Curso de Métodos Numéricos (DEMAT)

Tarea 4

Descripción:	Fechas
Fecha de publicación del documento:	**Septiembre 10, 2023**
Fecha límite de entrega de la tarea:	**Septiembre 17, 2023**

Indicaciones

Puede escribir el código de los algoritmos que se piden en una celda de este notebook o si lo prefiere, escribir las funciones en un archivo `.py` independiente e importar la funciones para usarlas en este notebook. Lo importante es que en el notebook aparezcan los resultados de la pruebas realizadas y que:

- Si se requieren otros archivos para poder reproducir los resultados, para mandar la tarea cree un archivo ZIP en el que incluya el notebook y los archivos adicionales.
- Si todos los códigos para que se requieren para reproducir los resultados están en el notebook, no hace falta comprimir el notebook y puede anexar este archivo en la tarea del Classroom.
- Exportar el notebook a un archivo PDF y anexarlo en la tarea del Classroom como un archivo independiente. **No incluya el PDF dentro del ZIP**, porque la idea que lo pueda accesar directamente para poner anotaciones y la calificación de cada ejercicio.

Ejercicio 1 (5 puntos)

Calcular la solución de sistemas con matrices triangulares inferiores.

1. Programar la función `forwardSubstitution` que aplica el método de sustitución hacia adelante para resolver el sistema $\mathbf{Lx} = \mathbf{b}$, donde $\mathbf{L}$ es una matriz triangular inferior.

Entradas de la función:

- La matriz $\mathbf{L}$

- La matriz $\mathbf{L}$,
- El arreglo $\mathbf{b}$ y
- una tolerancia τ para prevenir la división entre un número muy pequeño.

Salida de la función:

- El arreglo $\mathbf{x}$ que es solución del sistema si el algoritmo termina exitósamente o **None** si no se pudo calcular la solución.

2. Escriba una función que reciba como parámetros los nombres de dos archivos `npy` de Numpy, uno que tiene la información de una matriz triangular inferior $\mathbf{L}$ y el otro con la información del vector de términos independientes $\mathbf{b}$.

   Haga que esta función lea la información de los archivos usando la función numpy.load().

   Cree la matriz $\mathbf{L}$ y el vector $\mathbf{b}$ y que imprima
   el tamaño de la matriz y el tamaño del vector $\mathbf{b}$.

   Llame a la función `forwardSubstitution` para resolver el sistema $\mathbf{Lx} = \mathbf{b}$, usando como tolerancia $\tau = \epsilon_m^{2/3}$, donde ϵ_m es el épsilon máquina. Si el sistema tuvo solución, imprima los primeros y últimos **3 elementos** del vector solución. En caso contrario, imprima el mensaje de que el sistema no tiene solución única.

   Imprima el valor del error $\|\mathbf{Lx} - \mathbf{b}\|$ si existe la solución. En caso contrario, haga que la función imprima un mensaje que indique que la matriz es singular.

3. Pruebe la función del punto anterior usando los datos del archivo `datosTarea04.zip`, usando las parejas de archivos:

Matriz	Vector
matL006	vecb005
matL010	vecb010
matL015	vecb015
matL020	vecb020
matL050	vecb050
matL500	vecb500

Solución:

```
# Celda es para escribir el código de la función o importarla de un archivo
import numpy as np
```

```python

def forwardSubstitution(L: np.ndarray, b: np.ndarray, t):
  d = L.shape[0]
  res = np.zeros(d)
  for i in range(d):
    sum = np.dot(res[:i], L[i, :i])
    if np.abs(L[i, i]) ⩽ t:
        print(L[i, i])
        return None
    x = (b[i] - sum)/L[i, i]
    res[i] = x

  return res

```

```python
def l_factorization(mat, vec):
    L = np.load(mat)
    b = np.load(vec)
    print(f"Tamaño de la matriz {L.shape}", f"Tamaño del vector {b.shape}", sep='\n')
    eps = np.finfo(float).eps
    x = forwardSubstitution(L, b, eps**(2/3))
    if x is not None:
        print(f"{x[:3]} {x[-3:]}")
        print(np.linalg.norm(np.matmul(L, x)-b))
    else:
        print("El sistema no tuvo solucion unica")
```

```python
# files information
files = ['006', '010', '015', '020', '050', '500']
files_path = "/content"
```

```python
np.load(f"{files_path}/matL{files[2]}.npy")
```

```
array([[ 1.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,
         0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
       [ 0.22,  1.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,
         0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
       [-0.26,  0.57,  1.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,
         0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
       [-0.99,  0.11, -0.55,  1.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,
```

```
  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
[ 0.05, -0.13,  0.46, -0.15,  1.  ,  0.  ,  0.  ,  0.  ,  0.  ,
  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
[ 0.21, -0.18, -0.34,  0.09, -0.11,  1.  ,  0.  ,  0.  ,  0.  ,
  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
[-0.33,  0.15,  0.31, -0.1 ,  0.38, -0.95,  1.  ,  0.  ,  0.  ,
  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
[ 0.4 ,  0.28, -0.62,  0.13, -0.13, -0.71,  0.45,  1.  ,  0.  ,
  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
[-0.83,  0.81,  0.57,  0.13, -0.06,  0.3 , -0.58, -0.38,  1.  ,
  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
[-0.12,  0.57, -0.59,  0.54, -0.62, -0.86,  0.24,  0.83, -0.2 ,
  1.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
[-0.06, -0.56, -0.05,  0.29, -0.41,  0.5 , -0.97,  0.42,  0.08,
  0.4 ,  1.  ,  0.  ,  0.  ,  0.  ,  0.  ],
[ 0.14,  0.32, -0.06,  0.48, -0.33,  0.75, -0.94,  0.1 ,  0.31,
  0.01, -0.06,  1.  ,  0.  ,  0.  ,  0.  ],
[-0.71, -0.34,  0.17,  0.01, -0.36,  0.11, -0.32, -0.26,  0.67,
  0.06, -0.18,  0.6 ,  1.  ,  0.  ,  0.  ],
[-0.71,  0.53,  0.78, -0.14,  0.06, -0.49, -0.5 , -0.51,  0.7 ,
  0.43,  0.49,  0.41,  0.59,  0.  ,  0.  ],
[-0.61, -0.35, -0.69, -0.21, -0.46, -0.63,  0.13, -0.08,  0.45,
  0.44, -0.42, -0.37,  0.79, -0.1 ,  1.  ]])
```

```python
# Pruebas
L_matrices = (f"{files_path}/matL{file}.npy" for file in files)
vectores = (f"{files_path}/vecb{file}.npy" for file in files)

for Lm, b in zip(L_matrices, vectores):
    l_factorization(Lm, b)
    print("")


```

```
Tamaño de la matriz (6, 6)
Tamaño del vector (6,)
[ 2.97  0.15 -8.5 ] [-11.15  -6.36 -12.47]
8.881784197001252e-16

Tamaño de la matriz (10, 10)
Tamaño del vector (10,)
[-0.19 -2.35 -0.57] [-17.41  -9.55  -8.55]
```

```
4.070144838902081e-15

Tamaño de la matriz (15, 15)
Tamaño del vector (15,)
0.0
El sistema no tuvo solucion unica

Tamaño de la matriz (20, 20)
Tamaño del vector (20,)
0.0
El sistema no tuvo solucion unica

Tamaño de la matriz (50, 50)
Tamaño del vector (50,)
[-5.18  1.31  8.58] [-4.59  7.08  2.14]
1.9870200881297512e-14

Tamaño de la matriz (500, 500)
Tamaño del vector (500,)
[-18.57   6.34  -5.31] [-8.45  4.03 -5.64]
4.503930018820819e-13
```

_

Ejercicio 2 (5 puntos)

Calcular la solución de sistemas con matrices triangulares superiores.

1. Programar la función `backwardSubstitution` que aplica el método de sustitución hacia atrás para resolver el sistema $\mathbf{U}\mathbf{x} = \mathbf{b}$, donde $\mathbf{U}$ es una matriz triangular superior.

Entradas de la función:

- La matriz $\mathbf{U}$,
- El arreglo $\mathbf{b}$ y
- una tolerancia τ para prevenir la división entre un número muy pequeño.

Salida de la función:

- El arreglo $\mathbf{x}$ que es solución del sistema si el algoritmo termina exitósamente o None si no se pudo calcular la solución.

2. Análogamente al ejercicio 2, escriba un programa para resolver el sistema $\mathbf{U}\mathbf{x} = \mathbf{b}$ siguiendo las mismas indicaciones y pruebe el programa usando los datos del archivo `datosTarea05.zip` con las siguiente parejas de archivos:

Matriz	Vector
matU006	vecb005
matU010	vecb010
matU015	vecb015
matU020	vecb020
matU050	vecb050
matU500	vecb500

Solución:

```
# Celda es para escribir el código de la función o importarla de un archivo
import numpy as np

def backwardSubstitution(U: np.ndarray, b: np.ndarray, t):
    d = U.shape[0]
    res = np.zeros(d)
    for i in reversed(range(d)):
        sum = np.dot(res[i:], U[i, i:])
        if abs(U[i, i]) < t:
            print(U[i, i], t)
            return None
```

```
        x = (b[i] - sum)/U[i, i]
        res[i] = x
    return res


```

```
def u_factorization(mat, vec):
    U = np.load(mat)
    b = np.load(vec)
    print(f"Tamaño de la matriz {U.shape}", f"Tamaño del vector {b.shape}", sep='\n')
    eps = np.finfo(float).eps
    x = backwardSubstitution(U, b, eps**(2/3))
    if x is not None:
        print(f"{x[:3]} {x[-3:]}")
        print(np.linalg.norm(np.matmul(U, x)-b))
    else:
        print("El sistema no tuvo solucion unica")
```

```
# Pruebas
U_matrices = (f"{files_path}/matU{file}.npy" for file in files)
vectores = (f"{files_path}/vecb{file}.npy" for file in files)

for Um, b in zip(U_matrices, vectores):

    u_factorization(Um, b)
    print()


```

```
Tamaño de la matriz (6, 6)
Tamaño del vector (6,)
[ 0.06129878  0.5384223  -0.72786399] [-1.18093464  0.71851093 -2.17089474]
2.0350724194510405e-15

Tamaño de la matriz (10, 10)
Tamaño del vector (10,)
[ 1.16781739 -3.1711499  -0.81257694] [ 0.87963772  1.59334807 -0.43604688]
1.6001179613975517e-14

Tamaño de la matriz (15, 15)
Tamaño del vector (15,)
[0.52227684 1.6381038  0.80245709] [-0.91354608 -0.08254994  0.48933034]
```

```
1.7004905926528295e-14

Tamaño de la matriz (20, 20)
Tamaño del vector (20,)
0.0 3.666852862501036e-11
El sistema no tuvo solucion unica

Tamaño de la matriz (50, 50)
Tamaño del vector (50,)
0.0 3.666852862501036e-11
El sistema no tuvo solucion unica

Tamaño de la matriz (500, 500)
Tamaño del vector (500,)
[ 4.85663989  3.34824432 -0.6369946 ] [0.543548   0.23729909 0.37602499]
1.313194945433668e-11
```

–

```
1
```

✓ 0s completed at 11:55 PM