UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DEPARTAMENTO DE INGENIERÍA MECÁNICA

Proyecto: EDP
Resolución de la Ecuación de Calor en una Placa Plana en Estado Transiente

Computación Científica

IMP-458 - 2022

Departamento Ing. Mecánica UTFSM

Martín Achondo Mercado

Rol: 201860005-9

Profesor: Franco Perazzo M.

18 de Agosto de 2022

Resumen

En este trabajo se resolvió la ecuación de calor transiente en una placa rectangular bajo ciertas condiciones de borde. Para su resolución se utilizaron diferencias finitas con esquemas de Euler Explícito y de Crank-Nicolson bajo distintas mallas espaciales y temporales. Del resultado de las modelaciones se obtuvo que a simple vista los distintos esquemas, mientras converjan, no generan diferencias notorias. De esta manera, ambos logran reproducir la evolución de la distribución de la temperatura y el flujo de calor en la placa. De todas formas, el método de Crank-Nicolson bajo mallas más finas logran residuales menores y por ende, mayor convergencia. Al alcanzar el estado estacionario, las soluciones de ambos esquemas fueron comparados con los obtenidos por parte del software comercial de ANSYS y el código adjunto en la publicación de P. Beckers y B. Beckers [4] bajo elementos finitos. Con esto, se obtuvo distribuciones similares en donde la temperatura máxima de la placa no variaba más de 4 %. A partir de las modelaciones, se estima que el estado estacionario se alcanzaría en 54000 segundos aproximadamente.

Índice

1. Introducción

En este trabajó se modelará la transferencia de calor en una placa rectangular. El propósito es encontrar la distribución de temperaturas en diversos instantes de tiempo junto al flujo de calor. Para esto, se discretizará con diferencias finitas la ecuación de calor con un esquema de Euler Explícito y de Crank-Nicolson para evaluar las variaciones respecto a los pasos temporales y espaciales. Además, se resolverá el mismo problema utilizando el software comercial de ANSYS y el código adjunto en la publicación de P. Beckers y B. Beckers [4] con elementos finitos. De esta manera se podrá comparar y validar la solución obtenida.

1.1. Presentación del Problema

Se tiene una placa rectangular de 2x1 [m] calentada en la parte superior e inferior con un flujo de calor constante de $q = 10$ [kW/m^2] desde un instante inicial en que toda la placa está a una temperatura $T = 0$ [°C]. Además, la placa tiene zonas en la que se encuentra térmicamente aislada y en los bordes laterales una condición de temperatura fija de $T = 0$ [°C]. Una mejor representación se visualiza en la siguiente imagen:

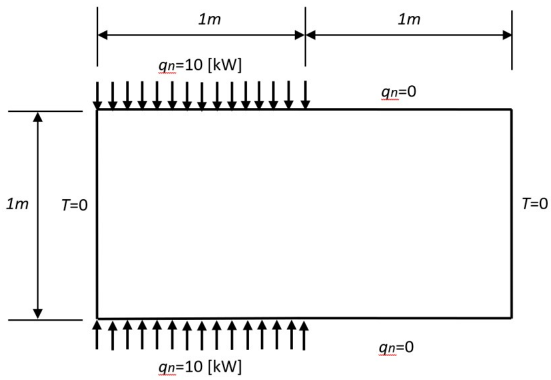


Figura 1: Placa a modelar

Las constantes características del material son: $\rho = 8862$ [kg/m^3], $k = 100$ [W/m°C] y $c_p = 421$ [J/kg°C].

Dicho esto, para el problema se pide:

1. Resolver el problema de transferencia de calor bidimensional transiente utilizando un esquema de Euler Explícito. Para ello se pide utilizar espaciamientos espaciales $\Delta x = \Delta y$ de 2 y 10 [cm].

2. Comparar los resultados al resolver el problema de transferencia de calor bidimensional transiente utilizando un esquema de Crank-Nicolson.

3. Estimar el tiempo necesario para que el sistema alcance un estado estacionario.

4. Comparar los resultados con los obtenidos al resolver el problema utilizando el método de elementos finitos en el software de ANSYS.
5. Comparar los resultados con los obtenidos al resolver el problema utilizando el método de elementos finitos programado por los autores P. Beckers y B. Beckers en [4].

2. Metodología

En este problema se pide resolver la ecuación de calor en una placa rectangular. Para simplificar el problema, se considerará una transferencia de calor bidimensional, propiedades de los materiales constantes y la no existencia de generación de energía. Dado esto, la ecuación con sus condiciones de borde e iniciales se puede plantear como:

$$\frac{\partial T}{\partial t} = \alpha \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) \qquad 0 \leq x \leq 2 \quad 0 \leq y \leq 1 \quad t \geq 0 \tag{1}$$

$$\begin{cases} T(x,y,0) = 0 & 0 \leq x \leq 2 \quad 0 \leq y \leq 1 \quad t = 0 \\ -k\dfrac{\partial T}{\partial y} = q_w & 0 \leq x \leq 1 \quad y = 0 \quad t \geq 0 \\ -k\dfrac{\partial T}{\partial y} = 0 & 1 < x \leq 2 \quad y = 0 \quad t \geq 0 \\ T(2,y,t) = 0 & x = 2 \quad 0 \leq y \leq 1 \quad t \geq 0 \\ -k\dfrac{\partial T}{\partial y} = -q_w & 0 \leq x \leq 1 \quad y = 1 \quad t \geq 0 \\ -k\dfrac{\partial T}{\partial y} = 0 & 1 < x \leq 2 \quad y = 1 \quad t \geq 0 \\ T(0,y,t) = 0 & x = 0 \quad 0 \leq y \leq 1 \quad t \geq 0 \end{cases}$$

En donde se resolverá para la función $T = T(x, y, t)$. En la ecuación α, k, q_w hacen referencia a la difusividad térmica, conducitividad térmica y al flujo de calor en la pared respectivamente. Notar la existencia de condiciones de borde de Dirichlet en las paredes laterales y Neumann en la superior e inferior.

2.1. Discretización

La ecuación 1 se resolverá numéricamente utilizando 2 esquemas de discretización para obtener la distribución de temperatura a lo largo de la placa para distintos instantes de tiempo por el método de diferencias finitas. Así, se podrá comparar el orden de convergencia de cada esquema. Se utilizará el mismo espaciamiento espacial $\Delta = \Delta x = \Delta y$. La cantidad de nodos espaciales entonces será:

$$\begin{aligned} N_x &= \frac{\ell_x}{\Delta x} + 1 \\ N_y &= \frac{\ell_y}{\Delta y} + 1 \end{aligned} \tag{2}$$

De esta manera, se tendrán $N_x \times N_y$ nodos espaciales, con: $i = 1, 2, 3, \cdots, N_x - 1, N_x$ y $j = 1, 2, 3, \cdots, N_y - 1, N_y$. El paso temporal será Δt, por lo que la cantidad de pasos temporales contando el instante inicial será de:

$$N_t = \frac{t_f}{\Delta t} + 1 \tag{3}$$

Por otra parte, un número adimensional importante que tiene relación con la discretización del problema es el número de Fourier, que puede ser definido en cada dirección $k = x, y$ como:

$$\sigma_k = \alpha \frac{\Delta t}{\Delta x_k^2} \tag{4}$$

Para todos los esquemas a presentar, se discretizará la segunda derivada de la temperatura respecto al espacio como:

$$\begin{aligned} \frac{\partial^2 T}{\partial x^2} &= \frac{T_{i-1j} - 2T_{ij} + T_{i+1j}}{\Delta x^2} + \mathcal{O}(\Delta x^2) \\ \frac{\partial^2 T}{\partial y^2} &= \frac{T_{ij-1} - 2T_{ij} + T_{ij+1}}{\Delta y^2} + \mathcal{O}(\Delta y^2) \end{aligned} \tag{5}$$

2.1.1. Condiciones de Borde

Para las condiciones de borde se utilizarán los siguientes esquemas:

- Dirichlet: Simplemente se fija la temperatura en los nodos laterales:

$$T_{1j} = 0 \quad ; \quad T_{N_x,j} = 0$$

- Neumann: Se discretiza el calor con diferencias centradas en la pared superior e inferior:

$$q_{i1} = -k\frac{T_{i2} - T_{i0}}{\Delta y} \quad ; \quad q_{iN_y} = -k\frac{T_{iN_y+1} - T_{iN_y-2}}{\Delta y} \tag{6}$$

  Así, se despeja T_{i0} o T_{iN_y+1} (nodos no existentes) y se reemplaza en la EDP para obtener ecuaciones adicionales. Estas obtenidas serán presentadas en las secciones siguientes. Notar que si la pared está aislada basta reemplazar $q = 0$.

2.1.2. Esquema Euler Explícito

Discretiza la ecuación de calor utilizando los valores en el paso anterior. Así, la ecuación queda como:

$$\frac{T^{n+1} - T^n}{\Delta t} = f(t^n, T^n) \tag{7}$$

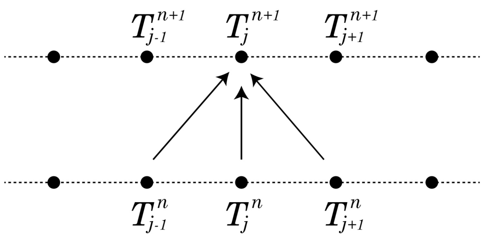


Figura 2: Esquema de discretización Euler Explícito

Reemplazando las derivadas espaciales se obtiene:

$$\frac{T_{ij}^{n+1} - T_{ij}^n}{\Delta t} = \alpha \left(\frac{T_{i-1j}^n - 2T_{ij}^n + T_{i+1j}^n}{\Delta x^2} + \frac{T_{ij-1}^n - 2T_{ij}^n + T_{ij+1}^n}{\Delta y^2} \right) \tag{8}$$

Despejando T^{n+1} se forman las siguientes ecuaciones para cada nodo interior ij. Con esto, el método queda con orden $\mathcal{O}(\Delta x^2, \Delta y^2, \Delta t)$, primer orden en tiempo y segundo en espacio.

$$T_{ij}^{n+1} = T_{ij}^n + \sigma(T_{i-1j}^n + T_{i+1j}^n + T_{ij-1}^n + T_{ij+1}^n - 4T_{ij}^n) \tag{9}$$

En donde $\sigma = \sigma_x = \sigma_y$. Un detalle importante sobre esta discretización es que es condicionalmente estable. Para su estabilidad se debe asegurar lo siguiente respecto al número de Fourier:

$$\alpha \frac{\Delta t}{\Delta x^2} + \alpha \frac{\Delta t}{\Delta y^2} = 2\sigma < \frac{1}{2} \tag{10}$$

Así, se necesita un valor de $\sigma = 1/4$. A esto se le llama a la condición CFL para difusión. Dada la utilización de nodos "fantasmas" para discretizar la condición de borde, las ecuaciones en los bordes inferior y superiores se modifican de la siguiente manera completando el sistema:

$$\begin{aligned} T_{i1}^{n+1} &= T_{i1}^n + \sigma(T_{i-11}^n + T_{i+11}^n + 2T_{i2}^n - 4T_{i1}^n - \theta q_w) \\ T_{iN_y}^{n+1} &= T_{iN_y}^n + \sigma(T_{i-1N_y}^n + T_{i+1N_y}^n + 2T_{iN_y-1}^n - 4T_{ij}^n - \theta q_w) \end{aligned} \tag{11}$$

En donde $\theta = -2\Delta/k$. En las ecuaciones presentadas cuando algún nodo hace referencia a una esquina o nodo lateral, basta reemplazar la condición de Dirichlet presentada.

2.1.3. Esquema Euler Crank Nicolson

Discretiza la ecuación de calor utilizando los valores del paso anterior y actual. Así, la ecuación queda como:

$$\frac{T^{n+1}-T^{n}}{\Delta t}=\frac{1}{2}\left(f(t^{n+1},T^{n+1})+f(t^{n},T^{n})\right) \tag{12}$$

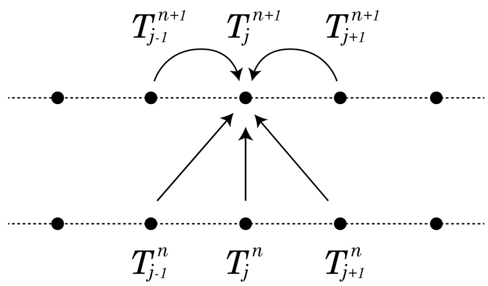


Figura 3: Esquema de discretización Crank Nicolson

Reemplazando las derivadas espaciales se obtiene:

$$\begin{aligned}\frac{T_j^{n+1}-T_j^{n}}{\Delta t}=\frac{\alpha}{2}\Bigg(&\frac{T_{i-1j}^{n+1}-2T_{ij}^{n+1}+T_{i+1j}^{n+1}}{\Delta x^2}+\frac{T_{ij-1}^{n+1}-2T_{ij}^{n+1}+T_{ij+1}^{n+1}}{\Delta y^2}+\\&+\frac{T_{i-1j}^{n}-2T_{ij}^{n}+T_{i+1j}^{n}}{\Delta x^2}+\frac{T_{ij-1}^{n}-2T_{ij}^{n}+T_{ij+1}^{n}}{\Delta y^2}\Bigg)\end{aligned} \tag{13}$$

Despejando T^{n+1} se forman las siguientes ecuaciones para cada nodo interior ij. Con esto, el método queda con orden $\mathcal{O}(\Delta x^2, \Delta y^2, \Delta t^2)$, segundo orden en espacio y tiempo.

$$\begin{aligned}-\mu T_{i+1j}^{n+1}+(1+4\mu)T_{ij}^{n+1}-\mu T_{i-1j}^{n+1}-\mu T_{ij+1}^{n+1}-\mu T_{ij-1}^{n+1}&=\mu T_{i+1j}^{n}+(1-4\mu)T_{ij}^{n}+\\&+\mu T_{i-1j}^{n}+\mu T_{ij+1}^{n}+\mu T_{ij-1}^{n}\end{aligned} \tag{14}$$

En donde $\mu=\sigma/2$. Notar que se forma un sistema de ecuaciones para obtener las temperaturas en el paso siguiente. Para resolverlo se utilizará el método de Gauss-Seidel con factor de relajación $\lambda = 0.5$. Se prefiere este método dado que para grandes cantidades de nodos un método directo puede ser muy costoso y por ende un método iterativo podría resolverlo de manera más rápida. Además, como vector de partida se utiliza el paso anterior, el cual debería estar cerca del paso siguiente. Adicionalmente para este sistema de ecuaciones se necesita convertir la matriz de temperatura en un vector para resolver el sistema lineal. Así, la matriz queda de $(N_x-2)(N_y)\times(N_x-2)(N_y)$.

Dada la utilización de nodos "fantasmas" para discretizar la condición de borde, las ecuaciones en los bordes inferior y superiores se modifican de la siguiente manera completando el sistema:

$$
\begin{aligned}
-\mu T_{i+1,1}^{n+1} + (1+4\mu)T_{i1}^{n+1} - \mu T_{i-1,1}^{n+1} - 2\mu T_{i2}^{n+1} &= \mu T_{i+11}^{n} + (1-4\mu)T_{i1}^{n} + \\
&+ \mu T_{i-11}^{n} + 2\mu T_{i2}^{n} - 2\mu\theta q_w \\
-\mu T_{i+1N_y}^{n+1} + (1+4\mu)T_{iN_y}^{n+1} - \mu T_{i-1N_y}^{n+1} - 2\mu T_{iN_y-1}^{n+1} &= \mu T_{i+1N_y}^{n} + (1-4\mu)T_{iN_y}^{n} + \\
&+ \mu T_{i-1N_y}^{n} + 2\mu T_{iN_y-1}^{n} - 2\mu\theta q_w
\end{aligned}
\tag{15}
$$

En las ecuaciones presentadas cuando algún nodo hace referencia a una esquina o nodo lateral, basta reemplazar la condición de Dirichlet presentada y por tanto se modifica la matriz dado que no es incógnita. También, si se en el nodo está el aislamiento reemplazar $q_w = 0$.

2.2. Modelación

Ambos esquemas presentados serán utilizados bajo distintas mallas ($\Delta x = \Delta y =$2, 5, 10 [cm]; $\Delta t =$1, 10, 100 [s]) para resolver el problema presentado. Para ambos esquemas se calculará el residual de la ecuación en cada iteración para evaluar la convergencia. Además, si la diferencia: $\max_{ij}(T_{ij}^{n+1} - T_{ij}^{n}) <$ 0.0001 se considerará que el estado estacionario fue alcanzado. En los resultados además se presentará el calor neto normal en cada nodo de la placa calculado mediante diferencias finitas centradas y adelantadas o atrasadas en los bordes (orden 2). Se utilizará el software de ANSYS transiente y estacionario por medio de elementos finitos para obtener la evolución y su estado final. Se comparará lo obtenido con el código de elementos finitos publicado en [4] para la parte simétrica.

3. Resultados

3.1. Residuales

Se evalúan residuales para distintos esquemas y mallas los cuales se presentan en el gráfico respecto al tiempo:

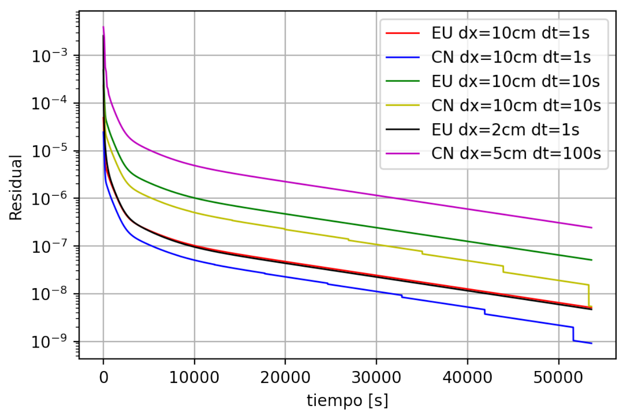
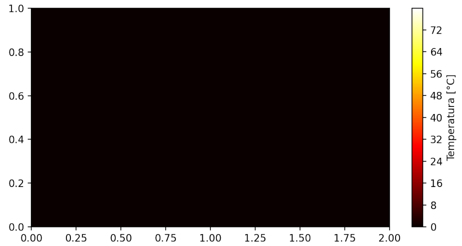


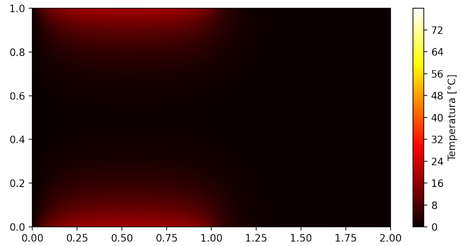
Figura 4: Evolución del residual para distintos esquemas

3.2. Evolución de la Solución

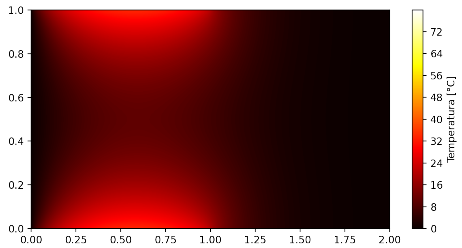
Se presenta la distribución para tiempos de 0, 1000, 4000, 10000, 20000 y 60000. Los resultados a presentar fueron realizas con una malla de $\Delta x = \Delta y = 2$ [cm] y $\Delta t = 1$ [s] mediante el esquema de Euler Explícito. De todas formas los resultados a simple vista son iguales al variar el esquema bajo esta misma malla. Las temperaturas en la placa se interpolaron para obtener una vista "suave" de la distribución.



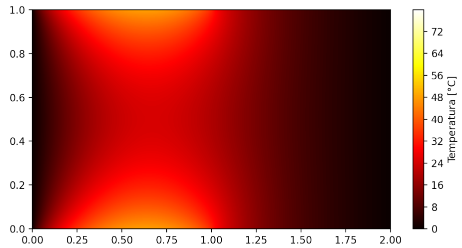
(a) $t = 0$ s



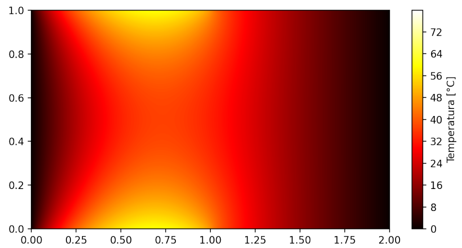
(b) $t = 1000$ s



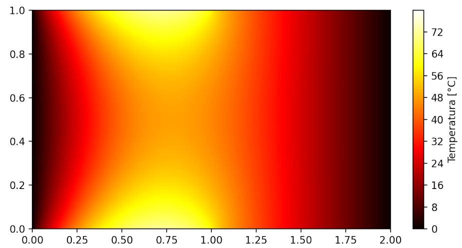
(c) $t = 4000$ s



(d) $t = 10000$ s



(e) $t = 20000$ s

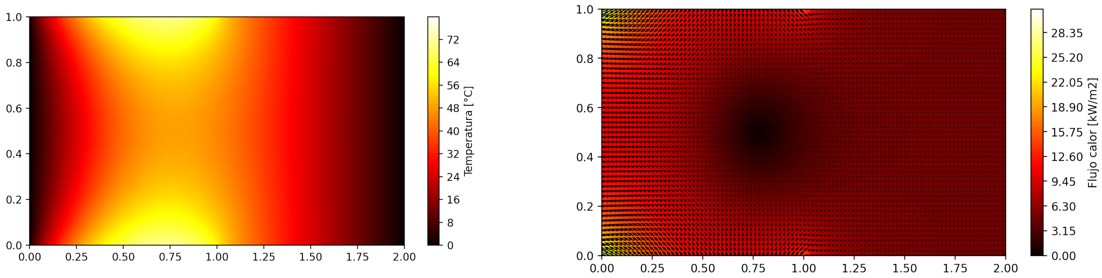


(f) $t = 60000$ s

Figura 5: Evolución de la temperatura en la placa

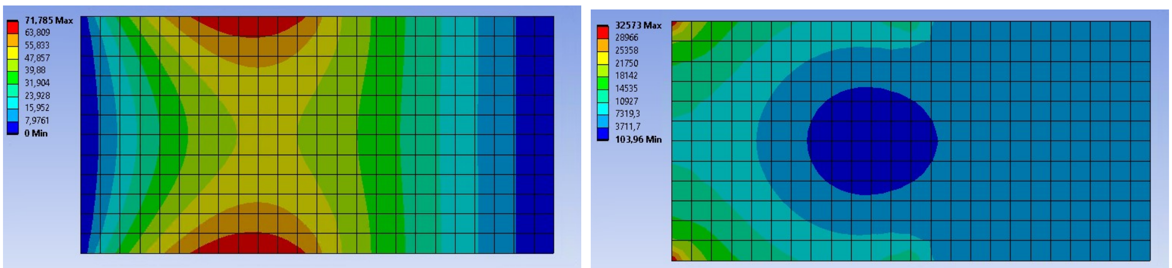
3.3. Estado Estacionario

Se muestra la distribución de temperatura y flujo de calor para el estado estacionario para los códigos elaborados y el software de ANSYS.



(a) Temperatura (b) Flujo de calor

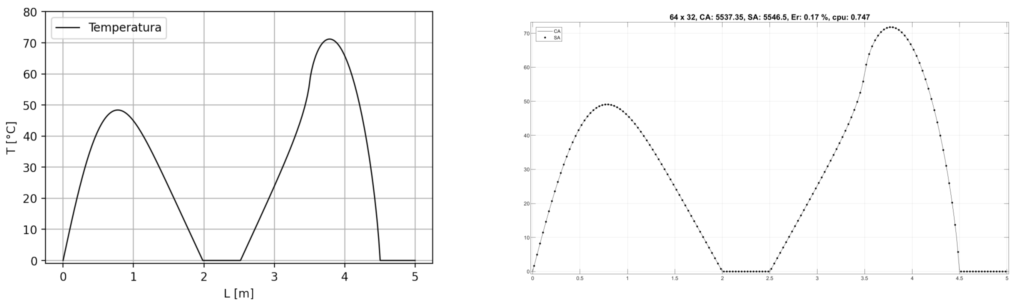
Figura 6: Temperatura y flujo de calor en la placa para estado estacionario con código propio



(a) Temperatura (b) Flujo de calor

Figura 7: Temperatura y flujo de calor en la placa para estado estacionario con ANSYS

Se adjunta la temperatura del borde de la parte simétrica de la placa obtenido por el código propio y el adjuntado en [4] por medio de elementos finitos.



(a) Código propio (b) Código Beckers y Beckers

Figura 8: Temperatura en el borde de parte simétrica

3.3.1. Evaluación Esquema

Tabla 1: Resultados por cada esquema

Método	$\Delta x = \Delta y$ [cm]	Δt [s]	Tiempo estacionario [s]	Temperatura máxima [°C]
Euler Explícito	10	1	54604	73.877
Euler Explícito	10	10	89480	75.091
Euler Explícito	2	1	53616	71.204
Crank-Nicolson	10	1	54605	73.877
Crank-Nicolson	10	10	89500	75.091
Crank-Nicolson	5	100	123700	73.681
ANSYS	10	-	-	71.785
MEF	3.125	-	-	71.791

4. Análisis de Resultados

Con los resultados obtenidos se nota que para ambos esquemas se obtuvo una solución aproximada que concuerda con la física. A medida que avanza el tiempo, las paredes en donde se encuentra el flujo constante de calor de 10 [kW] comienzan a calentar la placa de manera simétrica, fig. 5. Esta aumento de temperatura se difunde a través de la placa generando las distribuciones presentadas. Los esquemas de Euler Explícito y de Crank-Nicolson no generan mucha diferencia. Los gifs generados por medio del código propio y el software de ANSYS se comportan de manera similar. Con este último no se alcanzó el estado estacionario dado su alto costo computacional. Ya en el caso estacionario, si se comparan las figuras 6 y 7 (código y ANSYS) se visualizan distribuciones de temperaturas y flujos de calor similares. Notar que la transferencia de calor es máxima en las esquinas izquierdas dada las condiciones de borde existentes. De la misma forma, comparando con los resultados obtenidos por Beckers y Beckers, fig. 8, la temperatura en el borde de la parte simétrica se comporta de la misma forma.

Para una mejor evaluación de los esquemas, se presenta la tabla 1 en donde se compara la temperatura máxima obtenida y el tiempo necesario para llegar al criterio de estado estacionario presentado en la metodología. Se nota que a través de ANSYS y el código de elementos finitos se obtiene una temperatura máxima muy similar (variación de 0.008 %) de aproximadamente 71.79 [°C]. En esta tabla no se presenta para estos métodos un tiempo para el estado estacionario dado que ambos se resolvieron para obtener la solución estacionaria directamente. Respecto a los esquemas, se obtienen diferencias para la temperatura máxima de alrededor de 1 a 4 [°C], lo que lleva a variaciones de 1 % a 3 %, el cual es pequeño. Sin embargo, una diferencia notoria es el tiempo para alcanzar el estado estacionario. Pareciera darse que al utilizar mallas más gruesas en el espacio y tiempo, el tiempo para alcanzar el estado estacionario aumenta. Si bien las modelaciones se veían bien, esto se puede deber a un simple nodo en el cual la convergencia está siendo más lenta. Para las mallas más finas se obtiene un tiempo aproximado de 54000 segundos, lo que equivale a 15 horas. Se considera este como el "real" dado que se espera que una malla fina tenga un mejor orden de convergencia.

Continuando con lo anterior, en la figura 4 se nota como los residuales de la ecuación de calor varían con el paso del tiempo. Además, se nota que para las mallas más finas el residual siempre es menor al resto. El método de Crank-Nicolson con $\Delta x = 10$ [cm] y $\Delta t = 1$ [s] pareciera converger mejor que los métodos de Euler Explícito con $\Delta x = 2$, $\Delta x = 10$ [cm] y $\Delta t = 1$ [s] dado su mayor orden. Lo obvio se nota que al hacer más gruesas las mallas (espacio y tiempo), estos residuales tienden a aumentar. Algo intereante de mencionar es que el método de Crank-Nicolson es bastante sensible a la tolerancia impuesta en el método de Gauss-Seidel y al parecer por esto se producen esos escalones en los residuales, producto de algún tipo de error de redondeo y truncación.

Por último, se menciona que el método de Euler Explícito tiene un costo computacional mucho menor al método de Crank-Nicolson. Este último demora bastante en modelar para mallas espaciales muy finas y por esto se presenta como mínimo una malla con espaciamiento de 5 [cm]. De todas formas, su ventaja recae en que se pueden utilizar pasos temporales grandes, ejemplo de 100 [s] con el cual los resultados y residuales parecen no estar muy alejados. Una mejora que se puede implementar para hacer más eficiente el código es evitar crear la matriz del sist. lineal y utilizar un algoritmo parecido al método de Thomas para sistemas tridiagonales por bloques para la resolución en cada paso temporal.

5. Conclusión

Para finalizar, se nota que con los esquemas presentados se pudo modelar la distribución de temperatura en la placa para el estado transiente. La utilización de cada uno depende del objetivo y la precisión necesaria. El método de Euler termina siendo más rápido pero condicionalmente estable, por lo que diverge si no se selecciona una malla adecuada. El método de Crank-Nicolson permite utilización de mallas gruesas dado su carácter de estabilidad incondicional, permitiendo utilizar pasos temporales grandes a costa de su alto costo computacional. Se obtuvo que mientras más fina la malla los residuales terminan siendo menores y por ende, un mayor grado de convergencia. De todas formas, todas las mallas utilizadas logran representar la evolución temporal. Para una mayor eficiencia se podría haber resuelto para la mitad de la placa dada la simetría existente.

Ambos esquemas fueron comparados al alcanzar la solución estacionaria con el software de ANSYS y el código de elementos finitos presentado en [4]. Las temperaturas máximas presentadas no tienen variaciones mayores a 4 % y las distribuciones se ven muy similares. Esto de alguna forma valida el trabajo realizado al compararlo con la literatura existente. Por otra parte, de lo obtenido se estima que en 54000 segundos aproximadamente la placa alcanza el estado estacionario (bajo el criterio planteado). El variar la malla a una más gruesa puede generar variaciones en este tiempo de más del 100 %.

Por último, en este trabajo se refleja la importancia de los métodos numéricos dado que sirven para modelar cualquier situación real. Su conocimiento permite una buena selección, implementación y validación del resultado obtenido cuando se programa el método o se utiliza un software comercial.

6. Referencias

[1] Steven C. Chapra Raymond P. Canale. (2007). *Métodos numéricos para ingenieros*. The McGraw-Hill.

[2] Quarteroni A. Sacco, R. Saleri, F. (2000). *Numerical Mathematics*. Springer

[3] Cengel, Y. (2007). *Transferencia de Calor y Masa*. 3^a Edición.

[4] Beckers, P. Beckers, B. (2015) *A 66 line heat transfer finite element code to highlight the dual approach.* Computers and Mathematics with Applications.

[5] ANSYS INC. (2009) *Theory Reference*. Release 15.0.

7. Anexos

7.1. Código Implementado

Se presenta el código programado para la resolución:

Código 1: Programa con esquemas de Euler Explícito y Crank-Nicolson para la resolución de la ecuación de calor.

```

! Programa para resolver la ecuacion de calor
program main
    implicit none
    integer :: Nx,Ny,tout,nt
    real(8) :: time_f,d_time,dx,dy,Lx,Ly,q
    real(8), allocatable, dimension(:,:) :: Temperature_f,Temperature_i,qn,qx,qy
    real(8), allocatable, dimension(:) :: Res
    real(8) :: alpha,k,sigma,theta,cp,rho,sigma2
    character(25) :: file_T_eu,file_T_cn,file_qxeu,file_qyeu,file_q_cn,file_q_eu,file_qxcn,file_qycn,file_reseu,file_rescn
    logical :: euler,crank,flag_results

    ! Codigo exporta los siguientes archivos con resultados

    file_T_eu = 'Results_T_eu.txt'
    file_T_cn = 'Results_T_cn.txt'
    file_q_eu = 'Results_q_eu.txt'
    file_q_cn = 'Results_q_cn.txt'
    file_qxeu = 'Results_qx_eu.txt'
    file_qyeu = 'Results_qy_eu.txt'
    file_qxcn = 'Results_qx_cn.txt'
    file_qycn = 'Results_qy_cn.txt'
    file_reseu = 'Residual_eu.txt'
    file_rescn = 'Residual_cn.txt'

    k = 100.0d0
    cp = 421.0d0
    rho = 8862.0d0
    alpha = k/(cp*rho)
    q = 1.0d4

    time_f = 60000
    d_time = 1
    tout = 1000
    nt = floor(time_f/d_time)+1
    dx = 0.1d0
    dy = dx
    Lx = 2d0
    Ly = 1d0

    sigma = alpha*d_time/(dx**2)
    sigma2 = 2*sigma
    theta = -2*dx/k

```

```
    Nx = floor(Lx/dx) + 1
    Ny = floor(Ly/dy) + 1
    allocate (Temperature_i(Ny,Nx))
    allocate (Temperature_f(Ny,Nx))
    allocate (qn(Ny,Nx))
    allocate (qx(Ny,Nx))
    allocate (qy(Ny,Nx))
    allocate (Res(nt))

    Temperature_i = 0

    write(*,*)
    write(*,*) "Codigo resolucion ecuacion de calor placa"
    write(*,*)
    write(*,*) "Propiedades (k,alpha,rho,cp)"
    write(*,"(F7.1,ES12.4,F8.1,F7.1)") k,alpha,rho,cp
    write(*,*) "Numero de Fourier: ",sigma2
    write(*,*) "Nodos (Nx,Ny):",Nx,Ny
    write(*,*) "Espaciamientos (dt,dx=dy)"
    write(*,"(2F10.5)") d_time,dx

    write(*,*)
    write(*,*)

    call system('mkdir eu_rtxt')
    call system('mkdir cn_rtxt')
    call system('mkdir results')

    write(*,*)

    ! Condiciones para correr esquemas
    euler = .true.
    crank = .true.

    ! Condicion para imprimir los resultados cada tout
    flag_results = .false.

    if (euler) then
        write(*,*)
        write(*,*)
        write(*,*) "------- EULER EXPLICITO --------"
        write(*,*)

        call Euler_Explicito(Nx,Ny,Temperature_i,Temperature_f,d_time,time_f,
            sigma,q,theta,tout,flag_results,Res,nt)
        call WriteFile(Nx,Ny,Temperature_f,file_T_eu,.true.,'results/')
        call Calculate_Heat(Nx,Ny,Temperature_f,qn,theta,q,qx,qy)
        call WriteFile(Nx,Ny,qn,file_q_eu,.true.,'results/')
        call WriteFile(Nx,Ny,qx,file_qxeu,.true.,'results/')
        call WriteFile(Nx,Ny,qy,file_qyeu,.true.,'results/')
        call WriteFile2(nt,Res,file_reseu,.true.,'results/')

    end if

    if (crank) then
```

```
        write(*,*)
        write(*,*)
        write(*,*) "------- CRANK NICOLSON --------"
        write(*,*)

        call Crank_Nicolson(Nx,Ny,Temperature_i,Temperature_f,d_time,time_f,
           sigma,q,theta,tout,flag_results,Res,nt)
        call WriteFile(Nx,Ny,Temperature_f,file_T_cn,.true.,'results/')
        call Calculate_Heat(Nx,Ny,Temperature_f,qn,theta,q,qx,qy)
        call WriteFile(Nx,Ny,qn,file_q_cn,.true.,'results/')
        call WriteFile(Nx,Ny,qn,file_qxcn,.true.,'results/')
        call WriteFile(Nx,Ny,qn,file_qycn,.true.,'results/')
        call WriteFile2(nt,Res,file_rescn,.true.,'results/')
    end if

    write(*,*)
    write(*,*)
    write(*,*)

end program main


!============ Euler Explicito ===================!

subroutine Euler_Explicito(Nx,Ny,Ti,Tf,dt,timef,sigma,q,theta,tout,flag,Res,
   ntt)
    implicit none
    integer, intent(in) :: Nx,Ny,tout,ntt
    real(8), intent(in) :: dt,timef,sigma,Ti(Ny,Nx),q,theta
    real(8), intent(out) :: Tf(Ny,Nx),Res(ntt)
    logical, intent(in) :: flag
    integer :: i,j,nt,kk,ll,stat
    real(8) :: T_old(Ny,Nx),T(Ny,Nx),time,T_est(Ny,Nx),q2,Calc_Residual
    character(7) :: x1

    nt = 0

    T_old(:,:) = Ti(:,:)
    time = 0
    do while (time<timef)
        do i=2,Nx-1
            do j=2,Ny-1
                T(j,i)=T_old(j,i)+sigma*(T_old(j,i-1)-2*T_old(j,i)+T_old(j,i
                   +1))+sigma*(T_old(j-1,i)-2*T_old(j,i)+T_old(j+1,i))
            end do
        end do
        do i=2,Nx-1
            if (i.le.Ny) then
                q2 = q
            else
                q2 = 0d0
            end if
            j=1
            T(j,i)=T_old(j,i)+sigma*(T_old(j,i-1)-2*T_old(j,i)+T_old(j,i+1))+
               sigma*(-2*T_old(j,i)+2*T_old(j+1,i)-q2*theta)
```

```
            j=Ny
            T(j,i)=T_old(j,i)+sigma*(T_old(j,i-1)-2*T_old(j,i)+T_old(j,i+1))+
                sigma*(-2*T_old(j,i)+2*T_old(j-1,i)-q2*theta)
        end do

        T(:,1) = 0
        T(:,Nx) = 0

        time = time + dt

        ! Criterio estacionario
        T_est = T - T_old
        if (maxval(abs(T_est)).lt.0.0001) then
            write(*,*) 'Estado estacionario, tiempo:', time
            write(*,*) 'Max Temp:',maxval(T)
            exit
        end if

        ! Criterio si imprimir cada tout
        if (flag) then
            if (mod(nt,tout)==0 .or. nt==0) then
                write (x1,'(I7.7)') floor(time)
                open(unit=20,file='eu_rtxt/t_eu'//trim(x1)//'.txt',iostat=stat
                    ,action='write')
                if(stat/=0) then
                    write(*,*)'Error al abrir el archivo con iostat',stat
                end  if
                do kk=1,ny
                    write(20,*)(T(kk,ll), ll=1,nx)
                end do
                close(unit=20,iostat=stat)
            end if
        end if

        ! Residual
        Res(nt+1) = Calc_Residual(Nx,Ny,T_old,T,sigma,dt)

        nt = nt + 1
        T_old(:,:) = T(:,:)
    end do
    Tf(:,:) = T_old(:,:)


end subroutine Euler_Explicito


!===== Crank Nicolson ==============!

subroutine Crank_Nicolson(Nx,Ny,Ti,Tf,dt,timef,sigma,q,theta,tout,flag,Res,ntt
    )
    implicit none
    integer, intent(in) :: Nx,Ny,tout,ntt
    real(8), intent(in) :: dt,timef,sigma,Ti(Ny,Nx),q,theta
    real(8), intent(out) :: Tf(Ny,Nx),Res(ntt)
    logical, intent(in) :: flag
```

```
    integer :: Nv,nt,kk,ll,stat
    character(7) :: x1
    real(8) :: T_old(Ny,Nx),T(Ny,Nx),time,Calc_Residual,T_est(Ny,Nx)

    real(8), allocatable, dimension(:) :: b_vec,T_vec
    real(8), allocatable, dimension(:,:) :: A,A2

    Nv = (Nx-2)*(Ny)
    allocate(b_vec(Nv))
    allocate(T_vec(Nv))
    allocate(A(Nv,Nv))
    allocate(A2(Nv,Nv))

    ! Crear matriz
    call create_mat(A,sigma,Nv,Nx,Ny)
    write(*,*) 'Matrix created CN'

    T_old(:,:) = Ti(:,:)
    time = 0
    nt = 0
    do while (time<timef)
        ! Create b_system
        call b_system(T_old,b_vec,Nx,Ny,Nv,sigma,q,theta)
        ! resolver sistema
        call Gseid(A,b_vec,Nv,T_vec,300,1.0d-8,0.5d0)
        ! vec to mat T
        call vec_to_mat(T,T_vec,Nx,Ny,Nv)

        time = time + dt

        ! Criterio de estacionario
        T_est = T - T_old
        if (maxval(abs(T_est)).lt.0.0001) then
            write(*,*) 'Estado estacionario, tiempo:', time
            write(*,*) 'Max Temp:',maxval(T)
            exit
        end if

        ! Criterio de imprimir cada tout
        if (flag) then
            if (mod(nt,tout)==0 .or. nt==0) then
                write (x1,'(I7.7)') floor(time)
                open(unit=20,file='cn_rtxt/t_eu'//trim(x1)//'.txt',iostat=stat
                    ,action='write')
                if(stat/=0) then
                    write(*,*)'Error al abrir el archivo con iostat',stat
                end  if
                do kk=1,ny
                    write(20,*)(T(kk,ll), ll=1,nx)
                end do
                close(unit=20,iostat=stat)
            end if
        end if

        ! Residual
```

```
        Res(nt+1) = Calc_Residual(Nx,Ny,T_old,T,sigma,dt)

        nt = nt + 1
        T_old(:,:) = T(:,:)
    end do
    Tf(:,:) = T_old(:,:)

end subroutine Crank_Nicolson


! Funcion que calcula el residual en un paso
real(8) function Calc_Residual(Nx,Ny,Told,Tnew,sigma,dt)
    implicit none
    integer, intent(in) :: Nx,Ny
    real(8), intent(in) :: Told(Ny,Nx),Tnew(Ny,Nx),sigma,dt
    integer :: i,j
    real(8) :: Res(Ny-2,Nx-2),rxy

    do j=2,Ny-1
        do i=2,Nx-1
            Res(j-1,i-1)=(Tnew(j,i)-Told(j,i))/dt-(sigma/dt)*(Tnew(j,i-1)-4*
                Tnew(j,i)+Tnew(j,i+1)+Tnew(j-1,i)+Tnew(j+1,i))
        end do
    end do

    rxy = 0.0d0
    do j=1,Ny-2
        do i=1,Nx-2
            rxy = rxy + (1.0d0/((Nx-2)*(Ny-2)))*Res(j,i)**2
        end do
    end do

    Calc_Residual = sqrt(rxy)

end function Calc_Residual


! Creacion de matriz para CN
subroutine create_mat(A,sigma,Nv,Nx,Ny)
    implicit none
    integer, intent(in) :: Nv,Nx,Ny
    real(8), intent(in) :: sigma
    real(8), intent(out) :: A(Nv,Nv)
    integer :: i,j,k,nxx,nyy
    real(8) :: mu

    nxx = Nx-2
    nyy = Ny
    A = 0
    mu = sigma/2.0d0

    do j=1,nyy
        do i=1,nxx
            k = i + (j-1)*nxx
            if (j==1) then
```

```
                if (i==1) then
                    A(k,k) = 1 + 4*mu
                    A(k,k+1) = -mu
                elseif (i==nxx) then
                    A(k,k) = 1 + 4*mu
                    A(k,k-1) = -mu
                else
                    A(k,k) = 1 + 4*mu
                    A(k,k-1) = -mu
                    A(k,k+1) = -mu
                end if
                A(k,k+nxx) = -2*mu
            elseif (j==nyy) then
                if (i==1) then
                    A(k,k) = 1 + 4*mu
                    A(k,k+1) = -mu
                elseif (i==nxx) then
                    A(k,k) = 1 + 4*mu
                    A(k,k-1) = -mu
                else
                    A(k,k) = 1 + 4*mu
                    A(k,k-1) = -mu
                    A(k,k+1) = -mu
                end if
                A(k,k-nxx) = -2*mu
            else
                if (i==1) then
                    A(k,k) = 1 + 4*mu
                    A(k,k+1) = -mu
                elseif (i==nxx) then
                    A(k,k) = 1 + 4*mu
                    A(k,k-1) = -mu
                else
                    A(k,k) = 1 + 4*mu
                    A(k,k-1) = -mu
                    A(k,k+1) = -mu
                end if
                A(k,k-nxx) = -mu
                A(k,k+nxx) = -mu
            end if
        end do
    end do

end subroutine create_mat


! Creacion de vector b para Ax=b en CN
subroutine b_system(T_mat,b_vec,Nx,Ny,Nv,sigma,q,theta)
    implicit none
    integer, intent(in) :: Nx,Ny,Nv
    real(8), intent(in) :: T_mat(Ny,Nx),sigma,q,theta
    real(8), intent(out) :: b_vec(Nv)
    integer :: i,j,k
    real(8) :: mu,q2

```

```fortran
    mu = sigma/2.0d0
    k = 0
    do j=1,Ny
        do i=2,Nx-1
            k = k + 1
            if (i.le.Ny) then
                q2 = q
            else
                q2 = 0
            end if
            if (j==1) then
                b_vec(k)=mu*T_mat(j,i+1)+(1.0d0-4.0d0*mu)*T_mat(j,i)+mu*T_mat(
                    j,i-1)+2.0d0*mu*T_mat(j+1,i)-2*mu*q2*theta
            else if (j==Ny) then
                b_vec(k)=mu*T_mat(j,i+1)+(1.0d0-4.0d0*mu)*T_mat(j,i)+mu*T_mat(
                    j,i-1) +2.0d0*mu*T_mat(j-1,i)-2*mu*q2*theta
            else
                b_vec(k) = mu*T_mat(j,i+1) + (1.0d0-4.0d0*mu)*T_mat(j,i) + mu*
                    T_mat(j,i-1) + mu*T_mat(j+1,i)+mu*T_mat(j-1,i)
            end if
        end do
    end do

end subroutine b_system

! Paso del vector solucion a matriz de temperaturas en CN
subroutine vec_to_mat(T_mat,T_vec,Nx,Ny,Nv)
    implicit none
    integer, intent(in) :: Nx,Ny,Nv
    real(8), intent(in) :: T_vec(Nv)
    real(8), intent(out) :: T_mat(Ny,Nx)
    integer :: i,j,k

    T_mat = 0.0d0
    k = 0
    do j=1,Ny
        do i=2,Nx-1
            k = k + 1
            T_mat(j,i) = T_vec(k)
        end do
    end do

end subroutine vec_to_mat


!=========== Calcular Flujo de Calor ===========!
! Uitiliza diferencias centradas en nodos interiores y adelantadas o atrasadas
    en bordes.

subroutine Calculate_Heat(Nx,Ny,T,qn,theta,qw,qxx,qyy)
    implicit none
    integer, intent(in) :: Nx,Ny
    real(8), intent(in) :: theta,T(Ny,Nx),qw
    real(8), intent(out) :: qn(Ny,Nx),qxx(Ny,Nx),qyy(Ny,Nx)
    integer :: i,j
```

```
    real(8) :: qx,qy,qq(Ny,Nx)

    qq = 0
    do i=2,Nx-1
        do j=2,Ny-1
            qx = (1d0/theta)*(T(j,i+1)-T(j,i-1))
            qy = (1d0/theta)*(T(j+1,i)-T(j-1,i))
            qq(j,i) = sqrt(qx**2+qy**2)
            qxx(j,i) = qx
            qyy(j,i) = qy
        end do
    end do
    do i=2,Nx-1
        if (i.lt.Ny) then
            qx = (1d0/theta)*(T(1,i+1)-T(1,i-1))
            qy = qw
        else
            qx = (1d0/theta)*(T(1,i+1)-T(1,i-1))
            qy = 0
        end if
        qq(1,i) = sqrt(qx**2+qy**2)
        qxx(1,i) = qx
        qyy(1,i) = qy
    end do
    do i=2,Nx-1
        if (i.lt.Ny) then
            qx = (1d0/theta)*(T(Ny,i+1)-T(Ny,i-1))
            qy = -qw
        else
            qx = (1d0/theta)*(T(Ny,i+1)-T(Ny,i-1))
            qy = 0
        end if
        qq(Ny,i) = sqrt(qx**2+qy**2)
        qxx(Ny,i) = qx
        qyy(Ny,i) = qy
    end do
    do j=2,Ny-1
        qx = (1d0/theta)*(-3*T(j,1)+4*T(j,2)-T(j,3))
        qy = (1d0/theta)*(T(j+1,1)-T(j-1,1))
        qq(j,1) = sqrt(qx**2+qy**2)
        qxx(j,1) = qx
        qyy(j,1) = qy
    end do
    do j=2,Ny-1
        qx = (1d0/theta)*(3*T(j,Nx)-4*T(j,Nx-1)+T(j,Nx-2))
        qy = (1d0/theta)*(T(j+1,Nx)-T(j-1,Nx))
        qq(j,Nx) = sqrt(qx**2+qy**2)
        qxx(j,1) = qx
        qyy(j,1) = qy
    end do
    qn(:,:) = qq(:,:)
end subroutine Calculate_Heat

! Metodo de Gauss Seidel con relajacion
subroutine Gseid(a_mat,b2,n,x,imax,es,lambda)
```

```fortran
    implicit none
    integer, intent(in) :: n,imax
    real(8), intent(in) :: a_mat(n,n),b2(n),es,lambda
    real(8), intent(out) :: x(n)
    integer :: iter,centinela,i,j
    real(8) :: a(n,n),b(n),dummy,sum,old,ea

    a = a_mat
    b = b2

    do i=1,n
        dummy = a(i,i)
        do j=1,n
            a(i,j) = a(i,j)/dummy
        end do
        b(i) = b(i)/dummy
    end do
    do i=1,n
        sum = b(i)
        do j=1,n
            if(i.ne.j) then
                sum = sum - a(i,j)*x(j)
            end if
        end do
        x(i) = sum
    end do
    iter = 1
    do while(.true.)
        centinela = 1
        do i=1,n
            old = x(i)
            sum = b(i)
            do j=1,n
                if(i.ne.j) then
                    sum = sum - a(i,j)*x(j)
                end if
            end do
            x(i) = lambda*sum + (1-lambda)*old
            if (centinela == 1 .and. x(i) .ne.0 ) then
                ea = abs((x(i)-old)/x(i))*100
                if (ea.gt.es) then
                    centinela = 0
                end if
            end if
        end do
        iter = iter + 1
        if (centinela==1 .or. iter.ge.imax) then
            exit
        end if
    end do

end subroutine Gseid


! Para escribir archivo salida
```

```fortran
subroutine WriteFile(nx,ny,T,file,flag,dir)
    implicit none
    integer, intent(in) :: nx,ny
    character(25), intent(in) :: file
    character(8), intent(in) :: dir
    logical, intent(in) :: flag
    integer :: i,j,stat
    real(kind=8), intent(in) :: T(ny,nx)

    open(unit=20,file=dir//file,iostat=stat,action='write')

        if(stat/=0) then
            write(*,*)'Error al abrir el archivo con iostat',stat
        end  if

        do j=1,ny
            write(20,*)(T(j,i), i=1,nx)
        end do
    close(unit=20,iostat=stat)

    if(stat/=0) then
        write(*,*)'Error al cerrar el archivo con iostat',stat
    end if

    if (flag) then
        write(*,*) "Archivo Guardado: "//file
    end if

end subroutine WriteFile

! Para escribir archivo de salida
subroutine WriteFile2(nt,V,file,flag,dir)
    implicit none
    integer, intent(in) :: nt
    character(25), intent(in) :: file
    character(8), intent(in) :: dir
    logical, intent(in) :: flag
    integer :: j,stat
    real(kind=8), intent(in) :: V(nt)

    open(unit=20,file=dir//file,iostat=stat,action='write')

        if(stat/=0) then
            write(*,*)'Error al abrir el archivo con iostat',stat
        end  if

        do j=1,nt
            write(20,*)(V(j))
        end do
    close(unit=20,iostat=stat)

    if(stat/=0) then
        write(*,*)'Error al cerrar el archivo con iostat',stat
    end if

```

```
    if (flag) then
        write(*,*) "Archivo Guardado: "//file
    end if

end subroutine WriteFile2
```

Código 2: Programa para leer archivos exportados del programa y crear las gráficas.

```

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 200
plt.rcParams['savefig.dpi'] = 200


# Grafico de temperatura ---------

T = pd.read_csv("results/Results_T_eu.txt", header=None, delim_whitespace=True)
Lx = 2
Ly = 1
Ny,Nx = T.shape

x = np.linspace(0,Lx,Nx)
y = np.linspace(0,Ly,Ny)

X,Y = np.meshgrid(x,y)

fig0, ax0 = plt.subplots(1, 1, )
cf0 = ax0.contourf(X,Y,T,np.arange(0, 80, .1), cmap="hot")
cbar0 = plt.colorbar(cf0,)

cbar0.set_label('Temperatura [ C ]')
fig = plt.gcf()
fig.set_size_inches(8, 4)

name = 'Temperature'
path_dir = os.path.join(os.getcwd(),name+'.png')
fig.savefig(path_dir)


# Grafico de flujo de calor ------

q = pd.read_csv("results/Results_q_eu.txt", header=None, delim_whitespace=True)
qx = pd.read_csv("results/Results_qx_eu.txt", header=None, delim_whitespace=True)
qy = pd.read_csv("results/Results_qy_eu.txt", header=None, delim_whitespace=True)

Lx = 2
Ly = 1
Ny,Nx = q.shape
```

```python

x = np.linspace(0,Lx,Nx)
y = np.linspace(0,Ly,Ny)
X,Y = np.meshgrid(x,y)

plt.contourf(X,Y,q/1000,300, cmap="hot")
cbar = plt.colorbar()
plt.quiver(X,Y,qx*100,qy*100,color='k')
fig = plt.gcf()
cbar.set_label('Flujo calor [kW/m2]')
fig = plt.gcf()
fig.set_size_inches(8, 4)
plt.show()
name = 'heat'
plt.savefig(os.path.join(os.getcwd(),name+'.png'))


# Grafico de parte simetrica --------

T_b = np.zeros(Nx+Nx+Ny-2)
k = -1
Tx = T.to_numpy()

T_b[:Nx] = Tx[0,:]
T_b[Ny//2+Nx+1:Ny//2+Nx+Nx] = Tx[Ny//2,-1:0:-1]
kk = 0
T2 = np.zeros(Nx+Nx+Ny-2+Ny//2)
for k in range(len(T_b)-1,0,-1):
    T2[kk] = T_b[k]
    kk = kk+1

ll = np.linspace(0,5,253)
plt.plot(ll,T2[Ny//2-2:],linewidth=1,color='k', label='Temperatura')
plt.grid()
plt.legend(loc='upper left')
plt.xlabel('L [m]')
plt.ylabel('T [ C ]')
plt.ylim([-1,80])
fig = plt.gcf()
fig.set_size_inches(6, 4)
plt.show()
name = 'tsym'
plt.savefig(os.path.join(os.getcwd(),name+'.png'))
```