- **Desarrollo Pregunta 1**:

(a) **[25 puntos]**

De los antecedentes entregados en el contexto notamos que:

- $1 \leq t \leq 3$.
- $0 \leq y \leq \log(3)$. *Se agrega por completitud dado que se usará en la siguiente pregunta.*
- $\dfrac{\mathrm{d}^n\,(\log(t))}{\mathrm{d}t^n} = \dfrac{(-1)^{n-1}\,(n-1)!}{t^n}$.

De lo cual podemos concluir que dado el dominio de t, la menor cota superior para el valor absoluto de la n-ésima derivada de $\log(t)$ es,

$$\begin{aligned}\left|\frac{\mathrm{d}^n\,(\log(t))}{\mathrm{d}t^n}\right| = \left|\frac{(-1)^{n-1}\,(n-1)!}{t^n}\right| &\leq \left|\frac{(-1)^{n-1}\,(n-1)!}{1^n}\right|\\ &\leq \left|\frac{(n-1)!}{1^n}\right|\\ &\leq (n-1)!,\end{aligned}$$

donde $(\cdot)!$ corresponde al operador factorial.

Entonces, la propuesta consiste en:

- Utilizar puntos de Chebyshev para interpolar $\log(t)$ en $[1,3]$.

Lo cual nos permite acotar la menor cota superior del error de la siguiente forma,

$$\begin{aligned}\left|\log(t) - p_n^{\log}(t)\right| &= \frac{|(t-t_1)\,\dots\,(t-t_n)|}{2^{n-1}\,n!}\left|\left.\frac{\mathrm{d}^n\,(\log(t))}{\mathrm{d}t^n}\right|_{t=c}\right|\\ &\leq \frac{\left(\dfrac{3-1}{2}\right)^n}{2^{n-1}\,n!}(n-1)!\\ &\leq \frac{1}{2^{n-1}\,n} \leq \varepsilon.\end{aligned}$$

En resumen:

- Dominio de interpolación: $[1,3]$.
- Menor cota superior del error: $\dfrac{1}{2^{n-1}\,n}$.
- Se debe elegir el número de puntos n a interpolar tal que $\dfrac{1}{2^{n-1}\,n} \leq \varepsilon$.

(b) **[25 puntos]**

```
'''
input:
N       : (int) Degree of p(y).
n_log   : (int) Number of nodes to be used for the interpolation of the log function.

output:
p_fast  : (callable) A fast implementation of p(y).
log_fast : (callable) A fast implementation of log(t).
'''
def build_fast_g(N,n_log):

    # Interpolación de p(y). Se usan N+1 puntos para
    # interpolar exactamente un polinomio de grado N.
    y_cheb_p = myChebyshev(0,log_expensive(3),N+1)
    py_cheb_p = p_expensive(y_cheb_p)
    p_fast = BarycentricInterpolation(y_cheb_p,py_cheb_p)

    # Interpolación de log(t).
    t_cheb_log = myChebyshev(1,3,n_log)
    y_cheb_log = log_expensive(t_cheb_log)
    log_fast = BarycentricInterpolation(t_cheb_log,y_cheb_log)

    return p_fast, log_fast
```

- **Desarrollo Pregunta 2**:

(a) **[20 puntos]**

Alternativa 1:

- Ventajas:
    - Se procesa una cantidad finita de vectores, por lo cual asegura que el algoritmo terminará después de una cantidad finita de operaciones.
    - Solo se utiliza $\widecheck{Q}_n$ para obtener $C = I_m - \widehat{Q}_n\,\widehat{Q}_n^\top$.
    - La secuencia de vectores $\mathbf{c}_i$ solo pertenecen a Range($\widecheck{Q}_{m-n}$).
- Desventajas:
    - Se requiere construir $C = I_m - \widehat{Q}_n\,\widehat{Q}_n^\top$ en memoria.

Alternativa 2:

- Ventajas:
    - No se requiere definir una estructura particular a la secuencia de vectores generados.
- Desventajas:
    - No se asegura que el algoritmo finalizará luego de una cantidad finita de vectores, por lo cual podría ejecutarse por mucho tiempo hasta encontrar los $m-n$ vectores ortonormales para construir $\widecheck{Q}_{m-n}$.

Alternativa 3:

- Ventajas:
    - Se procesa una cantidad finita de vectores, por lo cual asegura que el algoritmo terminará después de una cantidad finita de operaciones.
    - Requiere el uso de $C = I_m$.
    - Utiliza $\widehat{Q}_n$ previamente obtenida.
- Desventajas:
    - La secuencia de vectores canónicos pertenecen a $\mathbb{R}^m$, es decir contienen componentes pertenecientes al Range($\widehat{Q}_n$) y al Range($\widecheck{Q}_{m-n}$).

(b) **[5 puntos]**

Para determinar numéricamente si un vector $\mathbf{c}$ es una combinación lineal de los vectores $\mathbf{q}_k$ ortonormales indicados, debemos encontrar los coeficientes α_k indicados que minimizan el error cuadrático. Teóricamente se puede resolver utilizando las ecuaciones normales asociadas para minimizar el residuo $\mathbf{r} = \mathbf{c} - \widehat{Q}_n\,\boldsymbol{\alpha}$, es decir,

$$\begin{aligned}\widehat{Q}_n^\top \widehat{Q}_n\,\overline{\boldsymbol{\alpha}} &= \widehat{Q}_n^\top\,\mathbf{c}, \overline{\boldsymbol{\alpha}}\\ &= \widehat{Q}_n^\top\,\mathbf{c}.\end{aligned}$$

Luego se necesita obtener $\mathbf{r}_{\min} = \mathbf{c} - \widehat{Q}_n\,\overline{\boldsymbol{\alpha}}$. Entonces, si $\|\mathbf{r}_{\min}\| < \gamma$ se concluye que existe dependencia lineal "numérica".

Desde el punto de vista de código, uno puede modificar la ortonormalización de Gram-Schmidt "modificada" para obtener los coeficientes:

```
def determine_linear_independence(Q,n,c,gamma):
    alphas = np.zeros(n)
    for i in np.arange(k):
        alphas[k] = np.dot(Q[:,i],c)
        c=c-alphas[k]*Q[:,i]
    if np.linalg.norm(c)<gamma:
        return True
    return False
```

(c) **[25 puntos]**

```
'''
input:
A       : (ndarray) Input matrix A of size m x n.
m       : (int) Number of rows of matrix A.
n       : (int) Number of columns of matrix A.
Qhat    : (ndarray) Matrix Qhat of 'reduced' QR of A, such that A=Qhat @ Rhat.
gamma   : (float) Threshold to determine linear independence.

output:
Qcheck  : (ndarray) The Qcheck matrix described before.
'''
def find_Qcheck1(A,m,n,Qhat,gamma=1e-12):
C = np.eye(m)-np.dot(Qhat,Qhat.T)
Qcheck = np.zeros((m,m-n))
rs = np.zeros(m)
l = 0
for k in np.arange(m):
    y = C[:,k]
    for i in np.arange(l):
        rs[i] = np.dot(Qcheck[:,i],y)
        y=y-rs[i]*Qcheck[:,i]
    norm_y = np.linalg.norm(y)
    if norm_y>=gamma:
        Qcheck[:,l] = y/norm_y
        l = l+1
    if l == m-n+1:
        break
return Qcheck

def find_Qcheck3(A,m,n,Qhat,gamma=1e-12):
    C = np.eye(m)
    Q = np.zeros((m,m))
    Q[:,:n] = Qhat
    rs = np.zeros(m)
    l = 0
    for k in np.arange(m):
        y = C[:,k]
        for i in np.arange(n+l):
            rs[i] = np.dot(Q[:,i],y)
            y=y-rs[i]*Q[:,i]
        norm_y = np.linalg.norm(y)
        if norm_y>=gamma:
            Q[:,n+l] = y/norm_y
            l = l+1
        if l == m-n+1:
            break
    Qcheck = Q[:,n:]
    return Qcheck
```