Sea $A \in \mathbb{R}^{m \times n}$ una matriz que recibe como input. Construya un algoritmo que entregue los índices de columnas linealmente independientes de $A$.

Considere que la primera columna de la matriz $A$ **no** es el vector nulo y que su procedimiento analiza la independencia lineal de izquierda a derecha, por ejemplo, para la siguiente matriz,

$$A_0 = \begin{pmatrix} 0.5 & 0 & 20 \\ 0 & 0.2 & -4 \\ 0 & 0 & 0 \end{pmatrix}$$

su resultado debe ser el vector li=[0,1].

Esto significa que la primera columna y la segunda columna de $A_0$ son linealmente independientes, sin embargo la tercera columna es linealmente dependiente respecto de las anteriores. La firma de la función debe ser la siguiente,

```
def find_li_columns_indices(A,delta):
    # Your own code. It may be useful to consider the modified Gram-Schmidt algorithm!
    return li
```

donde delta=$\delta$ se definirá a continuación.

Recordando que la dependencia lineal se puede obtener por medio de encontrar una solución no nula de los valores $\alpha_i$, para $i \in \{1, 2, \ldots, k\}$, de la siguiente ecuación:

$$\alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \cdots + \alpha_k \mathbf{v}_k = \mathbf{0}$$

Sin embargo, dado que estamos trabajando con *double precision*, consideraremos la dependencia lineal de la siguiente forma,

$$\|\alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \cdots + \alpha_k \mathbf{v}_k\|_2 < \delta$$

es decir, definiremos un umbral denominado $\delta$. Esto significa que se considerará que existe algún vector linealmente dependiente si es que la norma-2 de $\alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \cdots + \alpha_k \mathbf{v}_k$ es menor a $\delta$. Esto relaja un poco la noción de dependencia lineal pero de todos modos se cumple el objetivo si es que se utiliza el valor de $\delta$ adecuado. De todos modos considere que no pueden haber más de $m$ vectores linealmente independientes, por lo tanto su implementación debe detenerse si la cantidad de vectores linealmente independientes es $m$ y debe retornar esa lista de coeficientes.

**Jupyter notebook con código base para el desarrollo de su respuesta:** LINK

1. **[5 puntos]** [Verdadero y Falso] Si $n > m$, ¿Es posible solo considerar las primeras $n$ columnas de $A$ para determinar que columnas son linealmente independiente? **Ingrese 0 si es Falso y 1 si es Verdadero.**

1

La aseveración es falsa, por lo que la respuesta es 0.

2. **[5 puntos]** [Verdadero y Falso] Si $n > m$, ¿Es realmente necesario analizar todas las columnas de $A$ para obtener la lista de columnas linealmente independientes? **Ingrese 0 si es Falso y 1 si es Verdadero.**

0

La aseveración es verdadera, por lo que la respuesta es 1.

3. **[30 puntos]** Implemente *find_li_columns_indices*. Esto se ejecutará offline y se ingresará el puntaje posteriormente.

4. **[5 puntos]** Indique cuales son las columnas linealmente independientes de la matriz $A_1$ con $\delta = 1e-10$,

$$A_1 = \begin{pmatrix} 0.5 & 0 & 0.1 & 0 & -1 \\ 0 & 0.2 & -0.4 & 1 & 0 \\ 0 & 1 & -2 & 0 & 0 \\ 1 & 2 & -3.8 & 2 & 1 \\ -1 & 2 & -4.2 & 2 & -1 \end{pmatrix}$$

Nota: El jupyter notebook contiene la definición de la matriz $A_1$ en formato de np.array.

**Por ejemplo, si la salida de su código entrega [0,4], usted debiera ingresa un 0 en la primera casilla, un 4 en la segunda, y -1 en las demás casillas.**

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 3 | 4 | -1 |

El output de la función debería haber sido [0 1 3 4], por lo que el mismo arreglo con un -1 al final para completar era la respuesta esperada de acuerdo al ejemplo dado.

5. **[5 puntos]** Considere $A_2$ entregado en el jupyter notebook: ¿Cuántas columnas linealmente independiente tiene $A_2$ con $\delta = 10^{-10}$? **La respuesta a ingresar debe ser un número natural.**

126

El output de la función debería haber sido un arreglo de largo 126, por lo cual ese sería le numero de vectores linealmente independientes en A2.

6. **[10 puntos]** Considere $A_3$ entregado en el jupyter notebook: ¿Cuántas columnas linealmente independiente tiene $A_3$ con $\delta = 10^{-10}$? **La respuesta a ingresar debe ser un número natural.**

128

El output de la función debería haber sido un arreglo de largo 128, por lo cual ese sería le numero de vectores linealmente independientes en A3.

En la primera parte del curso estudiamos varios algoritmos de búsqueda de ceros. Una posible aplicación de búsqueda de ceros es la construcción de la función inversa de cualquier función $f(x)$ inyectiva, es decir $f^{-1}(y)$. Por ejemplo, si consideramos la búsqueda de ceros de la siguiente función $g(x) = \exp(x) - y$ dado $y$ como un valor conocido, entonces lo que estamos obteniendo con la búsqueda de ceros es el valor $x = \log(y)$. Esto corresponde efectivamente a la función inversa de la función exponencial $f(x) = \exp(x)$, es decir $f^{-1}(y) = \log(y)$. Recuerde que la relación entre una función y su inversa cumple que $f^{-1}(f(x)) = x$ ó $f(f^{-1}(x)) = x$. Entonces la búsqueda de ceros de la función $g(x) = f(x) - y$ entrega $x = f^{-1}(y)$. Este procedimiento es bastante efectivo cuando uno quiere aplicarlo unas pocas veces, sin embargo podría ser muy costoso cuando se requiere aplicar muchas veces. Por ejemplo, si quisiéramos aplicar la función inversa $f^{-1}(x)$ para valores de $x$ en un intervalo definido, digamos $[a, b]$, entonces podría convenir utilizar otra familia de algoritmos, como por ejemplo interpolación polinomial. Recuerde que en este caso $\log(y)$ es el logaritmo natural.

En particular, utilizaremos $f(x) = \exp(x)$ para $x \in [-1, 1]$ y su función inversa $f^{-1}(y) = \log(y)$. Esto significa que solo aplicaremos la función inversa $f^{-1}(y)$ a valores de la función $f(x) = \exp(x)$ restringiendo $x$ al intervalo indicado. Esto es muy útil ya que podemos definir una cota de los valores que utilizaremos en el dominio de la función inversa $f^{-1}(y)$. Por ejemplo, si nosotros recibimos el valor $y = 1.6487212707001282$, entoces nos interesa obtener el valor de $x$ tal que $f(x) = 1.6487212707001282$, o también podemos decir $f^{-1}(1.6487212707001282) = x$. En este caso la solución es $0.5$ porque $f^{-1}(1.6487212707001282) = 0.5$.

La tarea que se encomienda es construir un polinomio interpolador de la función $f^{-1}(y)$, digamos el polinomio $\mathrm{pLog}_n(y)$, donde $n$ corresponde a la cantidad de puntos utilizados en la interpolación. Además se agrega la restricción que se pueden usar como mínimo 2 puntos de interpolación y como máximo 100 puntos de interpolación, este rango es muy útil para restringir nuestra búsqueda del número de puntos de interpolación.

Para evaluar la calidad del interpolador obtenido consideraremos la siguiente definición de error,

$$\mathrm{Error}(n) = \max_{x \in [-1,1]} |\mathrm{pLog}_n(\exp(x)) - x|.$$

Sin embargo, como queremos obtener una estimación solamente, se propone utilizar la siguiente discretización de la función de error $\mathrm{Error}(n)$,

$$\mathrm{ErrorD}(n) = \max_{i \in \{0,1,2,\ldots,N\}} |\mathrm{pLog}_n(\exp(x_i)) - x_i|$$

donde $x_i = -1 + \frac{2\,i}{N}$ y $N = 100000$, es decir los $x_i$ son puntos equiespaciados en $[-1, 1]$.

**Jupyter notebook con código base para el desarrollo de su respuesta:** LINK

1. **[10 puntos]** Construya una interpolación polinomial que interpole $\log(y)$ para el rango indicado anteriormente sin que se vea afectado por el fenómeno de Runge.

El algoritmo debe retornar una función *callable* y debe tener la siguiente firma:

```
def interpolateLog(n):
    # Your code
    return pLog
```

Por ejemplo, con la función *interpolateLog* podemos construir nuestro interpolador *pLog=interpolateLog(n)* y luego la evaluación *vectorial* se realiza de la siguiente forma *out=pLog(x)*, donde *x=np.linspace(a,b,10)*, para un intervalo $[a, b]$.

Como caso particular considere que *pLog=interpolateLog(5)*, entonces el resultado al evaluar *pLog([0.9, 1.5])* debe ser [-0.102978  0.40662723]. Esto se ejecutará offline y se ingresará el puntaje posteriormente.

2. **[10 puntos]** Implemente $\mathrm{ErrorD}(n)$, debe considerar la siguiente firma:

```
def ErrorD(n):
    # Yout code
    return discrete_error
```

Esto se ejecutará offline y se ingresará el puntaje posteriormente.

3. **[5 puntos]** Evalúe el error de interpolación discreto $\mathrm{ErrorD}(n)$ para $n = 1$. **Por ejemplo, si su respuesta es 25.231897698123746, usted debe ingresar 25.23189, es decir el número truncado a 5 decimales utilizando un punto como separador de decimales.**

1.43378

El output de la función debería haber sido 1.433780830483027, por lo que, de acuerdo a las instrucciones, la respuesta esperada sería 1.43378.

4. **[10 puntos]** Construya la siguiente función:

```
def find_n_log(threshold):
    # Your code
    return n
```

Esta función debe recibir la variable *threshold* que corresponde al máximo error discreto permitido y debe retornar el **mínimo** valor de $n$ tal que se cumpla con el error *threshold* definido como parámetro.

Por ejemplo, si la variable *threshold* se define como $1e-1$, usted debe retornar el **menor valor posible de** *n* tal que se cumpla que $\mathrm{ErrorD}(n) <$ *threshold*. En caso de que no pueda encontrar un valor de *n* que cumpla con el *threshold* requerido en el rango de valores de $n$ indicado, la función debe retornar el valor *-1*.

Esto se ejecutará offline y se ingresará el puntaje posteriormente.

5. **[10 puntos]** Obtenga el valor de *n* que cumplan con el *threshold=1e-8, es decir ejecute find_n_log(threshold).* ***La respuesta a ingresar debe ser un número natural.***

22

El output de la evaluación que se pregunta era 22, por lo que esta es la respuesta esperada.

6. **[5 puntos]** [Verdadero y Falso] ¿Sigue siendo correcto el análisis de error anterior si se alternar el orden de evaluación entre $\mathrm{pLog}(x)$ y $\exp(x)$ en la definición de $\mathrm{ErrorD}(n)$, es decir pasar de $|\mathrm{pLog}_n(\exp(x_i)) - x_i|$ a $|\exp(\mathrm{pLog}_n(x_i)) - x_i|$? **Ingrese 0 si es Falso y 1 si es Verdadero**.

0

La aseveración es falsa, por lo que la respuesta es 0.