- **Rúbrica Pregunta 1**:

(a)
- **[15 puntos]** Primero, se define el procedimiento de suavizamiento de datos:
  - (I) Se deben seleccionar los índices de datos x_i en el intervalo $[z-\delta, z+\delta]$, los cuales podemos denotar por $S(z)$, entonces $x_{S(z)_j} \in [z-\delta, z+\delta]$, para $j \in \{0, 1, \ldots, |S(z)|-1\}$. Notar que estos índices cambian a medida que cambia z. z se define en el siguiente punto.
  - (II) Se debe construir el problema de ajuste de mínimos cuadrados correspondiente:

$$\min_{\alpha_0(z),\alpha_1(z),\alpha_2(z)} \|\mathbf{r}\|_2 = \min_{\alpha_0(z),\alpha_1(z),\alpha_2(z)} \left\| \begin{bmatrix} y_{S(z)_0} \\ y_{S(z)_1} \\ \vdots \\ y_{S(z)_{|S(z)|-1}} \end{bmatrix} - \begin{bmatrix} 1 & x_{S(z)_0} - z & \frac{(x_{S(z)_0} - z)^2}{2} \\ 1 & x_{S(z)_1} - z & \frac{(x_{S(z)_1} - z)^2}{2} \\ \vdots & \vdots & \vdots \\ 1 & x_{S(z)_{|S(z)|-1}} - z & \frac{(x_{S(z)_{|S(z)|-1}} - z)^2}{2} \end{bmatrix} \begin{bmatrix} \alpha_0(z) \\ \alpha_1(z) \\ \alpha_2(z) \end{bmatrix} \right\|_2$$

  - (III) Se resuelve el problemas de mínimos cuadrados con la factorización QR reducida.
- **[10 puntos]** Segundo, se define el procedimiento de interpolación de datos:
  - (I) Se definen $n+1$ puntos de Chebyshev en el intervalo $[-1, 1]$, esto asegura que el polinomio resultante sea de grado n, y estos $n+1$ puntos de Chebyshev serán los nodos que se utilizarán en la interpolación, es decir los valores $\widehat{x}_l$. Esto reduce el fenómeno de Runge. Es importante destacar que al elegir $n+1$ puntos de interpolación se da la posibilidad de que el polinomio resultante sea de grado n, sin embargo este puede ser de grado menor si los datos que se obtiene que interpolar requieren un polinomio de grado menor. Por ejemplo, si la data fuera una constante, entonces aunque se utilicen $n+1$ para interpolarla, el grado del polinomio resultante será 0.
  - (II) En cada punto de Chebyshev, se suavizará la data según el procedimiento antes mencionado, es decir para cada punto de Chebyshev $\widehat{x}_l$, se llamará a la función para suavizar la data y se obtendrá el coeficiente $\overline{\alpha_0(\widehat{x}_l)}$ y se usará como $\widehat{y}_l$. En resumen, los datos a interpolar son $(\widehat{x}_l, \overline{\alpha_0(\widehat{x}_l)})$.
  - (III) Para interpolar los datos se utilizará la Interpolación Baricéntrica dado que es el algoritmo que requiere menos operaciones elementales para ser construrida y evaluada que las alternativas presentadas.

(b)
```
'''
input:
xi      : (ndarray) Input data point xi.
yi      : (ndarray) Input data point yi.
delta   : (float) Size of the window to be used.
n       : (int) Degree of polynomial to be built.

output:
p_smoothed  : (callable) The callable polynomial approximation of the
                quadratically smoothed data.
'''
def quadratically_smoothed_pol(xi,yi,delta,n):
```

- [**15 puntos**] Definir el procedimiento para obtener los α's.

```
# This function computes the "alphas" that
# minimize the least-square error when
# approximating a parabola.
def obtain_alphas(x,y,z,delta):
    # Selecting the data logically within the indicated window
    selected = np.logical_and(x<=z+delta,z-delta<=x)
    # Getting the actual data to be used for interpolation
    x_local = x[selected]
    y_local = y[selected]
    # getting the number of points to be used
    n = len(x_local)
    # Initializing the truncated Vandermonde matrix.
    # Notice that since we use a matrix of ones
    # the first columns is defined correctly right away.
    V = np.ones((n,3))
    # It computes the second column of the Vandermonde matrix.
    V[:,1] = x_local-z
    # It computes the third column of the Vandermonde matrix.
    V[:,2] = np.power(x_local-z,2.)/2.
    # Now we compute the reduced QR factorization of the Vandermondematrix
    Qh, Rh = np.linalg.qr(V)
    # First step: Computing the right-hand-side
    RHS = np.dot(Qh.T,y_local)
    # Second step: Computing the 'alphas' from the upper-triangular matrix 'Rh'
    alphas = solve_triangular(Rh, RHS)
    # Finally, returning the 'alphas'
    return alphas
```

- [**10 puntos**] Definir el procedimiento para obtener el polinomio interpolador a partir de los datos suavizados uttilizando puntos de Chebyshev.

```
# Defining which 'z_i' we will be using
zs = myChebyshev(-1,1,n+1)
# Variable to onoy store the 'alpha_0'
alpha_0_cheb = np.zeros(n+1)
# Going over the whole range of Chebyshev points
for i in np.arange(n+1):
    # Just storing the alpha we need
    alpha_0_cheb[i] = obtain_alphas(xi,yi,zs[i],delta)[0]
# Building the 'Barycentric Interpolator'.
# NOTE: If you want to implement this, I suggest to use
# BarycentricInterpolator from SciPy.
p_smoothed = BarycentricInterpolation(zs, alpha_0_cheb)
```

```
    # It returns the interpolator.
    return p_smoothed
```

- **Rúbrica Pregunta 2**:

(a)
- **[5 puntos]** La transformación lineal que se aplica a cada mensaje encriptado viene dado por $\widetilde{\mathbf{s}}_i\, K_R = \mathbf{s}_i$ para cada $i \in \{1, 2, \dots\}$. Esto significa que:

$$\begin{aligned} \widetilde{\mathbf{s}}_1\, K_R &= \mathbf{s}_1 \\ \widetilde{\mathbf{s}}_2\, K_R &= \mathbf{s}_2 \\ &\vdots \end{aligned}$$

Lo que es análogo a:

$$\begin{bmatrix} \widetilde{\mathbf{s}}_1 \\ \widetilde{\mathbf{s}}_2 \\ \vdots \end{bmatrix} K_R = \begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \vdots \end{bmatrix} \Rightarrow A_T\, K_R = A$$

donde cada fila i de A_T y A contiene el mensaje encriptado $\widetilde{\mathbf{s}}_i$ y desencriptado $\mathbf{s}_i$ respectivamente.

- **[10 puntos]** Para resolver el sistema matricial $A_T\, KR = A$ basta con utilizar N mensajes encriptados/desencriptados en las matrices A_T y A respectivamente. Nos debemos asegurar eso sí, que los mensajes sean linealmente independientes, por lo que cada vez que agreguemos un mensaje encriptado a la matriz A_T y su correspondiente mensaje desencriptado a la matriz A, debemos comprobar que sea linealmente independiente a los mensajes ya agregados. Sea $A_T^{(k)} \in \mathbb{R}^{k \times N}$ la matriz que contiene k mensajes encriptados y linealmente independientes, entonces si se quiere agregar el mensaje encriptado $\widetilde{\mathbf{s}}_i$ a la matriz $A_T^{(k)}$, se debe comprobar que al ejecutar la función $LI(A_T^{(k)}, \widetilde{\mathbf{s}}_i)$ retorne `True`.
- **[10 puntos]** Después de obtener los N mensajes encriptados linealmente independientes, es decir, al haber obtenido $A_T = A_T^{(N)}$, se debe construir la matriz K_R a partir de las matrices A_T y A. Para esto, se deben resolver N sistemas de ecuaciones lineales de la forma:

$$A_T\, \mathbf{kr}_i = \mathbf{a}_i \text{ para } i \in \{1, \dots, N\}$$

donde $\mathbf{kr}_i$ es la i-ésima columna de K_R y $\mathbf{a}_i$ es la i-ésima columna de la matriz A con los mensajes desencriptados. Luego, se calcula la factorización $P\, A_T = L\, U$ para obtener cada vector columna $\mathbf{kr}_i$ de la matriz K_R. La factorización se utiliza de la siguiente forma para cada $i \in \{1, \dots, N\}$:
  - Resolver $L\, \mathbf{y}_i = P\, \mathbf{a}_i$ para $\mathbf{y}$
  - Resolver $U\, \mathbf{x}_i = \mathbf{y}_i$ para $\mathbf{x}_i$
  - Asignar $\mathbf{x}_i$ como la i-ésima columna $\mathbf{kr}_i$ de la matriz K_R.

(b)
```
'''
input:
St     : (ndarray)  Array of encrypted messages with dimension M x N.
S      : (ndarray)  Array of unencrypted with dimension M x N.
N      : (int)      Dimension of the messages.
M      : (int)      Number of pair of messages.

output:
KR     : (ndarray) The key to apply to encrypted messages.
'''
def obtain_key(St,S,N,M):
```

- **[4 puntos]** Crear matrices con mensajes encriptados (originales) A_t y desencriptados A. Se agrega el primer mensaje.

```
At,A = np.zeros((N,N)),np.zeros((N,N))
At[0,:] = St[0,:]
A[0,:] = S[0,:]
```

- **[8 puntos]** Agregar los mensajes encriptados/desencriptados a las matrices A_t/A respectivamente verificando que, el mensaje encriptado a ser agregado a la matriz A_t, sea linealmente independiente con las filas ya existentes.

```
count = 1
i = 1
while count < N:
    st = St[i]
    if LI(At[:count],st):
        At[count,:] = St[count,:]
        A[count,:] = S[count,:]
        count += 1
    i += 1
```

- **[5 puntos]** Obtener factorización $P\,A_t = L\,U$ de la matriz A_t e inicializar la matriz KR para obtener la llave.

```
P,L,U = palu(At)
KR = np.zeros((N,N))
```

- **[8 puntos]** Resolver, con la factorización $P\,A_t = L\,U$ de la matriz A_t, para cada columna $\mathbf{kr}_i$ de K_R con $i \in \{0, \ldots, N-1\}$, el sistema $A_t\,\mathbf{kr}_i = \mathbf{a}_i$, donde $\mathbf{a}_i$ es la i-ésima columna de A.

```
for i in np.arange(N):
    b = P@A[:,i]
    y = solve_triangular(L,b,lower=True)
    x = solve_triangular(U,y,lower=False)
    KR[:,i] = x
return KR
```