93.54 Métodos Numéricos

Grupo 3

2022

Trabajo Práctico Nro. 2

Lesiuk, Alejandro	60202
Peralta, Sergio Andres	61433
Schneeberger, Bautista	61463

1. Implementación

Para resolver el problema de cuadrados mínimos lineal

$$\vec{x}^* = \underset{\vec{x}}{argmin} ||\mathbf{A}\vec{x} - \vec{b}||_2 \tag{1}$$

se creó una función **leastq** la cual recibe como parámetros **A** y $\vec{b}$, este último un arreglo con una columna, y devuelve una matriz con los valores ajustados del sistema a resolver. Internamente **leastq** utiliza las funciones **qr**, la cual calcula la descomposición por Gram-Schmidt, y **triangsup**, que utiliza la sustitución hacia atrás para encontrar el $\vec{x}$ en el sistema triangular superior.
Para un sistema de ecuaciones lineales $\mathbf{A}\vec{x} = \vec{b}$, si $\mathbf{A} \in \Re^{mxn}$ existe una matriz ortogonal $\mathbf{Q} \in \Re^{m \times m}$ y una matriz triangular superior $\mathbf{R} \in \Re^{mxn}$ tales que $\mathbf{A} = \mathbf{QR}$ por lo que

$$\mathbf{A}\vec{x} = \vec{b} \tag{2}$$

$$\mathbf{QR}\vec{x} = \vec{b} \tag{3}$$

si multiplicamos a ambos términos por Q^T

$$\mathbf{Q}^T\mathbf{QR}\vec{x} = \mathbf{Q}^T\vec{b} \tag{4}$$

$$\mathbf{R}\vec{x} = \mathbf{Q}^T\vec{b} \tag{5}$$

La función **qr** recibe una matriz A y devuelve las matrices Q_1 y R_1, las de su descomposición QR reducida. Una vez que encontramos la matriz Q_1 con la función **triangsup** se procede a resolver el sistema lineal $R_1\vec{x} = \vec{d}$ (donde $\vec{d} = Q_1^T\vec{b}$) para encontrar el vector $\vec{x}$.

Para la función de prueba, se tomaron dos ejemplos con matrices A de rango completo, una cuadrada de $2x2$ y otro ejemplo realizado en clase. Ambos casos se comparan con los resultados de la función que realiza cuadrados mínimos de *numpy.linalg*.

En la función *sonido()* se lee el archivo, se crean los vectores $\vec{t}$ y $\vec{b}$ con los valores de la primera y segunda columna respectivamente y se crea la matriz A de Nx6 (Donde N es la cantidad de filas del archivo). En cada par de columnas $(2\,i, 2\,i+1),\ \ 0 < i < 2$ se colocan los $(cos(1000(i+1)\pi t),\ sen(1000(i+1)\pi t))$ donde t es el valor de $\vec{t}$ que corresponde a esa fila. El sistema se ingresa en la función *leastq* y devuelve en $\vec{x}$ los coeficientes a_i y b_i. La solución ajustada será el producto $\vec{y} = A\vec{x}$. Se devuelve la solución $\vec{y}$ y el error calculado como $\vec{e} = \vec{b} - \vec{y}$.

Con esta función, se obtuvo el gráfico de la Figura 1.

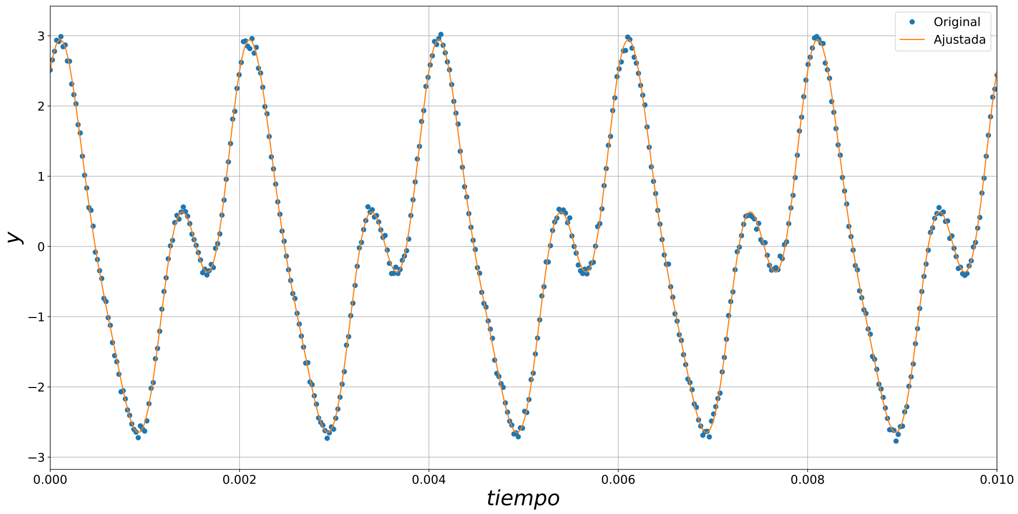

Figura 1: Amplitud "y" vs tiempo. **Función ajustada al conjunto de puntos**

2. Código

```
import matplotlib as mpl
import numpy as np

import pandas as pd
import matplotlib.pyplot as plt

# Resuelve el problema de cuadrados minimos lineal utilizando descomposicion QR
def leastq(A: np.ndarray, b: np.ndarray) -> np.ndarray:
    Q, R = qr(A)
    return triangsup(R, Q.T @ b)

# Calcula la descomposicion QR por Gram-Schmidt
def qr(A: np.ndarray):
    Q = np.ndarray(A.shape)
    R = np.zeros([A.shape[1], A.shape[1]])

    R[0][0] = np.linalg.norm(A[:, 0])
    Q[:, 0] = A[:, 0] / R[0][0]
    for i in range(1, A.shape[1]):
        Q[:, i] = A[:, i]

        for j in range(0, i):  # Resta de las proyecciones
            R[j][i] = np.dot(Q[:, j], A[:, i])
            Q[:, i] -= R[j][i] * Q[:, j]

        R[i][i] = np.linalg.norm(Q[:, i])        # Normalizacion
        Q[:, i] /= R[i][i]

    return Q, R


def triangsup(A: np.ndarray, b: np.ndarray):  #Ax = b ,  A triangular superior nxn
```

```
    x = np.zeros([A.shape[1], b.shape[1]])

    for j in range (0, x.shape[1]):
        x[-1][j] = b[-1][-1]/A[-1][-1]

        for i in range(x.shape[j] - 2, -1, -1):
            x[i][j] = (b[i][j] - np.dot(A[i, i+1: A.shape[1]], x[i + 1: x.shape[0], 0]))/A[i][i]

    return x

def test():

    A = []
    b = []

    # Ejemplo sencillo
    A.append(np.array([[1, 2], [0, 5]]))
    b.append(np.array([[5], [4]]))

    # Ejemplo de la clase
    A.append(np.array([[1.02,1],[1.01,1],[0.94,1], [0.99, 1]]))
    b.append(np.array([[2.05],[1.99],[2.02],[1.93]]))


    for i in range(len(A)):
        print("Matriz A:")
        print(A[i])
        print("Matriz b:")
        print(b[i])
        # Q, R = qr(A)
        # print(Q)
        # print(R)
        x = leastq(A[i], b[i])
        print("Solucion:")
        print(x)
        x = np.linalg.lstsq(A[i], b[i], rcond=None)[0]
        print("Solucion por numpy:")
        print(x)


def sonido():

    df  = pd.read_csv('sound.txt', header=None, names=['ti','yi'], dtype={'ti':np.float64,'yi':np.float64}, sep=' ')
    t  = np.array(df['ti'])

    N = len(t)

    b = np.ndarray([N, 1])
    b[:, 0]  = np.array(df['yi'])


    A = np.ndarray([N, 6])  # [ a1, b1, a2, b2, a3, b3 ]

    for i in range(3):
        A[:, 2*i] = np.cos(1000*(i+1)*np.pi*t)
        A[:, 2*i+1] = np.sin(1000*(i+1)*np.pi*t)

    x = leastq(A, b)
```

```python

    y = A @ x   # Solucion ajustada

    e = b - y # Error

    return y, e


if (__name__ == "__main__"):

    test()

# Ploteo de las curvas de sound original y ajustada
    # y, e = sonido()

    # df  = pd.read_csv('sound.txt', header=None, names=['ti','yi'], dtype={'ti':np.float64,'yi':np.float64}, sep=' ')
    # mpl.rcParams['font.size'] = 16

    # plt.plot(df['ti'], df['yi'], 'o', label='Original')
    # plt.plot(df['ti'], y, label='Ajustada')
    # # plt.plot(e, label='error')
    # plt.xlabel('$tiempo$', fontsize=28)
    # plt.ylabel('$y$', fontsize=28)
    # plt.xlim(0, 0.01)
    # plt.legend()
    # plt.grid()
    # plt.show()
```