1. **Desarrollo Pregunta 1**:

   (a) **[5 puntos]** Primero se debe aproximar el valor absoluto de la derivada de $f(x)$ para así obtener los puntos $z_k \approx |f'(x_k)|$ con alguna regla de aproximación. Por ejemplo, se puede aplicar **forward-difference** para aproximar la derivada en cada punto x_k para $k \in \{0, 1, \ldots, n\}$:

   $$z_k = \frac{|y_{k+1} - y_k|}{x_{k+1} - x_k} \quad \text{para } k \in \{0, 1, \ldots, n\}$$

   Segundo, nos falta la derivada para el punto x_{n+1}, en este caso utilizamos **backward-difference** para la aproximación:

   $$z_{n+1} = \frac{|y_{n+1} - y_n|}{x_{n+1} - x_n}$$

   Notar que el valor absoluto se aplica solo al numerador de z_k porque se conoce que el denominador es siempre positivo.
   [10 puntos] Finalmente, utilizando los puntos z_k para $k \in \{0, 1, \ldots, n+1\}$ obtenidos, se aplica el método del Trapecio, pero una versión modificada, ya que la data no está equiespaciada. Para cada intervalo $[x_k, x_{k+1}]$ con $k \in \{0, \ldots, n\}$ se aproxima la integral como:

   $$\int_{x_k}^{x_{k+1}} |f'(x)| \approx \frac{(z_k + z_{k+1})\,(x_{k+1} - x_k)}{2} = (z_k + z_{k+1})\,\frac{\Delta x_k}{2}$$

   donde $\Delta x_k = (x_{k+1} - x_k)$. Aplicando en todo el intervalo para $|f'(x)|$ se tiene que:

   $$\begin{aligned}
   \int_{-1}^{1} |f'(x)| &\approx \sum_{k=0}^{n} (z_k + z_{k+1})\,\frac{\Delta x_k}{2} \\
   &\approx (z_0 + z_1)\,\frac{\Delta x_0}{2} + (z_1 + z_2)\,\frac{\Delta x_1}{2} + \cdots + (z_n + z_{n+1})\,\frac{\Delta x_n}{2} \\
   &\approx \frac{\Delta x_0}{2}\,z_0 + \left(\frac{\Delta x_0}{2} + \frac{\Delta x_1}{2}\right) z_1 + \left(\frac{\Delta x_1}{2} + \frac{\Delta x_2}{2}\right) z_2 + \left(\frac{\Delta x_{n-1}}{2} + \frac{\Delta x_n}{2}\right) z_n + \frac{\Delta x_n}{2}\,z_{n+1} \\
   &\approx \frac{\Delta x_0}{2}\,z_0 + \frac{\Delta x_n}{2}\,z_{n+1} + \sum_{k=0}^{n-1} \left(\frac{\Delta x_k}{2} + \frac{\Delta x_{k+1}}{2}\right) z_{k+1}
   \end{aligned}$$

   Notar que si $h = \Delta x_0 = \Delta x_1 = \cdots = \Delta x_n$ obtenemos los pesos del método del Trapecio para puntos equiespaciados.

   (b) Se debe evaluar un punto x_j en la ecuación diferencial, por lo tanto, se debe obtener $f''(x_j)$, $f'(x_j)$ y $f(x_j)$.

   - **[5 puntos]** Para un punto x_j se puede utilizar la función AGSD$(\cdot)$ para aproximar $f''(x_j)$. Se debe entregar a la función los parámetros adecuados:

     $$f''(x_j) \approx \text{AGSD}(x_{j-1}, y_{j-1}, x_j, y_j, x_{j+1}, y_{j+1})$$

   - **[5 puntos]** Para la derivada $f'(x_j)$ se utiliza la aproximación realizada en el ítem anterior $f'(x_j) \approx (y_{j+1} - y_j)/(x_{j+1} - x_j)$ y para evaluar la función en x_j, es decir, $f(x_j)$, simplemente se utiliza los datos disponibles, por lo tanto, $f(x_j) = y_j$
   - **[5 puntos]** Finalmente se evalúa si el punto x_j aproxima la ecuación diferencial con una tolerancia δ:

     $$\left| \alpha\,\text{AGSD}(x_{j-1}, y_{j-1}, x_j, y_j, x_{j+1}, y_{j+1}) + \beta\,\frac{(y_{j+1} - y_j)}{(x_{j+1} - x_j)} + \gamma\,y_j - 1 \right| < \delta$$

(c)
```
'''
input:
n         : (integer) Value of n, where the length of the sequence in the table is n + 2.
x_k       : (ndarray) Array with the values of x_k.
y_k       : (ndarray) Array with the values of y_k.
alpha     : (float) Value of alpha.
beta      : (float) Value of beta.
gamma     : (float) Value of gamma.
j         : (integer) Index where the differential equation is verified for the data.
delta     : (float) Tolerance.

output:
total_var : (float) Approximation of total variation of f(x).
check     : (bool)  Boolean value that indicates if the data approximates the differential
equation at x_j. If the data satisfied the tolerance, the output must be True, otherwise False.
'''
def jones(n,x_k,y_k,alpha,beta,gamma,j,delta):
```

- [**4 puntos**] Calcular $y_{k+1} - y_k$ y $x_{k+1} - x_k$ para $k \in \{0, 1, \ldots, n\}$

```
delta_y = np.diff(y_k)
delta_x = np.diff(x_k)
```

- [**6 puntos**] Aproximación del valor absoluto de la derivada en x_k para $k \in \{0, 1, \ldots, n+1\}$

```
z = np.zeros(n + 2)
z[:n+1] = np.abs(delta_y) / delta_x
% using backward difference for the last node
z[n+1] = np.abs(y_k[n+1] - y_k[n])/delta_x[n]
```

- [**8 puntos**] Utilizar los pesos modificados y calcular la aproximación de la integral mediante el producto punto.

```
w = np.zeros(n+2)
w[0] = delta_x[0] / 2
w[-1] = delta_x[-1] / 2
w[1:-1] = (delta_x[:n] + delta_x[1:]) / 2
total_var = np.dot(w,z)
```

- [**5 puntos**] Evaluar la ecuación diferencial en el punto x_j

```
yp = delta_y[j] / delta_x[j]
ode = alpha*AGSD(x_k[j-1],y_k[j-1],x_k[j],y_k[j],x_k[j+1],y_k[j+1]) + beta*yp + gamma*y_k[j]
```

- [**2 puntos**] Verificar si en el punto x_j, la data aproxima la ecuación diferencial entregada,

```
check = np.abs(ode - 1) < delta
return total_var,check
```

2. **Desarrollo Pregunta 2**:

(a)
- **[12 puntos]** Dado que no tenemos acceso a la matriz A de forma explícita, es decir no está almacenada en memoria (sea RAM o disco o cualquier medio de almacenamiento), no podemos operar directamente con ella. Esto significa que no es posible hacer operaciones filas o columnas sobre ella. Lo que sí sabemos es que la matriz A se define como la suma de 2 matrices, la matriz B_1 y B_2, donde la matriz B_1 es *sparse* y sí está almacenada en memoria de forma explícita pero la matriz B_2 no. Para operar con la matrix B_2 tenemos acceso a la función SFP($\mathbf{v}$), que dado un vector $\mathbf{v}$ entrega el producto entre B_2 y $\mathbf{v}$. Entonces, para poder obtener el producto entre A y un vector arbitrario $\mathbf{v}$ lo podemos construir de la siguiente forma:

$$\begin{aligned} A\,\mathbf{v} &= (B_1 + B_2)\,\mathbf{v} \\ &= B_1\,\mathbf{v} + B_2\,\mathbf{v} \\ &= B_1\,\mathbf{v} + \mathrm{SFP}(\mathbf{v}), \end{aligned}$$

  donde A, B_1 y B_2 pertenecen a $\mathbb{R}^{n\times n}$, y $\mathbf{v} \in \mathbb{R}^n$. Lo que implica que tenemos a nuestra disposición el operador $A\,\mathbf{v}$, es decir el producto entre A y el vector $\mathbf{v}$ por medio de B_1 y SFP$(\cdot)$. Con estos antecedentes a nuestra disposición y dentro de los algoritmos estudiados, el algoritmo que se adapta idealmente a este caso es GMRes.
- **[12 puntos]** Ahora, recordemos que lo que se debe resolver es el sistema $A^l\,\mathbf{x} = \mathbf{b}$, por lo tanto, se necesita un algoritmo que nos permita calcular $A^l\,\mathbf{v}$ para cualquier $\mathbf{v} \in \mathbb{R}^n$ para luego conectarlo con GMRes, lo cual se puede lograr de la siguiente forma considerando que $l \in \mathbb{N}$:

$$\begin{aligned} A^l\,\mathbf{v} = A^{l-1}\,(A\,\mathbf{v}) &= A^{l-1}\underbrace{(B_1\,\mathbf{v} + \mathrm{SFP}(\mathbf{v}))}_{\mathbf{y}_1} = A^{l-2}\,(A\,\mathbf{y}_1) \\ &= A^{l-2}\underbrace{(B_1\,\mathbf{y}_1 + \mathrm{SFP}(\mathbf{y}_1))}_{\mathbf{y}_2} = A^{l-3}\,(A\,\mathbf{y}_2) \\ &= A^{l-3}\underbrace{(B_1\,\mathbf{y}_2 + \mathrm{SFP}(\mathbf{y}_2))}_{\mathbf{y}_3} \\ &\vdots \\ &= A\underbrace{(B_1\,\mathbf{y}_{l-2} + \mathrm{SFP}(\mathbf{y}_{l-2}))}_{\mathbf{y}_{l-1}} = B_1\,\mathbf{y}_{l-1} + \mathrm{SFP}(\mathbf{y}_{l-1}) \end{aligned}$$

  Notar entonces que podemos calcular $A^l\,\mathbf{x}$ como una sucesión de productos matriz-vector, sin necesidad de almacenar de forma explícita la matriz A o la matriz A^l. Un algoritmo para calcular $A^l\,\mathbf{v}$ tendría la siguiente forma:
  - Calcular el vector $\mathbf{y}_1 = B_1\,\mathbf{v} + \mathrm{SFP}(\mathbf{v})$.
  - Calcular $\mathbf{y}_{k+1} = B_1\,\mathbf{y}_k + \mathrm{SFP}(\mathbf{y}_k)$ para k desde 1 hasta $l-1$.

  Notar que no es necesario almacenar todos los vectores $\mathbf{y}_k$, simplemente se puede sobre-escribir en un vector único, digamos $\mathbf{y}$, pero por claridad de la explicación se presenta como una secuencia de vectores. El procedimiento anterior se puede almacenar en una función, digamos `afun` que recibe el vector $\mathbf{v}$ y retorna el vector $A^l\,\mathbf{v}$.
- **[6 puntos]** Por lo tanto, podemos utilizar el algoritmo de GMRes para obtener una aproximación $\tilde{\mathbf{x}}$, entregando a GMRes `afun` que obtiene $A^l\,\mathbf{v}$, el lado derecho $\mathbf{b}$, un *initial guess* $\mathbf{x}_0$ y una tolerancia δ, de tal forma que el algoritmo entrega la aproximación $\tilde{\mathbf{x}}$ asegurando que $\|\mathbf{b} - A\,\tilde{\mathbf{x}}\| \leq \delta$.

(b)
```
'''
input:
n       : (integer) dimension of the problem.
l       : (integer) Integer value of l.
B_1     : (ndarray) Sparse matrix B_1. Notice this is defined for simplicity as ndarray.
SFP     : (callable) Procedure that receives the vector 'v' and
computes the 'fast' product between B_2 and the vector 'v'.
b       : (ndarray) Right-hand-side vector.
delta   : (double) Numerical threshold for computation of residual.

output:
xt       : (ndarray) Array with the values of the numerical approximation x_tilde.
'''
def find_x(n,l,B_1,SFP,b,delta):
```

- [**10 puntos**] Calcular $A\,\mathbf{v}$ para algún vector $\mathbf{v}$.
  ```
  afun1 = lambda v: np.dot(B_1,v) + SFP(v)
  ```
- [**10 puntos**] Calcular $A^l\,\mathbf{v}$ para algún vector $\mathbf{v} \in \mathbb{R}^n$
  ```
  def afun2(v):
      y = afun1(v)
      for i in np.arange(l-1):
        y = afun1(y)
      return y
  ```
- [**5 puntos**] Definir el *initial guess*, utilizar GMRes y retornar la aproximación numérica `xt`.
  ```
  x0 = np.zeros(n)
  xt = GMRes_matrix_free(afun2,b,x0,delta)
  return xt
  ```