
- **Contexto Pregunta 1**: Considere que tiene a su disposición la estructura de la siguiente interpolación **no-polinomial**,

$$g(t) = a_0 + \sum_{i=1}^{N} a_i \, (\log(t))^i \,,$$

es decir, es una interpolación log-polinomial. En particular, se considera valores de $t \in \mathbb{R}$ restringidos a $0 \leq \log(t) \leq \log(3)$.

Una alternativa para convertir la función anterior en un polinomio es utilizar una expansión en series de Taylor entorno a $t = 1$, es decir,

$$\log(t) = (t-1) - \frac{1}{2}(t-1)^2 + \frac{1}{3}(t-1)^3 - \frac{1}{4}(t-1)^4 + \frac{1}{5}(t-1)^5 + \dots .$$

La cual podría usarse para cada término $(\log(t))^i$. Sin embargo, esta opción nos entrega un expresión con infinitos términos y no nos permite acotar el error de alguna forma. En la práctica uno podría truncarla y obtener una aproximación finita, pero lamentablemente el error es bajo solamente alrededor de $t = 1$.

Para el manejo de la interpolación log-polinomial, se propone el siguiente cambio de variable,

$$y = \log(t).$$

Lo cual implica que podemos re-escribir la función anterior de la siguiente forma,

$$p(y) = a_0 + \sum_{i=1}^{N} a_i \, y^i.$$

La cual es simplemente una aproximación **polinomial** en "y"!

En resumen, y como es tradicional, para la implementación computacional de la función log-polinomial $g(t)$ solo podemos utilizar la operaciones elementales. Esto significa que, adicional a la aproximación polinomial de $p(y)$, necesitamos aproximar computacionalmente el cambio de variable $y = \log(t)$.

Para hacer efectiva la construcción de la aproximación de $g(t)$ considere que tiene a su disposición las siguientes funciones "costosas" de evaluar: $g_{\text{expensive}}(t)$, $p_{\text{expensive}}(y)$, y $\log_{\text{expensive}}(t)$. Al indicar que son "costosas" significa que no queremos utilizarlas más de lo necesario. Esto significa que en la práctica no podemos utilizar $g_{\text{expensive}}(t)$ o $p_{\text{expensive}}(\log_{\text{expensive}}(t))$ para obtener el valor de $g(t)$, pero sí podemos utilizarlas para evaluaciones "puntuales".

- **Contexto Pregunta 2**: Considere que tiene a su disposición la siguiente familia de problemas de mínimos cuadrados:

$$\mathbf{r}_k = \mathbf{b}_k - A\,\mathbf{x}_k, \tag{1}$$

donde $A \in \mathbb{R}^{m \times n}$, $\mathbf{b}_k \in \mathbb{R}^m$, $\mathbf{r}_k \in \mathbb{R}^m$, $\mathbf{x}_k \in \mathbb{R}^n$, y $k \in \{1, 2, 3, \dots\}$. Dado que es un problema de mínimos cuadrados, podemos obtener la secuencia de vectores $\overline{\mathbf{x}}_k$ que minimizan el error cuadrático asociado de la siguiente forma,

$$\overline{\mathbf{x}}_k = \underset{\mathbf{x} \in \mathbb{R}^n}{\operatorname{argmin}} \, \|\mathbf{r}_k\|_2^2 = \underset{\mathbf{x} \in \mathbb{R}^n}{\operatorname{argmin}} \, \|\mathbf{b}_k - A\,\mathbf{x}_k\|_2^2 \,.$$

En particular, podemos obtener $\overline{\mathbf{x}}_k$ por medio de la factorización QR de la matriz A, es decir,

$$\overline{\mathbf{x}}_k = \widehat{R}_n^{-1} \, \widehat{Q}_n^{\top} \, \mathbf{b}_k, \tag{2}$$

donde $A = \widehat{Q}_n \, \widehat{R}_n$, es decir, la factorización QR reducida de A y $^{\top}$ corresponde al operador transpuesta. Ahora, reemplazando la solución de mínimos cuadrados (2) en el vector residual (1) y considerando también que A es igual a $\widehat{Q}_n \, \widehat{R}_n$, se obtiene,

$$\begin{aligned}
\mathbf{r}_k^{\text{mín}} &= \mathbf{b}_k - A\,\overline{\mathbf{x}}_k = \\
&= \mathbf{b}_k - A\,\widehat{R}_n^{-1} \, \widehat{Q}_n^{\top} \, \mathbf{b}_k, \\
&= \mathbf{b}_k - \widehat{Q}_n \, \widehat{R}_n \, \widehat{R}_n^{-1} \, \widehat{Q}_n^{\top} \, \mathbf{b}_k, \\
&= \mathbf{b}_k - \widehat{Q}_n \, \widehat{Q}_n^{\top} \, \mathbf{b}_k, \\
&= \left(I_m - \widehat{Q}_n \, \widehat{Q}_n^{\top} \right) \, \mathbf{b}_k,
\end{aligned}$$

donde I_m es la matriz identidad de dimensión $m \times m$. Notar que $\widehat{Q}_n \, \widehat{Q}_n^\top \neq I_m$ si $n < m$, por lo cual no se cancela I_m con $\widehat{Q}_n \, \widehat{Q}_n^\top$. Sin embargo, sí se obtiene la identidad si $n = m$, es decir $\widehat{Q}_m \, \widehat{Q}_m^\top = I_m$. Por otro lado, si el operador transpuesta es aplicado a la "primera" matriz y no a la "segunda" matriz del producto, $\widehat{Q}_n^\top \, \widehat{Q}_n$, se obtiene que es igual a I_n para todo n, lo cual genera la siguiente identidad $\widehat{Q}_n^\top \, \widehat{Q}_n = I_n$. Ahora, uno puede re-escribir el operador $\left(I_m - \widehat{Q}_n \, \widehat{Q}_n^\top\right)$ de la siguiente forma,

$$\begin{aligned} \mathbf{r}_k^{\text{mín}} &= \left(I_m - \widehat{Q}_n \, \widehat{Q}_n^\top\right) \mathbf{b}_k \\ &= \breve{Q}_{m-n} \, \breve{Q}_{m-n}^\top \, \mathbf{b}_k, \end{aligned}$$

donde $\breve{Q}_{m-n}$ es la parte **tradicionalmente no usada** de Q_m de la factorización QR "full" (o completa) de A, es decir,

$$\begin{aligned} A &= Q_m \, R_{m\times n} \leftarrow \text{Factorización QR "full"}, \\ &= \underbrace{\left[\widehat{Q}_n \quad | \quad \breve{Q}_{m-n}\right]}_{Q_m} \underbrace{\begin{bmatrix} \widehat{R}_n \\ \underline{\underline{0}} \end{bmatrix}}_{R_{m\times n}} \\ &= \underbrace{\widehat{Q}_n \, \widehat{R}_n}_{\text{QR reducida}} + \underbrace{\breve{Q}_{m-n} \, \underline{\underline{0}}}_{\text{producto matricial}}, \end{aligned}$$

donde $\underline{\underline{0}}$ es la matriz nula de dimensión $(m-n) \times n$. *Como dato adicional, se conoce que $m-n$ es un número entero pequeño, por lo cual, para ahorrar memoria en el almacenamiento de $\left(I_m - \widehat{Q}_n \, \widehat{Q}_n^\top\right)$ se necesita utilizar su representación equivalente $\breve{Q}_{m-n} \, \breve{Q}_{m-n}^\top$. En resumen, el problema consiste en construir una versión "comprimida" de la matriz $\left(I_m - \widehat{Q}_n \, \widehat{Q}_n^\top\right)$, es decir $\breve{Q}_{m-n} \, \breve{Q}_{m-n}^\top$*. Notar que es solo necesario obtener $\breve{Q}_{m-n}$ y no realizar el producto $\breve{Q}_{m-n} \, \breve{Q}_{m-n}^\top$.
Algunas observaciones:

- Conocemos que: Range $\left(I_m - \widehat{Q}_n \, \widehat{Q}_n^\top\right)$ = Range $\left(\breve{Q}_{m-n} \, \breve{Q}_{m-n}^\top\right)$ = Range $\left(\breve{Q}_{m-n}\right)$.
- $\left(I_m - \widehat{Q}_n \, \widehat{Q}_n^\top\right)$ es singular.
- $\left(I_m - \widehat{Q}_n \, \widehat{Q}_n^\top\right)$ no es *full rank*.
- Rank $\left(I_m - \widehat{Q}_n \, \widehat{Q}_n^\top\right) = m - n$.
- $\left(I_m - \widehat{Q}_n \, \widehat{Q}_n^\top\right)$ tiene $m - n$ columnas linealmente independientes.

Entonces, para obtener $\breve{Q}_{m-n}$ se debe "adaptar" la ortonormalización de Gram-Schmidt. La adaptación consiste en que a partir de una secuencia de vectores, que – idealmente – sean linealmente independientes, se les vaya removiendo las proyecciones sobre los vectores ya conocidos para que finalmente se pueda normalizar el vector resultante (si es que no es el vector nulo) y así generar un nuevo vector ortonormal. El proceso debe repetirse hasta obtener la cantidad de nuevos vectores ortonormales necesarios.

Por ejemplo, tenemos varias alternativas para generar secuencias finitas o infinitas de vectores en $\mathbb{R}^m$:

- **Alternativa 1**: Los vectores columnas de la matriz $\left(I_m - \widehat{Q}_n \, \widehat{Q}_n^\top\right)$.
- **Alternativa 2**: Generar vectores aleatorios, digamos $\mathbf{w}$, tantas veces como sea requerido.
- **Alternativa 3**: Otra alternativa, que no depende de generar vectores aleatorios, es elegir una secuencia de vectores linealmente independientes que sean una base de $\mathbb{R}^m$. En particular, se pueden considerar los vectores canónicos $\mathbf{e}_i \in \mathbb{R}^m$, los cuales se definen con un 1 en la coordenada i y 0 en todas las otras coordenadas, para $i \in \{1, 2, 3, \ldots, m\}$.

NOMBRE:_____ ROL:_____ PARALELO:_____

- **Desarrollo Pregunta 1**:

(a) **[25 puntos]** Proponga una aproximación numérica por medio de una interpolación polinomial del cambio de variable $y = \log(t)$ en el dominio correspondiente. Usted debe obtener la menor cota superior del máximo error de la aproximación tal que se pueda acotar por ε.

Usted debe indicar explícitamente,

- Dominio de interpolación.
- La menor cota superior del error, considerando todos los parámetros necesarios.
- Cómo se conecta la menor cota superior con el parámetro ε. Notar que no debe calcular el valor del n correspondiente, solo indicar cómo se conectan.

Hint: Recall that:

$$\begin{aligned}
\frac{\mathrm{d}\left(\log(t)\right)}{\mathrm{d}t} &= \frac{1}{t}, \\
\frac{\mathrm{d}^2\left(\log(t)\right)}{\mathrm{d}t^2} &= (-1)\frac{1}{t^2} &&= -\frac{1}{t^2}, \\
\frac{\mathrm{d}^3\left(\log(t)\right)}{\mathrm{d}t^3} &= (-1)^2\frac{2}{t^3} &&= \frac{2}{t^3}, \\
\frac{\mathrm{d}^4\left(\log(t)\right)}{\mathrm{d}t^4} &= (-1)^3\frac{2\cdot 3}{t^4} &&= -\frac{6}{t^4}, \\
&\vdots
\end{aligned}$$

(b) **[25 puntos]** Implemente la aproximación de la función $\log(t)$ y $p(y)$ considerando:

- Que el error de interpolación se reduzca al aumentar la cantidad de puntos de interpolación o que sea 0.
- Que su implementación minimice el uso de operaciones elementales al implementar las interpolaciones respectivas.

Para su implementación, considere que **solo** tiene a su disposición las siguientes funciones de la librería NumPy, además de las operaciones elementales, ciclos y condicionales propios de Python:

- `np.arange(n)`: Para `n` un número entero positivo entrega un vector de largo `n` con números enteros desde `0` a `n-1`.
- `np.ceil(a)`: Entrega la parte entera superior de cada elemento en el vector `a` o, respectivamente, la parte entera superior del escalar `a`.
- `np.linspace(a, b, n)`: Entrega `n` puntos equiespaciados en el dominio $[a, b]$.
- `{pV,pL,pB}={Vandermonde,Lagrange,BarycentricInterpolation}(xi,yi)`: Esta función recibe los puntos de interpolación x_i en el vector `xi` e y_i en el vector `yi`, y retorna la función *callable* {`pV`,`pL`,`pB`} que se construyó utilizando la interpolación con la {matriz de Vandermonde, interpolación de Lagrange, interpolación Baricéntrica}.
- `np.cos(x)`: Evalúa la función $\cos(x)$, donde `x` puede ser un vector o un escalar.
- `myChebyshev(a, b, n)`: Entrega los `n` puntos de Chebyshev en el intervalo definido $[a, b]$. Si se utiliza `myChebyshev(-1, 1, n)`, se retorna los puntos de Chebyshev "originales".
- {`g_expensive(t)`, `p_expensive(y)`, `log_expensive(t)`}: Retorna la evaluación **costosa** de la función $\{g_{\text{expensive}}(t)$ en t, $p_{\text{expensive}}(y)$ en y, $\log_{\text{expensive}}(t)$ en $t\}$, por lo que se quiere reducir su uso. Sin embargo se puede usar de forma *puntual*.

Notar que las funciones `pV`, `pL`, `pB`, `g_expensive(t)`, `p_expensive(y)`, y `log_expensive(t)` están vectorizadas, por lo que pueden recibir un vector `x` y retorna un vector `y` donde se evaluó el polinomio interpolador para cada elemento de `x`.

Al momento de implementar usted **debe decidir** qué componentes se deben vectorizar y qué componentes no, considerando las funciones de NumPy antes mencionadas. Considere la siguiente firma:

```
'''
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):
    # Your own code.
```

```
    return p_fast, log_fast
```

NOMBRE:_____ ROL:_____ PARALELO:_____

- **Desarrollo Pregunta 2**:

  (a) **[20 puntos]** Explique ventajas y desventajas de cada una de las 3 alternativas propuestas para la generación de secuencias de vectores para obtener $\breve{Q}_{m-n}$, e indique cuál alternativa usaría.

  (b) **[5 puntos]** Explique claramente cómo determinar "numéricamente" si un vector $\mathbf{c}$ es o no una combinación lineal de un conjunto de l vectores ortonormales $\mathbf{q}_k$, para $k \in \{1, 2, \ldots, l\}$ utilizando la **ortonormalización de Gram-Schmidt *"modificada"***. Para determinar "numéricamente" si efectivamente es una combinación lineal deberá obtener $\text{mín}_{\alpha_1,\ldots,\alpha_l} \left\| \mathbf{c} - \sum_{k=1}^{l} \alpha_k \, \mathbf{q}_k \right\|_2$ y comprobar si es menor a $\gamma = 10^{-12}$, en caso contrario no es linealmente dependiente.
  Hint: This is a well-known problem, usually called squares-least by their friends!

(c) **[25 puntos]** Implemente en Python utilizando adecuadamente la librería NumPy (en especial su capacidad de vectorización) la alternativa elegida para la obtención de la matriz $\breve{Q}_{m-n}$ utilizando el desarrollo teórico de las preguntas previas. **Recuerde que debe utilizar Gram-Schmidt modificado.** Para su implementación, considere que **solo** tiene a su disposición las siguientes funciones de la librería NumPy, además de las operaciones elementales, ciclos y condicionales propios de Python:

- `np.arange(n)`: Para `n` un número entero positivo entrega un vector de largo `n` con números enteros desde `0` a `n-1`.
- `np.arange(n1,n2)`: Para `n1` y `n2` números enterros positivo y `n2`>`n1` entrega un vector de largo `n2-n1` con números enteros desde `n1` a `n2-1`.
- `np.abs(x)`: Entrega el valor absoluto de `x` de forma *element-wise*.
- `np.dot(a,b)`: Obtiene el producto interno entre el vector `a` y `b`. En caso de que `a` sea una matriz, entrega el producto matriz-vector respectivo. Para esto último también es posible utilizar el operador `@`.
- `np.zeros(m)`: Genera un vector nulo de dimensión `m`.
- `np.zeros((m, n))`: Genera una matriz nula de dimensión $m \times n$.
- `np.eye(k)`: Genera la matriz identidad I_k de dimensión $k \times k$.
- `myRandom(m)`: Genera un vector pseudo-aleatorio de dimensión `m`.
- `myRandom((m,n))`: Genera una matriz pseudo-aleatoria de dimensión $m \times n$.
- `np.linalg.norm(x)`: Obtiene la norma 2 del vector `x`.
- `np.transpose(A)`: Obtiene la transpuesta de la matriz `A`, lo cual es equivalente a `A.T`.

Notar que al momento de implementar usted **debe decidir** qué componentes se deben vectorizar y qué componentes no, considerando las funciones de NumPy antes mencionadas. Considere la siguiente firma:

```
'''
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_Qcheck(A,m,n,Qhat,gamma=1e-12):
    # Your own code.




    return Qcheck
```