Instituto Tecnológico de Buenos Aires

Trabajo Práctico N$^{\underline{o}}$2

Cuadrados mínimos

93.54 Métodos Numéricos

Grupo N°4

Legajo N°	Nombre
61428	Kevin Amiel Wahle
61430	Francisco Basili
61431	Nicolás Bustelo

21/04/2022

Índice

1. Introducción

El objetivo de este informe es presentar y explicar el funcionamiento de un programa realizado en *Python*, el cual consiste en resolver el problema de cuadrados mínimos lineal, utilizando la descomposición de Cholesky. A su vez, también se escribió una función que ajustase los datos de un archivo a una curva dada. El código se encuentra adjunto en la entrega del TP, y también se lo puede ver en en el Anexo (sección 6).

2. Función *leastsq (A,b)*

La función *leastsq(A,b)* resuelve el sistema de ecuaciones $A\vec{x} = \vec{b}$ utilizando la descomposición Cholesky, y luego el método de ecuaciones normales.

Para ello, dado $A \in \mathbb{R}^{n\times m}$, se obtuvo $A' = A^T A \in \mathbb{R}^{n\times n}$. Se corroboró que esta matriz A' cumpliese con las hipótesis necesaria para aplicar la descomposición Cholesky: se verificó que la matriz fuera definida positiva y que fuera simétrica.

Luego, una vez obtenida la G al usar la descomposición Cholesky (que se encuentra explicada en la sección 3.1) para A', se procedió a resolver el sistema $GG^T\vec{x} = A^T\vec{b}$. Para ello, se resolvieron los sistemas $G\vec{y} = A^T\vec{b}$ y $\vec{y} = G^T\vec{x}$.

Sabiendo que tanto G como G^T son matrices triangulares conocidas, con las funciones *LsolverLower(mat, B)* y *LsolverUpper(mat, B)* (que se encuentran explicadas en la sección 3.2), se resolvieron los sistemas triangulares, obteniendo primero $\vec{y}$ y, con ella, $\vec{x}$.

3. Funciones Utilizadas

3.1. *Cholesky(A, ncols)*

La función *Cholesky(A, ncols)* calcula la matriz G triangular inferior tal que $GG^T = A^T A$. Esta función no verifica que la matriz sea definida positiva, ya que esta verificación se realiza en la función *leastsq (A,b)*.

3.2. Resolución de sistemas triangulares

Las funciones *LsolverLower(mat, B)* y *LsolverUpper(mat, B)* resuelven sistemas lineales de ecuaciones triangulares. Para la primera de ellas *mat* debe ser una matriz triangular inferior, mientras que para la segunda debe ser una matriz triangular superior. Una vez resuelto el sistema retornan el vector solución.

3.3. Funciones auxiliares

La función *transpuesta(mat, nrows, ncols)* toma una matriz de cualquier dimensión y retorna su matriz transpuesta pasando elemento a elemento.

La función *esSimetrica(mat, ncols)* verifica que *mat* sea simétrica recorriendo únicamente el triángulo superior de la misma.

La función *autovalores(mat)* calcula todos los autovalores de la matriz *mat*. Luego, sabiendo si la matriz es simétrica, se analizan estos autovalores para determinar si la matriz es definida positiva.

4. TestBench

El objetivo del TestBench es comprobar el correcto funcionamiento de la función *leastsq()*. Para ello, se la ejecutó con diferentes matrices, algunas cuyo producto con su transpuesta no es una matriz definida positiva o no es una matriz simétrica. Para comparar con los resultados de nuestra función se utilizó la función *lstsq()* del paquete *numpy.linalg* que sabemos que brinda resultados fiables.

5. Aplicación práctica de cuadrados mínimos

Se utilizó la función *leastsq()* para encontrar la mejor aproximación a los $N = 441.000$ datos, que se encontraban en una archivo "sound.txt" a la función de la ecuación (1). En dicho archivo interpretamos que la primera columna como valores **y** la segunda columna como **t**.

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

Interpretamos los datos para lograr representarlos como un sistema lineal de matrices $A^T \cdot \vec{x} = \vec{b}$. Donde $\vec{b}$ es un vector con todos los valores de **y**, el vector $\vec{x} = [a_1, a_2, a_3, b_1, b_2, b_3]^T$ y por último en la matriz A se encontraban, dispuestos por columna, los valores obtenidos de evaluar cada una de las funciones para un dado t_n.

$$A = \begin{bmatrix} cos(1000 \cdot \pi \cdot t_0) & ..... & cos(1000 \cdot \pi \cdot t_N) \\ cos(2000 \cdot \pi \cdot t_0) & ..... & cos(2000 \cdot \pi \cdot t_N) \\ cos(3000 \cdot \pi \cdot t_0) & ..... & cos(3000 \cdot \pi \cdot t_N) \\ sen(1000 \cdot \pi \cdot t_0) & ..... & sen(1000 \cdot \pi \cdot t_N) \\ sen(2000 \cdot \pi \cdot t_0) & ..... & sen(2000 \cdot \pi \cdot t_N) \\ sen(3000 \cdot \pi \cdot t_0) & ..... & sen(3000 \cdot \pi \cdot t_N) \end{bmatrix}$$

Gracias la función *leastsq* se obtuvieron los valores de a_k y b_k y se obtuvo un arreglo de errores de ajuste. Para ver la similitud entre los valores de $y(t)$ calculados con la ecuación (1) y los del archivo, se las graficó (ver figura 1) y se puede apreciar en la siguiente figura la gran similitud .

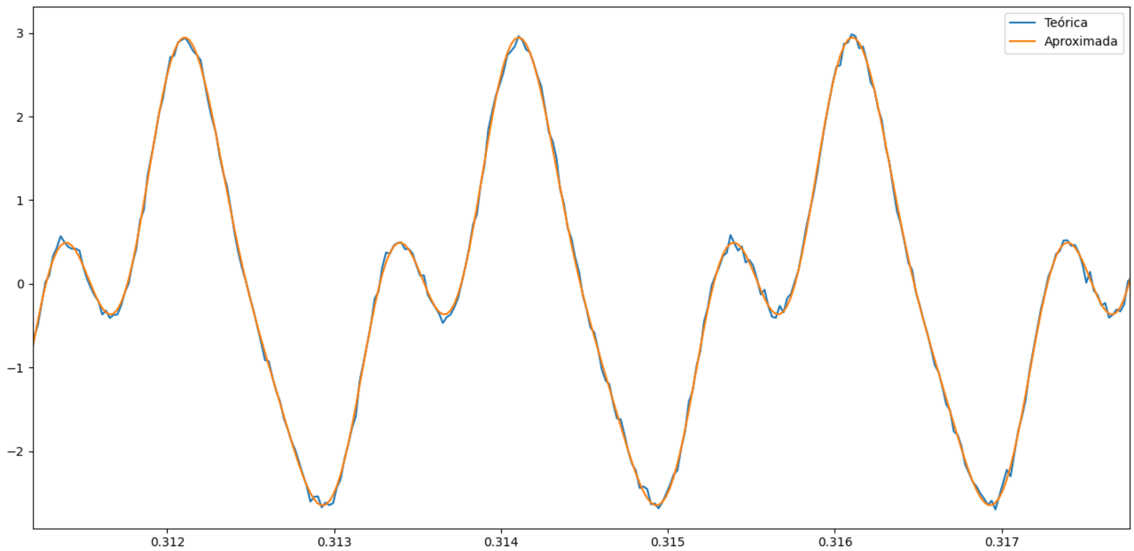

Figura 1: Comparación de los valores con la curva aproximada.

6. Anexo

6.1. Código completo en Python

```python
# ----------------------------------------------------------------------------
#  @file     +leastchol.py+
#  @brief    +Cuadrados mínimos+
#  @author   +Grupo 4+
# ----------------------------------------------------------------------------

# ----------------------------------------------------------------------------
# LIBRARIES
# ----------------------------------------------------------------------------
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math as mt

# ----------------------------------------------------------------------------
# FUNCTION DEF
# ----------------------------------------------------------------------------
# Resolución de sistemas de ecuaciones con la descomposición Cholesky
def leastsq(A, b):
    N = len(A)      #Numero de filas
    Nc = len(A[0])  #Numero de columnas

    At = transpuesta(A, N, Nc)

    if (autovalores(At@A).min() <= 0 or not esSimetrica(At@A, Nc)):
        print("La matriz no es definida positiva")
        return None

    X = np.zeros(Nc)

    G = Cholesky(At@A, Nc)
    Gt = transpuesta(G, Nc, Nc)

    Y = LsolverLower(G, At@b)
    X = LsolverUpper(Gt, Y)
    return X

# Calculo de la descomposición Cholesky
def Cholesky(A, N):
    G = np.zeros((N,N)) # G es la matriz de Cholesky
    for i in range(N):
        for j in range(i+1):
            suma = 0
            for k in range(j):
                suma += G[i][k]*G[j][k] # suma es la suma de los elementos de la fila anterior
            if (i == j):
                G[i][i] = np.sqrt(A[i][i] - suma) # G[i][i] es el elemento diagonal de la matriz de Cholesky
            else:
                G[i][j] = (A[i][j] - suma)/G[j][j] # G[i][j] es el elemento de la matriz de Cholesky
    return G # Retornamos la matriz de Cholesky

#Resuelve el cálculo de un sistema lineal con una matriz triangular inferior
def LsolverLower(Lm, B):
    n = len(B)
    Y = np.zeros(n)
    for i in range(n):
        suma = 0
        for j in range(i):
            suma += Lm[i][j]*Y[j]
        Y[i] = (B[i] - suma)/Lm[i][i]
    return Y

#Resuelve el cálculo de un sistema lineal con una matriz triangular superior
def LsolverUpper(Um, Y):
    n = len(Y)
```

```python
    X = np.zeros(n)
    for i in range(n-1,-1,-1):
        suma = 0
        for j in range(n-1,i-1,-1):
            suma += Um[i][j]*X[j]
        X[i] = (Y[i] - suma)/Um[i][i]
    return X

# Calculo de autovalores de la matriz A
def autovalores(A):
    aVa = np.linalg.eigvals(A)
    return aVa

# Calculo de la transpuesta
def transpuesta(mat, N, Nc):
    trans = np.empty((Nc, N))
    for i in range(Nc):
        for j in range(N):
            trans[i][j] = mat[j][i]
    return trans

# True si es simetrica, False sino
def esSimetrica(mat, N):
    for i in range(N):
        for j in range(i,N):
            if (mat[i][j] != mat[j][i]):
                return False
    return True


# ---------------------------------------------------------------------------
# TEST
# ---------------------------------------------------------------------------
def comp(A, b):
    At = transpuesta(A, len(A), len(A[0]))
    print("\n")
    # Método del grupo 4 para resolución de sistemas de ecuaciones
    X = leastsq(A, b)
    print("Resultado calculado: ",X)

    # Método de numpy para resolución de sistemas de ecuaciones
    X1= np.linalg.lstsq(A, b, rcond=None)[0]
    print("Resultado de linalg: ", X1)

    if (X is None): # Si no es definida positiva
        return 1    # No se puede resolver por Cholesky

    if np.allclose(X, X1):      # Comparamos las dos matrices
        print("Prueba exitosa")
        return 1
    else:
        print("Prueba fallida")
        return 0


def test():
    pOk=0; ptotales=0

    A = np.array([[-1,-1],[1,0]])
    b = np.zeros((2,1)); b=[0,1]
    pOk += comp(A, b)
    ptotales+=1

    A = np.array([[2,-1],[-1,2]])
    b = np.zeros((2,1)); b=[3,8]
    pOk += comp(A, b)
    ptotales+=1

    A = np.array([[0,1],[0,1]])     # Matriz no definida positiva
    b = np.zeros((2,1)); b=[0,0]
    pOk += comp(A, b)
    ptotales+=1
```

```

    A = np.array([[0,-0.45],[10000,0]])
    b = np.zeros((2,1)); b=[0.003,-87]
    pOk += comp(A, b)
    ptotales+=1

    A = np.array([[2, -1, 0],[-1, 2, -1],[0, -1, 2]])
    b = np.zeros((3,1)); b=[1,2,3]
    pOk += comp(A, b)
    ptotales+=1

    A = np.array([[-8, 1.1, 0],[-1, 3, 1],[0, 7, 5]])
    b = np.zeros((3,1)); b=[15,8,3.2]
    pOk += comp(A, b)
    ptotales+=1

    print(f"Se superaron: {pOk} de {ptotales} pruebas totales")


# ------------------------------------------------------------------------------
# Aplicación práctica de cuadrados mínimos
# ------------------------------------------------------------------------------
def sonido():
    #Cargamos y mostramos los datos del TP
    df  = pd.read_csv('sound.txt',header=None,names=['ti','yi'],dtype={'ti':np.float64,'yi':np.float64},sep=' ')

    ti  = np.array(df['ti'].tolist())
    yi  = np.array(df['yi'].tolist())


    A = [np.cos(1000*np.pi*ti), np.cos(2000*np.pi*ti), np.cos(3000*np.pi*ti),np.sin(1000*np.pi*ti), np.sin(2000*np.pi*ti), np.sin(3000*np.pi*ti)]
    At = np.transpose(A)
    b = np.asarray(yi)

    Xvect = leastsq(At, b)  # Se carga la transpuesta de A y se resuelve el sistema de ecuaciones

    ycalc= At@Xvect #Calculamos los valores de y a partir de las contantes calculadas

    error = np.subtract(yi, ycalc) #Calculamos el error

    #Graficamos los datos para ver la similitud
    plt.plot(ti,yi, label="Teórica")
    plt.plot(ti, ycalc, label="Aproximada")
    plt.get_current_fig_manager().window.showMaximized()
    plt.tight_layout()
    plt.legend()
    plt.show()

    return Xvect, error
```