Métodos Numéricos

Trabajo práctico N°2

Cuadrados Mínimos

Grupo 11: CHEN, Carlos Angel, Legajo
MOLDOVAN LOAYZA, Alexander Stephan, Legajo 60498
MOLINA, Facundo Nicolás, Legajo 60526

Profesor: Pablo Ignacio Fierens

Fecha de entrega: 22 de abril de 2022

Buenos Aires, Argentina

Secciones

Índice de Códigos

1. Función leastsq()

Para el siguiente ejercicio se decidió utilizar la descomposición QR reducida, para ello se escribieron 3 algoritmos:

- `thinQRFactorization(A)`: esta función toma a la matriz A y comienza a realizarle la descomposición QR reducida y devuelve las matrices `Q1, R1` reducidas.
- `solveTriangular(A,b)`: esta función recibe la matriz A cuadrada y triangular superior(nxn) y un vector b de terminos independientes (nx1). Devuelve el vector $\overline{x}$ de la ecuación $A\overline{x} = b$
- `leastsq(A,b)`: esta función verifica el rango de las matrices y llama a las funciones mencionadas anteriormente y resuelve la descomposición QR reducida. Devuelve el vector $\overline{x}$ de solución.

Código 1: `leastsq()` y funciones subordinadas

```
import numpy as np
from math import sqrt

# Función thinQRFactorization
# Recibe: matriz no singular de dimensiones mxn, con m>=n
# Devuelve: matrices Q1 y R1, correspondientes a la factorización QR reducida
# de la matriz de entrada
def thinQRFactorization(A):
    (m,n) = A.shape      # Se obtienen las dimensiones de la matriz de entrada...
    Q1 = np.zeros((m,n)) # ...para definir las matrices Q1 y R1, inicialmente en 0
    R1 = np.zeros((n,n))

    # Metodo Gram-Schmidt (tradicional)
    for k in range(n):      # Por cada columna de Q1...
        Q1[:,k] = A[:,k]    # se extrae una nueva columna de A
        for j in range(k):  # y por cada una de las columnas anteriores de Q1 (ahora versores) ...
            Q1[:,k] = Q1[:,k] - (Q1[:,j]@A[:,k])*Q1[:,j] # ... se le resta la proyección de la nueva columna
     ↪  en cada versor
        Q1[:,k] = Q1[:,k] / sqrt(Q1[:,k].T@Q1[:,k]) # Se normaliza la nueva columna
        for j in range(k,n): # Por cada una de las siguientes columnas
            R1[k,j] = Q1[:,k].T@A[:,j] # Se calcula el valore de R1
    return Q1,R1

# Función solveTriangular
# Recibe: matriz de coeficientes A cuadrada (nxn), y vector de terminos independientes (nx1)
# A debe ser triangular superior
# Devuelve: matriz (nx1) solución al sistema Ax=b
def solveTriangular(A,b):
    n = A.shape[0]      # Se obtiene el tamaño de A...
    x = np.zeros((n,1)) # ...para crear la matriz resultado, inicialmente en 0
    for k in range(n):  # Por cada fila del resultado
        row = n-1-k     # (comenzando por la última)
        x[row,0] = b[row,0] # Se iguala al término independiente
        for j in range(row+1,n): # Y por cada una de las filas ya calculadas
            x[row,0] -= A[row,j]*x[j,0] # Se sustrae su contribución con la incógnita actual
```

```
35        x[row,0]/= A[row,row] # Finalmente, se divide por el coeficiente de la incógnita actual
36    return x
37
38 # Función leastsq
39 # Recibe: matrices de numpy A (tamaño mxn) y b (tamaño mx1) correspondientes
40 # al sistema Ax = b
41 # Devuelve: matriz x de tamaño nx1 que minimice la norma del error
42 # e = Ax-b
43 def leastsq(A,b):
44    try:
45       (m,n) = A.shape
46       (o,p) = b.shape
47       if m < n:       # Comuncar error... Si A tiene mas columnas que filas;
48          print("leastsq: ISSUE: Dimensions of A (mxn)\nm must be greater of equal than n")
49          return np.array([[]])
50       elif m != o:    #... Si A y b tienen un número distinto de filas
51          print("leastsq: ISSUE: Dimensions of A (mxn) and b (nx1)\nDimensions of A and b
     ↪ mismatch")
52          return np.array([[]])
53       elif p != 1:    #... o si b tiene mas (o menos) de 1 columna
54          print("leastsq: ISSUE: Dimensions of b (nx1)\nb must have 1 column")
55          return np.array([[]])
56       else: # Si las entradas son válidas
57          Q1,R1 = thinQRFactorization(A) # Se factoriza a A en Q1 y R1 (factorización QR
     ↪ reducida)
58          x = solveTriangular(R1,Q1.T@b) # y se resuelve el sistema de ecuaciones R1x = Q1'b
59          return x
60
61    except: # Mensaje enviado en caso de que las entradas no sean válidas
62       print("leastsq: INVALID INPUT")
63       return np.array([[]])
```

2. Función test()

La función *test*() implementa un banco de pruebas con un arreglo de matrices predeterminadas que contemplan casos representativos. Se tiene en cuenta que para la resolución del problema de cuadrados mínimos lineal utilizando la descomposición QR, la matriz $A \epsilon \mathbb{R}^{mxn}$ y el vector $b \epsilon \mathbb{R}^{px1}$ deben cumplir con las siguientes condiciones:

- Rango(A) == dim(A). Es decir, A no singular.
- $m \geq n$
- rows(b) == 1

Se analizan los siguientes casos:

- Tres casos que cumplen los requerimientos.
- $rango(A) \leq dim(A)$

- $m \leq n$
- $m \leq n$ y $rango(A) \leq dim(A)$
- $m \neq n$
- $b \epsilon \mathbb{R}^{pxk}$ con $k \geq 1$

Para los casos que cumplen las condiciones, se comparó el vector solución devuelto por la función *leastsq*() contra el resultado de la operación empleando la función *lstsq*() de la biblioteca *linalg* de *numpy*. Para la comparación, se evaluó que

Código 2: Definición de test()

```
# Funcion test()
#   Testbench para funcion leastsq()
#   Analiza 3 casos sin errores y 5 casos que no cumplen condiciones.
#   Condiciones para factorizacion QR:
#       A de rango completo
#       m >= n
#       p == m
def test():
    # Test bank: A, b
    testCases = (
    #4x2
    (np.array([[5.1, 0], [3.4, 1], [-4, 1], [0.11, 1]]), np.array([[5, 1.8, 9.9, -1.5]])),
    #3x1
    (np.array([[-1], [0], [-1]]), np.array([[1, 5, 2]])),
    #3x3
    (np.array([[1, 0, 0], [1, 1, 0], [1, 1, 1]]), np.array([[0.7, 3, 4.1]])),
    #4x2        rank(A)<n
    (np.array([[1, 0], [1, 0], [1, 0], [1, 0]]), np.array([[5, 1.8, 9.9, -1.5]])),
    #2x3        m<n
    (np.array([[4, 1, 1], [1, 0, 1]]), np.array([[1, 2, -4]])),
     #2x3        m<n     rank(A)<n
    (np.array([[1, 1, 1], [1, 1, 1]]), np.array([[1, 2, -4]])),
    #4x2        Dimensions of A (mxn) and b (nx1) mismatch
    (np.array([[5.1, 0], [3.4, 1], [-4, 1], [0.11, 1]]), np.array([[5, 1.8, 9.9]])),
    #4x2        b with more columns
    (np.array([[5.1, 0], [3.4, 1], [-4, 1], [0.11, 1]]), np.array([[5, 1.8, 9.9, -1.5], [1, 0, 0, 1]])),
    )

    eps = 10**(-6)
    passed = failed = 0

    print("_____________________________________________________________________")
    print("TEST BENCH")

    for i in range(len(testCases)):
        A = testCases[i][0]
        b = testCases[i][1]

        #b = testCases[n] if n < n_casos_particulares else createAb()
```

```
        print("TEST N°: ", i)
        print("A =")
        print(A)
        print("b =")
        print(b)

        x = leastsq(A, b.T)

        if x.size == 0:
            (m,n) = A.shape
            (o,p) = b.shape
            if m < n or np.linalg.matrix_rank(A) < n or m != o or p != 1:
                print("ISSUE DETECTED. PASSED \n")
                passed +=1
            else:
                print("ISSUE NOT DETECTED. FAILED \n")
                failed += 1
        else:
            print("x =")
            print(x)
            x_prime = np.linalg.lstsq(A,b.T, rcond=None)
            print("x_prime =")
            print(x_prime[0])
            diff = np.linalg.norm(x - x_prime[0])

            if diff < eps:
                print("norm(x-x_prime) < eps. PASSED \n")
                passed +=1
            else:
                print("norm(x-x_prime) >= eps. FAILED \n")
                failed +=1

    print("CASES: ", len(testCases))
    print("PASS: ", passed)
    print("FAIL: ", failed)
```

3. Función sonido()

En el siguiente ejercicio se trabaja con el archivo **sonido.txt** el cual contiene 2 columnas. La primera son los valores de t y la segunda son los valores de y. Estas 2 variables corresponden a la siguiente ecuación:

$$y = \sum_{k=1}^{3} [a_k cos(1000k\pi t) + b_k sin(1000k\pi t)] \tag{1}$$

Se busca hallar los valores ajustados a_1, a_2, a_3, b_1, b_2 y b_3 de la ecuación y el error del ajuste utilizando la función `leastqs()` que se hizo para el ejercicio 1. Por lo que se definio las siguientes matrices:

$$A = \begin{pmatrix} cos(1000\pi t_0) & sin(1000\pi t_0) & cos(2000\pi t_0) & sin(2000\pi t_0) & cos(3000\pi t_0) & sin(3000\pi t_0) \\ cos(1000\pi t_1) & sin(1000\pi t_1) & cos(2000\pi t_1) & sin(2000\pi t_1) & cos(3000\pi t_1) & sin(3000\pi t_1) \\ cos(1000\pi t_2) & sin(1000\pi t_2) & cos(2000\pi t_2) & sin(2000\pi t_2) & cos(3000\pi t_2) & sin(3000\pi t_2) \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \end{pmatrix}$$

$$\overline{x} = \begin{pmatrix} a_1 \\ b_1 \\ a_2 \\ b_2 \\ a_3 \\ b_3 \end{pmatrix}$$

$$\overline{b} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ . \\ . \\ . \end{pmatrix}$$

Los valores de t_i e y_i van de 0 a 440999. Para obtener estas matrices, se implemento el siguiente codigo:

Código 3: Función sonido()

```
import pandas as pd
# Funcion sounido()
#   Toma los datos del archivo sound.txt y los evalua en las funciones coseno y seno,
#   obteniendo asi la matriz A y b
#   Hace llamado a la funcion leastsq(A,b) y les envia la matriz A y b
#   Devuelve la xsol y el error
def sonido():
    df  = pd.read_csv('.\TP2\sound.txt',header=None,names=['ti','yi'],dtype={'ti':np.float64,'yi':np.
     ↪ float64},sep=' ')
    ti  = np.array(df['ti'].tolist())
    b  = np.array([df['yi'].tolist()]).T

    A = np.zeros((441000, 6))

    for i in range(len(ti)):
        count = 0
        if count<6:
            row = []
            row.append(np.cos(1000*np.pi*ti[i]))
            row.append(np.sin(1000*np.pi*ti[i]))
            row.append(np.cos(2000*np.pi*ti[i]))
```

```
        row.append(np.sin(2000*np.pi*ti[i]))
        row.append(np.cos(3000*np.pi*ti[i]))
        row.append(np.sin(3000*np.pi*ti[i]))
        A[i] = row
        count+=1

    xsol = leastsq(A, b)
    error = np.matmul(A,xsol) - b

    return xsol, error
```