Curso de Métodos Numéricos (DEMAT)

Tarea 9

Descripción:	Fechas
Fecha de publicación del documento:	**Octubre 23, 2023**
Fecha límite de entrega de la tarea:	**Noviembre 1, 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 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 (6 puntos)

Programe la función que calcula la factorización QR de una matriz usando rotaciones de Givens.

1. Escriba la función que implemente el algoritmo descrito en la diapositiva 10 de la Clase 20.

- La función recibe una como entrada una matriz $\mathbf{A}$ de tamaño $m \times n$ y una tolerancia $\tau > 0$.
- Use la tolerancia τ en lugar de ϵ_m, de modo que la condición para aplicar la rotación de Givens sea $|s| > \tau$.
- La función debe devolver las matrices $\mathbf{Q}$ y $\mathbf{R}$.

2. Pruebe la función con las matrices que se definen en la celda siguiente con tolencia $\tau = \epsilon_m^{2/3}$.
3. Cada caso reporte para saber si el algoritmo funciona, reporte el valor de

$$\|\mathbf{Q}^\top \mathbf{Q} - \mathbf{I}\|,$$
$$\|\mathbf{A} - \mathbf{QR}\|$$

y

$$\|\mathbf{R} - np.triu(\mathbf{R})\|,$$

donde np.triu es la función que devuelve una copia de la matriz $\mathbf{R}$ con ceros por debajo de la diagonal principal, incluso si la matriz $\mathbf{R}$ es rectangular.

Solución:

```python
# Matrices para prueba

import numpy as np

np.random.seed(123)

# Una matriz simétrica
A  = np.round(np.random.randn(10,10), 2)
A1 = A + A.T

# Una matriz cuadrada arbitraria
A2 = np.round(np.random.randn(10,10), 2)

# Matrices rectangulares
A3 = np.round(np.random.randn(50,10), 2)

A4 = np.round(np.random.randn(1000,100), 2)

```

```python
# Código de la funcion

import numpy as np

def qr_givens(A: np.ndarray, t:float = 0):
    m, n = A.shape
    Q = np.eye(m)
    R = A.copy().astype(float)

    for j in range(n):
        for i in range(j+1, m):
            c, s = R[j, j], R[i, j]
            if np.abs(s) > t: # queremos que s sea cero
                d = np.sqrt(c**2 + s**2)
                c, s = c/d, s/d
                R[[j, i], :] = np.array([[c, s], [-s, c]]) @ R[[j, i], :]
                R[i, j] = 0 # esa posición debe ser un cero
                Q[[j, i], :] = np.array([[c, s], [-s, c]]) @ Q[[j, i], :]

    return Q.T, R
```

```python
from collections.abc import Callable
# Pruebas

def test_qr(A:np.ndarray, QR: tuple[np.ndarray, np.ndarray]):
    from numpy.linalg import norm
    Q, R = QR
    print(f"||Q.T Q - I|| = {norm(Q.T@Q - np.eye(Q.shape[0]))}")
    print(f"||A - QR|| = {norm(A-Q@R)}")
    print(f"||R - np.triu(R)|| = {norm(R-np.triu(R))}")

# def test_qr(A: np.ndarray, qr_descom: Callable):
#     from numpy.linalg import norm
#     eps = np.finfo(float).eps**(2/3)
#     Q, R = qr_descom(A, eps)
#     print(f"||Q.T Q - I|| = {norm(Q.T@Q - np.eye(Q.shape[0]))}")
```

```
#     print(f"||A - QR|| = {norm(A-Q@R)}")
```

```
# Pruebas
from numpy.linalg import norm
# test_qr(A4, qr_givens), test_qr(A4, np.linalg.qr)
matrices = [A1, A2, A3, A4]

# comparamos nuestra implementación con la de numpy
for m in matrices:
    eps = np.finfo(float).eps**(2/3)
    print("Mi implementacion")
    test_qr(m, qr_givens(m, eps))
    print("La implementacion de numpy")
    Q, R = np.linalg.qr(m)
    print(f"||Q.T Q - I|| = {norm(Q@Q.T - np.eye(Q.shape[0]))}")
    print(f"||A - QR|| = {norm(m-(Q@R))}")
    print(f"||R - np.triu(R)|| = {norm(R-np.triu(R))}")
    print()
```

```
Mi implementacion
||Q.T Q - I|| = 1.378538344847531e-15
||A - QR|| = 8.935714024050041e-15
||R - np.triu(R)|| = 0.0
La implementacion de numpy
||Q.T Q - I|| = 1.5201772867032951e-15
||A - QR|| = 5.424455622660468e-15
||R - np.triu(R)|| = 0.0

Mi implementacion
||Q.T Q - I|| = 1.523510175501811e-15
||A - QR|| = 4.523908720264494e-15
||R - np.triu(R)|| = 0.0
La implementacion de numpy
||Q.T Q - I|| = 1.3729492913293084e-15
||A - QR|| = 3.3972990372280907e-15
||R - np.triu(R)|| = 0.0

Mi implementacion
||Q.T Q - I|| = 4.885939345863889e-15
||A - QR|| = 1.8936122355906838e-14
||R - np.triu(R)|| = 0.0
La implementacion de numpy
||Q.T Q - I|| = 6.324555320336758
||A - QR|| = 5.12911379948457e-15
||R - np.triu(R)|| = 0.0

Mi implementacion
||Q.T Q - I|| = 6.735292029767495e-14
||A - QR|| = 1.130323945538053e-12
||R - np.triu(R)|| = 0.0
La implementacion de numpy
||Q.T Q - I|| = 30.000000000000004
||A - QR|| = 1.8892056894281572e-13
||R - np.triu(R)|| = 0.0
```

De lo anterior podemos incluso notar que nuestra implementación tiene errores cercanos a los de `numpy` (en un caso incluso es "mejor", mas pequeño).

--

Ejercicio 2 (4 puntos)

Calcular la solución $\mathbf{x}$ del problema de mínimos cuadrados

$$\min \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$$

usando la factorización QR de una matriz.

1. Escriba una función que reciba una matriz $\mathbf{A}$, el vector $\mathbf{b}$ y la tolerancia $\tau > 0$. La función calcula la factorización QR usando la función del Ejercicio 1.
2. La función devuelve la solución $\mathbf{x}_1$ del problema de mínimos cuadrados calculada de acuerdo con la diapositiva 2 de la Clase 21.
3. Use las matrices A_3 y A_4 del Ejercicio 1 para probar el algoritmo, con $\mathbf{b}$ que tiene todas sus entradas iguales a 1.
4. Imprima las primeras y últimas componentes del vector $\mathbf{x}_1$
5. Calcula la solución $\mathbf{x}_2$ del problema de mínimos cuadrados usando la fórmula

$$\mathbf{A}^\top \mathbf{A}\mathbf{x}_2 = \mathbf{A}^\top \mathbf{b}$$

6. Imprima el error $\|\mathbf{x}_1 - \mathbf{x}_2\|$ para verificar que coinciden las soluciones.

Solución:

```
## Funciones de la Tarea 4

import numpy as np

def backwardSubstitution(U: np.ndarray, b: np.ndarray, t):
    d = U.shape[1] # Para matrices no cuadradas nos interesa el numero de columnas
    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
```

```python
# Código de la función
import numpy as np

# podemos usar backwardsustitution porque R es Upper Triangular
def ls_qr(A:np.ndarray, b: np.ndarray, t:float):
    Q, R = qr_givens(A)
    x = backwardSubstitution(R, Q.T@b, t)
    return x



```

```python
# Pruebas
def test_ls_qr(A: np.ndarray, b: np.ndarray):
    eps = np.finfo(float).eps

    # Para le matriz A3
    x1 = ls_qr(A, b, eps)
    x2 = np.linalg.lstsq(A, b, rcond=None)

    print(f"x1 = {x1}", f"||x1-x2|| = {norm(x1-x2[0])}", sep='\n')
    print("Usando el truco A.T A x = A.T b obtenemos que:")
    print(f"||x1- x2|| = {norm(x1-np.linalg.solve(A.T@A, A.T@b))}")
```

```python
# Pruebas
eps = np.finfo(float).eps

# Para le matriz A3
print("Para A3")
test_ls_qr(A3, np.ones(A3.shape[0]))

print("Para A4")
test_ls_qr(A4, np.ones(A4.shape[0]))

```

```
Para A3
x1 = [ 0.01019281 -0.01749163 -0.16823942 -0.32273097 -0.0715259   0.23638792
 -0.05137134 -0.15842806 -0.0033877   0.07130793]
||x1-x2|| = 7.495937657100415e-16
Usando el truco A.T A x = A.T b obtenemos que:
||x1- x2|| = 3.031330003775621e-16
Para A4
x1 = [ 0.03355378  0.01330705 -0.01973571 -0.07850015  0.00276066 -0.01567825
  0.02628051 -0.00424231  0.01939108  0.02688731 -0.03604838 -0.01499184
  0.05620531  0.04571636 -0.01159996 -0.03756171 -0.0108619   0.00770347
 -0.03255031 -0.0155421  -0.02451084 -0.03176701 -0.03929835 -0.09014355
 -0.0091517  -0.01471299 -0.04974389  0.04821385  0.02434659  0.00692902
  0.01164744  0.05190751  0.0063978   0.0337241  -0.01649982  0.03403816
  0.05693563 -0.00483791  0.01157347  0.09351066 -0.00221624 -0.01095856
 -0.01326026  0.03030159 -0.03992685 -0.00601227 -0.01838074 -0.03607218
  0.02302453  0.01236152  0.02713527  0.053495   -0.03680839  0.02939482
  0.00706036 -0.00013625  0.00587317 -0.05360152  0.00905734  0.02497073
 -0.11457335 -0.01915847  0.02277082  0.01712966  0.01142295 -0.02490634
 -0.00780696  0.00994454 -0.02713449 -0.04121067  0.08509076  0.01315327
 -0.0196848   0.02844983  0.05241722 -0.00539394 -0.02648388  0.01064417
  0.0015824  -0.00382244 -0.03743758  0.06315128 -0.02095549  0.04688225
 -0.01040856  0.00303383  0.08195261  0.00509424  0.01613911  0.04183215
  0.0113859   0.01993732 -0.03143386 -0.07619282 -0.00329295 -0.04361427
 -0.01977154  0.00308723  0.00170532 -0.02796887]
```

```
||x1-x2|| = 1.0664589555291126e-15
Usando el truco A.T A x = A.T b obtenemos que:
||x1- x2|| = 9.28161189163082e-16
```

De lo anterior podemos notar que las soluciones son muy cercanas a las encontradas por `numpy`.