

Creamos una clase una clase llamada desempaquetar la cual nos ayudará a tener un solo array con todos los datos para x y otro para y. Debido a que la clase Applicaion(más adelante presentada), nos brinda un array y después anidamos ese array entre otro. Por lo que al momento de rotarlos datos y trasladarlos tenemos un array entero y no un array multidimensional el cual nos daría problemas con la propiedad zip la cual solo puede obtener una lista o un array y no arrays multidimensionales.

Ejemplo:

```
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
```

```
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
```

> *Rotación*

Creamos la clase Rotacion la cual nos ayuda a rotar todos los datos x, y multiplicando los vectores x,y por la matriz rotacion.

[np.cos(angulo), -np.sin(angulo)]

[np.sin(angulo), np.cos(angulo)]

```
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
```

> *Traslación*

Creamos una clase Traslación la cual nos ayuda a trasladar nuestros datos en los puntos x,y. Mutiliplicando el los vectores.

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

> *Guardar Datos*

Creamos una clase para guardarlos datos obtenidos en un documento aparte.

```
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*

Creamos una clase App en la cual abrimos nuestro documento con los datos encriptados y realizamos la lógica para poder llamar a nuestra función encriptar y guardarlos datos. Los datos se guardan en un array anidado creando un array multidimensional el cual tiene que ser desempaquetado.

```
class App():
    def abrir_document(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 == "decifrar":
                    resultado = cir.decifrar(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*

Creamos una clase para podergraficar nuestros resultados de x,y.

```
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)

        # Mostrar el gráfico
        plt.show()
```

> *Main*

> *Ecuación*

Código utilizado para correr nuestro programa.

Resolvemos nuestra ecuación de 10x10

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

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

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

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

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

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

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

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

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

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

```
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)

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

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)

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)

print(matriz)
```

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

> *Ecuaciones*

Primer ecuación para encontrar U, V

$$U = (6, p, r) \quad V = (t, 8, 9)$$
$$3U - 10V = (-42, -68, -126)$$
$$3U = (18, 3p, 3r) \quad 10V = (10t, 80, 90)$$
$$(t) \quad 18 - 10t = -42$$
$$-10t = -60 \quad (t) = 6$$
$$(p) \quad 3p - 80 = -68$$
$$3p = 12 \quad (p) = 4$$
$$(r) \quad 3r - 90 = -126$$
$$3r = -36 \quad (r) = -12$$
$$(t) = 6, \quad (p) = 4, \quad (r) = -12$$

$$U = (\frac{-1}{2}, \frac{\sqrt{38}}{2}, \frac{5}{2})$$
$$|u| = -w$$

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

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

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

$$|U| = 4$$

$$(W) = -4$$

$$(z)$$

$$U * V = z$$

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

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

$$(u * v) = (\frac{-1}{2} * \frac{11}{1}) + (\frac{\sqrt{38}}{2} * \frac{3\sqrt{38}}{1}) + (\frac{5}{2} * \frac{-21}{1})$$

$$(u * v) = (\frac{-11}{2}) + (\frac{3\sqrt{38}^2}{2}) + (\frac{-105}{2})$$

$$(u * v) = (\frac{-11}{2} - \frac{-105}{2}) + (\frac{57}{1})$$

$$(u * v) = -58 + 57$$

$$(u * v) = -1$$

$$(z) = -1$$

Se tienen los vectores U = (3, 6, 7) y V = (k, m, n). El resultado de operar 2U X 3V es igual a 612i + 156j -396k y el producto U * V = 58

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

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

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

$$6n - 7m = 102$$

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

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

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.

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

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

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:

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

Ahora encontramos (k) sustituyendo n por 10:

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

Para finalizar tenemos los valores:

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

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

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

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]
```

> *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: {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]]
```

Descencriptar datos

Multiplicaremos nuestra matriz inversa por la matriz con los datos codificados y de esta manera poder descifrarlos. Así podemos corroborarque 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.decifrar(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 = 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 = create.valores_y(datos_desencriptados)
#mostramos los datos de y
print(f"Los valores de y son: {valor_prueba_y}")

#estos datos aún no están rotados, la rotación será ahora con la clase Rotacion()
```

```
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, val
#mostramos los valores
print(f"Los valores de x son: {valor_prueba_desempaquetar_x}")
print(f"Los valores de y son: {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
#Mostramos los valores

print(f"Los valores x rotados son: {valor_prueba_rotados_x}")
print(f"Los valores y rotados son: {valor_prueba_rotados_y}")
```

```
Los valores de x son: [np.float64(20.000000000000007), np.float64(23.499999999999986), np.float64(21.5000000000
Los valores de y son: [np.float64(36.69999999999996), np.float64(24.900000000000002), np.float64(8.400000000000
Los valores x rotados son: [np.float64(-40.09295449327722), np.float64(-34.22396820942889), np.float64(-21.1424
Los valores y rotados son: [np.float64(-11.808683245815308), np.float64(-0.9899494936611762), np.float64(9.2630
```

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

Traslación

Ahora buscaremos comprobar que la clase Traslacion() funciona pasándole los datos anteriomente 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 anteriomente rotados para obtener los datos finales.
valor_prueba_transladado_x, valor_prueba_trasladado_y = traslacion.trasladar_matriz(valor_prueba_rotados_x, va

#pasamos los valores (a,b) como (20,30)
#mostramos los datos.

print(f"Los valores trasladados para x son: {valor_prueba_transladado_x}")
print(f"Los valores trasladados para y son: {valor_prueba_trasladado_y}")
```

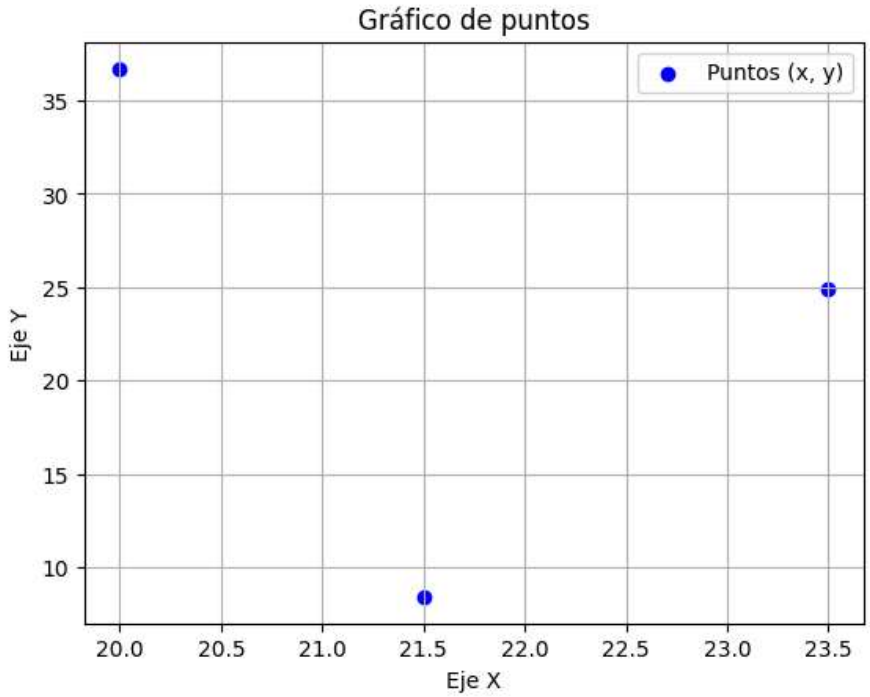
```
Los valores trasladados para x son: [np.float64(-20.09295449327722), np.float64(-14.22396820942889), np.float64
Los valores trasladados para y son: [np.float64(18.19131675418469), np.float64(29.010050506338825), np.float64(
```

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.

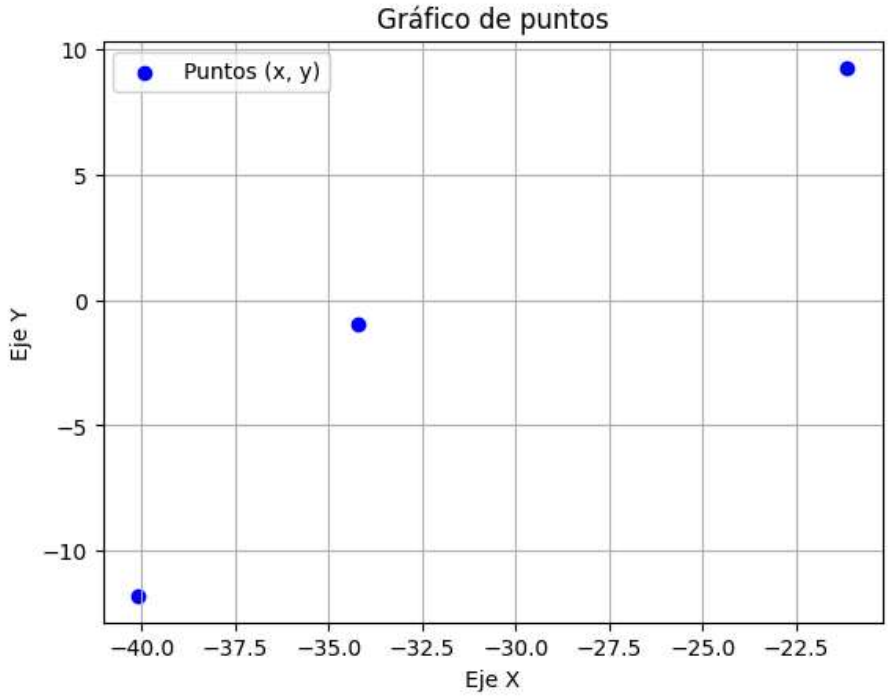
Mostraremos los datos para corroborar las gráficas

Gráficas

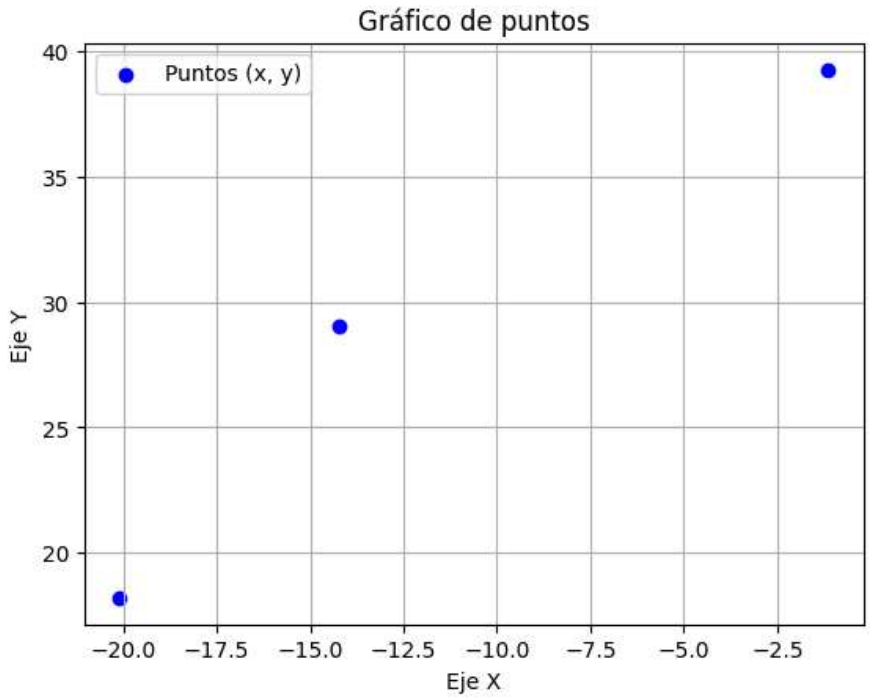
```
#Grafica para los puntos desencriptados únicamente.
graficar.graficadora(valor_prueba_x, valor_prueba_y)
```




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




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



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 línea por línea de nuestro documento.

Obtenemos cada valor para nuestros datos

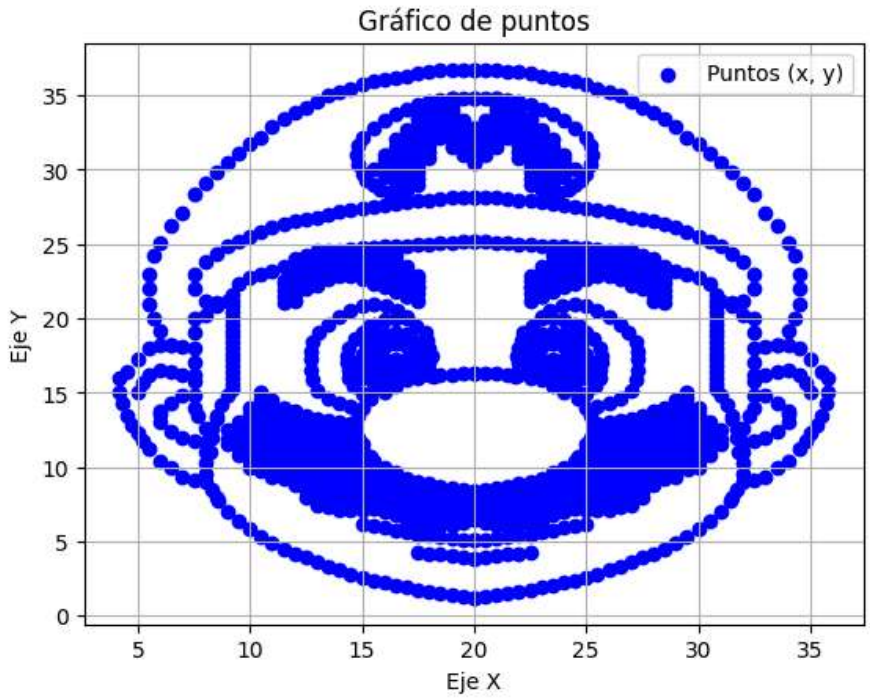
```
#Desencriptamos todos los valores para (x, y)
valores_x, valores_y = aplicacion.abrir_document(matriz_codificacion, "/content/drive/MyDrive/Colab Notebooks/
#Los datos están anidados, así que los resolvemos con la clase Desempaquetar_Array()
desempaquetar_x, desempaquetar_y = desempaquetador.desempaquetar(valores_x,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

> *Gráfico desencriptado*

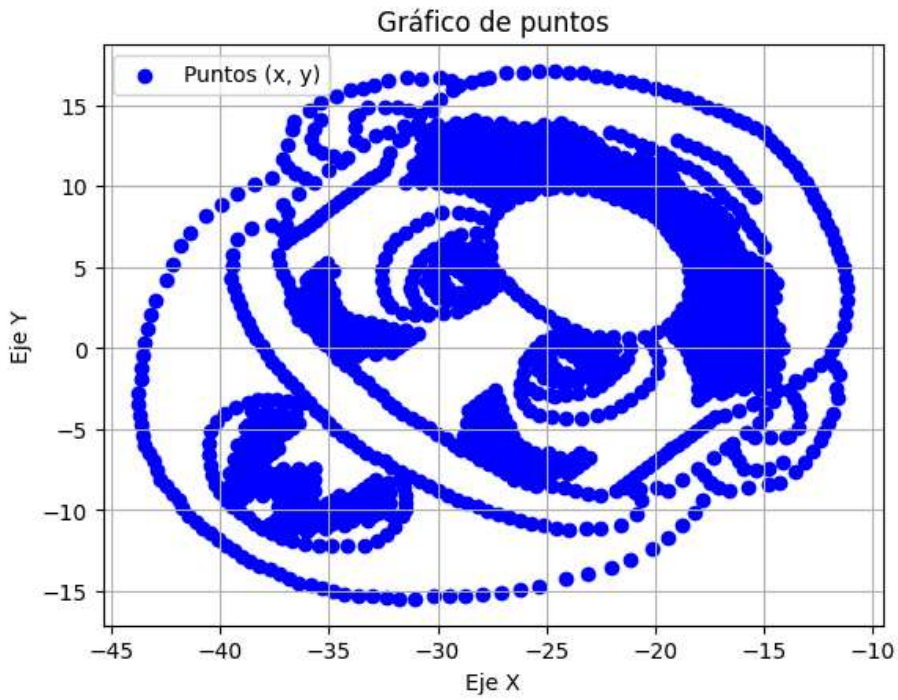
```
graficar.graficadora(desempaquetar_x, desempaquetar_y)
```



.

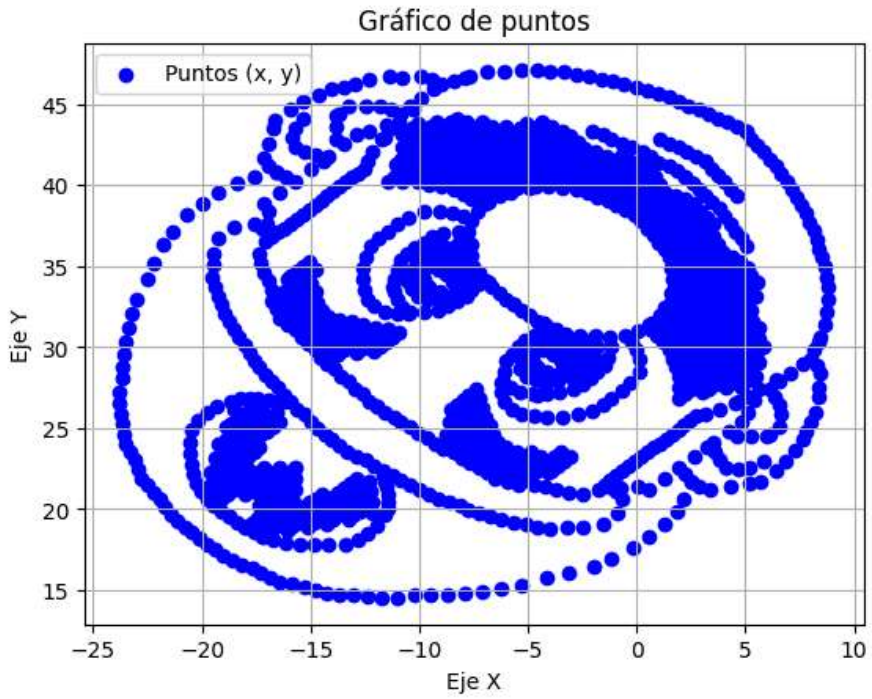
> *Gráfico Matriz Rotación*

```
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 desencriptaods, rotados y trasladados en nuevos archivos.

```
guardar_datos.save(desempaquetar_x, desempaquetar_y, "/content/drive/MyDrive/Colab Notebooks/valores desencrip
guardar_datos.save(valores_rotados_x, valores_rotados_y, "/content/drive/MyDrive/Colab Notebooks/valores rotad
guardar_datos.save(valores_trasladados_x,valores_trasladados_y,"/content/drive/MyDrive/Colab Notebooks/valores
```

Autor

Desarrollado por **Junior Eduardo Garniga Rojas**
Contacto: [jgarnigar@miumg.edu.gt] GitHub: jgarnigar