



Instituto Tecnológico
de Buenos Aires

TRABAJO PRÁCTICO N°2

CUADRADOS MÍNIMOS

93.54 Métodos Numéricos

Grupo N°4

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

8/04/2022

Índice

1. Introducción	2
2. Función <i>leastsq</i> (<i>A,b</i>)	2
3. Funciones Utilizadas	2
3.1. <i>Cholesky</i> (<i>A, ncols</i>)	2
3.2. Resolución de sistemas triangulares	2
3.3. Funciones auxiliares	2
4. TestBench	2
5. Aplicación práctica de cuadrados mínimos	3
6. Anexo	4
6.1. Código completo en Python	4

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 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' cumpliera con las hipótesis necesarias 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 tomó como $\vec{y} = G^T \vec{x}$, $G\vec{y} = A^T \vec{b}$.

Sabiendo que tanto G como G^T son matrices triangulares conocidas, con las funciones *solverLower*(*mat, B*) y *LsolverUpper*(*mat, B*) (que se encuentran explicadas en la sección 3.2), se resolvieron los sistemas triangulares, obteniendo \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.

3.2. Resolución de sistemas triangulares

Las funciones *LsolverLower*(*mat, B*) y *tsolUpper*(*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 una matriz triangular superior. Una vez resuelto el sistema retornan el vector solución.

3.3. Funciones auxiliares

La función *transpuesta*(*mat, nrow, 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 *lstq*(*)* del paquete *numpy.linalg* que sabemos que brinda resultados fiables.

5. Aplicación práctica de cuadrados mínimos

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

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

Interpretamos los datos para lograr representarlos como un sistema lineal de matrices $A \cdot \vec{x} = \vec{b}$. Donde \vec{b} es un vector de todos los valores de y , el vector $\vec{x} = [a_1, a_2, a_3, b_1, b_2, b_3]$ y por último la matriz A se encontraban los valores de t pero en una función según la t lo indicaba.

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

Gracias la función *leastsq* se obtuvieron los valores de a_k y b_k y se obtuvo un error de ajuste de 33.2. Para ver la similitud se gráfico la medición del archivo .txt con lo calculado y la formula de la ecuación (1) y se puede apreciar en la siguiente figura la gran similitud.

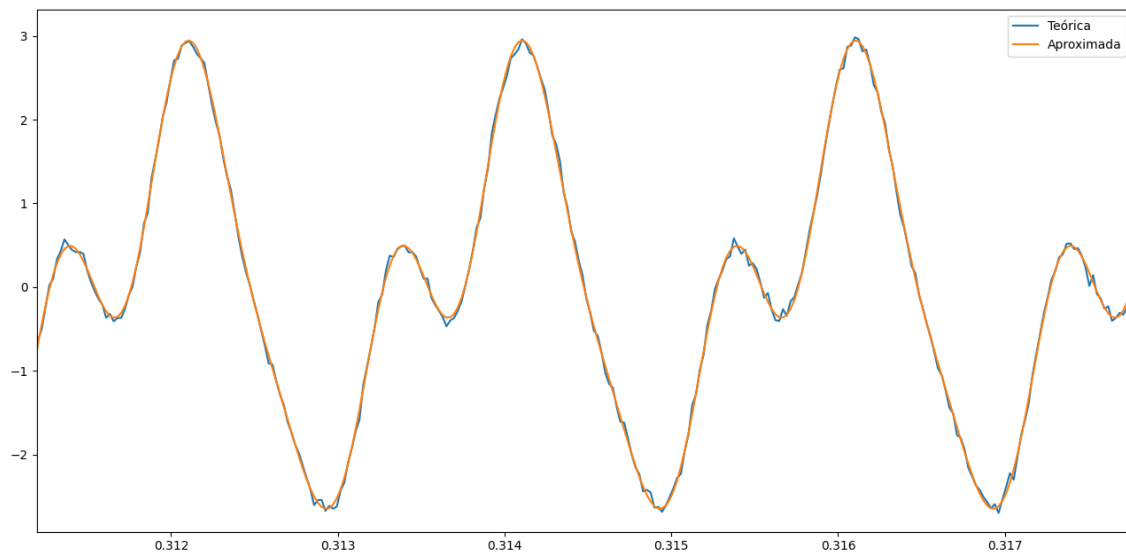


Figura 1: Comparación de los valores con la curva aproximada

6. Anexo

6.1. Código completo en Python

```

1 # -----
2 # @file      +leastchol.py+
3 # @brief     +Cuadrados mínimos+
4 # @author    +Grupo 4+
5 # -----
6
7 # -----
8 # LIBRARIES
9 # -----
10 import numpy as np
11 import pandas as pd
12 import matplotlib.pyplot as plt
13 import math as mt
14
15 # -----
16 # FUNCTION DEF
17 # -----
18 # Resolución de sistemas de ecuaciones con la descomposición Cholesky
19 def leastsq(A, b):
20     N = len(A)      #Numero de filas
21     Nc = len(A[0])  #Numero de columnas
22
23     At = transpuesta(A, N, Nc)
24
25     if (autovalores(At@A).min() <= 0 or not esSimetrica(At@A, Nc)):
26         print("La matriz no es definida positiva")
27         return None
28
29     X = np.zeros(Nc)
30
31     G = Cholesky(At@A, Nc)
32     Gt = transpuesta(G, Nc, Nc)
33
34     Y = LsolverLower(G, At@b)
35     X = LsolverUpper(Gt, Y)
36     return X
37
38 # Calculo de la descomposición Cholesky
39 def Cholesky(A, N):
40     G = np.zeros((N,N)) # G es la matriz de Cholesky
41     for i in range(N):
42         for j in range(i+1):
43             suma = 0
44             for k in range(j):
45                 suma += G[i][k]*G[j][k] # suma es la suma de los elementos de la fila
46             anterior
47             if (i == j):
48                 G[i][i] = np.sqrt(A[i][i] - suma) # G[i][i] es el elemento diagonal de la
49             matriz de Cholesky
50             else:
51                 G[i][j] = (A[i][j] - suma)/G[j][j] # G[i][j] es el elemento de la matriz
52             de Cholesky
53     return G # Retornamos la matriz de Cholesky
54
55 #Resuelve el cálculo de un sistema lineal con una matriz triangular inferior
56 def LsolverLower(Lm, B):
57     n = len(B)
58     Y = np.zeros(n)
59     for i in range(n):
60         suma = 0
61         for j in range(i):
62             suma += Lm[i][j]*Y[j]
63         Y[i] = (B[i] - suma)/Lm[i][i]
64     return Y
65
66 #Resuelve el cálculo de un sistema lineal con una matriz triangular superior
67 def LsolverUpper(Um, Y):
68     n = len(Y)

```

```

66 X = np.zeros(n)
67 for i in range(n-1,-1,-1):
68     suma = 0
69     for j in range(n-1,i-1,-1):
70         suma += Um[i][j]*X[j]
71     X[i] = (Y[i] - suma)/Um[i][i]
72 return X
73
74 # Calculo de autovalores de la matriz A
75 def autovalores(A):
76     aVa = np.linalg.eigvals(A)
77     return aVa
78
79 # Calculo de la transpuesta
80 def transpuesta(mat, N, Nc):
81     trans = np.empty((Nc, N))
82     for i in range(Nc):
83         for j in range(N):
84             trans[i][j] = mat[j][i]
85     return trans
86
87 # True si es simetrica, False sino
88 def esSimetrica(mat, N):
89     for i in range(N):
90         for j in range(i,N):
91             if (mat[i][j] != mat[j][i]):
92                 return False
93     return True
94
95
96 # -----
97 # TEST
98 # -----
99 def comp(A, b):
100     At = transpuesta(A, len(A), len(A[0]))
101     print("\n")
102     # Método del grupo 4 para resolución de sistemas de ecuaciones
103     X = leastsq(A, b)
104     print("Resultado calculado: ",X)
105
106     # Método de numpy para resolución de sistemas de ecuaciones
107     X1= np.linalg.lstsq(A, b, rcond=None)[0]
108     print("Resultado de linalg: ", X1)
109
110     if (X is None): # Si no es definida positiva
111         return 1
112
113     if np.allclose(X, X1): # Comparamos las dos matrices
114         print("Prueba exitosa")
115         return 1
116     else:
117         print("Prueba fallida")
118         return 0
119
120
121 def test():
122     pOk=0; ptotales=0
123
124     A = np.array([[ -1, -1], [1, 0]])
125     b = np.zeros((2,1)); b=[0,1]
126     pOk += comp(A, b)
127     ptotales+=1
128
129     A = np.array([[2, -1], [-1, 2]])
130     b = np.zeros((2,1)); b=[3,8]
131     pOk += comp(A, b)
132     ptotales+=1
133
134     A = np.array([[0, 1], [0, 1]])
135     b = np.zeros((2,1)); b=[0,0]
136     pOk += comp(A, b)
137     ptotales+=1

```

```

138
139     A = np.array([[0, -0.45], [10000, 0]])
140     b = np.zeros((2,1)); b=[0.003, -87]
141     p0k += comp(A, b)
142     ptotales+=1
143
144     A = np.array([[2, -1, 0], [-1, 2, -1], [0, -1, 2]])
145     b = np.zeros((3,1)); b=[1,2,3]
146     p0k += comp(A, b)
147     ptotales+=1
148
149     A = np.array([[-8, 1.1, 0], [-1, 3, 1], [0, 7, 5]])
150     b = np.zeros((3,1)); b=[15,8,3.2]
151     p0k += comp(A, b)
152     ptotales+=1
153
154     print(f"Se superaron: {p0k} de {ptotales} pruebas totales")
155
156
157 # -----
158 # Aplicación práctica de cuadrados mínimos
159 # -----
160 def sonido():
161     #Cargamos y mostramos los datos del TP
162     df = pd.read_csv('sound.txt', header=None, names=['ti', 'yi'], dtype={'ti': np.float64, 'yi': np.float64}, sep=' ')
163
164     ti = np.array(df['ti'].tolist())
165     yi = np.array(df['yi'].tolist())
166
167
168     A = [np.cos(1000*np.pi*ti), np.cos(2000*np.pi*ti), np.cos(3000*np.pi*ti), np.sin(1000*
169 np.pi*ti), np.sin(2000*np.pi*ti), np.sin(3000*np.pi*ti)]
170     At = np.transpose(A)
171     b = np.asarray(yi)
172
173     Xvect = leastsq(At, b) # Se carga la transpuesta de A y se resuelve el sistema de
174 ecuaciones
175
176     ycalc= At@Xvect #Calculamos los valores de y a partir de las constantes calculadas
177
178     #error = np.linalg.norm(yi-ycalc)
179     error = np.sqrt(np.square(np.subtract(yi, ycalc)).sum())
180     print("Error: ", error)
181
182     # Graficamos los datos para ver la similitud
183     plt.plot(ti, yi, label="Teórica")
184     plt.plot(ti, ycalc, label="Aproximada")
185     plt.get_current_fig_manager().window.showMaximized()
186     plt.tight_layout()
187     plt.legend()
188     plt.show()

```