Ecuaciones Diferenciales Parciales: Solución Numérica por el Método de Diferencias Finitas

Rodrigo Cárdenas Domínguez

30 de abril de 2018

1. EDP Lineales de Segundo Orden

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + D = 0$$

$B^2 - 4AC$	Ecuación	Ejemplo
>0	Elípticas	$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y)$
$= 0$	Parabólicas	$\frac{\partial u}{\partial t} - \alpha^2 \frac{\partial^2 u}{\partial x^2} = 0$
<0	Hiperbólicas	$\alpha^2 \frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial t^2} = 0$

2. Método de Diferencias Finitas: Ecuaciones Elípticas

Estas ecuaciones se presentan en problemas en estado estacionario con valores de frontera. Por lo que en este caso, el flujo debe ser igual en la entrada y en la salida.

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2}$$

$$\frac{\partial^2 T}{\partial y^2} = \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y^2}$$

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y^2} = 0$$

$$T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1} - 4T_{i,j} = 0$$

$$T_{i,j} = \frac{T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1}}{4}$$

```python
# Made by RODRIGO CARDENAS DOMINGUEZ.
# March 2nd, 2018.
# UNIVERSIDAD IBEROAMERICANA, ENGINEERING PHYSICS.

# FINITE DIFFERENCE: ELYPTIC EQUATIONS.
# LAPLACE EQUATION FOR A HEATED PLATE.
# DIRICHLET TYPE FRONTIER CONDITIONS.

import matplotlib.pyplot as plt
import numpy as np
import math as mt

# dx = 1 cm

M = ['Aluminum', 'Copper', 'Silver', 'Steel', 'Tin']
a = np.array([0.835, 1.11, 1.6563, 1.55, 0.4])
p = np.array([2.7, 8.96, 10.49, 8.05, 7.27])
C = np.array([0.2174, 0.0923, 0.056, 0.107, 0.05])
k = np.zeros(5)

# a : coefficient of thermal diffusity [cm^2/s]
# p : density [g/cm^3]
# C : heat capacity [cal/(g C)]
# k : coefficient of thermal conductivity [cal/(s cm C)]

X = np.arange(6)
Y = np.arange(6)

plt.figure(figsize=(15, 30))
plt.suptitle('Temperature Distribution and Flux', fontsize=18)

for n in range(0, 5):

    k[n] = a[n]*p[n]*C[n]

    A = np.array([[75.0, 0.0, 0.0, 0.0, 0.0, 50.0],
                  [75.0, 0.0, 0.0, 0.0, 0.0, 50.0],
                  [75.0, 0.0, 0.0, 0.0, 0.0, 50.0],
                  [75.0, 0.0, 0.0, 0.0, 0.0, 50.0],
                  [75.0, 0.0, 0.0, 0.0, 0.0, 50.0],
                  [100.0, 100.0, 100.0, 100.0, 100.0, 100.0]])

    error = 1.0

    qx = np.zeros((6,6))
    qy = np.zeros((6,6))
```

```
while( error > 0.001):
    T = np.array(A)
    for i in range(1, 5):
        for j in range(1, 5):
            T[i, j] = (T[i+1, j+1] + T[i-1, j] + T[i, j+1] + T[i, j-1])/4
            T[i, j] = 1.5*T[i, j] - 0.5*A[i, j]
    error = abs((T[1,1] - A[1,1])/T[1,1])
    A = T

for i in range(1, 5):
    for j in range(1, 5):
        qx[i,j] = (-k[n])*((A[i+1, j] - A[i-1, j])/20)
        qy[i,j] = (-k[n])*((A[i, j+1] - A[i, j-1])/20)

plt.subplot(5,2,2*n+1)
plt.contourf(X,Y,A,150)
plt.xlabel('dX')
plt.ylabel('dY')
plt.title(M[n])
plt.set_cmap('jet')
plt.colorbar()

plt.subplot(5,2,2*n+2)
plt.streamplot(X,Y,qy,qx, cmap='autum')
plt.xlabel('dX')
plt.ylabel('dY')
plt.title(M[n])
```

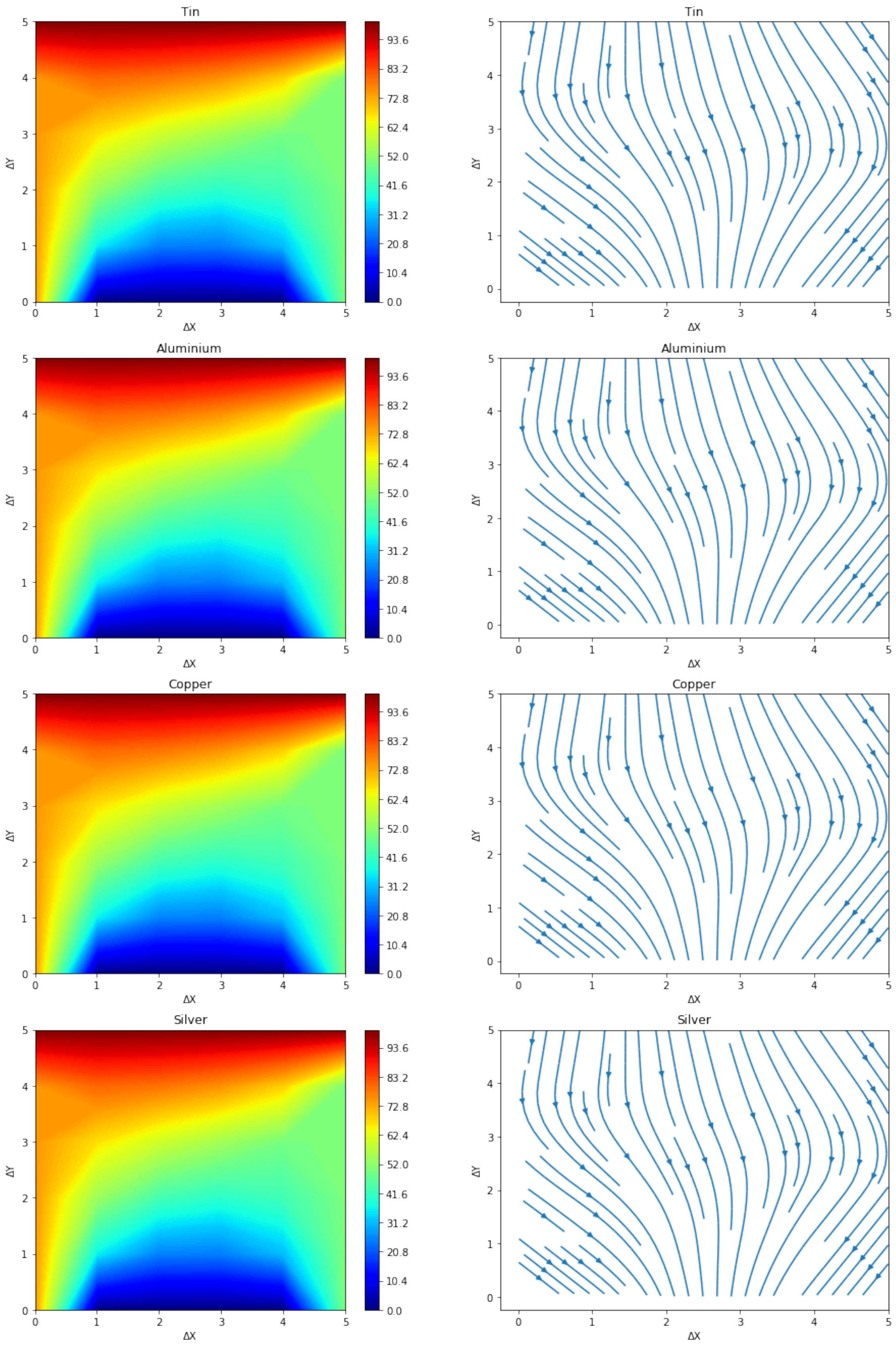

Figura 1: Distribución de calor en estado estacionario para una placa de Sn, Al, Cu y Ag.

2.1. Condiciones de Frontera Tipo Neumann

Cuando las condiciones de frontera no son constantes en el tiempo, es necesario incorporar la derivada en la ecuación Laplaciana en diferencias. Para el nodo en frontera $(0, j)$, la ecuación se reescribe de la forma:

$$T_{1,j} + T_{-1,j} + T_{0,j+1} + T_{0,j-1} - 4T_{0,j} = 0$$

$$\frac{\partial T}{\partial x} = \frac{T_{1,j} - T_{-1,j}}{2\Delta x}$$

$$T_{-1,j} = T_{1,j} - 2\Delta x \frac{\partial T}{\partial x}$$

$$2T_{1,j} + T_{0,j+1} + T_{0,j-1} - 4T_{0,j} - 2\Delta x \frac{\partial T}{\partial x} = 0$$

Similarmente para un nodo en frontera $(i, 0)$ se obtiene:

$$2T_{i,1} + T_{i+1,0} + T_{i-1,0} - 4T_{i,0} - 2\Delta y \frac{\partial T}{\partial y} = 0$$

```
# Made by RODRIGO CARDENAS DOMINGUEZ.
# March 18, 2018.
# UNIVERSIDAD IBEROAMERICANA, ENGINEERING PHYSICS.

# FINITE DIFFERENCE: ELYPTIC EQUATIONS.
# LAPLACE EQUATION FOR A HEATED PLATE.
# NEUMANN TYPE FRONTIER CONDITIONS.

import matplotlib.pyplot as plt
import numpy as np
import math as mt

M = ['Aluminum', 'Copper', 'Silver', 'Steel', 'Tin']
a = np.array([0.835, 1.11, 1.6563, 1.55, 0.4])
p = np.array([2.7, 8.96, 10.49, 8.05, 7.27])
C = np.array([0.2174, 0.0923, 0.056, 0.107, 0.05])
k = np.zeros(5)

# a : coefficient of thermal diffusity [cm^2/s]
# p : density [g/cm^3]
# C : heat capacity [cal/(g C)]
# k : coefficient of thermal conductivity [cal/(s cm C)]

# dx = 10 cm
plt.figure(figsize=(15, 10))
plt.suptitle('Temperature Distribution', fontsize=18)

for n in range(0, 4):

    k = a[n]*p[n]*C[n]
    A = np.array([[75.0, 0.0, 0.0, 0.0, 50.0],
                  [75.0, 0.0, 0.0, 0.0, 50.0],
                  [75.0, 0.0, 0.0, 0.0, 50.0],
                  [75.0, 0.0, 0.0, 0.0, 50.0],
                  [100.0, 100.0, 100.0, 100.0, 100.0]])

    error = 1.0
    X = np.arange(5)
    Y = np.arange(5)
    qx = np.zeros((5,5))
    qy = np.zeros((5,5))

    while( error > 0.001):
        T = np.array(A)
        for i in range(0, 4):
            for j in range(1, 4):
                if(i > 0):
                    T[i, j] = (T[i+1, j] + T[i-1, j] + T[i, j+1] + T[i, j-1])/4
                    T[i, j] = 1.5*T[i, j] - 0.5*A[i, j]
                if(i == 0):
                    T[i, j] = (T[i+1, j] + T[i-1, j] + 2*T[i, j+1])/4
        error = abs((T[1,1] - A[1,1])/T[1,1])
        A = T

    for i in range(1, 4):
        for j in range(1, 4):
            qx[i,j] = (-k)*((A[i+1, j] - A[i-1, j])/20)
            qy[i,j] = (-k)*((A[i, j+1] - A[i, j-1])/20)

    plt.subplot(2,4,2*n+1)
    plt.contourf(X,Y,A,200)
    plt.xlabel('dX')
    plt.ylabel('dY')
    plt.title('Temperature Distribution')
    plt.set_cmap('jet')
    plt.colorbar()
    plt.subplot(2,4,2*n+2)
    plt.streamplot(X,Y,qy,qx,)
    plt.xlabel('dX')
    plt.ylabel('dY')
    plt.title('Temperature Flux')
    plt.set_cmap('autumn')

plt.show()
```

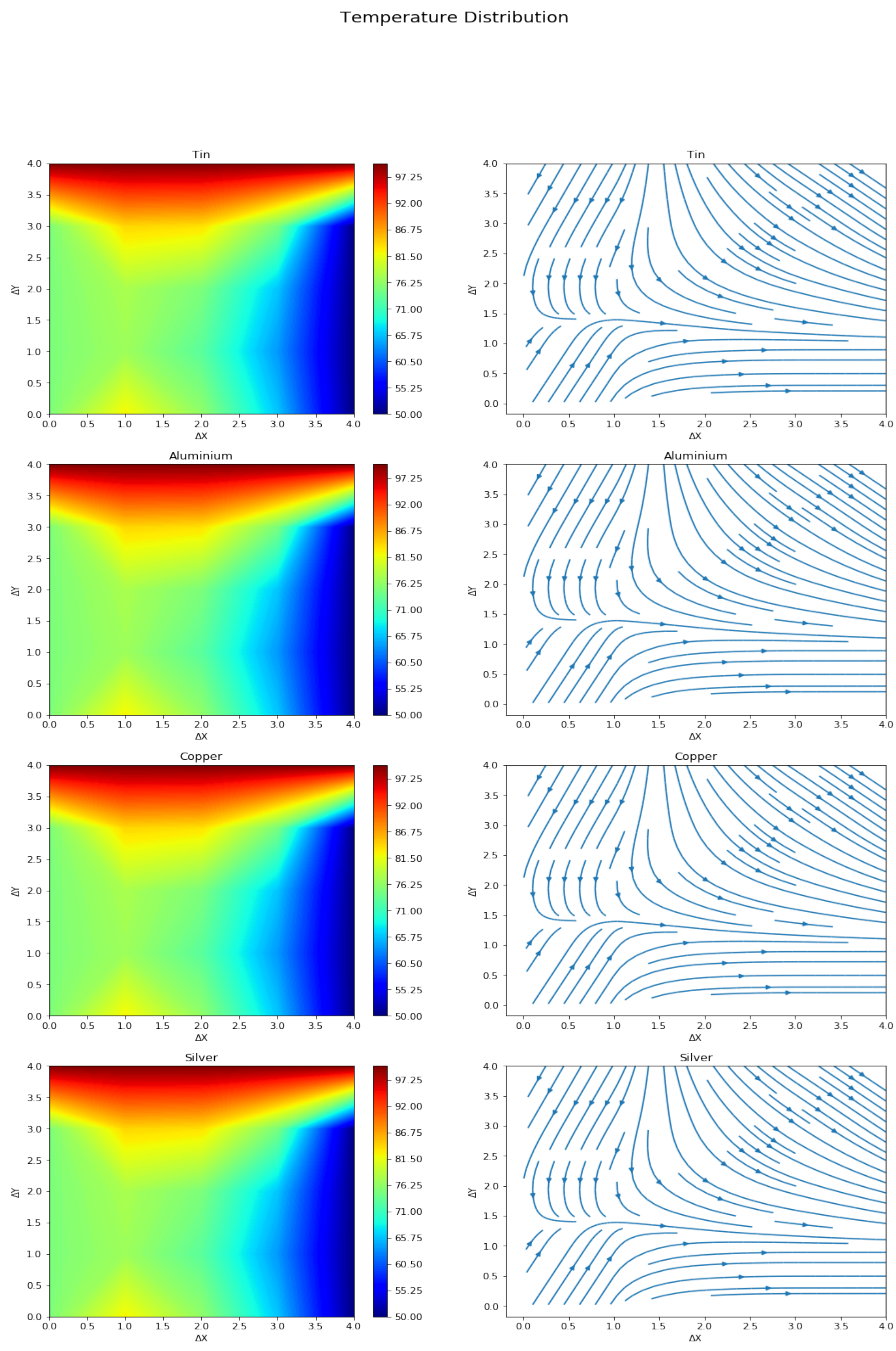

Figura 2: GDistribución de calor en estado estacionario para una placa de Sn, Al, Cu y Ag. Frontera inferior aislada.

3. Método de Diferencias Finitas: Ecuaciones Parabólicas

Estas ecuaciones se caracterizan por tener dependencia tanto en el tiempo como en el espacio, por lo que mostraran como una magnitud cambiara espacialmente conforme el tiempo incrementa. Un ejemplo de ecuación parabólica es la ecuación de conducción de calor:

$$\frac{\partial T}{\partial t} = \alpha\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}\right)$$

$$\alpha = \frac{k}{\rho C_p}$$

3.1. Método Explícito

$$\frac{u_j^{n+1} - u_j^n}{k} = \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{h^2}$$

$$\frac{\partial T}{\partial t} = \frac{T_i^{l+1} - T_i^l}{\Delta t} \qquad \frac{\partial^2 T}{\partial x^2} = \frac{T_{i+1}^l - 2T_i^l + T_{i-1}^l}{(\Delta x)^2}$$

$$\frac{T_i^{l+1} - T_i^l}{\Delta t} = \alpha\frac{T_{i+1}^l - 2T_i^l + T_{i-1}^l}{(\Delta x)^2}$$

$$T_i^{l+1} = \alpha\frac{\Delta t}{(\Delta x)^2}\left[T_{i+1}^l - 2T_i^l + T_{i-1}^l\right] + T_i^l$$

$$T_i^{l+1} = \lambda\left[T_{i+1}^l - 2T_i^l + T_{i-1}^l\right] + T_i^l$$

Donde $\lambda = \alpha\frac{\Delta t}{(\Delta x)^2}$

```python
# Made by RODRIGO CARDENAS DOMINGUEZ.
# March 27th, 2018.
# UNIVERSIDAD IBEROAMERICANA, ENGINEERING PHYSICS.

# FINITE DIFFERENCE: PARABOLIC EQUATIONS.
# HEAT CONDUCTION EQUATION.
# EXPLICIT SOLUTION OF THE ONE-DIMENSIONAL HEAT-CONDUCTION EQUATION.

import matplotlib.pyplot as plt
import numpy as np
import math as mt

M = ['Tin', 'Aluminium', 'Copper', 'Silver']
a = np.array([0.4, 0.835, 1.11, 1.6563])
# a : coefficient of thermal diffusity [cm^2/s]

dt = 0.1 # s
dx = 0.5 # cm

plt.figure(figsize=(15, 17.5))
plt.suptitle('Temperature Distribution', fontsize=18)

for n in range(0, 4):

    B = a[n]*(dt/(dx*dx))
    A = np.zeros(21);
    A[0] = 100; A[20] = 50
    X = np.arange(0, 10.5, 0.5)
    t = 0.0

    A1 = A
    T = A

    while( t < 48):
        for i in range(1, 20):
            A1[i] = A[i] + B*(A[i+1] - 2*A[i] + A[i-1])

        T = np.vstack((T, A1))
        A = A1
        t = t + 0.1

    plt.subplot(3,2,n+1)
    plt.plot(X, T[30, :], label='3 sec')
    plt.plot(X, T[60, :], label='6 sec')
    plt.plot(X, T[90, :], label='9 sec')
    plt.plot(X, T[120, :], label='12 sec')
    plt.plot(X, T[480, :], label='48 sec')
    plt.legend()
    plt.xlabel('dx')
    plt.ylabel('Temperature C')
    plt.title(M[n])

plt.show()
```

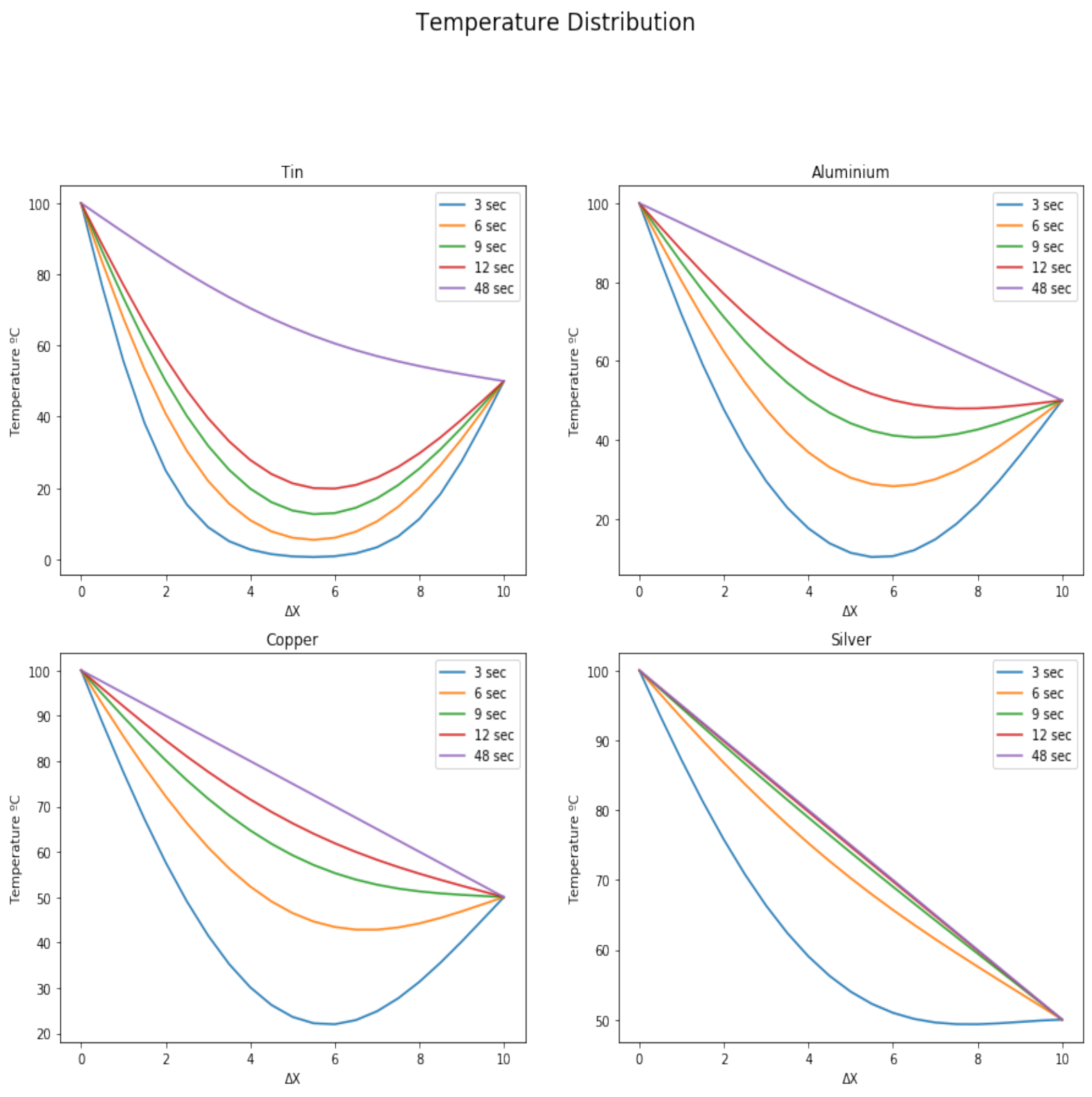

Figura 3: Evolución de la distribución de calor para una barra de Sn, Al, Cu y Ag.

3.2. Método Implícito Simple

$$\frac{\partial T}{\partial t} = \frac{T_i^{l+1} - T_i^l}{\Delta t} \qquad \frac{\partial^2 T}{\partial x^2} = \frac{T_{i+1}^{l+1} - 2T_i^{l+1} + T_{i-1}^{l+1}}{(\Delta x)^2}$$

$$\frac{T_i^{l+1} - T_i^l}{\Delta t} = \alpha \frac{T_{i+1}^{l+1} - 2T_i^{l+1} + T_{i-1}^{l+1}}{(\Delta x)^2}$$

$$T_i^{l+1} = \alpha \frac{\Delta t}{(\Delta x)^2} \left[T_{i+1}^{l+1} - 2T_i^{l+1} + T_{i-1}^{l+1} \right] + T_i^l$$

$$-\lambda T_{i+1}^{l+1} + (1 + 2\lambda) T_i^{l+1} - \lambda T_{i-1}^{l+1} = T_i^l$$

3.3. Método de Crank-Nichelson

$$\frac{\partial T}{\partial t} = \frac{T_i^{l+1} - T_i^l}{\Delta t}$$

$$\frac{\partial^2 T}{\partial x^2} = \frac{1}{2} \left[\frac{T_{i+1}^l - 2T_i^l + T_{i-1}^l}{(\Delta x)^2} + \frac{T_{i+1}^{l+1} - 2T_i^{l+1} + T_{i-1}^{l+1}}{(\Delta x)^2} \right]$$

$$\frac{T_i^{l+1} - T_i^l}{\Delta t} = \frac{\alpha}{2} \left[\frac{T_{i+1}^l - 2T_i^l + T_{i-1}^l}{(\Delta x)^2} + \frac{T_{i+1}^{l+1} - 2T_i^{l+1} + T_{i-1}^{l+1}}{(\Delta x)^2} \right]$$

$$-\lambda T_{i-1}^{l+1} + 2(1 + \lambda) T_i^{l+1} - \lambda T_{i-1}^{l+1} = \lambda T_{i-1}^l + 2(1 - \lambda) T_i^l + \lambda T_{i+1}^l$$

4. Ecuaciones Parabólicas en Dos Dimensiones

$$\frac{\partial T}{\partial t} = \alpha\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right)$$

4.1. Método Implícito de Dirección Alternante

El método resuelve ecuaciones en dos dimensiones usando matrices tridiagonales. Cada incremento de tiempo se ejecuta en dos pasos. En el primer paso, la ecuación queda:

$$\frac{T_{i,j}^{l+1/2} - T_{i,j}^{l}}{\Delta t/2} = \alpha\left[\frac{T_{i+1,j}^{l} - 2T_{i,j}^{l} + T_{i-1,j}^{l}}{(\Delta x)^2} + \frac{T_{i+1,j}^{l+1/2} - 2T_{i,j}^{l+1/2} + T_{i-1,j}^{l+1/2}}{(\Delta y)^2}\right]$$

La aproximación de $\partial^2 T/\partial x^2$ queda expresada de manera explícita, por lo que $\partial^2 T/\partial y^2$ queda de forma implícita. Para una malla donde $\Delta x = \Delta y$, la ecuación se escribe:

$$-\lambda T_{i,j-1}^{l+1/2} + 2(1+\lambda)T_{i,j}^{l+1/2} - \lambda T_{i,j+1}^{l+1/2} = \lambda T_{i-1,j}^{l} + 2(1-\lambda)T_{i,j}^{l} + \lambda T_{i+1,j}^{l}$$

$$\begin{bmatrix} 2(1+\lambda) & -\lambda & 0 \\ -\lambda & 2(1+\lambda) & -\lambda \\ 0 & -\lambda & 2(1+\lambda) \end{bmatrix} \begin{bmatrix} T_{i,j} \\ T_{i,j+1} \\ T_{i,j+2} \end{bmatrix} = \begin{bmatrix} \lambda(T_{i-1,j}^{l} + T_{i+1,j}^{l} + T_{i,j-1}^{l+1/2}) + 2(1-\lambda)T_{i,j}^{l} \\ \lambda(T_{i-1,j+1}^{l} + T_{i+1,j+1}^{l} + +2(1-\lambda)T_{i,j+1}^{l} \\ \lambda(T_{i-1,j+2}^{l} + T_{i+1,j+2}^{l} + T_{i,j+3}^{l+1/2}) + 2(1-\lambda)T_{i,j+2}^{l} \end{bmatrix}$$

En el segundo paso, que va desde $t^{l+1/2}$ hasta t^{l+1}, la ecuación se aproxima por:

$$\frac{T_{i,j}^{l+1} - T_{i,j}^{l+1/2}}{\Delta t/2} = \alpha\left[\frac{T_{i+1,j}^{l+1} - 2T_{i,j}^{l+1} + T_{i-1,j}^{l+1}}{(\Delta x)^2} + \frac{T_{i+1,j}^{l+1/2} - 2T_{i,j}^{l+1/2} + T_{i-1,j}^{l+1/2}}{(\Delta y)^2}\right]$$

En este caso, la aproximación de $\partial^2 T/\partial y^2$ es explícita, por lo que $\partial^2 T/\partial x^2$ es implícita.

$$-\lambda T_{i-1,j}^{l+1} + 2(1+\lambda)T_{i,j}^{l+1} - \lambda T_{i+1,j}^{l+1} = \lambda T_{i,j-1}^{l+1/2} + 2(1-\lambda)T_{i,j}^{l+1/2} + \lambda T_{i,j+1}^{l+1/2}$$

$$\begin{bmatrix} 2(1+\lambda) & -\lambda & 0 \\ -\lambda & 2(1+\lambda) & -\lambda \\ 0 & -\lambda & 2(1+\lambda) \end{bmatrix} \begin{bmatrix} T_{i,j} \\ T_{i+1,j} \\ T_{i+2,j} \end{bmatrix} = \begin{bmatrix} \lambda(T_{i,j-1}^{l+1/2} + T_{i,j+1}^{l*1/2} + T_{i-1,j}^{l+1}) + 2(1-\lambda)T_{i,j}^{l+1/2} \\ \lambda(T_{i+1,j-1}^{l+1/2} + T_{i+1,j+1}^{l+1/2} + 2(1-\lambda)T_{i+1,j}^{l+1/2} \\ \lambda(T_{i-1,j+2}^{l+1/2} + T_{i+1,j+2}^{l+1/2} + T_{i,j+3}^{l+1}) + 2(1-\lambda)T_{i,j+2}^{l+1/2} \end{bmatrix}$$

```python
# Made by RODRIGO CARDENAS DOMINGUEZ.
# April 7th, 2018.
# UNIVERSIDAD IBEROAMERICANA, ENGINEERING PHYSICS.

# FINITE DIFFERENCE: PARABOLIC EQUATIONS IN TWO SPATIAL DIMENSIONS.
# HEAT CONDUCTION EQUATION.
# ALTERNATING-DIRECTION IMPLICIT METHOD.

import matplotlib.pyplot as plt
import numpy as np
import math as mt

M = ['Tin', 'Aluminium', 'Copper', 'Silver']
a = np.array([0.4, 0.835, 1.11, 1.6563])
p = np.array([7.27, 2.7, 8.96, 10.49])
C = np.array([0.05, 0.2174, 0.0923, 0.056])
k = np.zeros(4)

# a:  coefficient of thermal diffusity [cm^2/s]
# p : density [g/cm^3]
# C : heat capacity [cal/(g C)]
# k : coefficient of thermal conductivity [cal/(s cm C)]

dt = 10 # s
dx = 10 # cm
X = np.arange(5)
Y = np.arange(5)

plt.figure(figsize=(15, 11))
plt.suptitle('Temperature Distribution', fontsize=18)

for n in range(0, 4):

    t = 0.0
    B = a[n]*(dt/(dx*dx))
    k[n] = a[n]*p[n]*C[n]

    A = np.array([[75.0, 0.0, 0.0, 0.0, 50.0],
                  [75.0, 0.0, 0.0, 0.0, 50.0],
                  [75.0, 0.0, 0.0, 0.0, 50.0],
                  [75.0, 0.0, 0.0, 0.0, 50.0],
                  [100.0, 100.0, 100.0, 100.0, 100.0]])

    U = np.array([[2*(1+B),      -B,        0.0],
                  [-B,      2*(1+B),         -B],
                  [0.0,          -B,   2*(1+B)]])
    while( t < 30):
        i = 1
        for j in range(1, 4):
            L = np.array([B*(A[i, j-1] + A[i, j+1] + A[i-1, j]) + 2*(1-B)*A[i, j],
                          B*(A[i+1, j-1] + A[i+1, j+1]) + 2*(1-B)*A[i+1, j],
                          B*(A[i+2, j-1] + A[i+2, j+1] + A[i+3, j]) + 2*(1-B)*A[i+2, j]])
            T = np.linalg.solve(U, L)
            A[1:4, j] = T

        j = 1
        for i in range(1, 4):
            L = np.array([B*(A[i-1, j] + A[i+1, j] + A[i, j-1]) + 2*(1-B)*A[i, j],
                          B*(A[i-1, j+1] + A[i+1, j+1]) + 2*(1-B)*A[i, j+1],
                          B*(A[i-1, j+2] + A[i+1, j+2] + A[i, j+3]) + 2*(1-B)*A[i, j+2]])
            T = np.linalg.solve(U, L)
            A[i, 1:4] = T
        t = t + 1

    plt.subplot(2,2,n+1)
    plt.contourf(X,Y,A,150)
    plt.xlabel('dX')
    plt.ylabel('dY')
    plt.title(M[n] + ' at '+ str(t) +'s')
    plt.set_cmap('jet')
    plt.colorbar()

plt.show()
```

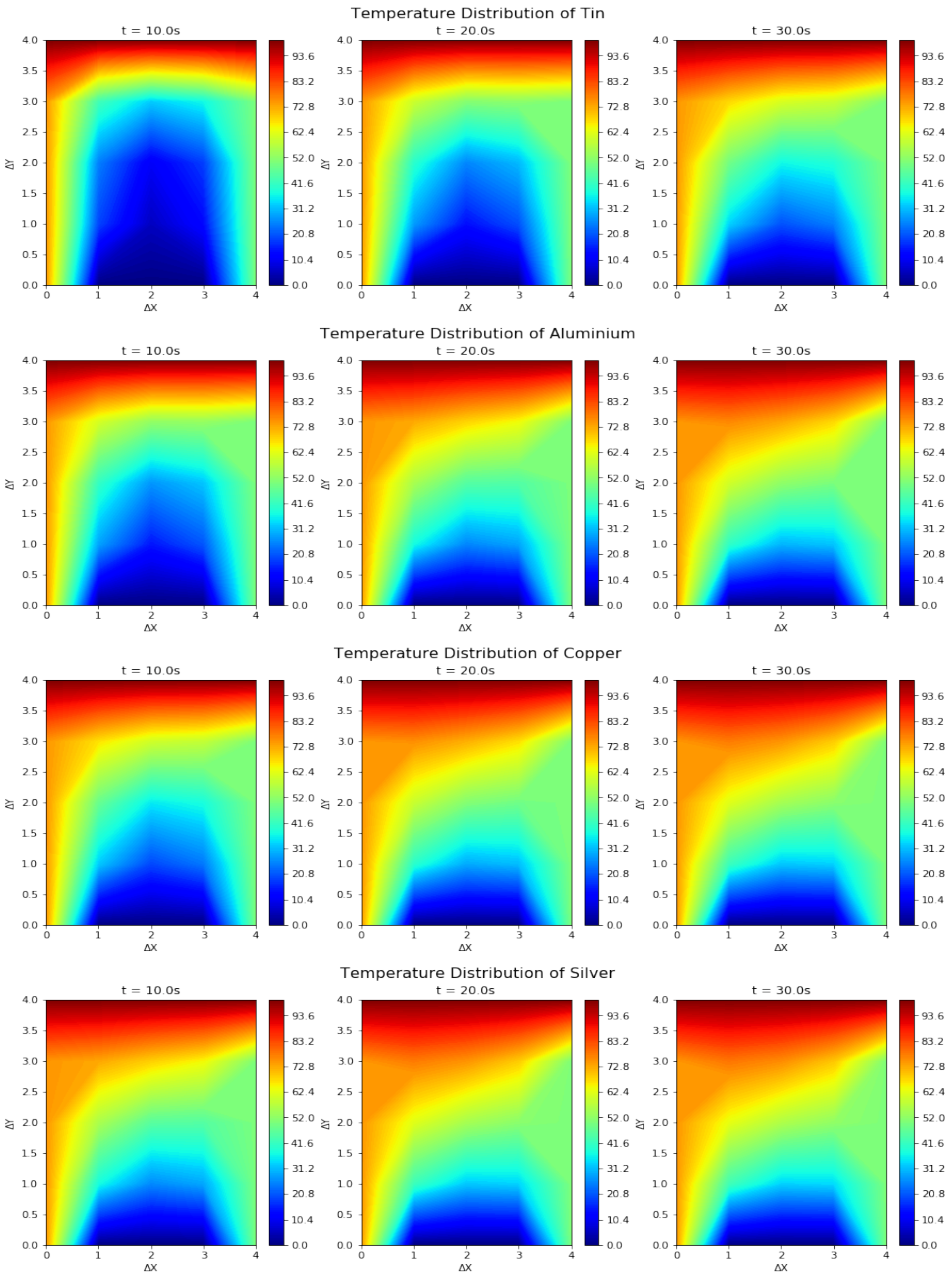

Figura 4: Evolución en la distribución de calor para una placa de Sn, Al, Cu y Ag.

5. Referencias

Steven C. Chapra & Raymond P. Canale. "Numerical Methods for Engineers". *Mc Graw-Hill*, 2015. Seventh Edition.

Richard L. Burden & J. Douglas Faires. "Numerical Analysis". *Brooks/Cole*, 2011. Ninth Edition.