

Problema 1

Ecuación de Calor

Resolvemos, mediante diferentes algoritmos, la ecuación del calor

$$\frac{\partial T}{\partial t} = \frac{\kappa_T}{C\rho} \nabla^2 [T(x, t)] \quad (1)$$

para el caso particular de una barra de aluminio de longitud $L = 1[m]$ y diámetro ω , alineada a lo largo del eje x ; que está aislada en sus lados, pero no en sus extremos. Inicialmente la barra se encuentra a $100[^\circ C]$, y se colocan sus extremos a $0[^\circ C]$. El calor fluye sólo por los extremos. La conductividad térmica (κ_T), el calor específico (C) y la densidad (ρ) del aluminio son

$$\kappa_T = 237 \left[\frac{W}{mK} \right]; \quad C = 900 \left[\frac{J}{kgK} \right]; \quad \rho = 2700 \left[\frac{kg}{m^3} \right] \quad (2)$$

Para ello:

- a) Adimensionalizar la ecuación para llevarla a la forma

$$\frac{\partial [T(\tilde{x}, \tilde{t})]}{\partial \tilde{t}} = \frac{\partial^2 [T(\tilde{x}, \tilde{t})]}{\partial^2 \tilde{x}} \quad (3)$$

- b) Escriba el código que resuelva la ecuación utilizando el método (de diferencias finitas) explícito de Euler "hacia adelante" (forward Euler). Utilice 100 divisiones en el eje x , y prevea hacer miles de pasos en t (es buena costumbre estimar tamaños de archivos antes de calcular (discuta en informe)! (Almacene sólo los valores de t que necesita, no todos). use $\Delta t = 0.3[s]$ (adimensionalizar y verificar que cumpla la condición de estabilidad). Escriba cada aproximadamente 300 pasos temporales los valores de la temperatura en la barra, y haga un gráfico de superficie mostrando $T(x, t)$ versus (x, t) .
- c) Controle que el programa dé una distribución de temperatura que varía suavemente a lo largo de la barra, y que está de acuerdo con las condiciones de contorno.
- d) Verifique que el programa dé una distribución de temperatura que varía suavemente a lo largo de la barra, y que está de acuerdo con las condiciones de contorno.
- e) Compare la solución analítica:

$$T(x, t) = \sum_{n=1,3,\dots}^{\infty} \frac{4T_0}{n\pi} \sin(k_n x) \exp \left[\frac{-(k_n)^2 \kappa_T t}{C\rho} \right] \quad (4)$$

con la solución numérica. Para ello, grafique $T(x, t)$, para $t_1 = 180[s]$ y $t_2 = 1800[s]$. Utilice ahora 10 divisiones en el eje x (y elija Δt de acuerdo a la condición de estabilidad) y compare los resultados obtenidos para t_1 y t_2 .

- f) Haga un gráfico de superficie mostrando $T(x, t)$ versus (x, t) , y un gráfico de contornos mostrando las isotermas.
- g) Repita los incisos (b)-(f) para los algoritmos: implícito y Crank-Nicolson. Presente figuras donde se comparen, y discutan: si las diferencias son apreciables al ojo, sus tiempos de CPU, sus errores, complejidad de métodos, etc.
- h) Compare los tiempos de CPU para llegar a $t_2 = 1800[s]$ con cada método y con una precisión de 10^{-4} (error relativo).

Resultados y Discusiones

Inciso a)

Análisis dimensional

Adimensionalización, partimos de la ecuación unidimensional

$$\frac{\partial T}{\partial t} = D \frac{\partial^2 [T(x, t)]}{\partial^2 x}; \quad D = \frac{\kappa_T}{C\rho} \quad (5)$$

notemos que las unidades de D (en unidades generales de tiempo (t), temperatura (T), y espacio (L) y masa (M)) las podremos averiguar conociendo las unidades de κ_T , C y ρ , entonces de (2)

$$[\kappa_T] = \left[\frac{\text{potencia}}{\text{distancia} \cdot \text{temperatura}} \right] \equiv \left[\frac{\text{potencia}}{L \cdot T} \right]$$

pero las unidades de potencia se pueden expresar de forma fundamental como

$$\begin{aligned} [\text{potencia}] &= \left[\frac{\text{trabajo}}{\text{tiempo}} \right] = \left[\frac{\text{fuerza} \cdot \text{distancia}}{\text{tiempo}} \right] = \left[\frac{\text{masa} \cdot \text{aceleración} \cdot \text{distancia}}{\text{tiempo}} \right] \\ \Rightarrow [\text{potencia}] &= \left[\frac{\text{masa} \cdot (\text{distancia})^2}{(\text{tiempo})^3} \right] \equiv \left[\frac{M \cdot L^2}{t^3} \right] \therefore [\kappa_T] \equiv \left[\frac{M \cdot L}{t^3 \cdot T} \right] \end{aligned}$$

y por otro lado tendremos,

$$[C] = \left[\frac{\text{trabajo}}{\text{masa} \cdot \text{temperatura}} \right] \equiv \left[\frac{L^2}{T \cdot t^2} \right] \wedge [\rho] = \left[\frac{\text{masa}}{\text{volumen}} \right] \equiv \left[\frac{M}{L^3} \right]$$

luego

$$[D] = \frac{[\kappa_T]}{[C] \cdot [\rho]} \equiv \left[\frac{L^2}{t} \right] \equiv \left[\frac{(\text{distancia})^2}{\text{tiempo}} \right] \quad (6)$$

y si ahora definimos las siguientes coordenadas adimensionales de espacio (\tilde{x}) y tiempo (\tilde{t})

$$\alpha \tilde{x} = x \wedge \beta \tilde{t} = t \quad (7)$$

donde α y β son parámetros característicos a definir para realizar las adimensionalidades. Si por un lado, adimensionalizamos x con la longitud característica de la barra $L \Rightarrow \alpha = L$ entonces deberá cumplirse que de (5), (7) (y usando regla de la cadena)

$$\begin{aligned} \frac{\partial}{\partial t} &= \frac{\partial}{\partial \tilde{t}} \underbrace{\frac{\partial \tilde{t}}{\partial t}}_{=1/\beta} = \frac{1}{\beta} \frac{\partial}{\partial \tilde{t}}; \quad \frac{\partial^2}{\partial x^2} = \left[\underbrace{\frac{\partial^2}{\partial \tilde{x}^2} \left(\frac{\partial \tilde{x}}{\partial x} \right)^2}_{=1/\alpha^2} + \underbrace{\frac{\partial}{\partial \tilde{x}} \frac{\partial^2 \tilde{x}}{\partial x^2}}_{=0} \right] = \frac{1}{\alpha^2} \frac{\partial^2}{\partial \tilde{x}^2} \\ \Rightarrow \frac{\partial T}{\partial \tilde{t}} &= \frac{\beta D}{\alpha^2} \frac{\partial^2 [T(\tilde{x}, \tilde{t})]}{\partial^2 \tilde{x}} \quad (8) \end{aligned}$$

entonces, para obtener la ecuación (3) deberá cumplirse que

$$\frac{\beta D}{\alpha^2} = 1 \Rightarrow \beta = \frac{\alpha^2}{D} = \frac{L^2}{D} \quad (9)$$

es decir, encontramos el tiempo característico del sistema para adimensionalizar el la coordenada temporal y donde se verifican las unidades pues, $[\beta] = [L^2] \cdot [t/L^2] = [t] \equiv [\text{tiempo}]$.

Inciso b)

Estimación de tamaño de archivos

Se llevo a cabo una estimación de tamaño de archivos de la siguiente manera, por ejemplo, si tenemos las siguientes configuraciones en nuestro código

```
1 program estimated_size
2   use module_precision
```

```

3      implicit none
4      real(dp) :: valor1=4._dp*atan(1.0_dp), valor2=4._dp*atan(1.0_dp)
5      integer(sp) :: i, istat
6      open(10, file='estimated_size.dat', status='replace', action='write', iostat=istat)
7      write(*,*) 'Input/Output file. istat10 = ', istat
8      20 format(E9.2,x,E9.2)
9      do i=1,200; write(10,20) valor1, valor2; end do
10 end program estimated_size
11
12 ! gfortran -o estimated_size.o module_precision.f90 estimated_size.f90 && ./estimated_size.o && rm
    *.mod *.o

```

donde (dp) designa doble precisión (flotante de 8[bytes] \equiv 64[bits]) que utiliza 26 dígitos para su representación. En primera instancia, si no imprimimos los datos con formato tamaño del archivo sería tamaño = $(\# \text{filas}) \cdot \left(\frac{\# \text{valores}}{\text{fila}} \right) \cdot \left(\frac{\text{dígitos}}{\text{valor}} \right)$ [bytes], pero debemos tener en cuenta

también que el sistema operativo guarda en memoria el índice de fila y columna del archivo de datos y por cada índice consume un byte (en nuestro caso tenemos 200 filas y 2 columnas y habrá que considerar 202 bytes adicionales) entonces,

$$\text{tamaño} = (200 \cdot 2 \cdot 26 + 202) [\text{bytes}] = 10602 [\text{bytes}]$$

notemos que estuvimos muy cerca del tamaño real que fue de

```
-rw-rw-r-- 1 mendez mendez 10600 abr 24 09:51 estimated_size.dat
```

Ahora bien, esta es una estimación grosera pues el formato no nos imprimirá todos los dígitos de doble precisión sino que sólo utilizará 9 dígitos para guardar todo el número (contando signos, puntos, exponente, decimales, etc.), además, como en el formato tenemos un espacio en blanco por renglón el sistema operativo también gasta un byte para su representación (esto, sin embargo, no se sabe muy bien por qué razón los espacios en blanco utilizan memoria quizás se debe a los datos basura de los sistemas operativos y también podría deberse al editor de texto en si, comprimiendo o no datos para su almacenamiento, por ejemplo, en este [link](#) se discute acerca de esto y la respuesta podría estar en que si se almacenan los datos en cadenas de caracteres, cualquiera de ellos, incluso los espacios en blanco ocupan memoria) entonces debemos estimar el tamaño como

$$\text{tamaño} = (200 \cdot 2 \cdot 9 + 203 + 200) [\text{bytes}] = 4003 [\text{bytes}]$$

donde el segundo término de 203 bytes se debe al gasto de almacenar índices de filas y columnas (200 filas y 3 columnas) y el tercer término de 200 bytes se debe a el gasto adicional de almacenar un espacio en blanco. El tamaño real fue de

```
-rw-rw-r-- 1 mendez mendez 4000 abr 24 10:22 estimated_size.dat
```

Dirigiéndonos ahora a las simulaciones numéricas mostramos a continuación los resultados obtenidos

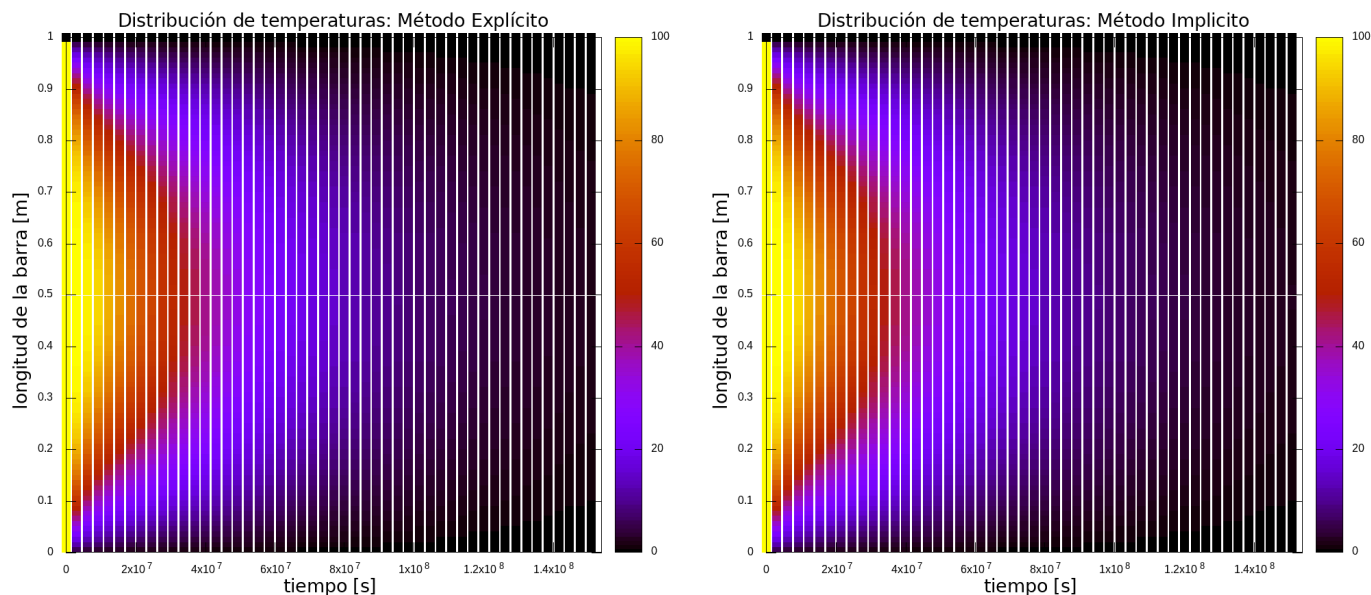


Figure 1:

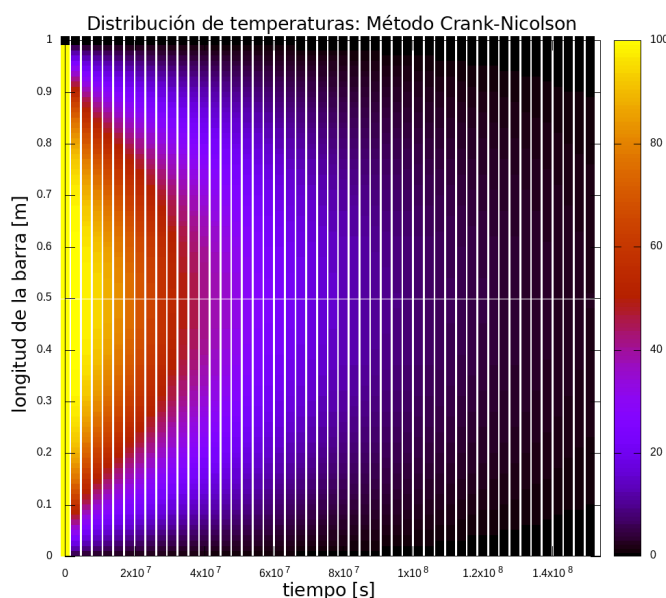


Figure 2:

Las gráficas anteriores muestran la distribución de temperatura (en colores) para cada posición de la barra (cada paso espacial simulado en el eje vertical) y cada 300 pasos de evolución temporal (eje horizontal). A simple vista no se aprecian diferencias significativas, lo que evidencia que los resultados para cada uno de los métodos son similares.

inciso e,f,g)

Control de estabilidad método explícito

Hubo mucha variación de los resultados dependiendo del parámetro de estabilidad elegido, por ejemplo, si bien teóricamente es necesario que este parámetro sea menor a $1/2$ para que el método explícito sea estable, si se elegía un valor cercano a $1/2$ la distribución de temperaturas era nula (como si la relajación fuese instantánea) lo cual sabemos que físicamente no ocurre, y se obtenía un incremento considerable del tiempo de CPU. Y, cuanto menor sea el parámetro la distribución de temperaturas será más gradual, sin embargo, tampoco debe ser muy pequeño pues no se tendrá

relajación alguna. Los resultados se muestran a continuación:

```
real(dp), parameter      :: alpha=0.45_dp (verificando estabilidad por muy poco)
mendez@mendez-notebook:~/my_repositories$ cat ../results/result_02_aprox.dat
0.00E+00  0.00E+00
0.11E+00  0.15-322
0.22E+00  0.30-322
0.33E+00  0.40-322
0.44E+00  0.44-322
0.56E+00  0.44-322
0.67E+00  0.40-322
0.78E+00  0.30-322
0.89E+00  0.15-322
0.10E+01  0.00E+00
```

```
real(dp), parameter      :: alpha=1.E-20_dp (muy por debajo del error de la maquina)
mendez@mendez-notebook:~/my_repositories$ cat ../results/result_02_aprox.dat
0.00E+00  0.00E+00
0.11E+00  0.10E+03
0.22E+00  0.10E+03
0.33E+00  0.10E+03
0.44E+00  0.10E+03
0.56E+00  0.10E+03
0.67E+00  0.10E+03
0.78E+00  0.10E+03
0.89E+00  0.10E+03
0.10E+01  0.00E+00
```

```
real(dp), parameter      :: alpha=1.E-7_dp (por encima del error de la maquina)
mendez@mendez-notebook:~/my_repositories$ cat ../results/result_02_aprox.dat
0.00E+00  0.00E+00
0.11E+00  0.84E+02
0.22E+00  0.99E+02
0.33E+00  0.10E+03
0.44E+00  0.10E+03
0.56E+00  0.10E+03
0.67E+00  0.10E+03
0.78E+00  0.99E+02
0.89E+00  0.84E+02
0.10E+01  0.00E+00
```

También, se probó con valores mayores a $1/2$ produciéndose errores numéricos (propagación de NaN) lo cual evidencia alguna división por cero o algún error aritmético similar.

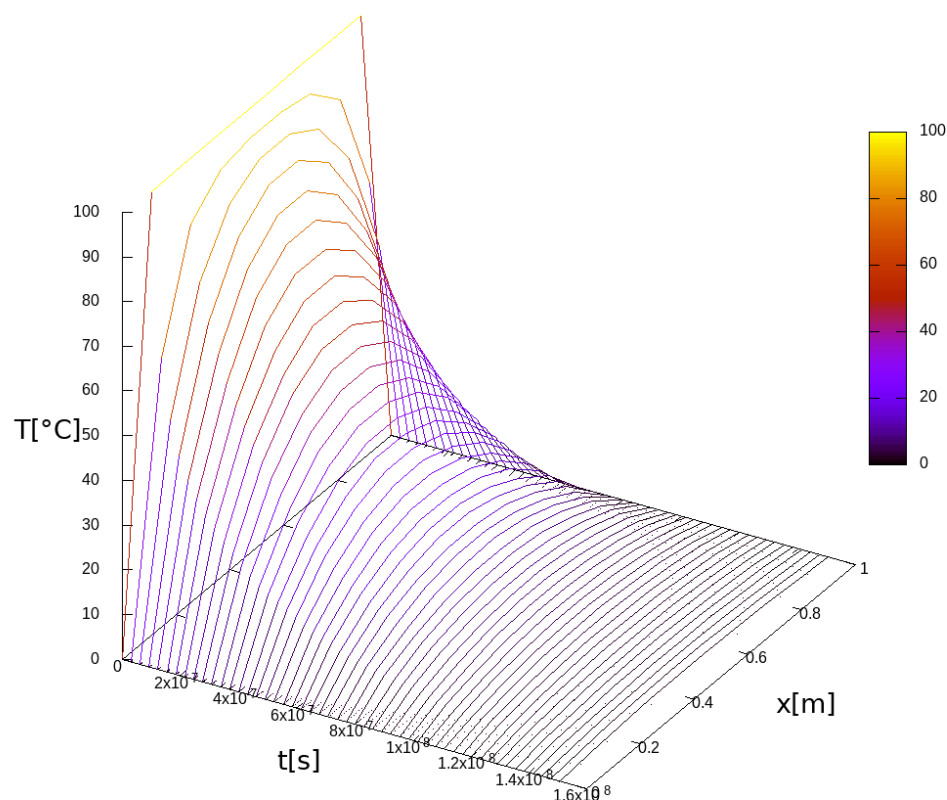
```
real(dp), parameter      :: alpha=4.5_dp (no verificando estabilidad)
mendez@mendez-notebook:~/my_repositories$ cat ../results/result_02_aprox.dat
0.00E+00  0.00E+00
0.11E+00  NaN
0.22E+00  NaN
0.33E+00  NaN
0.44E+00  NaN
0.56E+00  NaN
0.67E+00  NaN
0.78E+00  NaN
0.89E+00  NaN
0.10E+01  0.00E+00
```

Para todos los casos el α utilizado fue de 10^{-7} . Es necesario aclarar que, el análisis anterior no es estrictamente correcto, pues los errores de los métodos dependen de los valores de α pero específicamente de los valores de Δx y Δt de forma acoplada, es decir, en los casos anteriores el

Δx siempre se mantiene fijo y el valor de Δt se ve modificado por el valor de α , esto no es un método correcto de variar α pues el error numérico puede verse modificado groseramente ya que no se acompaña una discretización espacial con una discretización temporal (estamos haciendo elementos finitos más esbeltos en la dimensión temporal en vez de hacerlos más esbeltos en ambas dimensiones y obtener elementos finitos pequeños en su totalidad).

El gráfico de superficie obtenido se muestra en la siguiente figura

Distribución de temperaturas: Método Explícito



La figura anterior sólo se hizo para el método explícito, teniendo en cuenta los resultados anteriores que nos permitían asegurar que los resultados no variaban apreciablemente. En la figura se observa inicialmente la temperatura con los extremos de la barra a $0[^\circ C]$ y el resto del cuerpo de la barra a $100[^\circ C]$, luego de un tiempo se ve la termalización de la barra, es decir, se produce la relajación de la barra a una temperatura de equilibrio de $0[^\circ C]$.

Luego, graficando la distribución de temperaturas para los tiempos específicos de $180[s]$ y $1800[s]$, para cada uno de los métodos, tendremos:

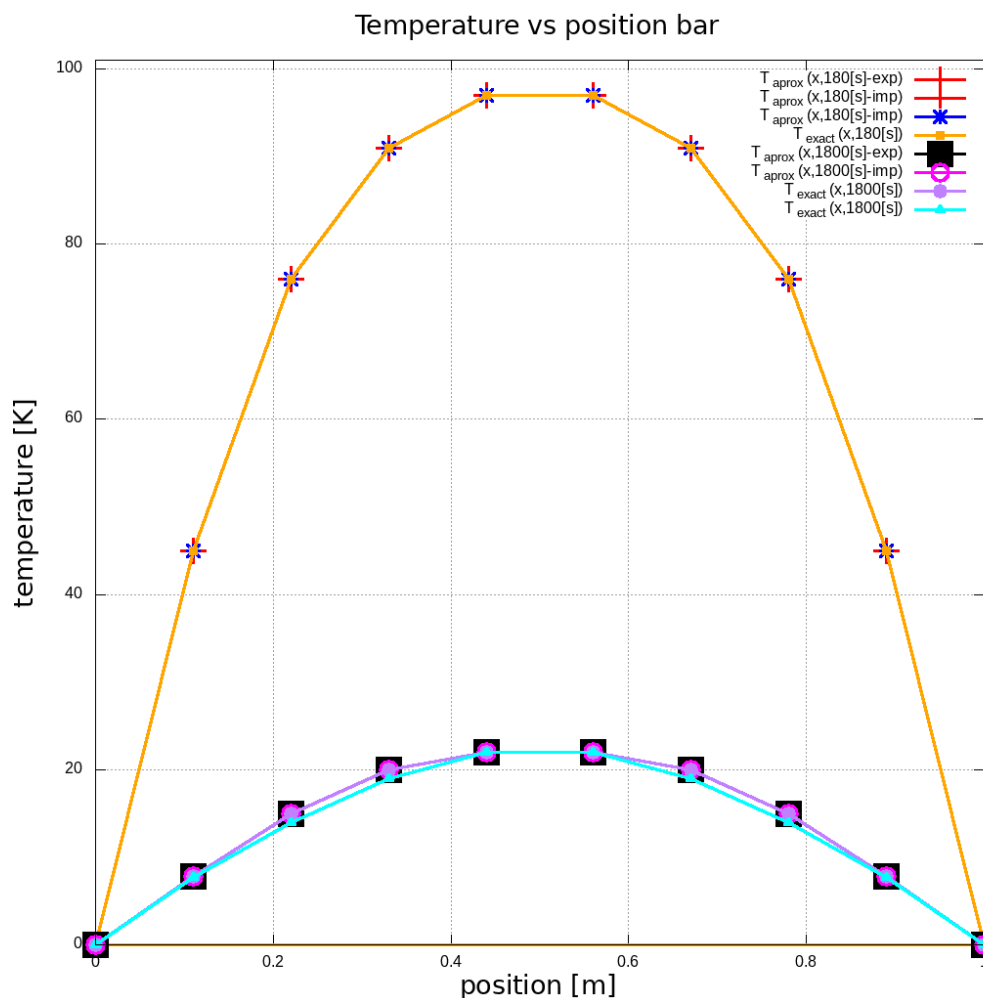


Figure 3:

En esta gráfica podemos ver que los tres métodos arrojan resultados muy próximos a la solución exacta, además, para tiempos mayores vemos que existen diferencias más apreciables respecto al valor exacto.

Inciso h)

Los tiempos fueron medidos para los distintos métodos utilizando la herramienta perf versión 5.13.19, con este comando se corrieron cada uno de los códigos una cinco veces y se pudo medir el valor medio y tolerancia del tiempo de CPU, como así también la cantidad de instrucciones por ciclo, referencias a memoria caché y cache-misses (cantidad de veces que los registros no entran completamente en memoria caché haciendo más dificultoso el procesamiento e incrementando el tiempo de cpu). Para todos los casos se relajó el sistema hasta los 1800[s], con 142200000 pasos temporales, y se midió el tiempo total de ejecución del programa, es decir, primero se compilieron los tres códigos (correspondientes a cada uno de los métodos), se generaron los códigos objeto y luego se aplicó el comando perf a estos códigos binarios para hacer las mediciones de performance.

Los resultados obtenidos fueron,

```
# ++++++
# METODO EXPLÍCITO
# ++++++
perf stat -e cycles,instructions,cache-references,cache-misses -r 5 ./heateq_comparison_01.o

Performance counter stats for './heateq_comparison_01.o' (5 runs):

 30.997.662.307    cycles                    ( +- 3,34% )
 95.295.666.771    instructions             #    3,07  insn per cycle    ( +- 0,00% )
  2.834.670       cache-references        ( +- 2,80% )
  1.012.570       cache-misses             #   35,721 % of all cache refs ( +- 3,01% )

11,455 +- 0,385 seconds time elapsed ( +- 3,36% )
```

```
# ++++++
# METODO IMPLÍCITO
# ++++++
perf stat -e cycles,instructions,cache-references,cache-misses -r 5 ./heateq_comparison_02.o

Performance counter stats for './heateq_comparison_02.o' (5 runs):

    91.829.376.951      cycles                      ( +-  1,08% )
   253.610.980.072      instructions                #    2,76  insn per cycle      ( +-  0,00% )
    15.309.764         cache-references              ( +-  9,38% )
     6.619.732         cache-misses                #  43,239 % of all cache refs   ( +- 11,09% )

    35,429 +- 0,872 seconds time elapsed ( +-  2,46% )
# ++++++
# METODO CRANK-NICOLSON
# ++++++
perf stat -e cycles,instructions,cache-references,cache-misses -r 5 ./heateq_comparison_03.o

Performance counter stats for './heateq_comparison_03.o' (5 runs):

   204.268.612.300      cycles                      ( +-  0,92% )
   540.648.910.045      instructions                #    2,65  insn per cycle      ( +-  0,00% )
    59.518.787         cache-references              ( +- 12,88% )
    17.188.257         cache-misses                #  28,879 % of all cache refs   ( +- 12,84% )

    78,652 +- 0,789 seconds time elapsed ( +-  1,00% )
```

Claramente, el método que más tiempo de CPU obtuvo fue el de Crank-Nicolson (C-N), seguido por el método implícito y el más rápido fue el método explícito. Sin embargo, el método de C-N obtuvo un 33% menos de cache-misses que el método implícito y un 19% menos que el método explícito. La complejidad de los códigos es, en términos generales, la misma aunque el aumento significativo de tiempo de cómputo quizás se deba a que los métodos de C-N e implícito trabajan con matrices mientras que el método explícito no, es más, el aumento de tiempo de CPU del método de C-N respecto del método implícito quizás podría deberse al agregado de operaciones de tipo multiplicación de matrices y vectores, ya que, si bien en ambos métodos se trabaja con matrices bandeadas y con una subrutina que trata con estos sistemas de ecuaciones específicos, el método de C-N realiza más operaciones matemáticas con una segunda matriz bandeada.

Códigos

Repositorio GitHub

<https://github.com/mendzmartin/fiscomp2022.git>

Repositorio GitHub del problema

<https://github.com/mendzmartin/fiscomp2022/tree/main/lab02/prob01>

Programas principales

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob01/code/heateq_explicit_method.f90

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob01/code/heateq_implicit_method.f90

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob01/code/heateq_crank_nicolson_method.f90

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob01/code/heateq_comparison_01.f90

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob01/code/heateq_comparison_02.f90

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob01/code/heateq_comparison_03.f90

Módulos necesarios

https://github.com/mendzmartin/fiscomp2022/blob/main/modules/module_precision.f90

https://github.com/mendzmartin/fiscomp2022/blob/main/modules/module_tridiag_matrix.f90

Bash script para correr los códigos

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob01/code/script_run.sh

Referencias

Chapman, S. (2007). *Fortran 95/2003 for Scientists & Engineers*. McGraw-Hill Education. https://books.google.com/books/about/Fortran_95_2003_for_Scientists_Engineers.html?hl=&id=c8cLDQEACAAJ

Chapra, S. C., & Canale, R. P. (2007). *Métodos numéricos para ingenieros*. https://books.google.com/books/about/M%C3%A9todos_num%C3%A9ricos_para_ingenieros.html?hl=&id=hoH0MAAACAAJ

Landau, R. H., Mejía, M. J. P., Páñez, M. J., Kowallik, H., & Jansen, H. (1997). *Computational Physics*. Wiley-VCH. https://books.google.com/books/about/Computational_Physics.html?hl=&id=MJ3vAAAAMAAJ

Códigos explícitos

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob01/code/heateq_explicit_method.f90

```

1  program heateq_explicit_method
2      use module_precision
3      implicit none
4      real(dp), allocatable :: ux_old(:),ux_new(:)
5      integer(sp), parameter :: n=98_sp                ! numero de elementos finitos
6      real(dp), parameter :: u0_Li=0._dp,u0_Lf=0._dp    ! condiciones de borde
7      real(dp) :: u0_ti=100._dp                        ! condicion inicial
8      real(dp) :: param_t,param_x
9      real(dp) :: D
10     real(dp) :: alpha
11     real(dp) :: t_step_adim,x_step_adim
12     integer(sp) :: i,j,k,istat
13     integer(sp), parameter :: t_write=300_sp          ! pasos temporales entre escrituras
14     real(dp) :: factor
15     20 format(E8.2,x,E8.2)
16     open(10,file='../results/result_01_explicit.dat',status='replace',action='write',iostat=istat)
17     write(*,*) 'istat(10file) = ',istat
18     ! parametros para adimensionalizar
19     D=237._dp*(1._dp/(900._dp*2700._dp)) ! D = K/C*rho [L^2]/[t]
20     param_x=1._dp ! longitud característica (long total barra) [L]
21     write(*,'(A20,E10.4)') 'param_x = ',param_x
22     param_t=param_x*param_x*(1._dp/D) ! tiempo característico (paso temporal) [t]
23     write(*,'(A20,E10.4)') 'param_t = ',param_t
24     t_step_adim=0.3_dp*(1._dp/param_t)
25     write(*,*) t_step_adim
26     x_step_adim=1._dp/(real(n,dp)+1._dp)
27     write(*,*) x_step_adim
28     alpha=t_step_adim*(1._dp/(x_step_adim*x_step_adim))
29     ! verificamos condición de estabilidad debe ser menor a 1/2
30     write(*,'(A20,E10.4)') 'alpha = ',alpha
31     allocate(ux_old(n+2))
32     ! inicializamos el vector espacial a tiempo inicial
33     ux_old(1)=u0_Li
34     write(10,20) 0._dp,ux_old(1)
35     do i=2,n+1
36         ux_old(i)=u0_ti
37         write(10,20) real(i-1,dp)*x_step_adim,ux_old(i)
38     end do
39     ux_old(n+2)=u0_Lf
40     write(10,20) real(n+1,dp)*x_step_adim,ux_old(n+2)
41     allocate(ux_new(n+2))
42     ux_new(1)=ux_old(1)
43     ux_new(n+2)=ux_old(n+2)
44     ! hacemos la evolución temporal (sin considerar los extremos)
45     ! asignamos valores al vector espacial a tiempo t
46     do k=1,50 ! evolucionamos (50*t_write) pasos temporales
47         do i=1_sp,(t_write-1_sp)
48             do j=2_sp,(n+1_sp)
49                 ux_new(j)=(1._dp-2._dp*alpha)*ux_old(j)+alpha*(ux_old(j+1)+ux_old(j-1))
50             end do
51             ux_old(2:n+1)=ux_new(2:n+1)
52         end do
53         ! escribimos valores luego de t_write pasos temporales
54         write(10,20) 0._dp,ux_new(1)
55         do j=2_sp,(n+1_sp)

```

```

56      ux_new(j)=(1._dp-2._dp*alpha)*ux_old(j)+alpha*(ux_old(j+1)+ux_old(j-1))
57      write(10,20) real(j-1,dp)*x_step_adim,ux_new(j)
58    end do
59    write(10,20) real(n+1,dp)*x_step_adim,ux_new(n+2)
60  end do
61  close(10)
62  ! controlar termalización en el centro de la barra
63  write(*,'(A20,E10.4)') 'T(L/2,t_final) = ', ux_new(50)
64  ! estimamos tamaño de archivo
65  factor=10._dp ! este factor depende de (20 format (2(E10.2))), en este ejemplo factor es 10
66  write(*,'(A30,E10.4,A10)') 'Tamaño aprox. de archivo = ', 2._dp*real(n,dp)*51._dp*factor, '[bytes]'
67  deallocate(ux_old,ux_new)
68 end program heateq_explicit_method

```

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob01/code/heateq_implicit_method.f90

```

1  program heateq_implicit_method
2    use module_precision
3    use module_tridiag_matrix
4    implicit none
5    integer(sp), parameter :: n=98_sp ! numero de elementos finitos
6    real(dp), parameter :: u0_Li=0._dp, u0_Lf=0._dp ! condiciones de borde
7    real(dp), parameter :: u0_ti=100._dp ! condicion inicial
8    integer(sp), parameter :: t_write=300_sp ! pasos temporales entre escrituras
9    real(dp), allocatable :: ux_old(:),ux_new(:),diag(:),diag_sup(:),diag_inf(:)
10   real(dp) :: param_t,param_x
11   real(dp) :: D ! thermal diffusivity
12   real(dp) :: alpha
13   real(dp) :: t_step_adim,x_step_adim
14   integer(sp) :: i,j,k,istat
15   20 format(E8.2,x,E8.2)
16   open(10,file='../results/result_01_implicit.dat',status='replace',action='write',iostat=istat)
17   write(*,*) 'istat(10file) = ',istat
18   ! parametros para adimensionalizar
19   D=237._dp*(1._dp/(900._dp*2700._dp)) ! D = K/C*rho [L^2]/[t]
20   param_x=1._dp ! longitud caracteristica (long total barra) [L]
21   param_t=param_x*param_x*(1._dp/D) ! tiempo caracteristico (paso temporal) [t]
22   t_step_adim=0.3_dp*(1._dp/param_t)
23   x_step_adim=1._dp/(real(n,dp)+1._dp)
24   alpha=t_step_adim*(1._dp/(x_step_adim*x_step_adim))
25   ! cargamos diagonales central, superior e inferior
26   allocate(diag(n),diag_sup(n),diag_inf(n))
27   diag_sup(n)=0._dp
28   diag_inf(1)=0._dp
29   do i=1,n
30     diag(i)=1._dp+2._dp*alpha
31     if (i/=1) diag_inf(i)=-alpha
32     if (i/=n) diag_sup(i)=-alpha
33   end do
34   ! cargamos datos iniciales de temperatura
35   allocate(ux_old(n+2))
36   ux_old(1)=u0_Li
37   write(10,20) 0._dp,ux_old(1)
38   do i=2,n+1
39     ux_old(i)=u0_ti
40     write(10,20) real(i-1,dp)*x_step_adim,ux_old(i)
41   end do
42   ux_old(n+2)=u0_Lf
43   write(10,20) real(n+1,dp)*x_step_adim,ux_old(n+2)
44   ! aplicamos método implícito
45   allocate(ux_new(n+2))
46   ux_new(1)=ux_old(1)
47   ux_new(n+2)=ux_old(n+2)
48   do i=1,50
49     do j=1,(t_write-1)
50       call implicit_method(n,diag,diag_sup,diag_inf,ux_old(2:n+1),ux_new(2:n+1))
51       ux_old(2:n+1)=ux_new(2:n+1)
52     end do
53     call implicit_method(n,diag,diag_sup,diag_inf,ux_old(2:n+1),ux_new(2:n+1))
54     do j=1,n+2
55       write(10,20) real(j-1,dp)*x_step_adim,ux_new(j)

```

```

56     end do
57 end do
58 close(10)
59 deallocate(diag,diag_sup,diag_inf,ux_old,ux_new)
60 end program heateq_implicit_method

```

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob01/code/heatq_crank_nicolson_method.f90

```

1  program heateq_crank_nicolson_method
2  use module_precision
3  use module_tridiag_matrix
4  implicit none
5  real(dp)                :: param_t,param_x
6  real(dp)                :: D                ! thermal diffusivity
7  real(dp)                :: alpha
8  real(dp)                :: t_step_adim,x_step_adim
9  integer(sp)             :: i,j,istat
10 real(dp), allocatable :: ux_old(:),ux_new(:),diag(:),diag_sup(:),diag_inf(:)
11 real(dp), allocatable :: B_matrix(:,:),A_matrix(:,:)
12 integer(sp), parameter :: n=98_sp          ! numero de elementos finitos
13 real(dp), parameter :: u0_Li=0._dp, u0_Lf=0._dp ! condiciones de borde
14 real(dp), parameter :: u0_ti=100._dp        ! condicion inicial
15 integer(sp), parameter :: t_write=300_sp     ! pasos temporales entre escrituras
16 20 format(E8.2,x,E8.2)
17 open(10,file='../results/result_01_cranknicolson.dat',status='replace',action='write',iostat=istat)
18 write(*,*) 'istat(10file) = ',istat
19 ! ANALISIS DIMENSIONAL
20 D=237._dp*(1._dp/(900._dp*2700._dp))      ! D = K/C*rho [L^2]/[t]
21 param_x=1._dp                             ! longitud caracteristica (long total barra) [L]
22 param_t=param_x*param_x*(1._dp/D)          ! tiempo caracteristico (paso temporal) [t]
23 t_step_adim=0.3_dp*(1._dp/param_t)
24 x_step_adim=1._dp/(real(n,dp)+1._dp)
25 write(*,*) 'x_step_adim = ',x_step_adim
26 alpha=t_step_adim*(1._dp/(x_step_adim*x_step_adim))
27 ! CARGAMOS DIAGONALES CENTRAL, SUPERIOR E INFERIOR
28 allocate(diag(n),diag_sup(n),diag_inf(n))
29 diag_sup(n)=0._dp
30 diag_inf(1)=0._dp
31 do i=1,n
32     diag(i)=2._dp*(1._dp/alpha+1._dp)
33     if (i/=1) diag_inf(i)=-1._dp
34     if (i/=n) diag_sup(i)=-1._dp
35 end do
36 ! CARGAMOS MATRIZ CUADRADA PARA USAR MÉTODO IMPLÍCITO
37 allocate(B_matrix(n,n))
38 B_matrix=0._dp ! elementos nulos fuera de la tribanda
39 do i=1,n
40     B_matrix(i,i)=2._dp*(1._dp/alpha-1._dp) ! diagonal principal
41     if (i/=1) B_matrix(i,i-1)=1._dp        ! diagonal inferior
42     if (i/=n) B_matrix(i,i+1)=1._dp        ! diagonal superior
43 end do
44 ! CARGAMOS DATOS INICIALES DE TEMPERATURA
45 allocate(A_matrix(n,1)) ! matriz auxiliar p/matmul (column vector rank=2)
46 allocate(ux_old(n+2))
47 ux_old(1)=u0_Li
48 write(10,20) 0._dp, ux_old(1)
49 do i=2,n+1
50     A_matrix(i-1,1)=u0_ti
51     write(10,20) real(i-1,dp)*x_step_adim, A_matrix(i-1,1)
52 end do
53 ux_old(n+2)=u0_Lf
54 write(10,20) real(n+1,dp)*x_step_adim, ux_old(n+2)
55 ! CARGAMOS DATOS INICIALES PARA APLICAR MÉTODO IMPLÍCITO
56 A_matrix=matmul(B_matrix,A_matrix)
57 ux_old(2:n+1)=A_matrix(1:n,1)
58 ! APLICAMOS MÉTODO IMPLÍCITO
59 allocate(ux_new(n+2))
60 ux_new(1)=ux_old(1)
61 ux_new(n+2)=ux_old(n+2)
62 do i=1,50
63     do j=1,(t_write-1)
64         call implicit_method(n,diag,diag_sup,diag_inf,data_vector=ux_old(2:n+1),unknown_vector=ux_new(2:n+1))
65         A_matrix(1:n,1)=ux_new(2:n+1)
66         A_matrix=matmul(B_matrix,A_matrix)
67         ux_old(2:n+1)=A_matrix(1:n,1)
68     end do

```

```

69     call implicit_method(n,diag,diag_sup,diag_inf,ux_old(2:n+1),ux_new(2:n+1))
70     do j=1,n+2
71         write(10,20) real(j-1,dp)*x_step_adim, ux_new(j)
72     end do
73 end do
74 close(10)
75 ! CONTROLAMOS TEMALIZACIÓN EN EL CENTRO DE LA BARRA
76 write(*,'(A20,E10.4)') 'T(L/2,t_final) = ', ux_new(50)
77 deallocate(diag,diag_sup,diag_inf,ux_old,ux_new)
78 deallocate(B_matrix,A_matrix)
79 end program heateq_crank_nicolson_method

```

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob01/code/heateq_comparison_01.f90

```

1  ! Comparación entre el método explícito y la solución exacta
2  program heateq_comparison_01
3      use module_precision
4      implicit none
5      real(dp), allocatable :: ux_old(:),ux_new(:)
6      integer(dp), parameter :: n=8_dp ! numero de elementos finitos
7      real(dp), parameter :: u0_Li=0._dp,u0_Lf=0._dp ! condiciones de borde
8      real(dp) :: u0_ti=100._dp ! condicion inicial
9      real(dp) :: param_t,param_x ! parametros para adimensionalizar (espacial y temporal)
10     real(dp), parameter :: D=237._dp*(1._dp/(900._dp*2700._dp)) ! thermal diffusivity(D=(K/C*rho)[L^2]/[t])
11     real(dp), parameter :: alpha=1.E-7_dp ! se asegura estabilidad (alpha < 0.5)
12     real(dp) :: T_exact ! temperatura exacta
13     real(dp) :: t_step_adim,x_step_adim ! variables adimensionales (espacial y temporal)
14     integer(dp) :: i,j,k,istat ! variables de loops
15     real(dp), parameter :: t_write_1_noadm=180._dp,t_write_2_noadm=1800._dp ! tiempos en dimensiones para escrituras
16     integer(dp) :: t_write_1,t_write_2 ! pasos temporales entre escrituras
17     ! parametros para adimensionalizar
18     param_x=1._dp ! longitud caracteristica (long total barra en metros) [L]
19     param_t=param_x*param_x*(1._dp/D) ! tiempo caracteristico [t]
20     x_step_adim=1._dp/(real(n,dp)+1._dp)
21     t_step_adim=alpha*x_step_adim*x_step_adim
22     allocate(ux_old(n+2))
23     ! inicializamos el vector espacial a tiempo inicial
24     ux_old(1)=u0_Li
25     ux_old(2:n+1)=u0_ti
26     ux_old(n+2)=u0_Lf
27     allocate(ux_new(n+2))
28     ! ADIMENSIONALIZAMOS LOS TIEMPOS DE ESCRITURA
29     t_write_1=int(t_write_1_noadm*(1._dp/(param_t*t_step_adim)),dp)
30     t_write_2=int(t_write_2_noadm*(1._dp/(param_t*t_step_adim)),dp)
31     write(*,*) t_write_2
32     ! HACEMOS LA EVOLUCIÓN TEMPORAL (SIN CONSIDERAR LOS EXTREMOS)
33     ux_new(1)=u0_Li;ux_new(n+2)=u0_Lf
34     do i=1,(t_write_1-1)
35         do j=2,(n+1)
36             ux_new(j)=(1._dp-2._dp*alpha)*ux_old(j)+alpha*(ux_old(j+1)+ux_old(j-1))
37             ux_old(j)=ux_new(j)
38         end do
39     end do
40     ! ESCRIBIMOS VALORES LUEGO DE t_write_1 PASOS TEMPORALES
41     !20 format (E9.2,x,E9.2)
42     !open(10,file='../results/result_02_aprox_explicit.dat',status='replace',action='write',iostat=istat)
43     !write(*,*) 'Input/Output file. istat10 = ',istat
44     !open(11,file='../results/result_02_exact.dat',status='replace',action='write',iostat=istat)
45     !write(*,*) 'Input/Output file. istat11 = ',istat
46     !call exact_solution(0._dp,t_write_1_noadm*(1._dp/param_t),u0_Li,T_exact)
47     call exact_solution(0._dp,t_write_1_noadm,D,u0_Li,T_exact)
48     !write(10,20) 0._dp,ux_new(1)
49     !write(11,20) 0._dp,T_exact
50     do j=2,(n+1)
51         ux_new(j)=(1._dp-2._dp*alpha)*ux_old(j)+alpha*(ux_old(j+1)+ux_old(j-1))
52         ux_old(j)=ux_new(j)
53         !call exact_solution(real(j-1,dp)*x_step_adim,t_write_1_noadm*(1._dp/param_t),u0_ti,T_exact)
54         call exact_solution(real(j-1,dp)*x_step_adim,t_write_1_noadm,D,u0_ti,T_exact)
55         !write(10,20) real(j-1,dp)*x_step_adim,ux_new(j)
56         !write(11,20) real(j-1,dp)*x_step_adim,T_exact
57     end do
58     !call exact_solution(real(n+1,dp)*x_step_adim,t_write_1_noadm*(1._dp/param_t),u0_Lf,T_exact)
59     call exact_solution(real(n+1,dp)*x_step_adim,t_write_1_noadm,D,u0_Lf,T_exact)
60     !write(10,20) real(n+1,dp)*x_step_adim,ux_new(n+2)
61     !write(11,20) real(n+1,dp)*x_step_adim,T_exact
62     do i=t_write_1,(t_write_2-1)
63         do j=2,(n+1)
64             ux_new(j)=(1._dp-2._dp*alpha)*ux_old(j)+alpha*(ux_old(j+1)+ux_old(j-1))
65             ux_old(j)=ux_new(j)
66         end do
67     end do
68     ! ESCRIBIMOS VALORES LUEGO DE t_write_2 PASOS TEMPORALES
69     !call exact_solution(0._dp,t_write_2_noadm*(1._dp/param_t),u0_Li,T_exact)
70     call exact_solution(0._dp,t_write_2_noadm,D,u0_Li,T_exact)
71     !write(10,20) 0._dp,ux_new(1)

```

```

72     !write(11,20) 0._dp,T_exact
73     do j=2,(n+1)
74         ux_new(j)=(1._dp-2._dp*alpha)*ux_old(j)+alpha*(ux_old(j+1)+ux_old(j-1))
75         ux_old(j)=ux_new(j)
76         !call exact_solution(real(j-1,dp)*x_step_adim,t_write_2_noadm*(1._dp/param_t),u0_ti,T_exact)
77         call exact_solution(real(j-1,dp)*x_step_adim,t_write_2_noadm,D,u0_ti,T_exact)
78         !write(10,20) real(j-1,dp)*x_step_adim,ux_new(j)
79         !write(11,20) real(j-1,dp)*x_step_adim,T_exact
80     end do
81     !call exact_solution(real(n+1,dp)*x_step_adim,t_write_2_noadm*(1._dp/param_t),u0_Lf,T_exact)
82     call exact_solution(real(n+1,dp)*x_step_adim,t_write_2_noadm,D,u0_Lf,T_exact)
83     !write(10,20) real(n+1,dp)*x_step_adim,ux_new(n+2)
84     !write(11,20) real(n+1,dp)*x_step_adim,T_exact
85     !close(10)
86     !close(11)
87     deallocate(ux_old,ux_new)
88 end program heateq_comparison_01
89 subroutine exact_solution(x,t,D,Temp0,Temp)
90     use module_precision
91     implicit none
92     ! variables de entrada/salida
93     real(dp), intent(in) :: x,t,Temp0,D ! ojo, x y t no son adimensionales
94     real(dp), intent(out) :: Temp
95     ! variables locales
96     real(dp), parameter :: pi=4._dp*atan(1._dp)
97     integer(dp), parameter :: n_max=81_dp ! debe ser impar
98     real(dp) :: kn
99     integer(dp) :: n
100     Temp=0._dp
101     do n=1,n_max,2 ! recorro numeros impares
102         kn=real(n,dp)*pi
103         Temp=Temp+(1._dp/real(n,dp))*sin(kn*x)*exp(-(kn*kn*t*D))
104     end do
105     Temp=4._dp*Temp0*(1._dp/pi)*Temp
106 end subroutine exact_solution

```

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob01/code/heateq_comparison_02.f90

```

1  ! Comparación entre el método implícito y la solución exacta
2  program heateq_comparison_02
3      use module_precision
4      use module_tridiag_matrix
5      implicit none
6      real(dp), allocatable :: ux_old(:),ux_new(:) ! vectores de temperatura (paso anterior y paso posterior)
7      real(dp), allocatable :: diag(:),diag_sup(:),diag_inf(:) ! diagonales de la matriz tridiagonal
8      integer(sp), parameter :: n=8_sp ! numero de elementos finitos dimension espacial
9      real(dp), parameter :: u0_Li=0._dp, u0_Lf=0._dp ! condiciones de borde
10     real(dp) :: u0_ti=100._dp ! condicion inicial
11     real(dp) :: param_t ! tiempo caracteristico (paso temporal) [t]
12     real(dp) :: param_x ! longitud caracteristica (long total barra) [L]
13     real(dp) :: D ! thermal diffusivity (D=K/C*rho [L^2]/[t])
14     real(dp) :: alpha=1.E-7_dp ! coeficiente discretización
15     real(dp) :: t_step_adim, x_step_adim ! deltatiempo y deltaespacio (adimensionales)
16     integer(sp) :: i,j,k,istat ! variables de loop y variable de control
17     real(dp), parameter :: t_write_1_noadm=180._dp,t_write_2_noadm=1800._dp ! tiempos en dimensiones para escrituras
18     integer(dp) :: t_write_1,t_write_2 ! pasos temporales entre escrituras
19     ! 20 format (E8.2,x,E8.2)
20     ! open( 10, file = '../results/result_02_aprox_implicit.dat', status = 'replace', action = 'write', iostat = istat )
21     ! ANÁLISIS DIMENSIONAL
22     D=237._dp*(1._dp/(900._dp*2700._dp))
23     param_x = 1._dp
24     param_t = param_x*param_x*(1._dp/D)
25     x_step_adim=1._dp/(real(n,dp)+1._dp)
26     t_step_adim=alpha*x_step_adim*x_step_adim
27     ! CARGAMOS DIAGONALES CENTRAL, SUPERIOR E INFERIOR
28     allocate(diag(n),diag_sup(n),diag_inf(n))
29     diag_sup(n)=0._dp
30     diag_inf(1)=0._dp
31     do i=1,n
32         diag(i)=1._dp+2._dp*alpha
33         if (i/=1) diag_inf(i) = -alpha
34         if (i/=n) diag_sup(i) = -alpha
35     end do
36     ! CARGAMOS DATOS INICIALES DE TEMPERATURA
37     allocate(ux_old(n+2))
38     ux_old(1)=u0_Li
39     ux_old(2:n+1)=u0_ti
40     ux_old(n+2)=u0_Lf
41     ! ADIMENSIONALIZAMOS LOS TIEMPOS DE ESCRITURA
42     t_write_1=int(t_write_1_noadm*(1._dp/(param_t*t_step_adim)),dp)
43     t_write_2=int(t_write_2_noadm*(1._dp/(param_t*t_step_adim)),dp)
44     ! APLICAMOS MÉTODO IMPLÍCITO
45     allocate(ux_new(n+2))
46     ux_new(1)=ux_old(1)
47     ux_new(n+2)=ux_old(n+2)
48     do j=1,(t_write_1-1)

```

```

49     call implicit_method(n,diag,diag_sup,diag_inf,ux_old(2:n+1),ux_new(2:n+1))
50     do k=2,(n+1); ux_old(k)=ux_new(k); end do
51 end do
52 ! ESCRIBIMOS VALORES LUEGO DE t_write_1 PASOS TEMPORALES
53 call implicit_method(n,diag,diag_sup,diag_inf,ux_old(2:n+1),ux_new(2:n+1))
54 !do j=1,n+2; write(10,20) real(j-1,dp)*x_step_adim, ux_new(j); end do
55 do j=t_write_1,(t_write_2-1)
56     call implicit_method(n,diag,diag_sup,diag_inf,ux_old(2:n+1),ux_new(2:n+1))
57     do k=2,(n+1); ux_old(k)=ux_new(k); end do
58 end do
59 ! ESCRIBIMOS VALORES LUEGO DE t_write_2 PASOS TEMPORALES
60 call implicit_method(n,diag,diag_sup,diag_inf,ux_old(2:n+1),ux_new(2:n+1))
61 !do j=1,n+2; write(10,20) real(j-1,dp)*x_step_adim, ux_new(j); end do
62 !close(10)
63 deallocate(diag,diag_sup,diag_inf,ux_old,ux_new)
64 end program heateq_comparison_02

```

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob01/code/heateq_comparison_03.f90

```

1  program heateq_comparison_03
2      use module_precision
3      use module_tridiag_matrix
4      implicit none
5      real(dp), allocatable :: ux_old(:),ux_new(:) ! vectores de temperatura (paso anterior y paso posterior)
6      real(dp), allocatable :: diag(:),diag_sup(:),diag_inf(:) ! diagonales de la matriz tridiagonal
7      real(dp), allocatable :: B_matrix(:,:),A_matrix(:,:)
8      integer(sp), parameter :: n=8_sp ! numero de elementos finitos dimension espacial
9      real(dp), parameter :: u0_Li=0._dp, u0_Lf=0._dp ! condiciones de borde
10     real(dp) :: u0_t=100._dp ! condicion inicial
11     real(dp) :: param_t ! tiempo caracteristico (paso temporal) [t]
12     real(dp) :: param_x ! longitud caracteristica (long total barra) [L]
13     real(dp) :: D ! thermal diffusivity (D=K/C*rho [L^2]/[t])
14     real(dp) :: alpha=1.E-7_dp ! coeficiente discretización
15     real(dp) :: t_step_adim, x_step_adim ! deltatiempo y deltaespacio (adimensionales)
16     integer(sp) :: i,istat ! variables de loop y variable de control
17     real(dp), parameter :: t_write_1_noadm=180._dp,t_write_2_noadm=1800._dp ! tiempos en dimensiones para escrituras
18     integer(dp) :: t_write_1,t_write_2 ! pasos temporales entre escrituras
19     ! 20 format (E8.2,x,E8.2)
20     ! open( 10, file = '../results/result_02_aprox_cranksnicolson.dat', status = 'replace', action = 'write', iostat = istat )
21     ! ANÁLISIS DIMENSIONAL
22     D=237._dp*(1._dp/(900._dp*2700._dp))
23     param_x = 1._dp
24     param_t = param_x*param_x*(1._dp/D)
25     x_step_adim=1._dp/(real(n,dp)+1._dp)
26     t_step_adim=alpha*x_step_adim*x_step_adim
27     ! ADIMENSIONALIZAMOS LOS TIEMPOS DE ESCRITURA
28     t_write_1=int(t_write_1_noadm*(1._dp/(param_t*t_step_adim)),dp)
29     t_write_2=int(t_write_2_noadm*(1._dp/(param_t*t_step_adim)),dp)
30     ! CARGAMOS DIAGONALES CENTRAL, SUPERIOR E INFERIOR
31     allocate(diag(n),diag_sup(n),diag_inf(n))
32     diag_sup(n)=0._dp
33     diag_inf(1)=0._dp
34     do i=1,n
35         diag(i)=2._dp*(1._dp/alpha+1._dp)
36         if (i/=1) diag_inf(i)=-1._dp
37         if (i/=n) diag_sup(i)=-1._dp
38     end do
39     ! CARGAMOS MATRIZ CUADRADA PARA USAR MÉTODO IMPLÍCITO
40     allocate(B_matrix(n,n))
41     B_matrix=0._dp ! elementos nulos fuera de la tribanda
42     do i=1,n
43         B_matrix(i,i)=2._dp*(1._dp/alpha+1._dp) ! diagonal principal
44         if (i/=1) B_matrix(i,i-1)=1._dp ! diagonal inferior
45         if (i/=n) B_matrix(i,i+1)=1._dp ! diagonal superior
46     end do
47     ! CARGAMOS DATOS INICIALES DE TEMPERATURA
48     allocate(A_matrix(n,1)) ! matriz auxiliar p/matmul (column vector rank=2)
49     allocate(ux_old(n+2))
50     ux_old(1)=u0_Li
51     A_matrix(1:n,1)=u0_t
52     ux_old(n+2)=u0_Lf
53     !CARGAMOS DATOS INICIALES PARA APLICAR MÉTODO IMPLÍCITO
54     A_matrix=matmul(B_matrix,A_matrix)
55     ux_old(2:n+1)=A_matrix(1:n,1)
56     ! APLICAMOS MÉTODO IMPLÍCITO
57     allocate(ux_new(n+2))
58     ux_new(1)=ux_old(1)
59     ux_new(n+2)=ux_old(n+2)
60     do i=1,(t_write_1-1)
61         call implicit_method(n,diag,diag_sup,diag_inf,data_vector=ux_old(2:n+1),unknown_vector=ux_new(2:n+1))
62         A_matrix(1:n,1)=ux_new(2:n+1)
63         A_matrix=matmul(B_matrix,A_matrix)
64         ux_old(2:n+1)=A_matrix(1:n,1)
65     end do
66     ! ESCRIBIMOS VALORES LUEGO DE t_write_1 PASOS TEMPORALES
67     call implicit_method(n,diag,diag_sup,diag_inf,ux_old(2:n+1),ux_new(2:n+1))

```

```

68 ! do i=1,n+2
69 !   write(10,20) real(i-1,dp)*x_step_adim, ux_new(i)
70 ! end do
71 do i=t_write_1,(t_write_2-1)
72   call implicit_method(n,diag,diag_sup,diag_inf,data_vector=ux_old(2:n+1),unknown_vector=ux_new(2:n+1))
73   A_matrix(1:n,1)=ux_new(2:n+1)
74   A_matrix=matmul(B_matrix,A_matrix)
75   ux_old(2:n+1)=A_matrix(1:n,1)
76 end do
77 ! ESCRIBIMOS VALORES LUEGO DE t_write_2 PASOS TEMPORALES
78 call implicit_method(n,diag,diag_sup,diag_inf,ux_old(2:n+1),ux_new(2:n+1))
79 ! do i=1,n+2
80 !   write(10,20) real(i-1,dp)*x_step_adim, ux_new(i)
81 ! end do
82 !close(10)
83 deallocate(diag,diag_sup,diag_inf,ux_old,ux_new)
84 deallocate(B_matrix,A_matrix)
85 end program heateq_comparison_03

```

Módulos

https://github.com/mendzmartin/fiscomp2022/blob/main/modules/module_precision.f90

```

1 module module_precision
2   implicit none
3   integer, parameter :: sp = selected_real_kind(p=6,r=37) ! simple presicion (sp) class
4   integer, parameter :: dp = selected_real_kind(p=15,r=307) ! double presicion (dp) class
5   integer, parameter :: qp = selected_real_kind(p=33,r=4931) ! quad presicion (dp) class
6 end module module_precision
7 !-----
8 ! REFERENCES
9 !-----
10 ! https://fortranwiki.org/fortran/show/Real+precision
11 !-----

```

https://github.com/mendzmartin/fiscomp2022/blob/main/modules/module_tridiag_matrix.f90

```

1 module module_tridiag_matrix
2   use module_precision
3   implicit none
4   contains
5   subroutine implicit_method(n,diag,diag_sup,diag_inf,data_vector,unknown_vector)
6     ! variables de entrada/salida
7     integer(sp), intent(in) :: n ! dimension del vector diagonal
8     real(dp), intent(out) :: unknown_vector(n) ! vector de incognitas
9     real(dp), intent(in) :: diag(n),diag_sup(n),diag_inf(n) ! diagonales central, superior e inferior
10    real(dp), intent(in) :: data_vector(n) ! vector de datos conocidos
11    ! variables locales
12    integer(sp) :: i ! variable del loop
13    real(dp) :: diag_sup_new(n) ! diagonal superior redefinida
14    real(dp) :: data_vector_new(n) ! vector de datos redefinidos
15    real(dp) :: factor
16    ! descomposición LU
17    factor=1._dp/diag(1)
18    diag_sup_new(1)=diag_sup(1)*factor
19    data_vector_new(1)=data_vector(1)*factor
20    do i=2,n
21      factor=1._dp/(diag(i)-diag_inf(i)*diag_sup_new(i-1))
22      diag_sup_new(i)=diag_sup(i)*factor
23      data_vector_new(i)=(data_vector(i)-diag_inf(i)*data_vector_new(i-1))*factor
24    end do
25    ! sustitución hacia atrás
26    unknown_vector(n)=data_vector_new(n)
27    do i=(n-1),1,-1
28      unknown_vector(i)=data_vector_new(i)-diag_sup_new(i)*unknown_vector(i+1)
29    end do
30  end subroutine implicit_method
31 end module module_tridiag_matrix
32 ! Los vectores de entrada deben definirse de la siguiente manera
33 ! diag_sup = [ ds(1)  ds(2) ... ds(n-1) ds(n)=0 ]
34 ! diag_inf = [ di(1)=0 di(2) ... di(n-1) di(n) ]
35 ! diag = [ d(1)  d(2) ... di(n-1) d(n) ]

```