Universidad de Guanajuato

Método de Volúmenes Finitos (Proyecto 1)

García Sánchez Marco Antonio[1]

[1]División de Ingenierías, Campus Irapuato-Salamanca, Universidad de Guanajuato, Salamanca, Gto.

Docente: César Eduardo Damián Ascencio, Ph.D.

Mayo 2019

Índice

Resumen

En este trabajo se analiza un problema físico de transferencia de calor unidimensional mediante el Método de Volúmenes Finitos (MVF). El objetivo es determinar la distribución de temperatura y observar las diferencias resultantes debido a la sensibilidad de la malla utilizada.

1. Introducción

El Método de Volúmenes Finitos (MVF) fue introducido en la década de 1970 por McDonald, MacCormack y Paullay, y desde entonces se ha convertido en uno de los métodos preferidos por científicos e ingenieros en el campo de la mecánica de fluidos.

El punto de partida del método es la descomposición del dominio en pequeños volúmenes de control (VC) donde las variables son almacenadas en nodos. Usualmente, estos volúmenes de control se definen a partir de una malla estructurada, y las variables se evalúan en los centros o vértices de dichos volúmenes. Posteriormente, las ecuaciones de conservación se formulan en su forma integral para cada volumen de control, y el sistema resultante se resuelve numéricamente.

Aunque el Método de Elementos Finitos (MEF) ha tenido avances significativos en las últimas décadas, el MVF sigue siendo la opción más práctica para problemas complejos, especialmente aquellos que involucran flujos multifásicos, reactivos o altamente turbulentos.

2. Ecuaciones Gobernantes

Un esquema del problema se muestra en la Figura 1, mientras que los datos correspondientes se presentan en el Cuadro 1.

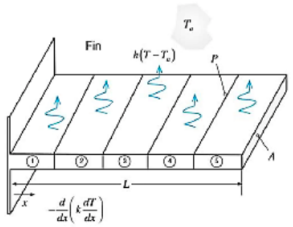

Figura 1: Esquema del problema físico considerado.

Para el análisis se toma un elemento diferencial de la aleta, como se muestra en la Figura 2.

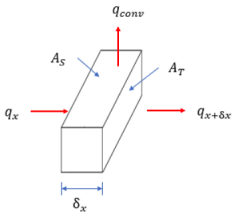

Figura 2: Elemento diferencial de la aleta.

Se hacen las siguientes suposiciones:

- Transferencia de calor unidimensional.
- Conductividad térmica constante.
- Radiación superficial despreciable.
- Sin generación interna de calor.
- Coeficiente de convección uniforme sobre la superficie.

A partir del balance de energía mostrado en la Figura 2:

$$q_x = q_{x+dx} + q_{conv} \tag{1}$$

Sabemos que $q_x = -A_t k \frac{dT}{dx}$ y $q_{conv} = -A_s h(T - T_\infty)$. Con estas relaciones, se obtiene la ecuación gobernante de transferencia de calor en la aleta:

$$-\frac{d}{dx}\left(k\frac{dT}{dx}\right) + \frac{hp}{A_t}(T - T_\infty) = 0 \tag{2}$$

3. Discretización

Integrando la ecuación anterior sobre un volumen de control y aplicando aproximaciones de diferencias finitas centrales, se obtiene:

$$\frac{T_W - 2T_P + T_E}{\Delta x^2} + \frac{hp}{A_t k}(T_P - T_\infty) = 0 \tag{3}$$

Esta ecuación es válida para los nodos internos. En las siguientes subsecciones se describen las condiciones de frontera para distintos casos.

3.1. Caso 1: Temperaturas fijas en ambos extremos

Para los nodos de frontera se introducen nodos fantasmas y se obtienen las ecuaciones modificadas:

Nodo inicial:

$$T_1\left(\frac{3k}{\Delta x}+\frac{hp\Delta x}{A_t}\right)+T_2\left(-\frac{k}{\Delta x}\right)=\frac{hp\Delta x}{A_t}T_\infty+2T_A\left(\frac{k}{\Delta x}\right) \tag{4}$$

Nodo final:

$$T_n\left(\frac{3k}{\Delta x}+\frac{hp\Delta x}{A_t}\right)+T_{n-1}\left(-\frac{k}{\Delta x}\right)=\frac{hp\Delta x}{A_t}T_\infty+2T_B\left(\frac{k}{\Delta x}\right) \tag{5}$$

3.2. Caso 2: Flujo de calor constante en ambos extremos

Nodo inicial:

$$T_1\left(\frac{k}{\Delta x}+\frac{hp\Delta x}{A_t}\right)+T_2\left(-\frac{k}{\Delta x}\right)=\frac{hp\Delta x}{A_t}T_\infty+q_{x=0} \tag{6}$$

Nodo final:

$$T_n\left(\frac{k}{\Delta x}+\frac{hp\Delta x}{A_t}\right)+T_{n-1}\left(-\frac{k}{\Delta x}\right)=\frac{hp\Delta x}{A_t}T_\infty-q_{x=L} \tag{7}$$

4. Resultados e Independencia de Malla

Las ecuaciones discretizadas se resolvieron mediante el método iterativo de Gauss–Seidel implementado en Python. El código empleado se muestra en el anexo.

Las propiedades y características del problema se presentan en el Cuadro 1.

Cuadro 1: Propiedades del problema

Parámetro	Valor	Unidades
T_A	200	°C
T_B	90	°C
T_∞	25	°C
h	100	$\mathrm{W/m^2K}$
k	160	W/mK
L	0.1	m
A_s	10^{-5}	$\mathrm{m^2}$
p	0.1004	m
$E_{propuesto}$	0.001	–

Los resultados de independencia de malla se presentan en el Cuadro 2. Se observa que para un número de nodos mayor a 30, la variación de la temperatura se estabiliza.

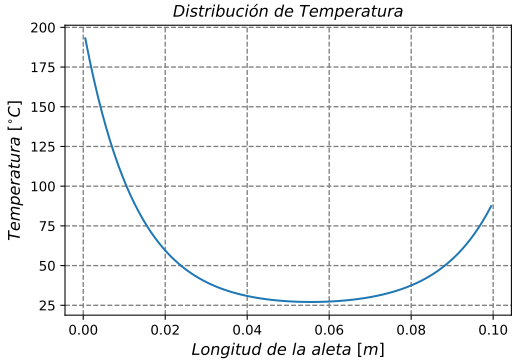

Figura 3: Distribución de temperatura en la aleta (Caso 1).

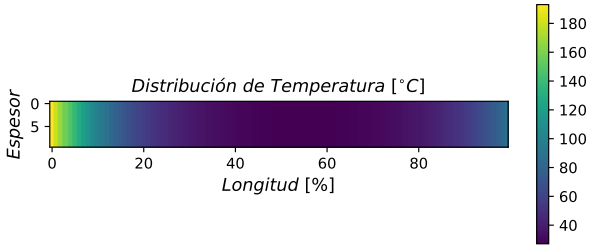

Figura 4: Campo de temperatura y mapa de color de la aleta (Caso 1).

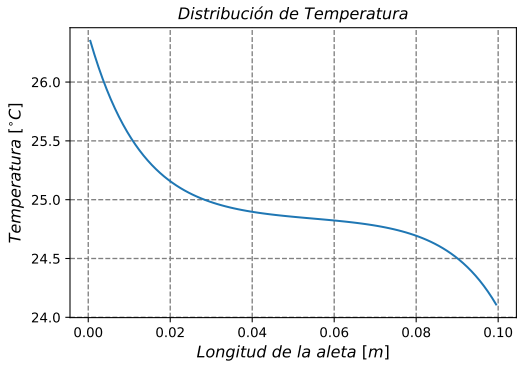

Figura 5: Distribución de temperatura en la aleta (Caso 2).

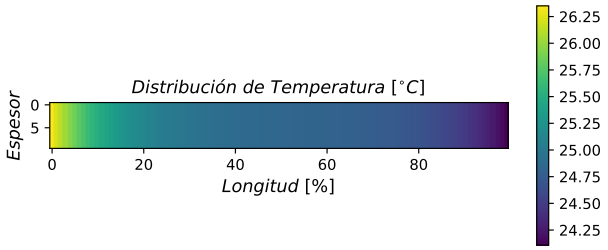

Figura 6: Campo de temperatura y mapa de color de la aleta (Caso 2).

5. Conclusiones

El Método de Volúmenes Finitos demuestra ser una herramienta fundamental para el análisis de fenómenos de transferencia de calor y flujo de fluidos. La aproximación discreta del dominio permite resolver problemas en los que las soluciones analíticas no son factibles, especialmente en geometrías complejas.

Se concluye que una malla entre 30 y 40 nodos proporciona resultados suficientemente precisos con un costo computacional razonable. El método de Gauss–Seidel mostró una buena convergencia para el error propuesto, validando su uso en este tipo de simulaciones.

Anexo I: Códigos para la Solución

```
print('\033[H\033[J')
import numpy as np
import matplotlib.pyplot as plt
from gauss_seidel import gauss_seidel
n=100 #orden del sistema (numero de nodos)
n_t=800 #nodos temporales
error=.01 #error propuesto
A=np.zeros(((n,n)),float)
T=70*np.ones(((n)),float)
B=np.zeros(((n)),float)
Ba=np.zeros(((n)),float)
Xc=np.zeros(((n)),float)
E=np.zeros(((n)),float)
Ln=np.zeros(((n)),float)
Lnt=np.zeros(((8)),float)

Tn0=np.ones(((8)),float)
Tnm=np.ones(((8)),float)
Tnf=np.ones(((8)),float)

Tp2=np.ones(((n)),float)
Tp4=np.ones(((n)),float)
Tp6=np.ones(((n)),float)

L=.5/12
t=8/3600
dx=L/n
dt=t/n_t
k_unam=13
k_fimee=22.9
alpha_unam=.1775
alpha_fimee=.298
Tfusion_unam=2570
Tfusion_fimee=2770
T_0h=4500
T_0c=70
h_h=1000
h_c=300
Bh_u=h_h*dx/k_unam
Bh_f=h_h*dx/k_fimee
F_u=alpha_unam*dt/(dx**2)
F_f=alpha_fimee*dx/(dx**2)
Bc_u=h_c*dx/k_unam
Bc_f=h_c*dx/k_fimee


fimee=1
unam=2
material=fimee

if material==fimee:
    alpha=alpha_fimee
    Tfusion=Tfusion_fimee
    Bh=Bh_f
    Bc=Bc_f
    k=k_fimee
    F=F_f

if material==unam:
    alpha=alpha_unam
    Tfusion=Tfusion_unam
    Bh=Bh_u
    Bc=Bc_u
    k=k_unam
```

```
    F=F_u

explicito=1
implicito=2
CrankN=3

case=implicito
if case==1:
   a=1
   b=0
if case==2:
   a=0
   b=1
if case==3:
   a=.5
   b=.5


for r in range(0,8):
   Ba=T
   Tn0[r]=T[0]
   Tnm[r]=T[50]
   Tnf[r]=T[99]
   Lnt[r]=dt*r*3600*100
   if r==2:
      Tp2=T
   if r==4:
      Tp4=T
   if r==6:
      Tp6=T

   for i in range(0,n):
       aux1=i+1
       aux2=i-1
       Ln[i]=dx*i


       if i==0:
          A[i,i]=a*(1)+b*(1+2*Bh*F+2*F)
          A[i,aux1]=a*(0)-b*(2*F)
          B[i]=a*(Ba[i]*(1-2*F-2*F*Bh)+2*F*(Ba[aux1]+Bh*T_0h))+b*(Ba[i]+2*Bh*F*T_0h)
       if i==(n-1):
          A[i,i]=a*(1)+b*(1+2*Bc*F+2*F)
          A[i,aux2]=a*(0)-b*(2*F)
          B[i]=a*(Ba[i]*(1-2*F-2*F*Bc)+2*F*(Ba[aux2]+Bc*T_0c))+b*(Ba[i]+2*Bc*F*T_0c)
       if i!=0 and i!=n-1:
          A[i,i]=a*1+b*(1+2*F)
          A[i,aux1]=-a*(0)-b*(F)
          A[i,aux2]=-a*(0)-b*(F)
          B[i]=a*(F*(Ba[aux1]+Ba[aux2])+(1-2*F)*Ba[i])+b*(Ba[i])
   X=gauss_seidel(A,B,T,Xc,E,n,error,np)


plt.figure()
plt.plot(Lnt,Tn0,label = "x=0 ", color = 'red')
plt.hold(True)
plt.plot(Lnt,Tnm,label = "x=L/2 ", color = 'blue')
plt.plot(Lnt,Tnf,label = "x=l", color = 'green')
plt.grid(color = '0.5', linestyle = '--', linewidth = 1)

plt.savefig('GraficaT-t.pdf')
plt.show()


plt.figure()

plt.hold(True)
```

```
plt.plot(Ln,Tp2,label = "t=2 s", color = 'blue')
plt.plot(Ln,Tp4,label = "t=4 s", color = 'green')
plt.plot(Ln,Tp6,label = "t=6 s", color = 'yellow')
plt.plot(Ln,T,label = "t=8 s", color = 'red')
#plt.grid(True)
plt.grid(color = '0.5', linestyle = '--', linewidth = 1)

plt.savefig('GraficaT-x.pdf')
plt.show()

Xg=np.zeros(((10*n)),float)
for k in range(1,11):
   for j in range(0,11*n):
       if j<k*n and j>=(k-1)*n:
          Xg[j]=T[j-(k-1)*n]

plt.close('all')
arr = Xg.reshape((10,n))
fig = plt.figure(figsize=(7, 3))
im = plt.imshow(arr,cmap='jet')
plt.colorbar()
plt.savefig('perfil.pdf')
```

Listing 1: Código principal: ProyectoTransfe2.py

```
def gauss_seidel(A,B,X,Xc,E,n,error,np):
   ite=0
   Error_prom=1
   while error<Error_prom and ite<10000:
      ite=ite+1
      for i in range(0,n):
         Xc[i]=X[i]
         suma=0
         for j in range(0,n):
            if j!=i:
               suma=suma+(A[i,j]*X[j])
         X[i]=(B[i]-suma)/A[i,i]
         E[i]=abs(X[i])-abs(Xc[i])
      #E[i]=abs(X[i])-abs(Xc[i])
      Error_prom=abs(np.sum(E,axis=0)/n)

   print('Matriz A')
   print(A)
   print('Vector B')
   print(B)
   print('X')
   print(X)
   print('Xc')
   print(Xc)
   print('iteracion')
   print(ite)
   print('Error Promedio')
   print(Error_prom)
   return(X)
```

Listing 2: Función Gauss–Seidel: gauss_seidel.py