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

Problema 1

Ecuación de Calor

Resolvemos, mediante diferentes algoritmos, la ecuación del calor

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

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 $100[°C]$, y se colocan sus extremos a $0[°C]$. El calor fluye sólo por los extremos. La conductividad térmica (κ_T), el calor específico (C) y la densidad (ρ) del aluminio son

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

Para ello:

a) Adimensionalizar la ecuación para llevarla a la forma

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

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:

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

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

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

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)

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

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

$$[potencia] = \left[\frac{trabajo}{tiempo}\right] = \left[\frac{fuerza \cdot distancia}{tiempo}\right] = \left[\frac{masa \cdot aceleración \cdot distancia}{tiempo}\right]$$

$$\Rightarrow [potencia] = \left[\frac{masa \cdot (distancia)^2}{(tiempo)^3}\right] \equiv \left[\frac{M \cdot L^2}{t^3}\right] \therefore [\kappa_T] \equiv \left[\frac{M \cdot L}{t^3 \cdot T}\right]$$

y por otro lado tendremos,

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

luego

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

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

$$\alpha\tilde{x} = x \wedge \beta\tilde{t} = t \tag{7}$$

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)

$$\frac{\partial}{\partial t} = \frac{\partial}{\partial \tilde{t}}\underbrace{\frac{\partial \tilde{t}}{\partial t}}_{=1/\beta} = \frac{1}{\beta}\frac{\partial}{\partial \tilde{t}};\ \frac{\partial^2}{\partial x^2} = \left[\frac{\partial^2}{\partial \tilde{x}^2}\underbrace{\left(\frac{\partial \tilde{x}}{\partial x}\right)^2}_{=1/\alpha} + \frac{\partial}{\partial \tilde{x}}\underbrace{\frac{\partial^2 \tilde{x}}{\partial x^2}}_{=0}\right] = \frac{1}{\alpha^2}\frac{\partial^2}{\partial \tilde{x}^2}$$

$$\Rightarrow \frac{\partial T}{\partial \tilde{t}} = \frac{\beta D}{\alpha^2}\frac{\partial^2[T(\tilde{x},\tilde{t})]}{\partial^2 \tilde{x}} \tag{8}$$

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

$$\frac{\beta D}{\alpha^2} = 1 \Rightarrow \beta = \frac{\alpha^2}{D} = \frac{L^2}{D} \tag{9}$$

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

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 $8[\text{bytes}] \equiv 64[\text{bits}]$) que utiliza 26 dígitos para su representación. En primera instancia, si no imprimimos los datos con formato tamaño del archivo sería $\text{tamaño} = (\#\text{filas}) \cdot \left(\frac{\#\text{valores}}{\text{fila}}\right) \cdot \left(\frac{\text{digitos}}{\text{valor}}\right)[\text{bytes}]$, pero debemos tener en cuenta también que el sistema operativo guarda en memoria el indice de fila y columna del archivo de datos y por cada indice 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 indices 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

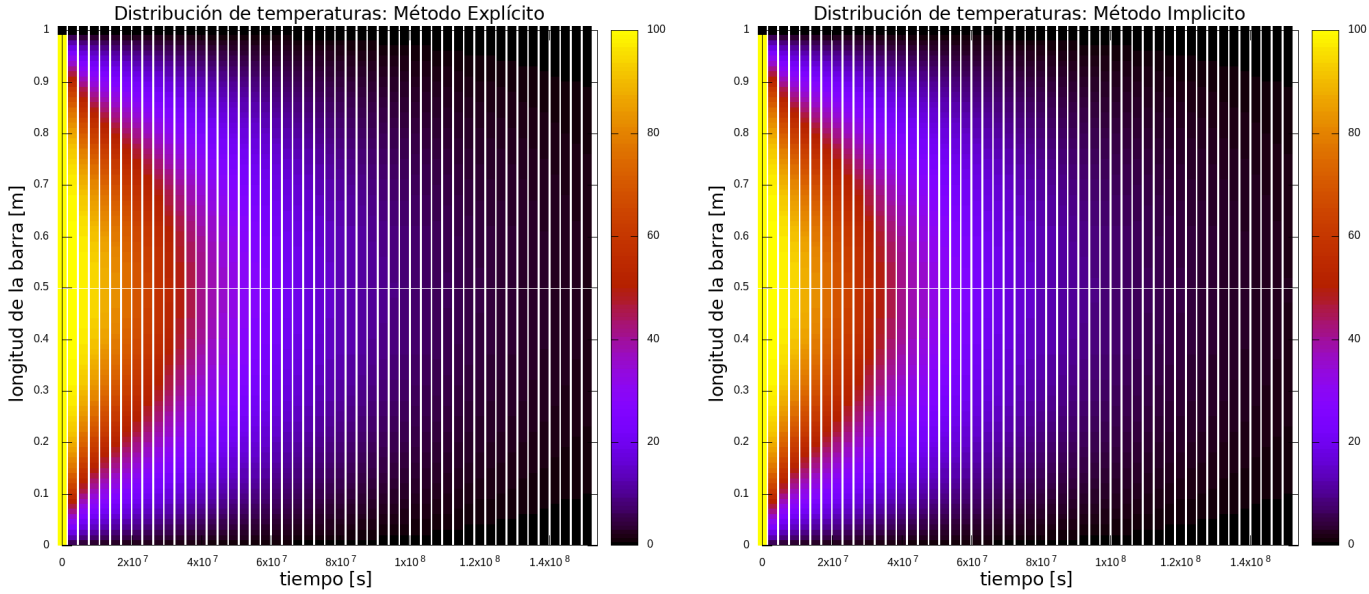


Figure 1:

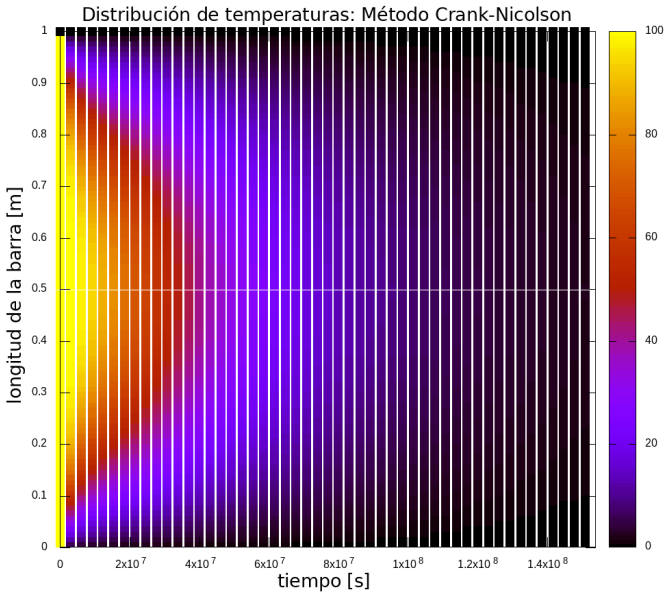


Figure 2:

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
```

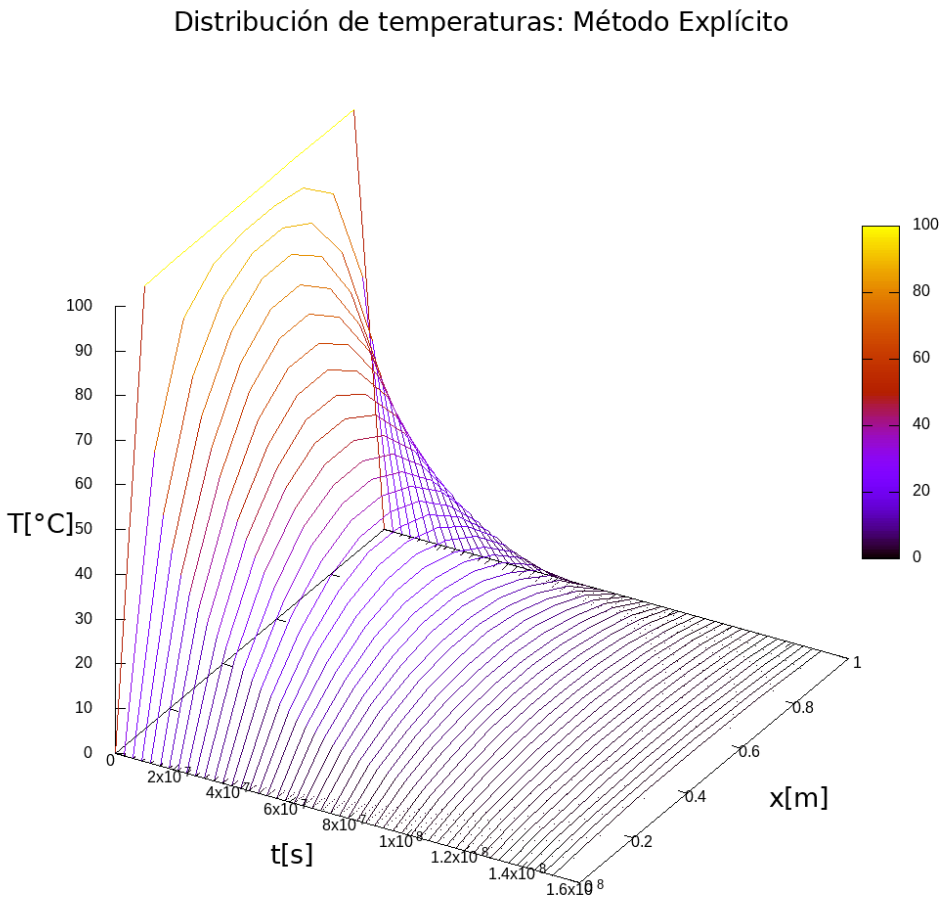
También, se probó con valores mayores a $^1/_2$ produciéndose errores numéricos (propagación de NaN) lo cual evidencia alguna división por cero o algún error aritmético similar.

```
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



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 $0[°C]$ y el resto del cuerpo de la barra a $100[°C]$, 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 $0[°C]$.

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

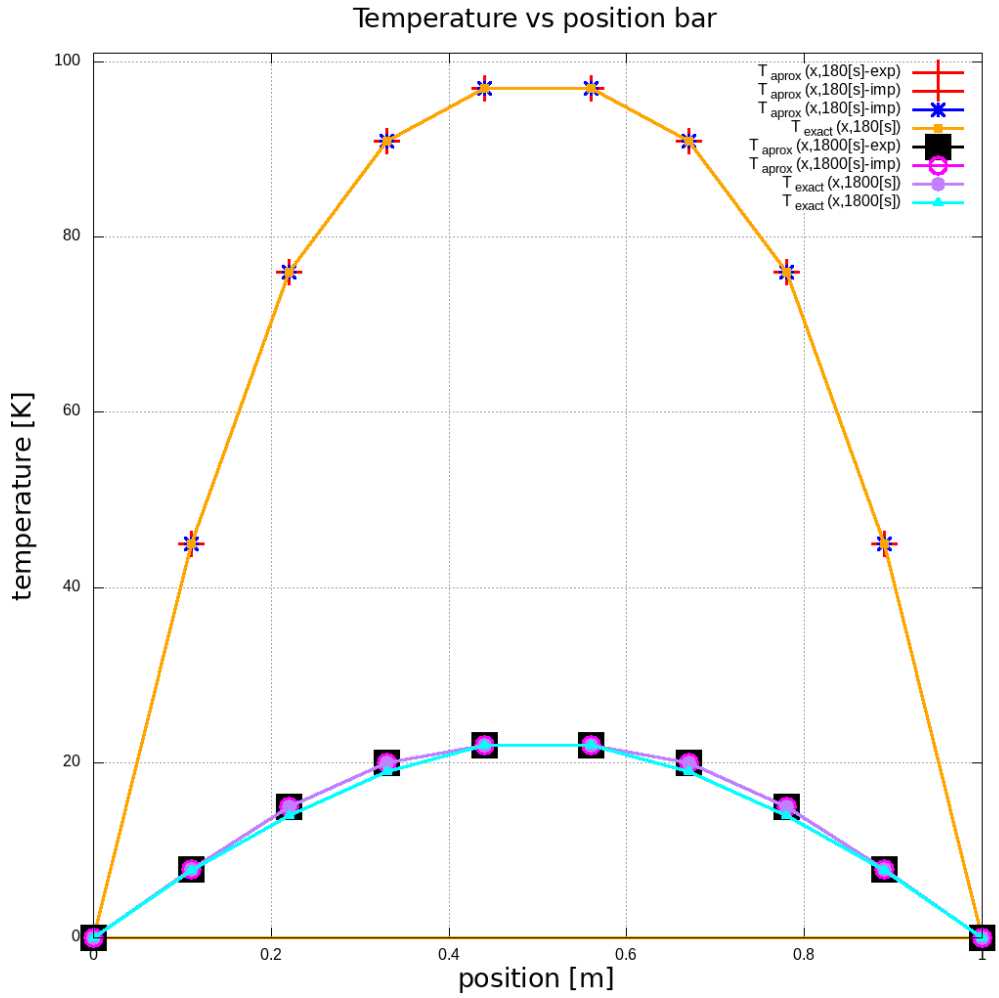


Figure 3:

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 cache 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 compilaron 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_02.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., 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/prob01/code/heateq_explicit_method.f90

```
program heateq_explicit_method
    use module_precision
    implicit none
    real(dp),    allocatable :: ux_old(:),ux_new(:)
    integer(sp), parameter   :: n=98_sp                          ! numero de elementos finitos
    real(dp),    parameter   :: u0_Li=0._dp,u0_Lf=0._dp          ! condiciones de borde
    real(dp)                 :: u0_ti=100._dp                    ! condicion inicial
    real(dp)                 :: param_t,param_x
    real(dp)                 :: D
    real(dp)                 :: alpha
    real(dp)                 :: t_step_adim,x_step_adim
    integer(sp)              :: i,j,k,istat
    integer(sp), parameter   :: t_write=300_sp                   ! pasos temporales entre escrituras
    real(dp)                 :: factor
    20 format(E8.2,x,E8.2)
    open(10,file='../results/result_01_explicit.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) [L]
    write(*,'(A20,E10.4)') 'param_x = ',param_x
    param_t=param_x*param_x*(1._dp/D)             ! tiempo caracteristico (paso temporal) [t]
    write(*,'(A20,E10.4)') 'param_t = ',param_t
    t_step_adim=0.3_dp*(1._dp/param_t)
    write(*,*) t_step_adim
    x_step_adim=1._dp/(real(n,dp)+1._dp)
    write(*,*) x_step_adim
    alpha=t_step_adim*(1._dp/(x_step_adim*x_step_adim))
    ! verificamos condición de estabilidad debe ser menor a 1/2
    write(*,'(A20,E10.4)') 'alpha = ',alpha
    allocate(ux_old(n+2))
    ! inicializamos el vector espacial a tiempo inicial
    ux_old(1)=u0_Li
    write(10,20) 0._dp,ux_old(1)
    do i=2,n+1
        ux_old(i)=u0_ti
        write(10,20) real(i-1,dp)*x_step_adim,ux_old(i)
    end do
    ux_old(n+2)=u0_Lf
    write(10,20) real(n+1,dp)*x_step_adim,ux_old(n+2)
    allocate(ux_new(n+2))
    ux_new(1)=ux_old(1)
    ux_new(n+2)=ux_old(n+2)
    ! hacemos la evolución temporal (sin considerar los extremos)
    ! asignamos valores al vector espacial a tiempo t
    do k=1,50 ! evolucionamos (50*t_write) pasos temporales
        do i=1_sp,(t_write-1_sp)
            do j=2_sp,(n+1_sp)
                ux_new(j)=(1._dp-2._dp*alpha)*ux_old(j)+alpha*(ux_old(j+1)+ux_old(j-1))
            end do
            ux_old(2:n+1)=ux_new(2:n+1)
        end do
        ! escribimos valores luego de t_write pasos temporales
        write(10,20) 0._dp,ux_new(1)
        do j=2_sp,(n+1_sp)
```

```
            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
        write(10,20) real(n+1,dp)*x_step_adim,ux_new(n+2)
    end do
    close(10)
    ! controlar termalización en el centro de la barra
    write(*,'(A20,E10.4)') 'T(L/2,t_final) = ', ux_new(50)
    ! estimamos tamaño de archivo
    factor=10._dp ! este factor depende de (20 format (2(E10.2))), en este ejemplo factor es 10
    write(*,'(A30,E10.4,A10)') 'Tamaño aprox. de archivo = ', 2._dp*real(n,dp)*51._dp*factor, '[bytes]'
    deallocate(ux_old,ux_new)
end program heateq_explicit_method
```

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

```
program heateq_implicit_method
    use module_precision
    use module_tridiag_matrix
    implicit none
    integer(sp), parameter   :: n=98_sp                  ! numero de elementos finitos
    real(dp),    parameter   :: u0_Li=0._dp, u0_Lf=0._dp ! condiciones de borde
    real(dp),    parameter   :: u0_ti=100._dp            ! condicion inicial
    integer(sp), parameter   :: t_write=300_sp           ! pasos temporales entre escrituras
    real(dp),    allocatable :: ux_old(:),ux_new(:),diag(:),diag_sup(:),diag_inf(:)
    real(dp)                 :: param_t,param_x
    real(dp)                 :: D                        ! thermal diffusivity
    real(dp)                 :: alpha
    real(dp)                 :: t_step_adim,x_step_adim
    integer(sp)              :: i,j,k,istat
    20 format(E8.2,x,E8.2)
    open(10,file='../results/result_01_implicit.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) [L]
    param_t=param_x*param_x*(1._dp/D)          ! tiempo caracteristico (paso temporal) [t]
    t_step_adim=0.3_dp*(1._dp/param_t)
    x_step_adim=1._dp/(real(n,dp)+1._dp)
    alpha=t_step_adim*(1._dp/(x_step_adim*x_step_adim))
    ! cargamos diagonales central, superior e inferior
    allocate(diag(n),diag_sup(n),diag_inf(n))
    diag_sup(n)=0._dp
    diag_inf(1)=0._dp
    do i=1,n
        diag(i)=1._dp+2._dp*alpha
        if (i/=1) diag_inf(i)=-alpha
        if (i/=n) diag_sup(i)=-alpha
    end do
    ! cargamos datos iniciales de temperatura
    allocate(ux_old(n+2))
    ux_old(1)=u0_Li
    write(10,20) 0._dp,ux_old(1)
    do i=2,n+1
        ux_old(i)=u0_ti
        write(10,20) real(i-1,dp)*x_step_adim,ux_old(i)
    end do
    ux_old(n+2)=u0_Lf
    write(10,20) real(n+1,dp)*x_step_adim,ux_old(n+2)
    ! aplicamos método implicito
    allocate(ux_new(n+2))
    ux_new(1)=ux_old(1)
    ux_new(n+2)=ux_old(n+2)
    do i=1,50
        do j=1,(t_write-1)
            call implicit_method(n,diag,diag_sup,diag_inf,ux_old(2:n+1),ux_new(2:n+1))
            ux_old(2:n+1)=ux_new(2:n+1)
        end do
        call implicit_method(n,diag,diag_sup,diag_inf,ux_old(2:n+1),ux_new(2:n+1))
        do j=1,n+2
            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/heateq_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 APLCIAR 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
```

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

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

```
! Comparación entre el método explícito y la solución exacta
program heateq_comparison_01
    use module_precision
    implicit none
    real(dp), allocatable   :: ux_old(:),ux_new(:)
    integer(dp), parameter  :: n=8_dp                   ! numero de elementos finitos
    real(dp),    parameter  :: u0_Li=0._dp,u0_Lf=0._dp  ! condiciones de borde
    real(dp)                :: u0_ti=100._dp            ! condicion inicial
    real(dp)                :: param_t,param_x          ! parametros para adimensionalizar (espacial y temporal)
    real(dp),    parameter  :: D=237._dp*(1._dp/(900._dp*2700._dp))               ! thermal diffusivity(D=(K/C*rho)[L^2]/[t])
    real(dp), parameter     :: alpha=1.E-7_dp           ! se asegura estabilidad (alpha < 0.5)
    real(dp)                :: T_exact                  ! temperatura exacta
    real(dp)                :: t_step_adim,x_step_adim  ! variables adimensionales (espacial y temporal)
    integer(dp)             :: i,j,k,istat              ! variables de loops
    real(dp), parameter     :: t_write_1_noadm=180._dp,t_write_2_noadm=1800._dp ! tiempos en dimensiones para escrituras
    integer(dp)             :: t_write_1,t_write_2      ! pasos temporales entre escrituras
    ! parametros para adimensionalizar
    param_x=1._dp                       ! longitud caracteristica (long total barra en metros) [L]
    param_t=param_x*param_x*(1._dp/D)    ! tiempo caracteristico [t]
    x_step_adim=1._dp/(real(n,dp)+1._dp)
    t_step_adim=alpha*x_step_adim*x_step_adim
    allocate(ux_old(n+2))
    ! inicializamos el vector espacial a tiempo inicial
    ux_old(1)=u0_Li
    ux_old(2:n+1)=u0_ti
    ux_old(n+2)=u0_Lf
    allocate(ux_new(n+2))
    ! ADIMENSIONALIZAMOS LOS TIEMPOS DE ESCRITURA
    t_write_1=int(t_write_1_noadm*(1._dp/(param_t*t_step_adim)),dp)
    t_write_2=int(t_write_2_noadm*(1._dp/(param_t*t_step_adim)),dp)
    write(*,*) t_write_2
    ! HACEMOS LA EVOLUCIÓN TEMPORAL (SIN CONSIDERAR LOS EXTREMOS)
    ux_new(1)=u0_Li;ux_new(n+2)=u0_Lf
    do i=1,(t_write_1-1)
        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))
            ux_old(j)=ux_new(j)
        end do
    end do
    ! ESCRIBIMOS VALORES LUEGO DE t_write_1 PASOS TEMPORALES
    !20 format (E9.2,x,E9.2)
    !open(10,file='../results/result_02_aprox_explicit.dat',status='replace',action='write',iostat=istat)
    !write(*,*) 'Input/Output file. istat10 = ',istat
    !open(11,file='../results/result_02_exact.dat',status='replace',action='write',iostat=istat)
    !write(*,*) 'Input/Output file. istat11 = ',istat
    !call exact_solution(0._dp,t_write_1_noadm*(1._dp/param_t),u0_Li,T_exact)
    call exact_solution(0._dp,t_write_1_noadm,D,u0_Li,T_exact)
    !write(10,20) 0._dp,ux_new(1)
    !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))
        ux_old(j)=ux_new(j)
        !call exact_solution(real(j-1,dp)*x_step_adim,t_write_1_noadm*(1._dp/param_t),u0_ti,T_exact)
        call exact_solution(real(j-1,dp)*x_step_adim,t_write_1_noadm,D,u0_ti,T_exact)
        !write(10,20) real(j-1,dp)*x_step_adim,ux_new(j)
        !write(11,20) real(j-1,dp)*x_step_adim,T_exact
    end do
    !call exact_solution(real(n+1,dp)*x_step_adim,t_write_1_noadm*(1._dp/param_t),u0_Lf,T_exact)
    call exact_solution(real(n+1,dp)*x_step_adim,t_write_1_noadm,D,u0_Lf,T_exact)
    !write(10,20) real(n+1,dp)*x_step_adim,ux_new(n+2)
    !write(11,20) real(n+1,dp)*x_step_adim,T_exact
    do i=t_write_1,(t_write_2-1)
        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))
            ux_old(j)=ux_new(j)
        end do
    end do
    ! ESCRIBIMOS VALORES LUEGO DE t_write_2 PASOS TEMPORALES
    !call exact_solution(0._dp,t_write_2_noadm*(1._dp/param_t),u0_Li,T_exact)
    call exact_solution(0._dp,t_write_2_noadm,D,u0_Li,T_exact)
    !write(10,20) 0._dp,ux_new(1)
```

```
    !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))
        ux_old(j)=ux_new(j)
        !call exact_solution(real(j-1,dp)*x_step_adim,t_write_2_noadm*(1._dp/param_t),u0_ti,T_exact)
        call exact_solution(real(j-1,dp)*x_step_adim,t_write_2_noadm,D,u0_ti,T_exact)
        !write(10,20) real(j-1,dp)*x_step_adim,ux_new(j)
        !write(11,20) real(j-1,dp)*x_step_adim,T_exact
    end do
    !call exact_solution(real(n+1,dp)*x_step_adim,t_write_2_noadm*(1._dp/param_t),u0_Lf,T_exact)
    call exact_solution(real(n+1,dp)*x_step_adim,t_write_2_noadm,D,u0_Lf,T_exact)
    !write(10,20) real(n+1,dp)*x_step_adim,ux_new(n+2)
    !write(11,20) real(n+1,dp)*x_step_adim,T_exact
    !close(10)
    !close(11)
    deallocate(ux_old,ux_new)
end program heateq_comparison_01
subroutine exact_solution(x,t,D,Temp0,Temp)
    use module_precision
    implicit none
    ! variables de entrada/salida
    real(dp), intent(in)    :: x,t,Temp0,D ! ojo, x y t no son adimensionales
    real(dp), intent(out)   :: Temp
    ! variables locales
    real(dp),    parameter  :: pi=4._dp*atan(1._dp)
    integer(dp), parameter  :: n_max=81_dp ! debe ser impar
    real(dp)                :: kn
    integer(dp)             :: n
    Temp=0._dp
    do n=1,n_max,2 ! recorro numeros impares
        kn=real(n,dp)*pi
        Temp=Temp+(1._dp/real(n,dp))*sin(kn*x)*exp(-(kn*kn*t*D))
    end do
    Temp=4._dp*Temp0*(1._dp/pi)*Temp
end subroutine exact_solution
```

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

```
! Comparación entre el método implícito y la solución exacta
program heateq_comparison_02
    use module_precision
    use module_tridiag_matrix
    implicit none
    real(dp), allocatable   :: ux_old(:),ux_new(:)      ! vectores de temperatura (paso anterior y paso posterior)
    real(dp), allocatable   :: diag(:),diag_sup(:),diag_inf(:) ! diagonales de la matriz tridiagonal
    integer(sp), parameter  :: n=8_sp                   ! numero de elementos finitos dimension espacial
    real(dp), parameter     :: u0_Li=0._dp, u0_Lf=0._dp ! condiciones de borde
    real(dp)                :: u0_ti=100._dp            ! condicion inicial
    real(dp)                :: param_t                  ! tiempo caracteristico (paso temporal) [t]
    real(dp)                :: param_x                  ! longitud caracteristica (long total barra) [L]
    real(dp)                :: D                        ! thermal diffusivity (D=K/C*rho [L^2]/[t])
    real(dp)                :: alpha=1.E-7_dp           ! coeficiente discretización
    real(dp)                :: t_step_adim, x_step_adim ! deltatiempo y deltaespacio (adimensionales)
    integer(sp)             :: i,j,k,istat              ! variables de loop y variable de control
    real(dp), parameter     :: t_write_1_noadm=180._dp,t_write_2_noadm=1800._dp ! tiempos en dimensiones para escrituras
    integer(dp)             :: t_write_1,t_write_2      ! pasos temporales entre escrituras
    ! 20 format (E8.2,x,E8.2)
    ! open( 10, file = '../results/result_02_aprox_implicit.dat', status = 'replace', action = 'write', iostat = istat )
    ! ANÁLISIS DIMENSIONAL
    D=237._dp*(1._dp/(900._dp*2700._dp))
    param_x = 1._dp
    param_t = param_x*param_x*(1._dp/D)
    x_step_adim=1._dp/(real(n,dp)+1._dp)
    t_step_adim=alpha*x_step_adim*x_step_adim
    ! CARGAMOS DIAGONALES CENTRAL, SUPERIOR E INFERIOR
    allocate(diag(n),diag_sup(n),diag_inf(n))
    diag_sup(n)=0._dp
    diag_inf(1)=0._dp
    do i=1,n
        diag(i)=1._dp+2._dp*alpha
        if (i/=1) diag_inf(i) = -alpha
        if (i/=n) diag_sup(i) = -alpha
    end do
    ! CARGAMOS DATOS INICIALES DE TEMPERATURA
    allocate(ux_old(n+2))
    ux_old(1)=u0_Li
    ux_old(2:n+1)=u0_ti
    ux_old(n+2)=u0_Lf
    ! ADIMENSIONALIZAMOS LOS TIEMPOS DE ESCRITURA
    t_write_1=int(t_write_1_noadm*(1._dp/(param_t*t_step_adim)),dp)
    t_write_2=int(t_write_2_noadm*(1._dp/(param_t*t_step_adim)),dp)
    ! APLICAMOS MÉTODO IMPLÍCITO
    allocate(ux_new(n+2))
    ux_new(1)=ux_old(1)
    ux_new(n+2)=ux_old(n+2)
    do j=1,(t_write_1-1)
```

```
        call implicit_method(n,diag,diag_sup,diag_inf,ux_old(2:n+1),ux_new(2:n+1))
        do k=2,(n+1); ux_old(k)=ux_new(k); end do
    end do
    ! ESCRIBIMOS VALORES LUEGO DE t_write_1 PASOS TEMPORALES
    call implicit_method(n,diag,diag_sup,diag_inf,ux_old(2:n+1),ux_new(2:n+1))
    !do j=1,n+2; write(10,20) real(j-1,dp)*x_step_adim, ux_new(j); end do
    do j=t_write_1,(t_write_2-1)
        call implicit_method(n,diag,diag_sup,diag_inf,ux_old(2:n+1),ux_new(2:n+1))
        do k=2,(n+1); ux_old(k)=ux_new(k); end do
    end do
    ! ESCRIBIMOS VALORES LUEGO DE t_write_2 PASOS TEMPORALES
    call implicit_method(n,diag,diag_sup,diag_inf,ux_old(2:n+1),ux_new(2:n+1))
    !do j=1,n+2; write(10,20) real(j-1,dp)*x_step_adim, ux_new(j); end do
    !close(10)
    deallocate(diag,diag_sup,diag_inf,ux_old,ux_new)
end program heateq_comparison_02
```

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

```
program heateq_comparison_03
    use module_precision
    use module_tridiag_matrix
    implicit none
    real(dp), allocatable   :: ux_old(:),ux_new(:)      ! vectores de temperatura (paso anterior y paso posterior)
    real(dp), allocatable   :: diag(:),diag_sup(:),diag_inf(:) ! diagonales de la matriz tridiagonal
    real(dp),    allocatable :: B_matrix(:,:),A_matrix(:,:)
    integer(sp), parameter  :: n=8_sp                   ! numero de elementos finitos dimension espacial
    real(dp), parameter     :: u0_Li=0._dp, u0_Lf=0._dp ! condiciones de borde
    real(dp)                :: u0_ti=100._dp            ! condicion inicial
    real(dp)                :: param_t                  ! tiempo caracteristico (paso temporal) [t]
    real(dp)                :: param_x                  ! longitud caracteristica (long total barra) [L]
    real(dp)                :: D                        ! thermal diffusivity (D=K/C*rho [L^2]/[t])
    real(dp)                :: alpha=1.E-7_dp           ! coeficiente discretización
    real(dp)                :: t_step_adim, x_step_adim ! deltatiempo y deltaespacio (adimensionales)
    integer(sp)             :: i,istat              ! variables de loop y variable de control
    real(dp), parameter     :: t_write_1_noadm=180._dp,t_write_2_noadm=1800._dp ! tiempos en dimensiones para escrituras
    integer(dp)             :: t_write_1,t_write_2      ! pasos temporales entre escrituras
    ! 20 format (E8.2,x,E8.2)
    ! open( 10, file = '../results/result_02_aprox_cranknicolson.dat', status = 'replace', action = 'write', iostat = istat )
    ! ANÁLISIS DIMENSIONAL
    D=237._dp*(1._dp/(900._dp*2700._dp))
    param_x = 1._dp
    param_t = param_x*param_x*(1._dp/D)
    x_step_adim=1._dp/(real(n,dp)+1._dp)
    t_step_adim=alpha*x_step_adim*x_step_adim
    ! ADIMENSIONALIZAMOS LOS TIEMPOS DE ESCRITURA
    t_write_1=int(t_write_1_noadm*(1._dp/(param_t*t_step_adim)),dp)
    t_write_2=int(t_write_2_noadm*(1._dp/(param_t*t_step_adim)),dp)
    ! CARGAMOS DIAGONALES CENTRAL, SUPERIOR E INFERIOR
    allocate(diag(n),diag_sup(n),diag_inf(n))
    diag_sup(n)=0._dp
    diag_inf(1)=0._dp
    do i=1,n
        diag(i)=2._dp*(1._dp/alpha+1._dp)
        if (i/=1) diag_inf(i)=-1._dp
        if (i/=n) diag_sup(i)=-1._dp
    end do
    ! CARGAMOS MATRIZ CUADRADA PARA USAR MÉTODO IMPLÍCITO
    allocate(B_matrix(n,n))
    B_matrix=0._dp ! elementos nulos fuera de la tribanda
    do i=1,n
        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) B_matrix(i,i+1)=1._dp         ! diagonal superior
    end do
    ! CARGAMOS DATOS INICIALES DE TEMPERATURA
    allocate(A_matrix(n,1)) ! matriz auxiliar p/matmul (column vector rank=2)
    allocate(ux_old(n+2))
    ux_old(1)=u0_Li
    A_matrix(1:n,1)=u0_ti
    ux_old(n+2)=u0_Lf
    !CARGAMOS DATOS INICIALES PARA APLCIAR MÉTODO IMPLÍCITO
    A_matrix=matmul(B_matrix,A_matrix)
    ux_old(2:n+1)=A_matrix(1:n,1)
    ! APLICAMOS MÉTODO IMPLÍCITO
    allocate(ux_new(n+2))
    ux_new(1)=ux_old(1)
    ux_new(n+2)=ux_old(n+2)
    do i=1,(t_write_1-1)
        call implicit_method(n,diag,diag_sup,diag_inf,data_vector=ux_old(2:n+1),unknown_vector=ux_new(2:n+1))
        A_matrix(1:n,1)=ux_new(2:n+1)
        A_matrix=matmul(B_matrix,A_matrix)
        ux_old(2:n+1)=A_matrix(1:n,1)
    end do
    ! ESCRIBIMOS VALORES LUEGO DE t_write_1 PASOS TEMPORALES
    call implicit_method(n,diag,diag_sup,diag_inf,ux_old(2:n+1),ux_new(2:n+1))
```

```
    ! do i=1,n+2
    !     write(10,20) real(i-1,dp)*x_step_adim, ux_new(i)
    ! end do
    do i=t_write_1,(t_write_2-1)
        call implicit_method(n,diag,diag_sup,diag_inf,data_vector=ux_old(2:n+1),unknown_vector=ux_new(2:n+1))
        A_matrix(1:n,1)=ux_new(2:n+1)
        A_matrix=matmul(B_matrix,A_matrix)
        ux_old(2:n+1)=A_matrix(1:n,1)
    end do
    ! ESCRIBIMOS VALORES LUEGO DE t_write_2 PASOS TEMPORALES
    call implicit_method(n,diag,diag_sup,diag_inf,ux_old(2:n+1),ux_new(2:n+1))
    ! do i=1,n+2
    !     write(10,20) real(i-1,dp)*x_step_adim, ux_new(i)
    ! end do
    !close(10)
    deallocate(diag,diag_sup,diag_inf,ux_old,ux_new)
    deallocate(B_matrix,A_matrix)
end program heateq_comparison_03
```

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)    ]
```

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)

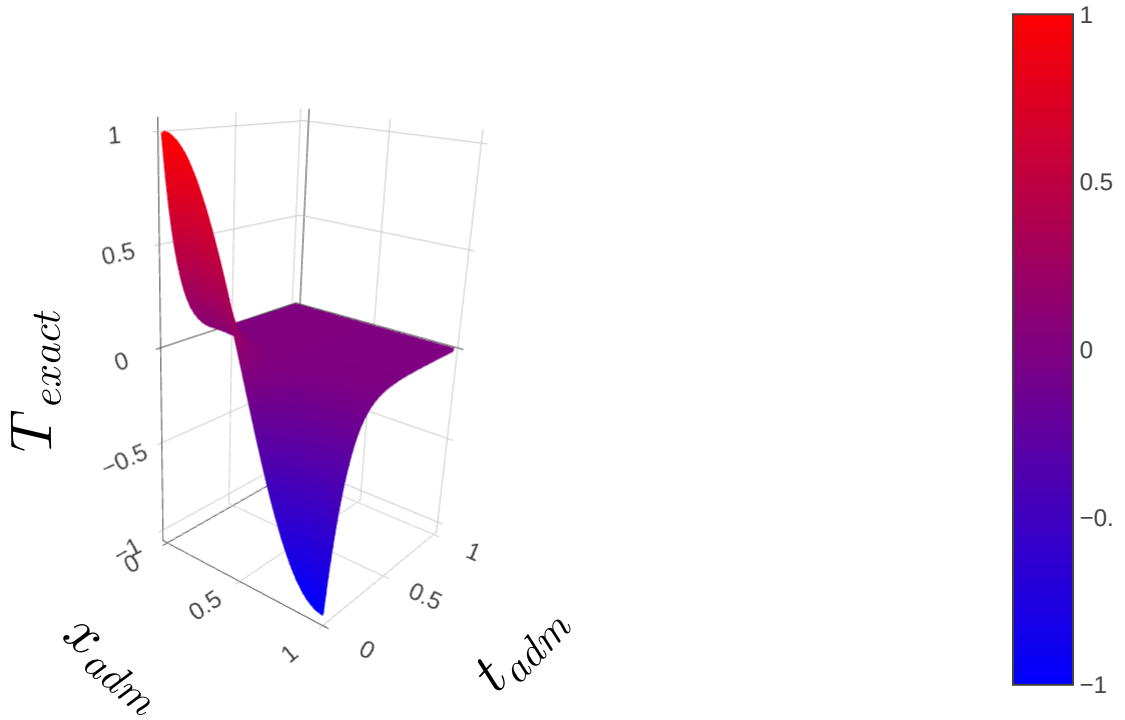
$$T_{1,(j+1)} = -2\eta T_{1,j} + (1-2\eta)T_{0,j} + 2\eta\Delta x g_0; i = 1$$
$$T_{i,(j+1)} = (1+2\eta)T_{i,j} - \eta[T_{(i-1),j} + T_{(i+1),j}] \ \forall \ 2 \leqslant i \leqslant (n+1)$$
$$T_{n+2,(j+1)} = -2\eta T_{(n+2),j} + (1+2\eta)T_{(n+2),j} - 2\eta\Delta x g_{(n+2)}; i = (n+2)$$

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

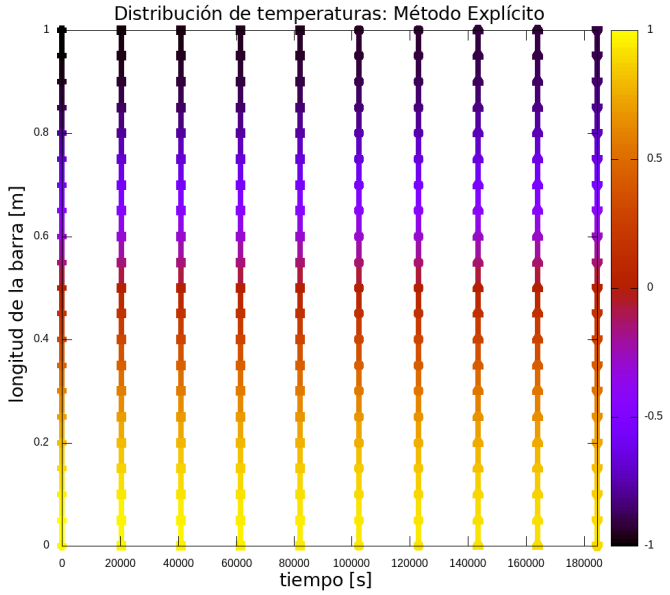


Figure 1:

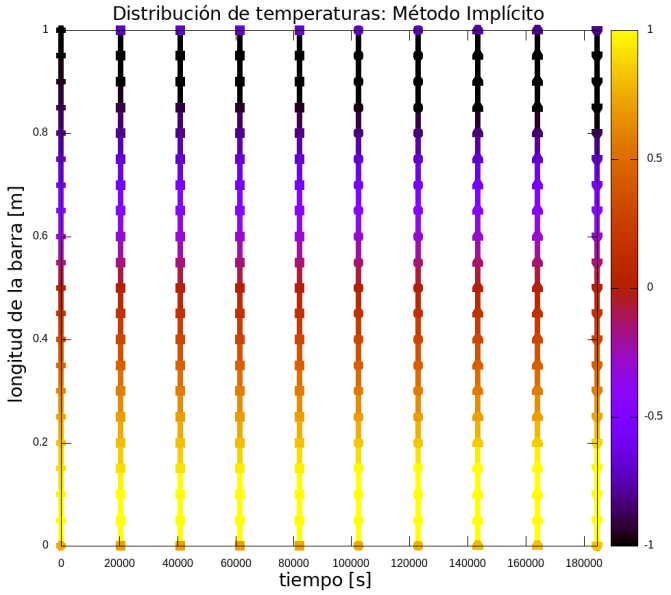


Figure 2:

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,

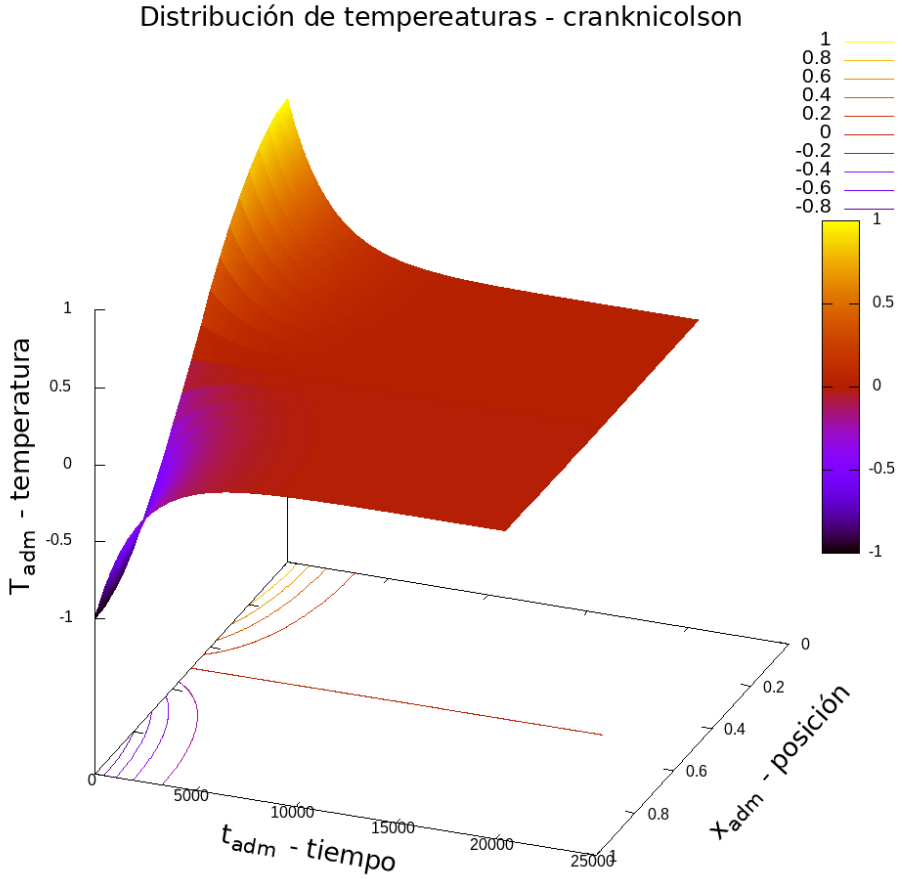


Figure 3:

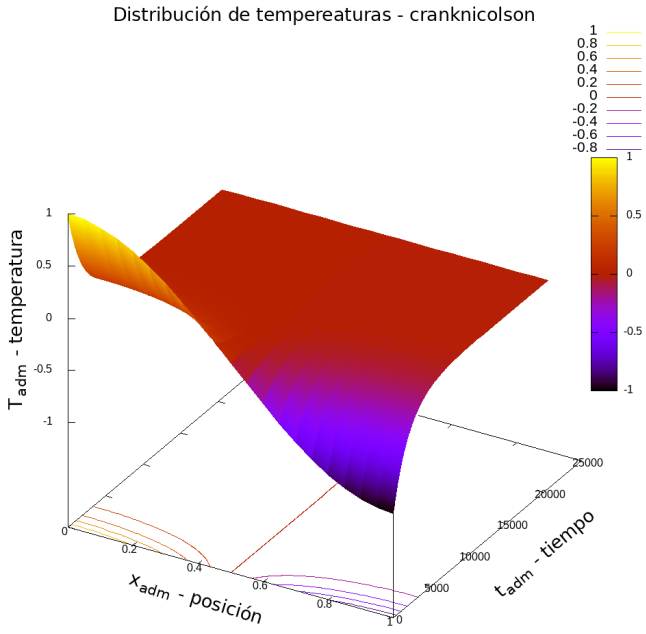


Figure 4:

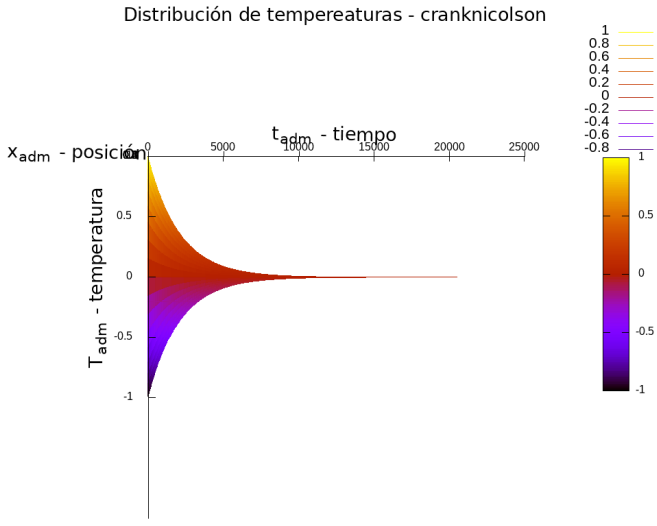


Figure 5:

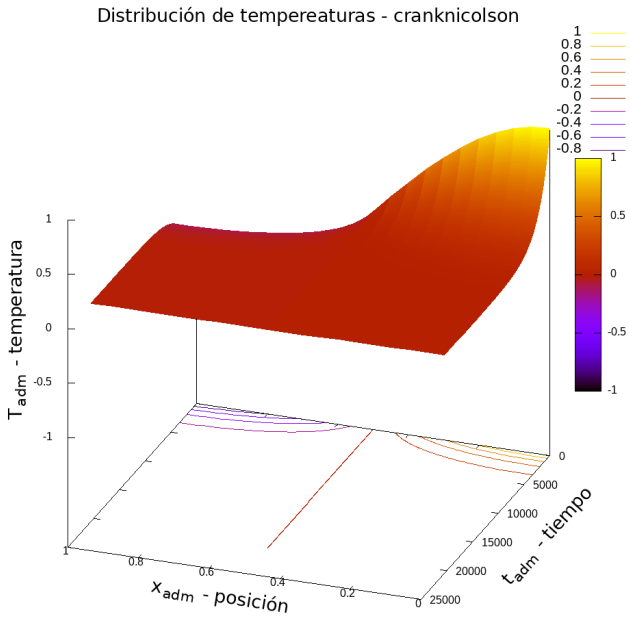


Figure 6:

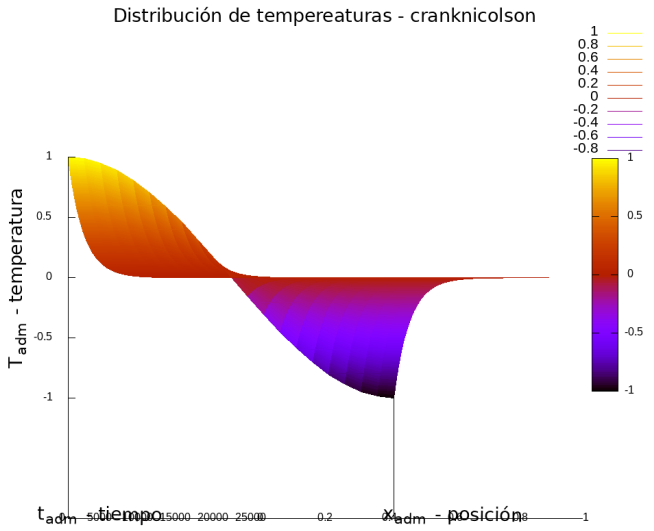


Figure 7:

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,

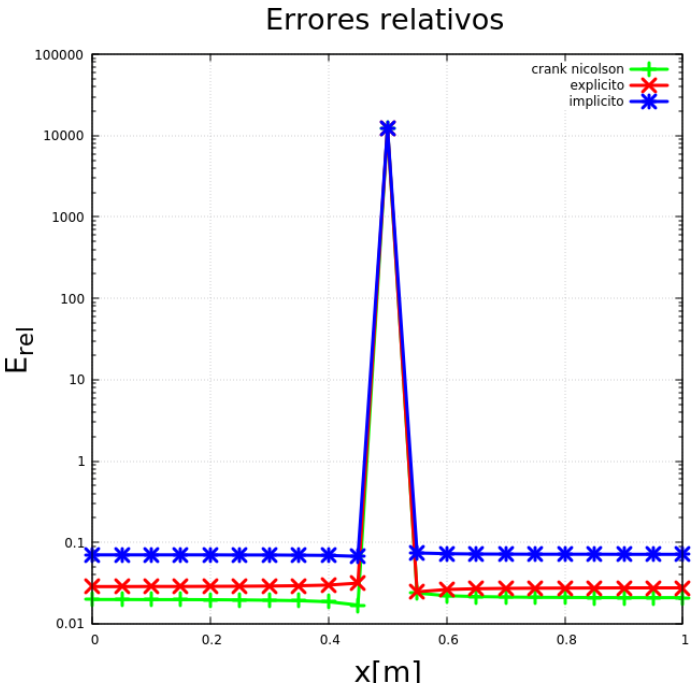


Figure 8:

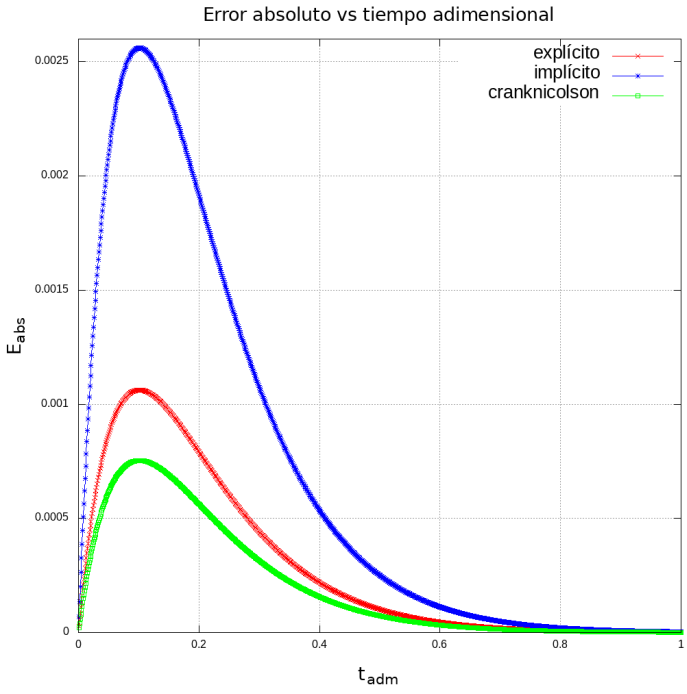


Figure 9:

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,

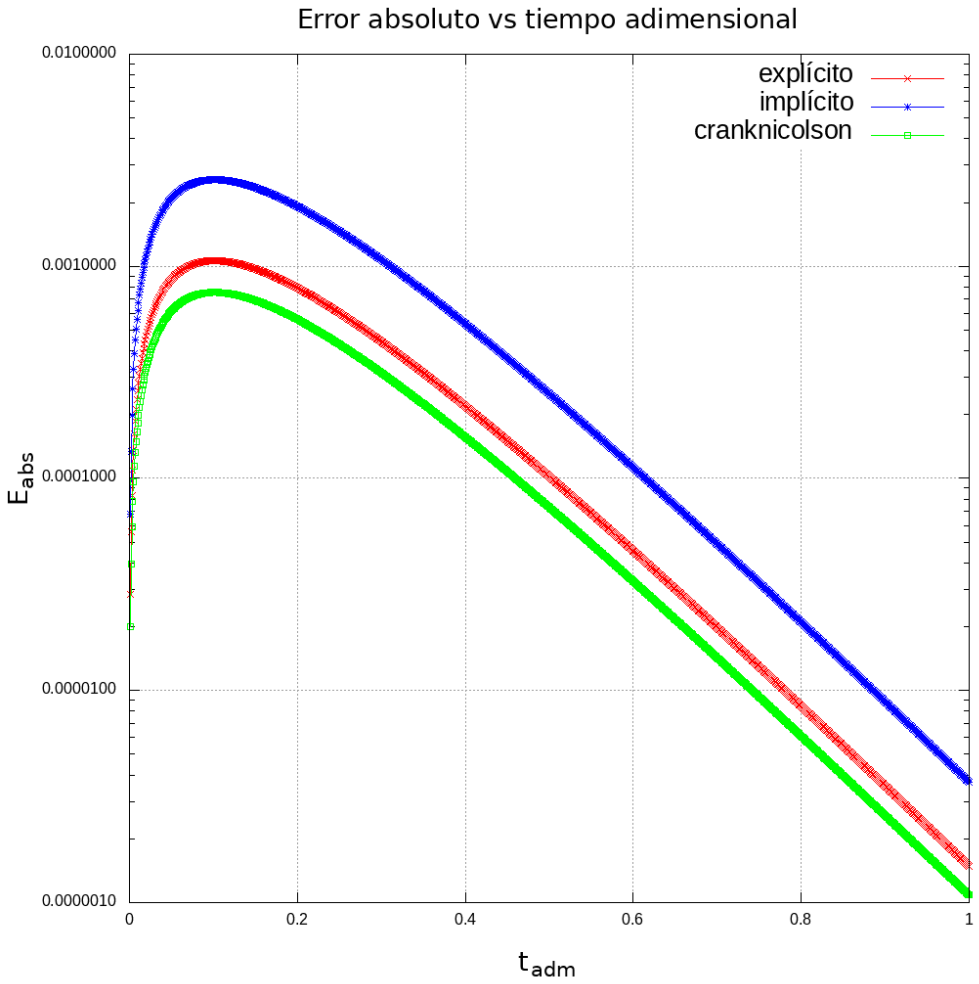


Figure 10:

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)    ]
```