Memoria de calculo para el ajuste automático de la constante de perdida geométrica

Hugo E. Sosa[1]

[1]Div. de Estudios y Ensayos de Componentes Estructurales

CNEA - Centro Atomico Constituyentes

hsosa@cnea.gov.ar

10 de marzo de 2021

1. Descripción del problema y Base del modelo

Se pretende obtener un algoritmo que ajuste por cuadrados mínimos, las mediciones de perdidas de carga en el CEBP mientras estas se realizan, es decir, un algoritmo que se actualice con cada nueva medición. Para esto se busca minimizar la suma:

$$\sum_{i=0} C_m = \sum_{i=0} (a_0 + a_1 x_i + a_2 x_i^2 - y_i)^2 \tag{1}$$

donde y_i y x_i son las mediciones de perdida de carga y caudal respectivamente. Teniendo en cuenta que $\nabla \sum_{i=0} C_m = \sum_{i=0} \nabla C_m$, y entendiendo a este gradiente como el operador $\left(\frac{\partial}{\partial a_0}, \frac{\partial}{\partial a_1}, \frac{\partial}{\partial a_2}\right)$. La ecuación que minimiza 1 es:

$$\sum_{i=0} \nabla C_m = 0$$

cuyas componentes son:

$$\sum_{i=0} 2(a_0 + a_1 x_i + a_2 x_i^2 - y_i) = 0$$

$$\sum_{i=0} 2(a_0 + a_1 x_i + a_2 x_i^2 - y_i) x_i = 0$$

$$\sum_{i=0} 2(a_0 + a_1 x_i + a_2 x_i^2 - y_i) x_i^2 = 0$$

Reordenando un poco estas ecuaciones obtenemos el sistema:

$$\sum_{i=0} \begin{pmatrix} 1 & x_i & x_i^2 \\ x_i & x_i^2 & x_i^3 \\ x_i^2 & x_i^3 & x_i^4 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \sum_{i=0} \begin{pmatrix} y_i \\ y_i x_i \\ y_i x_i^2 \end{pmatrix} \tag{2}$$

$$\sum_{i=0} M_i \vec{a} = \sum_{i=0} \vec{v}_i \tag{3}$$

Donde la matriz M_i de la ecuación 3 solo es inversible a partir de $i \geq 1$. Por lo que puede obtenerse de manera iterativa los coeficientes $(a_0, a_1, a_2)_{i+1}$, a partir de la medición $i = 1$ como:

$$\begin{aligned} \vec{a}_{i+1} &= M_{i+1}^{-1} \vec{v}_{i+1} \quad \forall i \geq 1 \\ \text{donde:} \quad M_{i+1} &= M_i + M_{i-1} \quad \& \quad \vec{v}_{i+1} = \vec{v}_i + \vec{v}_{i-1} \end{aligned}$$

El área de paso de la perdida geométrica puede ser obtenida en cada iteración a partir de la tercer coordenada de $\vec{a}$, es decir a_2, por $A_{i+1} = \sqrt{\frac{1}{2\rho a_{2_{i+1}}}}$. En el **ANEXO** de la presente memoria de calculo se muestra un ejemplo de implementación de código en lenguaje Python:

2. ANEXO

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import linalg as npl

x=[0]
y=[np.random.random(1)-0.5]
c=0
X0 = np.array([[1, x[0], x[0] ** 2]])
Y0 = np.array([np.array(y[0]) * np.ones(3)])
M = np.dot(np.transpose(X0), X0)
V = np.transpose(X0) * np.transpose(Y0)
plt.figure()
while x[-1]<=5:
    c = c+1
    x.append(x[-1]+0.01)
    y.append([2*x[-1]**2+np.random.random(1)-0.5])
    X0 = np.array([[1, x[c], x[c] ** 2]])
    Y0 = np.array([np.array(y[c][0]) * np.ones(3)])
    M1 = np.dot(np.transpose(X0),X0)
    V1 = np.transpose(X0)*np.transpose(Y0)
    M = M + M1
    V = V + V1
    A = np.dot(npl.inv(M),V)
    plt.plot(x, y, '.r')
    q = np.array(x)
    p = A[0]+A[1]*np.array(x)+A[2]*np.array(x)**2
    h, = plt.plot(q, p, '-b')
    ax = plt.gca()
    pos = [0.1*np.max(x),np.max(y)-(0.1*np.max(y))]
    tx = r'$f(x)␣=$␣%2.2f␣+␣%2.2f$x$␣+␣%2.2f$x^2$'%(A[0],A[1],A↩
       [2])
    ann = plt.text(pos[0], pos[1], tx)
    plt.pause(0.05)
    ann.remove()
    ax.lines.remove(h)
    plt.draw()
print(r'a_0=%g,␣a_1=%g,␣a_2=%g'%(A[0],A[1],A[2]))
```