Facultad de Matemática, Astronomía, Física y Computación (FaMAF-UNC)
Curso de Física Computacional 2022-Guía N°02

Problema 2

Ecuación de Calor: condiciones de contorno de Neumann

Resolveremos, mediante diferentes algoritmos, la ecuación de calor ya adimensionalizada

$$\frac{\partial[T(\tilde{x},\tilde{t})]}{\partial\tilde{t}} = \frac{\partial^2[T(\tilde{x},\tilde{t})]}{\partial^2\tilde{x}} \tag{1}$$

con las siguientes condiciones de contorno e iniciales

$$T(x,0) = cos(\pi x),\ \underbrace{\frac{\partial[T(0,t)]}{\partial x}}_{g_0} = \underbrace{\frac{\partial[T(L,t)]}{\partial x}}_{g_L} = 0 \tag{2}$$

a) Modifique los códigos escritos en el problema anterior para que consideren condiciones de contorno de Neumann.

b) Resuelva la ecuación utilizando los diferentes métodos. Utilice $\Delta\tilde{x} = 0.05$ y $\Delta\tilde{t} = 0.001$. Al menos para alguno de ellos, haga un gráfico de superficie mostrando $T(x,t)$ versus (x,t) y un mapa de temperatura incluyendo curvas de nivel.

c) La solución analítica es

$$T(x,t) = \exp\left(-\pi^2 t\right)\cos(\pi x) \equiv T(x,0)\exp\left(-\pi^2 t\right) \tag{3}$$

Compare en un gráfico de $T(x,1)$ versus x los tres métodos y la solución exacta. Grafique, además, el "máximo error absoluto" de la solución numérica a lo largo de la barra, para cada tiempo, versus tiempo. Compare los tres métodos (en la misma figura).

d) Si disminuye a $\Delta\tilde{t} = 0.00025$, ¿qué espera obtener? ¿habría que cambiar/variar algo más? Repita las gráficas del inciso anterior, compare con resultados previos y discuta la teoría.

Introducción

Aquí para calcular la cantidad de elementos finitos tuvimos en cuenta que

$$\Delta\tilde{x} = \frac{1}{(n+1)} \Rightarrow n = \text{int}\left(\frac{1}{\Delta\tilde{x}} - 1\right)$$

sin embargo, el calculo anterior puede no ser correcto pues, como n es entero y $\Delta\tilde{x}$ es real, al hacer la conversión anterior pude darnos el entero inmediatamente anterior al correcto para obtener un $\Delta\tilde{x}_{aprox}$ lo más cercano al propuesto para ello, se utilizó la función módulo y se implementó el siguiente bloque de código

```
1 [...]
2   integer(sp)        :: n                ! numero de elementos finitos (FE)
3   integer(sp)        :: a,b              ! variables auxiliares
4   real(dp),parameter :: x_step_adim=0.05 ! deltaX adimensional
5   ! calculo parte entera del modulo y del resto de x_step_adim^(-1)
6   a=int(mod(1._dp,x_step_adim),sp); b=int(1._dp/x_step_adim,sp)
7   ! calculamos el número de FE
8   n=(a+b-1_sp)*(1_sp/(1_sp-a))
9 [...]
```

esto nos asegura que el número entero n obtenido siempre nos asegura obtener el $\Delta\tilde{x}_{aprox}$ más cercano al real. Y además, como el valor de $\Delta\tilde{x}$ debe estar entre cero y uno, la formula de la línea (10) nos asegura que no exista un error aritmético.

Inciso a)

Se tuvieron en cuenta las siguientes ecuaciones que modifican que nos permite modificar los códigos del problema 1 para utilizar las condiciones de contorno de Von-Neumann, en resumidas cuentas, los que hacemos es no sólo evolucionar los n puntos internos de la barra (de 2 a $n+1$ puntos) sino que también se evolucionan los puntos extremos, y en total se estarían evolucionando de 1 a $n+2$ puntos de la barra (todos los puntos físicos). Esta es la principal diferencia respecto a las condiciones de contorno de Dirichlet, es decir, el problema ahora se modifica agregando dos puntos "fantasmas" (los puntos ficticios 0 y $n+3$) y se encuentra una expresión de estos puntos en términos de los puntos reales de la grilla pero con el agregado de que ahora evolucionan todos los puntos de la barra (esto se hace aproximando la derivada, condiciones de borde de V-N, por su expresión en el método de diferencias centradas).

Método explícito (diferencia hacia adelante para derivada temporal)

$$T_{1,(j+1)} = 2\eta T_{1,j} + (1-2\eta)T_{0,j} - 2\eta\Delta x g_0; i = 1$$
$$T_{i,(j+1)} = (1-2\eta)T_{i,j} + \eta[T_{(i-1),j} + T_{(i+1),j}] \ \forall \ 2 \leqslant i \leqslant (n+1)$$
$$T_{n+2,(j+1)} = 2\eta T_{(n+2),j} + (1-2\eta)T_{(n+2),j} + 2\eta\Delta x g_{(n+2)}; i = (n+2)$$

Método implícito (diferencia hacia atrás para derivada temporal)

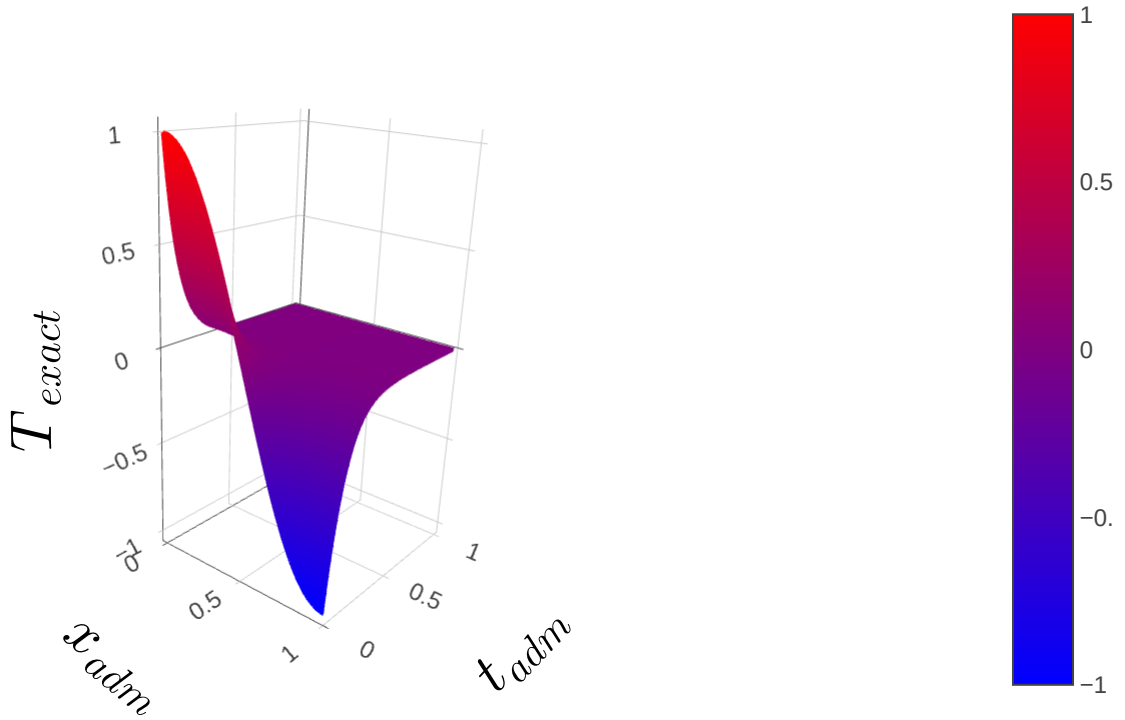
$$T_{1,(j+1)} = -2\eta T_{1,j} + (1-2\eta)T_{0,j} + 2\eta\Delta x g_0; i = 1$$
$$T_{i,(j+1)} = (1+2\eta)T_{i,j} - \eta[T_{(i-1),j} + T_{(i+1),j}] \ \forall \ 2 \leqslant i \leqslant (n+1)$$
$$T_{n+2,(j+1)} = -2\eta T_{(n+2),j} + (1+2\eta)T_{(n+2),j} - 2\eta\Delta x g_{(n+2)}; i = (n+2)$$

Método de Crank-Nicolson (diferencia centrada para derivada temporal - split lineal)

$$\begin{bmatrix} \frac{2}{\eta}+2 & -2 & 0 & \cdots & \cdots 0 \\ -1 & \frac{2}{\eta}+2 & -1 & \cdots & \cdots 0 \\ 0 & \cdots & \cdots & \cdots & \vdots \\ \vdots & 0\cdots & -1 & \frac{2}{\eta}+2 & -1 \\ 0 & 0\cdots & 0 & -2 & \frac{2}{\eta}+2 \end{bmatrix} \cdot \begin{bmatrix} T_{0,j+1} \\ T_{1,j+1} \\ \vdots \\ T_{n,j+1} \\ T_{n+1,j+1} \end{bmatrix} = \begin{bmatrix} \frac{2}{\eta}-2 & -2 & 0 & \cdots & \cdots 0 \\ -1 & \frac{2}{\eta}-2 & -1 & \cdots & \cdots 0 \\ 0 & \cdots & \cdots & \cdots & \vdots \\ \vdots & 0\cdots & -1 & \frac{2}{\eta}-2 & -1 \\ 0 & 0\cdots & 0 & -2 & \frac{2}{\eta}-2 \end{bmatrix} \cdot \begin{bmatrix} T_{0,j} \\ T_{1,j} \\ \vdots \\ T_{n,j} \\ T_{n+1,j} \end{bmatrix} + \begin{bmatrix} 2\eta\Delta x\,(g_{0,j} - g_{0,j+1}) \\ 0 \\ \vdots \\ 0 \\ -2\eta\Delta x\,(g_{n+1,j} - g_{n+1,j+1}) \end{bmatrix}$$

inciso c)

Si graficamos la solución exacta obtenemos el siguiente resultado



esto nos va a permitir comparar fielmente los resultados numéricos. Además, de la ecuación exacta y las condiciones de contorno vemos que las fórmulas consideran a todas las variables como adimensionales es decir, coordenada espacial (x), coordenada temporal (t) y temperatura (T). Ahora bien, como la temperatura, para cualquier tiempo, oscila entre -1 y 1 se deduce que estamos pensando en temperaturas adimensionales obtenidas a partir de temperaturas en grados centígrados pues, es posible obtener temperaturas negativas que sabemos que no puede ser así si consideramos temperaturas en Kelvins.

Por otro lado, como la temperatura de termalización es cero y las temperatura máximas que puede alcanzar la barra son de ± 1 (valor no muy lejano al cero) los tiempos de relajación al equilibrio son relativamente cortos, comparados, por ejemplo, con el problema 1 de la guía (que requería ≈ 15000 pasos temporales de evolución). Aquí, con solo 1000 pasos temporales, la temperatura se encuentra a valores del $\mathcal{O}\left(10^{-5}\right)$.

Resultados y Discusiones

Inciso b)

Los gráficos que muestran las distribuciones de temperatura vs el tiempo para los métodos explícito e implícito se muestran a continuación

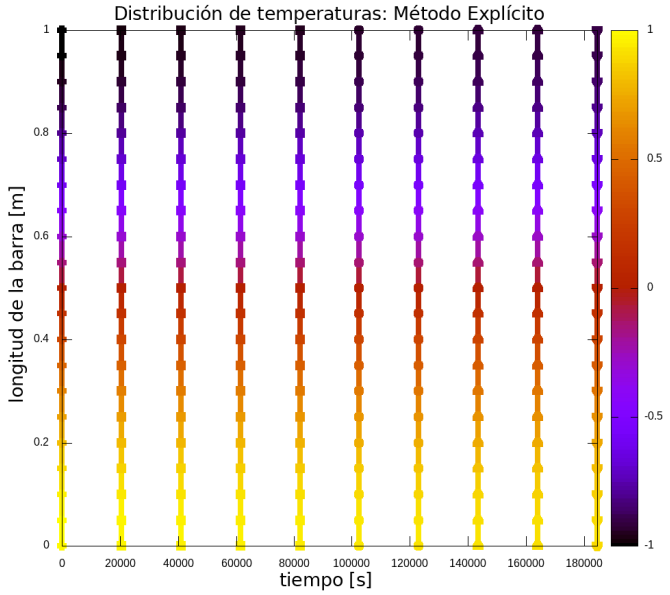


Figure 1:

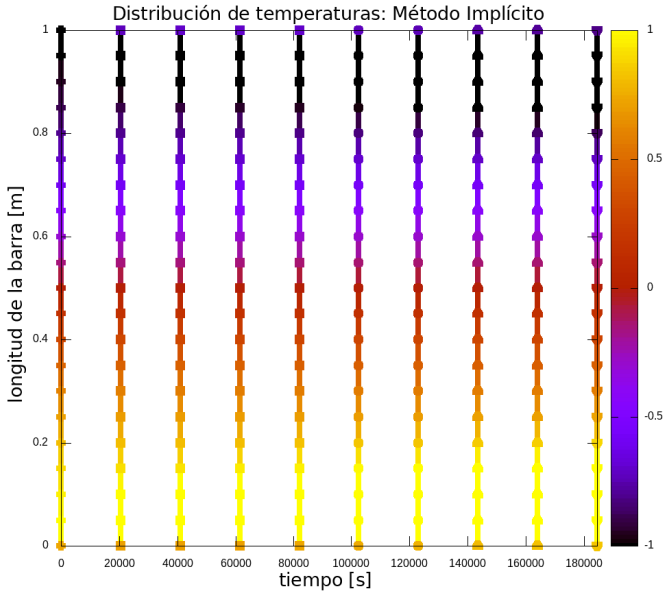


Figure 2:

Si bien las coordenadas temporales y espaciales están adimensionalizadas se supuso que se trataba de una barra de $1[m]$ de longitud (tal como fue definido en el problema 1 de la guía) y se dimensionalizó el tiempo con el tiempo característico de $0.1025E+05[s]$, además, se evolucionó la barra durante 20 pasos temporales y se imprimieron los datos cada 2 pasos temporales obteniendo 10 barras con su distribución de temperatura correspondiente.

Ahora bien, para el método de C-N se realizón un gráfico de superficie obteniendo,

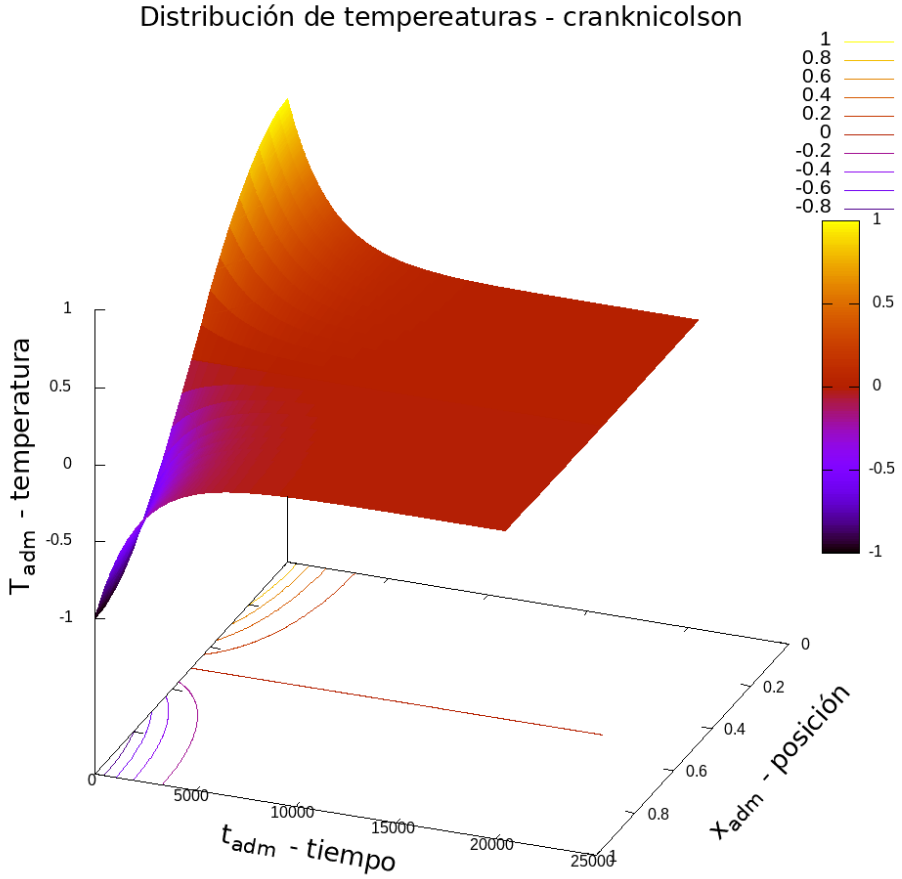


Figure 3:

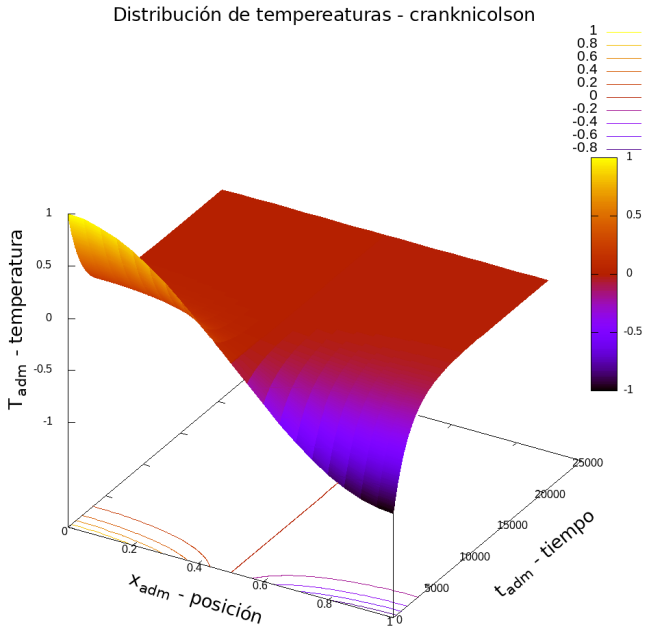


Figure 4:

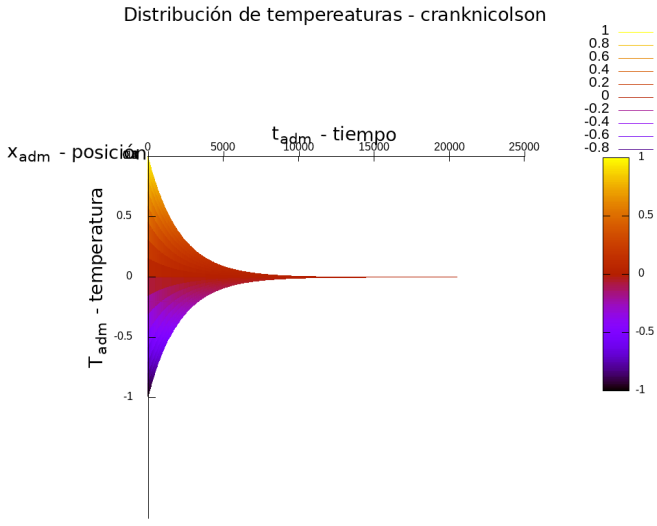


Figure 5:

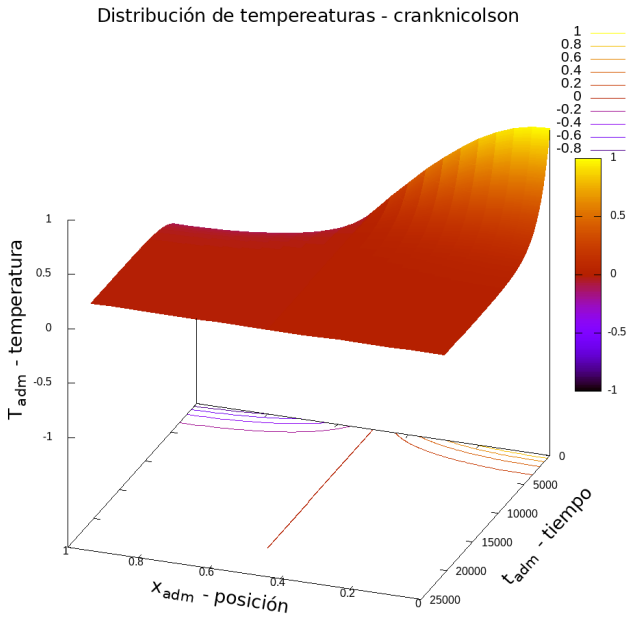


Figure 6:

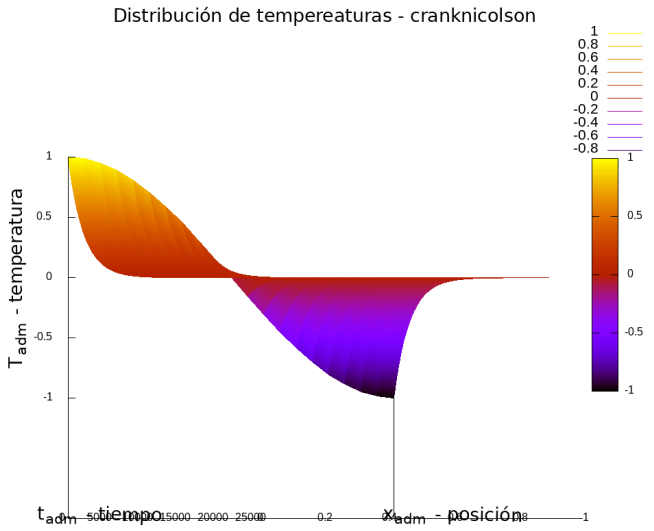


Figure 7:

Comparando esta gráfica de superficie con la superficie de la solución exacta vemos que son muy similares, en cada una de las dimensiones (por ello se han hecho vistas superior y lateral). Teniendo una distribución de temperaturas cosenoidal en el instante inicial y una distribución cosenoidal en el tiempo, modulada con una exponencial decreciente, lo cual nos muestra el rápido decaimiento al equilibrio térmico. En la base del gráfico se pueden apreciar algunas curvas de nivel.

Inciso c)

A continuación se muestran los errores numéricos obtenidos,

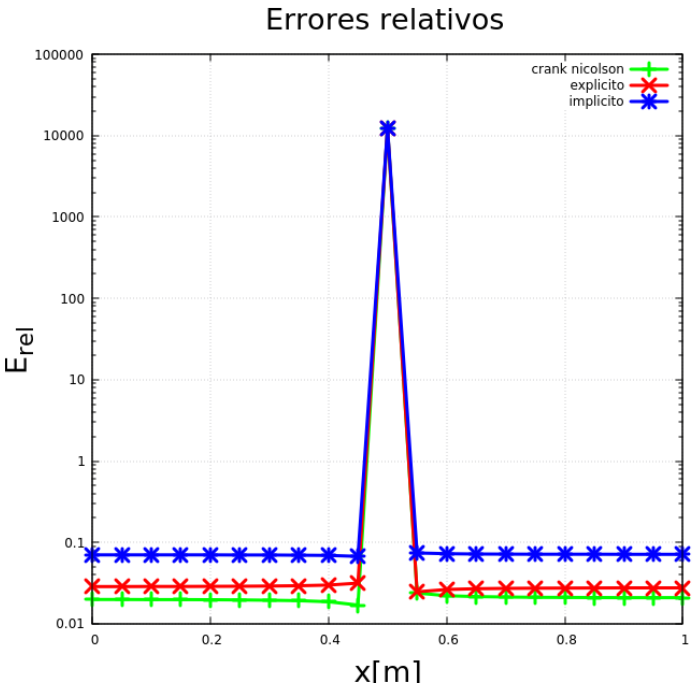


Figure 8:

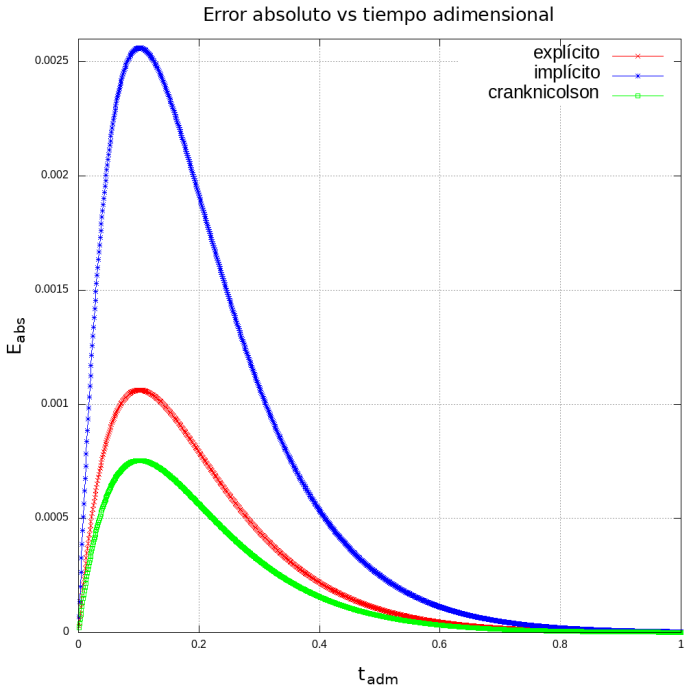


Figure 9:

En la primer figura, se observa el error relativo $\epsilon_{rel} = \left| {}^{(\epsilon_{eacto} - \epsilon_{aprox.})}/_{\epsilon_{eacto}} \right|$ para la distribución de temperaturas al tiempo fijo de $t_{adm} = 1$ para cada posición de la barra. El mayor error se obtuvo

para el centro de la barra y para el método implícito, seguido por el método explícito y finalmente el método de C-N. Como el eje vertical se graficó en escala logarítmica notamos que los órdenes de magnitud son similares para cada uno de los métodos.
En la segunda figura, se observa el mayor error absoluto definido como $\epsilon_{rel} = \max|(\epsilon_{eacto} - \epsilon_{aprox.})|$ para la distribución de temperaturas para todo tiempo (en el rango de $0 \leqslant t_{adm} \leqslant 1$), al igual que en el caso anterior se observa que el método más preciso es el de C-N, seguido por el método explícito y finalmente por el método explícito. Además, en vemos que esta gráfica es útil para observar a que tiempo los errores son mayores, y a medida que incluimos más pasos temporales el error disminuye convergiendo al valor exacto. Si cambiamos la escala del eje vertical a logarítmica podremos apreciar la ley de potencia del error,

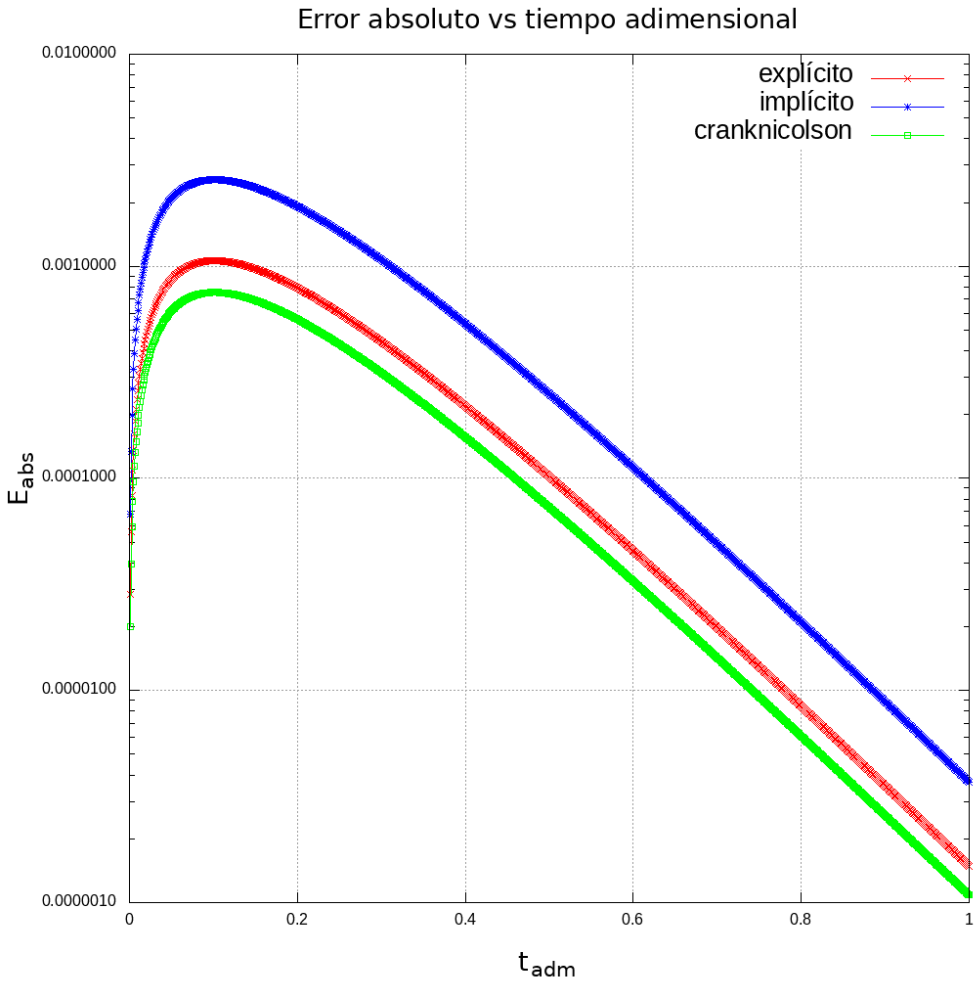


Figure 10:

al tener los tres métodos la misma pendiente siguen la misma ley de potencia.
Inciso d)
Si disminuimos la discretización temporal a un valor de $\Delta\tilde{t} \equiv \Delta t_{adim} = 0.00025$ entonces tendremos que el parámetro para controlar estabilidad será (sin modificar la discretización espacial $\Delta\tilde{x}$

$$\alpha = \frac{\Delta\tilde{t}}{(\Delta\tilde{x})^2} = \frac{0.00025}{(0.05)^2} = 0.1 < 0.5$$

los cual nos asegura que el método explícito es convergente y (como se ha discutido en el problema 1) si no modificamos la discretización en la coordenada espacial el error quizás disminuya pero no apreciablemente, pues la discretización de la barra seguirá siendo la misma que en los casos anteriores y el error mínimo estaría condicionado a esta discretización, lo que si se esperaría es que los tiempos de CPU sean mayores, pues para un tiempo final de termalización fijo, al achicar el

paso temporal nos tomaría más pasos evolucionar para llegar a dicho tiempo.

Códigos

Repositorio GitHub

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

Repositorio GitHub del problema

https://github.com/mendzmartin/fiscomp2022/tree/main/lab02/prob02

Programas principales

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob02/code/heateq_explicit_von_neumann.f90

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob02/code/heateq_implicit_von_neumann.f90

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob02/code/heateq_crank_nicolson_von_neumann.f90

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob02/code/heateq_comparison_01.f90

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob02/code/heateq_comparison_02.f90

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob02/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/prob02/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., PÃ¡ez, 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/prob02/code/heateq_explicit_von_neumann.f90

```
1  program heateq_explicit_von_neumann
2      use module_precision
3      implicit none
4      real(dp),    parameter   :: u0_Li=0._dp,u0_Lf=0._dp  ! condiciones de contorno von neumann
5      integer(sp), parameter   :: t_write=2_sp             ! pasos temporales entre escrituras
6      real(dp),    parameter   :: t_step_adim=0.001_dp,x_step_adim=0.05_dp ! deltaT y deltaX adimensionales
7      real(dp)                 :: pi=4._dp*atan(1.0_dp)
8      real(dp)                 :: D                        ! coeficiente de difusión
9      real(dp)                 :: param_t,param_x          ! parametros para adimensionalizar (espacial y temporal)
10     real(dp)                 :: alpha                    ! coeficiente para controlar estabilidad
11     integer(sp)              :: i,j,k,istat
12     integer(sp)              :: n,a,b                    ! numero de elementos finitos y variables auxiliares
13     real(dp),    allocatable :: ux_old(:),ux_new(:)
14     20 format(E13.6,x,E13.6)
15     open(10,file='../results/result_01_explicit_vn.dat',status='replace',action='write',iostat=istat)
16     write(*,*) 'istat(10file) = ',istat
17     ! ANALISIS DIMENSIONAL
18     D=237._dp*(1._dp/(900._dp*2700._dp))  ! D = K/C*rho [L^2]/[t]
19     param_x=1._dp                         ! longitud caracteristica (long total barra en metros) [L]
20     param_t=param_x*param_x*(1._dp/D)     ! tiempo caracteristico [t]
21     alpha=t_step_adim*(1._dp/(x_step_adim*x_step_adim))
22     ! verificamos condición de estabilidad debe ser menor a 1/2
```

```
    write(*,'(A20,E10.4)') 'alpha = ',alpha; write(*,'(A20,E10.4)') 'param_t = ',param_t
    ! calculamos la parte entera del modulo y del resto del inverso de x_step_adim
    a=int(mod(1._dp,x_step_adim),sp);b=int(1._dp/x_step_adim);n=(a+b-1_sp)*(1_sp/(1_sp-a))
    allocate(ux_old(n+2))
    ! inicializamos el vector espacial a tiempo inicial
    do i=1,n+2
        ux_old(i)=cos(pi*real(i-1,dp)*x_step_adim)
        write(10,20) real(i-1,dp)*x_step_adim,ux_old(i)
    end do
    allocate(ux_new(n+2))
    ! hacemos la evolución temporal (considerando los extremos)
    do k=1,10 ! evolucionamos (10*t_write) pasos temporales
        do i=1,(t_write-1)
            ux_new(1)=(1._dp-2._dp*alpha)*ux_old(1)+2._dp*alpha*(ux_old(2)-x_step_adim*u0_Li)
            do j=2,(n+1); ux_new(j)=(1._dp-2._dp*alpha)*ux_old(j)+alpha*(ux_old(j+1)+ux_old(j-1));end do
            ux_new(n+2)=(1._dp-2._dp*alpha)*ux_old(n+2)+2._dp*alpha*(ux_old(n+1)+x_step_adim*u0_Lf)
            ux_old(1:n+2)=ux_new(1:n+2)
        end do
        ! escribimos valores luego de t_write pasos temporales
        ux_new(1)=(1._dp-2._dp*alpha)*ux_old(1)+2._dp*alpha*(ux_old(2)-x_step_adim*u0_Li)
        write(10,20) 0._dp,ux_new(1)
        do j=2,(n+1)
            ux_new(j)=(1._dp-2._dp*alpha)*ux_old(j)+alpha*(ux_old(j+1)+ux_old(j-1))
            write(10,20) real(j-1,dp)*x_step_adim,ux_new(j)
        end do
        ux_new(n+2)=(1._dp-2._dp*alpha)*ux_old(n+2)+2._dp*alpha*(ux_old(n+1)+x_step_adim*u0_Lf)
        write(10,20) real(n+1,dp)*x_step_adim,ux_new(n+2)
    end do
    close(10);deallocate(ux_old,ux_new)
end program heateq_explicit_von_neumann
```

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob02/code/heateq_implicit_von_neumann.f90

```
program heateq_implicit_von_neumann
    use module_precision;use module_tridiag_matrix
    implicit none
    real(dp),    parameter   :: u0_Li=0._dp, u0_Lf=0._dp                ! condiciones de borde
    real(dp),    parameter   :: t_step_adim=0.001_dp,x_step_adim=0.05_dp ! deltaT y deltaX adimensionales
    integer(sp), parameter   :: t_write=2_sp                            ! pasos temporales entre escrituras
    real(dp)                 :: pi=4._dp*atan(1.0)
    real(dp)                 :: param_t,param_x
    real(dp)                 :: D                       ! thermal diffusivity
    real(dp)                 :: alpha
    integer(sp)              :: n,a,b                   ! numero de elementos finitos y variables auxiliares
    integer(sp)              :: i,j,istat
    real(dp),    allocatable :: ux_old(:),ux_new(:),diag(:),diag_sup(:),diag_inf(:)
    20 format(E13.6,x,E13.6)
    open(10,file='../results/result_01_implicit_vn.dat',status='replace',action='write',iostat=istat)
    write(*,*) 'istat(10file) = ',istat
    ! parametros para adimensionalizar
    D=237._dp*(1._dp/(900._dp*2700._dp)) ! D = K/C*rho [L^2]/[t]
    param_x=1._dp                        ! longitud caracteristica (long total barra en metros) [L]
    param_t=param_x*param_x*(1._dp/D)    ! tiempo caracteristico (paso temporal en segundos) [t]
    alpha=t_step_adim*(1._dp/(x_step_adim*x_step_adim))
    ! calculamos la parte entera del modulo y del resto del inverso de x_step_adim
    a=int(mod(1._dp,x_step_adim),sp);b=int(1._dp/x_step_adim);n=(a+b-1_sp)*(1_sp/(1_sp-a))
    ! cargamos diagonales central, superior e inferior
    allocate(diag(n+2),diag_sup(n+2),diag_inf(n+2))
    diag_sup(n+2)=0._dp;diag_inf(1)=0._dp
    do i=1,n+2
        diag(i)=1._dp+2._dp*alpha
        if (i/=1) diag_inf(i)=-alpha;if (i/=n+2) diag_sup(i)=-alpha
    end do
    ! cambiamos los extremos de las diagonales iferior y superior
    diag_inf(n+1)=diag_inf(n+1)*2._dp;diag_sup(2)=diag_sup(2)*2._dp
    ! cargamos datos iniciales de temperatura
    allocate(ux_old(n+2))
    do i=1,n+2
        ux_old(i)=cos(pi*real(i-1,dp)*x_step_adim)
        write(10,20) real(i-1,dp)*x_step_adim,ux_old(i)
    end do
    ! aplicamos método implicito
    allocate(ux_new(n+2))
    do i=1,10
        do j=1,(t_write-1)
            ux_old(1)=ux_old(1)+2._dp*alpha*x_step_adim*u0_Li
            ux_old(n+2)=ux_old(n+2)-2._dp*alpha*x_step_adim*u0_Lf
```

```
            call implicit_method(n+2,diag,diag_sup,diag_inf,ux_old(:),ux_new(:))
            ux_old(:)=ux_new(:)
        end do
        call implicit_method(n+2,diag,diag_sup,diag_inf,ux_old(:),ux_new(:))
        do j=1,n+2;write(10,20) real(j-1,dp)*x_step_adim,ux_new(j);end do
    end do
    close(10);deallocate(diag,diag_sup,diag_inf,ux_old,ux_new)
end program heateq_implicit_von_neumann
```

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob02/code/heateq_crank_nicolson_von_neumann.f90

```
program heateq_crank_nicolson_von_neumann
    use module_precision;use module_tridiag_matrix
    implicit none
    real(dp),    parameter   :: u0_Li=0._dp, u0_Lf=0._dp                ! condiciones de borde de Von Neumann (t=t0)
    real(dp),    parameter   :: u1_Li=0._dp, u1_Lf=0._dp                ! condiciones de borde de Von Neumann (t=t1)
    real(dp),    parameter   :: t_step_adim=0.001_dp,x_step_adim=0.05_dp ! deltaT y deltaX adimensionales
    integer(sp), parameter   :: t_write=2_sp                            ! pasos temporales entre escrituras
    real(dp),    parameter   :: pi=4._dp*atan(1.0)
    real(dp),    parameter   :: D=237._dp*(1._dp/(900._dp*2700._dp)) ! thermal diffusivity(D=(K/C*rho)[L^2]/[t])
    real(dp),    parameter   :: param_x=1._dp                    ! longitud caracteristica (long total barra en metros) [L]
    real(dp),    parameter   :: param_t=param_x*param_x*(1._dp/D) ! tiempo caracteristico (paso temporal en segundos) [t]
    real(dp),    parameter   :: alpha=t_step_adim*(1._dp/(x_step_adim*x_step_adim))
    integer(sp)              :: n,a,b                   ! numero de elementos finitos y variables auxiliares
    integer(sp)              :: i,j,istat
    real(dp),    allocatable :: ux_old(:),ux_new(:),diag(:),diag_sup(:),diag_inf(:)
    real(dp),    allocatable :: B_matrix(:,:),A_matrix(:,:)
    20 format(E13.6,x,E13.6,x,E13.6)
    open(10,file='../results/result_01_cranknicolson_vn.dat',status='replace',action='write',iostat=istat)
    write(*,*) 'istat(10file) = ',istat
    write(*,*) 't_adim = ',10*t_step_adim
    ! calculamos la parte entera del modulo y del resto del inverso de x_step_adim
    a=int(mod(1._dp,x_step_adim),sp);b=int(1._dp/x_step_adim);n=(a+b-1_sp)*(1_sp/(1_sp-a))
    ! CARGAMOS DIAGONALES CENTRAL, SUPERIOR E INFERIOR
    allocate(diag(n+2),diag_sup(n+2),diag_inf(n+2))
    diag(:)=2._dp*(1._dp/alpha+1._dp);diag_inf(:)=-1._dp;diag_sup(:)=-1._dp
    ! cambiamos los extremos de las diagonales iferior y superior
    diag_inf(n+2)=diag_inf(n+2)*2._dp;diag_sup(1)=diag_sup(1)*2._dp
    diag_sup(n+2)=0._dp;diag_inf(1)=0._dp
    ! CARGAMOS MATRIZ CUADRADA PARA USAR MÉTODO IMPLÍCITO
    allocate(B_matrix(n+2,n+2))
    B_matrix=0._dp ! elementos nulos fuera de la tribanda
    do i=1,n+2
        B_matrix(i,i)=2._dp*(1._dp/alpha-1._dp) ! diagonal principal
        if (i/=1) B_matrix(i,i-1)=1._dp         ! diagonal inferior
        if (i/=n+2) B_matrix(i,i+1)=1._dp       ! diagonal superior
    end do
    ! cambiamos los extremos de las diagonales iferior y superior de la matriz B
    B_matrix(n+2,n+1)=B_matrix(n+2,n+1)*2._dp;B_matrix(1,2)=B_matrix(1,2)*2._dp
    ! CARGAMOS DATOS INICIALES DE TEMPERATURA
    allocate(A_matrix(n+2,1)) ! matriz auxiliar p/matmul (column vector rank=2)
    allocate(ux_old(n+2))
    do i=1,n+2
        ux_old(i)=cos(pi*real(i-1,dp)*x_step_adim);A_matrix(i,1)=ux_old(i)
        write(10,20) 0.0_dp,real(i-1,dp)*x_step_adim,ux_old(i)
    end do
    write(10,*) ''
    ! CARGAMOS DATOS INICIALES PARA APLCIAR MÉTODO IMPLÍCITO
    A_matrix=matmul(B_matrix,A_matrix);ux_old(:)=A_matrix(:,1)
    ! APLICAMOS MÉTODO IMPLÍCITO
    allocate(ux_new(n+2))
    do i=1,1000
        do j=1,(t_write-1)
            ux_old(1)=ux_old(1)+2._dp*alpha*x_step_adim*(u0_Li-u1_Lf)
            ux_old(n+2)=ux_old(n+2)-2._dp*alpha*x_step_adim*(u0_Lf-u1_Lf)
            call implicit_method(n+2,diag,diag_sup,diag_inf,data_vector=ux_old(:),unknown_vector=ux_new(:))
            A_matrix(:,1)=ux_new(:);A_matrix=matmul(B_matrix,A_matrix);ux_old(:)=A_matrix(:,1)
        end do
        call implicit_method(n+2,diag,diag_sup,diag_inf,ux_old(:),ux_new(:))
        do j=1,n+2;write(10,20) t_write*i*t_step_adim*param_t,real(j-1,dp)*x_step_adim, ux_new(j);end do
        write(10,*) ''
    end do
    close(10);deallocate(diag,diag_sup,diag_inf,ux_old,ux_new,B_matrix,A_matrix)
end program heateq_crank_nicolson_von_neumann
```

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob02/code/heateq_comparison_01.f90

```
program heateq_comparison_01
    use module_precision
    implicit none
    real(dp),    parameter   :: u0_Li=0._dp,u0_Lf=0._dp                          ! condiciones de contorno von neumann
    real(dp),    parameter   :: t_step_adim=0.001_dp,x_step_adim=0.05_dp         ! deltaT y deltaX adimensionales
    real(dp),    parameter   :: t_write_adim=1._dp                              ! tiempo adimensional de evolución
    real(dp),    parameter   :: pi=4._dp*atan(1.0)
    real(dp),    parameter   :: D=237._dp*(1._dp/(900._dp*2700._dp))            ! thermal diffusivity(D=(K/C*rho)[L^2]/[t])
    real(dp),    parameter   :: param_x=1._dp                                   ! longitud caracteristica (long total barra en metros) [L]
```

```
    real(dp),    parameter   :: param_t=param_x*param_x*(1._dp/D)                  ! tiempo caracteristico (paso temporal en segundos) [t]
    real(dp),    parameter   :: alpha=t_step_adim*(1._dp/(x_step_adim*x_step_adim))
    integer(sp), parameter   :: t_write=int(t_write_adim*(1._dp/t_step_adim),sp)   ! pasos temporales de evolución
    integer(sp)              :: i,j,k,istat
    real(dp)                 :: T_exact                                           ! Temperatura exacta
    real(dp)                 :: err_new                                           ! errores absolutos
    integer(sp)              :: n,a,b                                             ! numero de elementos finitos y variables auxiliares
    real(dp),    allocatable :: ux_old(:),ux_new(:),err(:)
    20 format(E13.6,x,E13.6);21 format(E13.6,x,E13.6,x,E13.6)
    open(10,file='../results/result_02_explicit_vn.dat',status='replace',action='write',iostat=istat)
    write(*,*) 'istat(10file) = ',istat
    open(11,file='../results/result_02_exact_vn.dat',status='replace',action='write',iostat=istat)
    write(*,*) 'istat(11file) = ',istat
    open(12,file='../results/result_02_explicit_vn_err.dat',status='replace',action='write',iostat=istat)
    write(*,*) 'istat(12file) = ',istat
    ! verificamos condición de estabilidad debe ser menor a 1/2
    write(*,'(A20,E10.4)') 'alpha = ',alpha; write(*,'(A20,E10.4)') 'param_t = ',param_t
    ! calculamos la parte entera del modulo y del resto del inverso de x_step_adim
    a=int(mod(1._dp,x_step_adim),sp);b=int(1._dp/x_step_adim);n=(a+b-1_sp)*(1_sp/(1_sp-a))
    allocate(ux_old(n+2))
    ! inicializamos el vector espacial a tiempo inicial
    do i=1,n+2;ux_old(i)=cos(pi*real(i-1,dp)*x_step_adim);end do
    ! hacemos la evolución temporal (considerando los extremos)
    allocate(ux_new(n+2),err(t_write))
    err(:)=0._dp
    do i=1,(t_write-1)
        ux_new(1)=(1._dp-2._dp*alpha)*ux_old(1)+2._dp*alpha*(ux_old(2)-x_step_adim*u0_Li)
        ! imprimimos errores absolutos para todo t
        call exact_solution(real(i,dp)*t_step_adim,0._dp,T_exact)
        err_new=abs(T_exact-ux_new(1));if (err_new>err(i)) err(i)=err_new
        do j=2,(n+1)
            ux_new(j)=(1._dp-2._dp*alpha)*ux_old(j)+alpha*(ux_old(j+1)+ux_old(j-1))
            call exact_solution(real(i,dp)*t_step_adim,real(j-1,dp)*x_step_adim,T_exact)
            err_new=abs(T_exact-ux_new(j));if (err_new>err(i)) err(i)=err_new
        end do
        ux_new(n+2)=(1._dp-2._dp*alpha)*ux_old(n+2)+2._dp*alpha*(ux_old(n+1)+x_step_adim*u0_Lf)
        ux_old(1:n+2)=ux_new(1:n+2)
        call exact_solution(real(i,dp)*t_step_adim,real(n+1,dp)*x_step_adim,T_exact)
        err_new=abs(T_exact-ux_new(j));if (err_new>err(i)) err(i)=err_new
        write(12,21) real(i,dp)*t_step_adim,err(i)
    end do
    ! escribimos valores luego de t_write pasos temporales
    ux_new(1)=(1._dp-2._dp*alpha)*ux_old(1)+2._dp*alpha*(ux_old(2)-x_step_adim*u0_Li)
    call exact_solution(t_write_adim,0._dp,T_exact)
    err_new=abs(T_exact-ux_new(1));if (err_new>err(t_write)) err(t_write)=err_new
    write(10,21) 0._dp,ux_new(1),abs((T_exact-ux_new(1))*(1._dp/T_exact));write(11,20) 0._dp,T_exact
    do j=2,(n+1)
        ux_new(j)=(1._dp-2._dp*alpha)*ux_old(j)+alpha*(ux_old(j+1)+ux_old(j-1))
        call exact_solution(t_write_adim,real(j-1,dp)*x_step_adim,T_exact)
        err_new=abs(T_exact-ux_new(j));if (err_new>err(t_write)) err(t_write)=err_new
        write(10,21) real(j-1,dp)*x_step_adim,ux_new(j),abs((T_exact-ux_new(j))*(1._dp/T_exact))
        write(11,20) real(j-1,dp)*x_step_adim,T_exact
    end do
    ux_new(n+2)=(1._dp-2._dp*alpha)*ux_old(n+2)+2._dp*alpha*(ux_old(n+1)+x_step_adim*u0_Lf)
    call exact_solution(t_write_adim,real(n+1,dp)*x_step_adim,T_exact)
    err_new=abs(T_exact-ux_new(j));if (err_new>err(t_write)) err(t_write)=err_new
    write(10,21) real(n+1,dp)*x_step_adim,ux_new(n+2),abs((T_exact-ux_new(n+2))*(1._dp/T_exact))
    write(11,20) real(n+1,dp)*x_step_adim,T_exact;write(12,20) t_write_adim,err(t_write)
    close(10);deallocate(ux_old,ux_new)
end program heateq_comparison_01
subroutine exact_solution(t_adim,x_adim,T_exact)
    use module_precision
    implicit none
    real(dp), intent(in)  :: t_adim,x_adim
    real(dp), intent(out) ::  T_exact
    real(dp), parameter   :: pi=4._dp*atan(1.0)
    T_exact=exp(-pi*pi*t_adim)*cos(pi*x_adim)
end subroutine exact_solution
```

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob02/code/heateq_comparison_02.f90

```
program heateq_comparison_02
    use module_precision;use module_tridiag_matrix
    implicit none
    real(dp),    parameter   :: u0_Li=0._dp, u0_Lf=0._dp                          ! condiciones de borde (von neumann)
    real(dp),    parameter   :: t_step_adim=0.001_dp,x_step_adim=0.05_dp          ! deltaT y deltaX adimensionales
    real(dp),    parameter   :: t_write_adim=1._dp                                ! tiempo adimensional de evolución
    real(dp),    parameter    :: pi=4._dp*atan(1.0)
    real(dp),    parameter   :: D=237._dp*(1._dp/(900._dp*2700._dp))              ! thermal diffusivity(D=(K/C*rho)[L^2]/[t])
    real(dp),    parameter   :: param_x=1._dp                                     ! longitud caracteristica (long total barra en metros) [L]
    real(dp),    parameter   :: param_t=param_x*param_x*(1._dp/D)                  ! tiempo caracteristico (paso temporal en segundos) [t]
    real(dp),    parameter   :: alpha=t_step_adim*(1._dp/(x_step_adim*x_step_adim))
    integer(sp), parameter   :: t_write=int(t_write_adim*(1._dp/t_step_adim),sp)   ! pasos temporales de evolución
    integer(sp)              :: n,a,b                                             ! numero de elementos finitos y variables auxiliares
    integer(sp)              :: i,j,istat
    real(dp)                 :: T_exact                                           ! Temperatura exacta
    real(dp)                 :: err_new                                           ! errores absolutos
    real(dp),    allocatable :: ux_old(:),ux_new(:),diag(:),diag_sup(:),diag_inf(:),err(:)
    20 format(E13.6,x,E13.6,x,E13.6);21 format(E13.6,x,E13.6)
    open(10,file='../results/result_02_implicit_vn.dat',status='replace',action='write',iostat=istat)
    write(*,*) 'istat(10file) = ',istat
    open(11,file='../results/result_02_implicit_vn_err.dat',status='replace',action='write',iostat=istat)
    write(*,*) 'istat(11file) = ',istat
    ! calculamos la parte entera del modulo y del resto del inverso de x_step_adim
    a=int(mod(1._dp,x_step_adim),sp);b=int(1._dp/x_step_adim);n=(a+b-1_sp)*(1_sp/(1_sp-a))
    ! cargamos diagonales central, superior e inferior
    allocate(diag(n+2),diag_sup(n+2),diag_inf(n+2))
    diag(:)=1._dp+2._dp*alpha;diag_inf(:)=-alpha;diag_sup(:)=-alpha
```

```
    ! cambiamos los extremos de las diagonales iferior y superior
    diag_inf(n+2)=diag_inf(n+2)*2._dp;diag_sup(1)=diag_sup(1)*2._dp
    diag_sup(n+2)=0._dp;diag_inf(1)=0._dp
    ! cargamos datos iniciales de temperatura
    allocate(ux_old(n+2))
    do i=1,n+2;ux_old(i)=cos(pi*real(i-1,dp)*x_step_adim);end do
    ! aplicamos método implicito, hacemos la evolución temporal (considerando los extremos)
    allocate(ux_new(n+2),err(t_write))
    err(:)=0._dp
    do j=1,(t_write-1)
        ux_old(1)=ux_old(1)+2._dp*alpha*x_step_adim*u0_Li
        ux_old(n+2)=ux_old(n+2)-2._dp*alpha*x_step_adim*u0_Lf
        call implicit_method(n+2,diag,diag_sup,diag_inf,ux_old(:),ux_new(:))
        ux_old(:)=ux_new(:)
        ! imprimimos errores absolutos para todo t
        do i=1,n+2
            call exact_solution(real(j,dp)*t_step_adim,real(i-1,dp)*x_step_adim,T_exact)
            err_new=abs(T_exact-ux_new(i));if (err_new>err(j)) err(j)=err_new
        end do;write(11,21) real(j,dp)*t_step_adim,err(j)
    end do
    ! escribimos valores luego de t_write pasos temporales
    call implicit_method(n+2,diag,diag_sup,diag_inf,ux_old(:),ux_new(:))
    do j=1,n+2
        call exact_solution(t_write_adim,real(j-1,dp)*x_step_adim,T_exact)
        err_new=abs(T_exact-ux_new(j));if (err_new>err(t_write)) err(t_write)=err_new
        write(10,20) real(j-1,dp)*x_step_adim,ux_new(j),abs((T_exact-ux_new(j))*(1._dp/T_exact))
    end do
    write(11,21) real(t_write,dp)*t_step_adim,err(t_write)
    close(10);deallocate(diag,diag_sup,diag_inf,ux_old,ux_new)
end program heateq_comparison_02
subroutine exact_solution(t_adim,x_adim,T_exact)
    use module_precision
    implicit none
    real(dp), intent(in)  :: t_adim,x_adim
    real(dp), intent(out) ::  T_exact
    real(dp), parameter   :: pi=4._dp*atan(1.0)
    T_exact=exp(-pi*pi*t_adim)*cos(pi*x_adim)
end subroutine exact_solution
```

https://github.com/mendzmartin/fiscomp2022/blob/main/lab02/prob02/code/heateq_comparison_03.f90

```
program heateq_comparison_03
    use module_precision;use module_tridiag_matrix
    implicit none
    real(dp),    parameter   :: u0_Li=0._dp, u0_Lf=0._dp                          ! condiciones de borde de Von Neumann (t=t0)
    real(dp),    parameter   :: u1_Li=0._dp, u1_Lf=0._dp                          ! condiciones de borde de Von Neumann (t=t1)
    real(dp),    parameter   :: t_step_adim=0.001_dp,x_step_adim=0.05_dp          ! deltaT y deltaX adimensionales
    real(dp),    parameter   :: t_write_adim=1._dp                                ! tiempo adimensional de evolución
    real(dp),    parameter   :: pi=4._dp*atan(1.0)
    real(dp),    parameter   :: D=237._dp*(1._dp/(900._dp*2700._dp))              ! thermal diffusivity(D=(K/C*rho)[L^2]/[t])
    real(dp),    parameter   :: param_x=1._dp                                     ! longitud caracteristica (long total barra en metros) [L]
    real(dp),    parameter   :: param_t=param_x*param_x*(1._dp/D)                  ! tiempo caracteristico (paso temporal en segundos) [t]
    real(dp),    parameter   :: alpha=t_step_adim*(1._dp/(x_step_adim*x_step_adim))
    integer(sp), parameter   :: t_write=int(t_write_adim*(1._dp/t_step_adim),sp)  ! pasos temporales de evolución
    integer(sp)              :: n,a,b                                             ! numero de elementos finitos y variables auxiliares
    integer(sp)              :: i,j,istat
    real(dp)                 :: T_exact                                           ! Temperatura exacta
    real(dp)                 :: err_new                                           ! errores absolutos
    real(dp),    allocatable :: ux_old(:),ux_new(:),diag(:),diag_sup(:),diag_inf(:)
    real(dp),    allocatable :: B_matrix(:,:),A_matrix(:,:),err(:)
    20 format(E13.6,x,E13.6,x,E13.6);21 format(E13.6,x,E13.6)
    open(10,file='../results/result_02_cranknicolson_vn.dat',status='replace',action='write',iostat=istat)
    write(*,*) 'istat(10file) = ',istat
    open(11,file='../results/result_02_cranknicolson_vn_err.dat',status='replace',action='write',iostat=istat)
    write(*,*) 'istat(11file) = ',istat
    ! calculamos la parte entera del modulo y del resto del inverso de x_step_adim
    a=int(mod(1._dp,x_step_adim),sp);b=int(1._dp/x_step_adim);n=(a+b-1_sp)*(1_sp/(1_sp-a))
    ! CARGAMOS DIAGONALES CENTRAL, SUPERIOR E INFERIOR
    allocate(diag(n+2),diag_sup(n+2),diag_inf(n+2))
    diag(:)=2._dp*(1._dp/alpha+1._dp);diag_inf(:)=-1._dp;diag_sup(:)=-1._dp
    ! cambiamos los extremos de las diagonales iferior y superior
    diag_inf(n+2)=diag_inf(n+2)*2._dp;diag_sup(1)=diag_sup(1)*2._dp
    diag_sup(n+2)=0._dp;diag_inf(1)=0._dp
    ! CARGAMOS MATRIZ CUADRADA PARA USAR MÉTODO IMPLÍCITO
    allocate(B_matrix(n+2,n+2))
    B_matrix=0._dp ! elementos nulos fuera de la tribanda
    do i=1,n+2
        B_matrix(i,i)=2._dp*(1._dp/alpha-1._dp)  ! diagonal principal
        if (i/=1)   B_matrix(i,i-1)=1._dp         ! diagonal inferior
        if (i/=n+2) B_matrix(i,i+1)=1._dp       ! diagonal superior
    end do
    ! cambiamos los extremos de las diagonales iferior y superior de la matriz B
    B_matrix(n+2,n+1)=B_matrix(n+2,n+1)*2._dp;B_matrix(1,2)=B_matrix(1,2)*2._dp
    ! CARGAMOS DATOS INICIALES DE TEMPERATURA
    allocate(A_matrix(n+2,1)) ! matriz auxiliar p/matmul (column vector rank=2)
    allocate(ux_old(n+2))
    do i=1,n+2;ux_old(i)=cos(pi*real(i-1,dp)*x_step_adim);A_matrix(i,1)=ux_old(i);end do
    ! CARGAMOS DATOS INICIALES PARA APLCIAR MÉTODO IMPLÍCITO
    A_matrix=matmul(B_matrix,A_matrix);ux_old(:)=A_matrix(:,1)
    ! APLICAMOS MÉTODO IMPLÍCITO: hacemos la evolución temporal (considerando los extremos)
    allocate(ux_new(n+2),err(t_write))
    err(:)=0._dp
    do j=1,(t_write-1)
        ux_old(1)=ux_old(1)+2._dp*alpha*x_step_adim*u0_Li*(u0_Li-u1_Lf)
        ux_old(n+2)=ux_old(n+2)-2._dp*alpha*x_step_adim*(u0_Lf-u1_Lf)
        call implicit_method(n+2,diag,diag_sup,diag_inf,data_vector=ux_old(:),unknown_vector=ux_new(:))
        A_matrix(:,1)=ux_new(:);A_matrix=matmul(B_matrix,A_matrix);ux_old(:)=A_matrix(:,1)
        do i=1,n+2
```

```
            call exact_solution(real(j,dp)*t_step_adim,real(i-1,dp)*x_step_adim,T_exact)
            err_new=abs(T_exact-ux_new(i));if (err_new>err(j)) err(j)=err_new
        end do;write(11,21) real(j,dp)*t_step_adim,err(j)
    end do
    ! escribimos valores luego de t_write pasos temporales
    call implicit_method(n+2,diag,diag_sup,diag_inf,ux_old(:),ux_new(:))
    do j=1,n+2
        call exact_solution(t_write_adim,real(j-1,dp)*x_step_adim,T_exact)
        err_new=abs(T_exact-ux_new(j));if (err_new>err(t_write)) err(t_write)=err_new
        write(10,20) real(j-1,dp)*x_step_adim,ux_new(j),abs((T_exact-ux_new(j))*(1._dp/T_exact))
    end do
    write(11,21) real(t_write,dp)*t_step_adim,err(t_write)

    close(10);deallocate(diag,diag_sup,diag_inf,ux_old,ux_new,B_matrix,A_matrix)
end program heateq_comparison_03
subroutine exact_solution(t_adim,x_adim,T_exact)
    use module_precision
    implicit none
    real(dp), intent(in)  :: t_adim,x_adim
    real(dp), intent(out) ::  T_exact
    real(dp), parameter   :: pi=4._dp*atan(1.0)
    T_exact=exp(-pi*pi*t_adim)*cos(pi*x_adim)
end subroutine exact_solution
```

Módulos

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

```
module module_precision
  implicit none
  integer, parameter :: sp = selected_real_kind(p=6,r=37)    ! simple presicion (sp) class
  integer, parameter :: dp = selected_real_kind(p=15,r=307)  ! double presicion (dp) class
  integer, parameter :: qp = selected_real_kind(p=33,r=4931)  ! quad presicion (dp) class
end module module_precision
!-------------------------------------------------------------
! REFERENCES
!-------------------------------------------------------------
! https://fortranwiki.org/fortran/show/Real+precision
!-------------------------------------------------------------
```

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

```
module module_tridiag_matrix
    use module_precision
    implicit none
    contains
    subroutine implicit_method(n,diag,diag_sup,diag_inf,data_vector,unknown_vector)
        ! variables de entrada/salida
        integer(sp),    intent(in)  :: n                              ! dimension del vector diagonal
        real(dp),       intent(out) :: unknown_vector(n)              ! vector de incognitas
        real(dp),       intent(in)  :: diag(n),diag_sup(n),diag_inf(n) ! diagonales central, superior e inferior
        real(dp),       intent(in)  :: data_vector(n)                 ! vector de datos conocidos
        ! variables locales
        integer(sp) :: i                    ! variable del loop
        real(dp)    :: diag_sup_new(n)      ! diagonal superior redefinida
        real(dp)    :: data_vector_new(n)   ! vector de datos redefinidos
        real(dp)    :: factor
        ! descomposición LU
        factor=1._dp/diag(1)
        diag_sup_new(1)=diag_sup(1)*factor
        data_vector_new(1)=data_vector(1)*factor
        do i=2,n
            factor=1._dp/(diag(i)-diag_inf(i)*diag_sup_new(i-1))
            diag_sup_new(i)=diag_sup(i)*factor
            data_vector_new(i)=(data_vector(i)-diag_inf(i)*data_vector_new(i-1))*factor
        end do
        ! sustitución hacia atrás
        unknown_vector(n)=data_vector_new(n)
        do i=(n-1),1,-1
            unknown_vector(i)=data_vector_new(i)-diag_sup_new(i)*unknown_vector(i+1)
        end do
    end subroutine implicit_method
end module module_tridiag_matrix
! Los vectores de entrada deben definirse de la siguiente manera
! diag_sup = [ ds(1)   ds(2) ... ds(n-1) ds(n)=0 ]
! diag_inf = [ di(1)=0 di(2) ... di(n-1) di(n)   ]
! diag     = [ d(1)    d(2)  ... di(n-1) d(n)    ]
```