Dinâmica 30/09/2021 a 30/09/2021
Métodos de diferenças finitas.

MET-576-4

Modelagem Numérica da Atmosfera

Dr. Paulo Yoshio Kubota

Os métodos numéricos, formulação e parametrizações utilizados nos modelos atmosféricos serão descritos em detalhe.

3 Meses
24 Aulas (2 horas cada)

Dinâmica 30/09/2021 a 30/09/2021

Dinâmica:

Métodos numéricos amplamente utilizados na solução numérica das equações diferencias parciais que governam os movimentos na atmosfera serão o foco, mas também serão analisados os novos conceitos e novos métodos.

Dinâmica 30/09/2021 a 30/09/2021

- ✓ **Métodos de diferenças finitas.**
- ✓ **Acurácia.**
- ✓ **Consistência.**
- ✓ **Estabilidade.**
- ✓ **Convergência.**
- ✓ **Grades de Arakawa A, B, C e E.**
- ✓ **Domínio de influência e domínio de dependência.**
- ✓ **Dispersão numérica e dissipação.**
- ✓ **Definição de filtros monótono e positivo.**
- ✓ **Métodos espectrais.**
- ✓ **Métodos de volume finito.**
- ✓ **Métodos Semi-Lagrangeanos.**
- ✓ **Conservação de massa local.**
- ✓ **Esquemas explícitos versus semi-implícitos.**
- ✓ **Métodos semi-implícitos.**

Esquema implícito: FTBS

Esquema implícito: FTBS

A equação de advecção linear com discretização Backward no tempo e backward no espaço pode ser escrita como:

$$\frac{\emptyset_j^{n+1} - \emptyset_j^n}{\Delta t} + u\frac{\emptyset_j^{n+1} - \emptyset_{j-1}^{n+1}}{\Delta x} = 0 \qquad (25)$$

$$\emptyset_j^{n+1} - \emptyset_j^n = -u\frac{\Delta t}{\Delta x}\left(\emptyset_j^{n+1} - \emptyset_{j-1}^{n+1}\right)$$

$$\emptyset_j^{n+1} = \emptyset_j^n - u\frac{\Delta t}{\Delta x}\left(\emptyset_j^{n+1} - \emptyset_{j-1}^{n-1}\right)$$

$$\emptyset_j^{n+1} = \emptyset_j^n - C\left(\emptyset_j^{n+1} - \emptyset_{j-1}^{n+1}\right)$$

$$\emptyset_j^{n+1} = \emptyset_j^n - C\emptyset_j^{n+1} + C\emptyset_{j-1}^{n+1}$$

$$\emptyset_j^{n+1} + C\emptyset_j^{n+1} = \emptyset_j^n + C\emptyset_{j-1}^{n+1}$$

$$(1 + C)\emptyset_j^{n+1} = \emptyset_j^n + C\emptyset_{j-1}^{n+1}$$

$$\emptyset_j^{n+1} = \frac{1}{(1 + C)}\left(\emptyset_j^n + C\emptyset_{j-1}^{n+1}\right) \qquad (26)$$

Exercício: mostrar que o fator de amplificação no esquema BTBS

$$|A|^2 = [1 + 2c(1 + c)(1 - cos(k\Delta x))]^{-1}$$

É incondicionalmente estável i.e.

$$\emptyset_j^{n+1} = \frac{1}{(1+C)}\left(\emptyset_j^n + C\emptyset_{j-1}^{n+1}\right) \quad \textbf{(27)}$$

Substituindo $\phi(x_j, t_n) = A^n e^{ikj\Delta x}$ Na eqn 26º temos.

$$A^{n+1}e^{ikj\Delta x} = \frac{1}{(1+C)}\left(A^n e^{ikj\Delta x} + CA^{n+1}e^{ik(j+1)\Delta x}\right)$$

$$A^n A e^{ikj\Delta x} = \frac{1}{(1+C)}\left(A^n e^{ikj\Delta x} + CA^n A e^{ikj\Delta x} e^{ik\Delta x}\right)$$

Cancelando os termo $A^n e^{ikj\Delta x}$.

$$A^n A e^{ikj\Delta x} = \frac{1}{(1+C)}\left(A^n e^{ikj\Delta x} + CA^n A e^{ikj\Delta x} e^{ik\Delta x}\right)$$

$$A = \frac{1}{(1+C)}\left(1 + CAe^{ik\Delta x}\right)$$

$$A = \frac{1}{(1+C)}\left(1 + CAe^{ik\Delta x}\right)$$

$$(1+C)A = \left(1 + CAe^{ik\Delta x}\right)$$

$$(1+C)A - CAe^{ik\Delta x} = 1$$

$$A\left(1 + C - Ce^{ik\Delta x}\right) = 1$$

$$A = \frac{1}{(1 + C - Ce^{ik\Delta x})}$$

$$|A|^2 = \frac{1}{(1 + C - Ce^{ik\Delta x})}\frac{1}{(1 + C - Ce^{-ik\Delta x})}$$

$$|A|^2 = \frac{1}{(1 + C - Ce^{-ik\Delta x} + C + C^2 - C^2e^{-ik\Delta x} - Ce^{ik\Delta x} - C^2\, e^{ik\Delta x} + C^2e^{ik\Delta x - ik\Delta x})}$$

$$|A|^2 = \frac{1}{(1 + C - Ce^{-ik\Delta x} + C + C^2 - C^2e^{-ik\Delta x} - Ce^{ik\Delta x} - C^2\, e^{ik\Delta x} + C^2)}$$

$$|A|^2 = \frac{1}{(1 + 2C + 2C^2 - (C + C^2)e^{-ik\Delta x} - (C + C^2)e^{ik\Delta x})}$$

$$|A|^2 = \frac{1}{(1 + 2C + 2C^2 - (C + C^2)e^{-ik\Delta x} - (C + C^2)e^{ik\Delta x})}$$

$$|A|^2 = \frac{1}{\left(1 + 2C + 2C^2 - (C + C^2)(e^{-ik\Delta x} + e^{ik\Delta x})\right)}$$

$$|A|^2 = \frac{1}{\left(1 + 2C + 2C^2 - 2(C + C^2)\frac{(e^{-ik\Delta x} + e^{ik\Delta x})}{2}\right)}$$

$$|A|^2 = \frac{1}{(1 + 2C + 2C^2 - 2(C + C^2)\cos(k\Delta x)\ )}$$

$$|A|^2 = \frac{1}{(1 + 2(C + C^2) - 2(C + C^2)\cos(k\Delta x)\ )}$$

$$|A|^2 = \frac{1}{(1 + 2(C + C^2)(1 - \cos(k\Delta x))\ )}$$

$$|A|^2 = \frac{1}{(1 + 2C(1 + C)(1 - \cos(k\Delta x))\ )}$$

$$|A|^2 = \frac{1}{(1 + 2C(1 + C)(1 - \cos(k\Delta x)))}$$

Para qualquer numero de onda exceto para k=0, $(1 - \cos(k\Delta x) > 0$

Então $2C(1 + C)$ **>0**

Portanto

$$|A|^2 = \frac{1}{(1 + 2C(1 + C)(1 - \cos(k\Delta x)))} \leq 1$$

A discretização no tempo realizada pelo método implícito implícita, pode ser escrito na forma: Aqui o nosso método upwind torna-se :

$$\frac{\emptyset_j^{n+1} - \emptyset_j^n}{\Delta t} + u\frac{\emptyset_j^{n+1} - \emptyset_{j-1}^{n+1}}{\Delta x} = 0$$

Nós podemos escrever a equação como um sistema linear de equações acopladas:

$$\emptyset_j^{n+1} - \emptyset_j^n = -u\frac{\Delta t}{\Delta x}\left(\emptyset_j^{n+1} - \emptyset_{j-1}^{n+1}\right)$$

Defini-se $\mathrm{C} = u\frac{\Delta t}{\Delta x}$

$$\emptyset_j^{n+1} - \emptyset_j^n = -C\left(\emptyset_j^{n+1} - \emptyset_{j-1}^{n+1}\right)$$

$$\emptyset_j^{n+1} - \emptyset_j^n = -C\emptyset_j^{n+1} + C\emptyset_{j-1}^{n+1}$$

$$\emptyset_j^{n+1} + C\emptyset_j^{n+1} - C\emptyset_{j-1}^{n+1} = \emptyset_j^n$$

$$-C\emptyset_{j-1}^{n+1} + (1+C)\emptyset_j^{n+1} = \emptyset_j^n$$

Em forma matricial, resolve-se para os pontos 1,. . . , j-1, isto é:

$$\begin{pmatrix} 1+C & 0 & 0 & 0 & 0 & 0 & -C \\ -C & 1+C & 0 & 0 & 0 & 0 & 0 \\ 0 & -C & 1+C & 0 & 0 & 0 & 0 \\ 0 & 0 & -C & 1+C & 0 & 0 & 0 \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & 0 & -C & 1+C & 0 \\ 0 & 0 & 0 & 0 & 0 & -C & 1+C \end{pmatrix} \begin{pmatrix} \emptyset_1^{n+1} \\ \emptyset_2^{n+1} \\ \emptyset_3^{n+1} \\ \emptyset_4^{n+1} \\ \vdots \\ \emptyset_{j-2}^{n+1} \\ \emptyset_{j-1}^{n+1} \end{pmatrix} = \begin{pmatrix} \emptyset_1^{n} \\ \emptyset_2^{n} \\ \emptyset_3^{n} \\ \emptyset_4^{n} \\ \vdots \\ \emptyset_{j-2}^{n} \\ \emptyset_{j-1}^{n} \end{pmatrix}$$

Isto requer a resolução de uma matriz isto torna o métodos implícitos geralmente mais caros do que os métodos explícitos. No entanto, a análise de estabilidade mostraria que esta discretização implícita é estável para qualquer escolha de C. (Mas não se deve confundir estabilidade com precisão. As soluções mais precisas com este método ainda deverá ter um C pequeno). Observe também que a forma da matriz vai mudar dependendo da escolha das condições de contorno.

Exercício

- **Resolver a equação advecção 10 numericamente no domínio 0 ≤ X ≤ 1000M. Deixe Δx = 0,5 M. Presume-se que, para a velocidade de Advecção U = 1 m/s. Deixe o estado inicial ser uma função passo . Para a condição limite use phi(0,t) = 0.**

$$\phi(x,0) = \begin{array}{ll} 0 & \text{for} \quad x < 40 \\ 10 & \text{for} \quad 40 \leq x \leq 200 \\ 0 & \text{for} \quad x > 200 \end{array} .$$

Usando o esquema BTBS e mostrar soluções para T = 0S
T = 100 º -s, T = 200S T = 300S , T = 800S.

1. **Compare as soluções de 26º com c=1,0 e c=3,0**

```
MODULE LinearSolve
 IMPLICIT NONE
 PRIVATE
 INTEGER, PARAMETER :: r8 = selected_real_kind(15, 307)
 INTEGER, PARAMETER :: r4 = selected_real_kind(6, 37)
 PUBLIC :: solve_tridiag
CONTAINS
subroutine solve_tridiag(a,b,c,d,x,n)
     implicit none
!a - sub-diagonal (diagonal abaixo da diagonal principal)
!b - diagonal principal
!c - sup-diagonal (diagonal acima da diagonal principal)
!d - parte à direita
!x - resposta
!n - número de equações
       integer, intent(in) :: n
       real (r8), dimension (n),intent (in   ) :: a,b,c,d
       real (r8), dimension (n),intent (out) :: x
       real (r8), dimension (n) :: cp, dp
       real (r8) :: m
       integer :: i
! inicializar c-primo e d-primo
       cp(1) = c(1)/b(1)
       dp(1) = d(1)/b(1)
! resolver para vetores c-primo e d-primo
        do i = 2,n
          m = b(i)-cp(i-1)*a(i)
          cp(i) = c(i)/m
          dp(i) = (d(i)-dp(i-1)*a(i))/m
        enddo
! inicializar x
       x(n) = dp(n)
! resolver para x a partir de vetores c-primo e d-primo
       do i = n-1, 1, -1
        x(i) = dp(i)-cp(i)*x(i+1)
       end do
   end subroutine solve_tridiag
END MODULE LinearSolve
```

```
MODULE Class_Fields
 IMPLICIT NONE
 PRIVATE
 INTEGER, PARAMETER :: r8 = selected_real_kind(15, 307)
 INTEGER, PARAMETER :: r4 = selected_real_kind(6, 37)
 REAL (KIND=r8), PUBLIC, ALLOCATABLE :: A0  (:)
 REAL (KIND=r8), PUBLIC, ALLOCATABLE :: A   (:)
 REAL (KIND=r8), PUBLIC, ALLOCATABLE :: Anew(:)
 REAL (KIND=r8), PUBLIC, ALLOCATABLE :: AA  (:,:)
 REAL (KIND=r8), PUBLIC, ALLOCATABLE :: B   (:)
 REAL (KIND=r8), PUBLIC, ALLOCATABLE :: X   (:)
 INTEGER     , PUBLIC            :: ilo
 INTEGER     , PUBLIC            :: ihi
 INTEGER     , PUBLIC            :: iMax
 PUBLIC ::  Init_Class_Fields

 CONTAINS

SUBROUTINE Init_Class_Fields(nx,dx, coef_C )
 IMPLICIT NONE
 INTEGER       , INTENT (IN   ) :: nx
 REAL (KIND=r8) , INTENT (IN   ) :: dx
 REAL (KIND=r8) , INTENT (IN   ) :: coef_C
 REAL (KIND=r8) :: coord_X(0:nx)
 INTEGER       :: i
 iMax=nx
 ALLOCATE (A0   (0:iMax)      )
 ALLOCATE (A    (0:iMax)      )
 ALLOCATE (Anew (0:iMax)      )
 ALLOCATE (AA   (0:iMax+1,0:iMax+1))
 ALLOCATE (B    (0:iMax)      )
 ALLOCATE (X    (0:iMax)      )

!# python is zero-based.  We are assuming periodic BCs, so
!# points 0 and N-1 are the same.  Set some integer indices to
!# allow us to easily access points 1 through N-1.  Point 0
!# won't be explicitly updated, but rather filled by the BC
!# routine.
```

```
ilo = 1
 ihi = iMax-1
 DO i=0,iMax
   coord_X(i) = REAL(i,KIND=r8)*dx
 END DO
 !
 ! initialize the data -- tophat
 !
 DO i=0,iMax
   IF(coord_X(i) >= 0.333_r8 .and. coord_X(i) <= 0.666_r8)
THEN
      A0(i) = COS(0.50_r8 - coord_X(i))
   ELSE
      A0(i) = 0.0_r8
   END IF
 END DO
 A=A0
 AA=0.0_r8
 !
 ! """ we don't explicitly update point 0, since it is identical
 !          to N-1, so fill it here """
 A(0) = A(ihi)

 !
 !  a(t+1,i)  -   a(t,i)              a(t+1,i) - a(t+1,i-1)
 !  ----------------------  = -  U ----------------------------------
 !        Dt                                  Dx
 !
 !                                      -                        -
 !                                 Dt  |                          |
 !  a(t+1,i)  -  a(t,i)  =  - U ---- | a(t+1,i) - a(t+1,i-1) |
 !                                 Dx |                          |
 !                                      -                        -

 !                                    -                      -
 !                                   |                        |
 !  a(t+1,i)  -  a(t,i)  =  - C  | a(t+1,i) - a(t+1,i-1) |
 !                                   |                        |
 !                                    -                      -
```

```
 !
  !  a(t+1,i)  -  a(t,i)  =  - Ca(t+1,i) + Ca(t+1,i-1)
  !
  !  a(t+1,i) +  Ca(t+1,i) - Ca(t+1,i-1)  =  a(t,i)

  !  -C a(t+1,i-1)  + (1 + C) a(t+1,i)  =  a(t,i)

  !  -C a(t+1,  0)  + (1 + C) a(t+1,1)  =  a(t,1)
  !  -C a(t+1,  1)  + (1 + C) a(t+1,2)  =  a(t,2)
  !  -C a(t+1,  2)  + (1 + C) a(t+1,3)  =  a(t,3)
  !  -C a(t+1,  3)  + (1 + C) a(t+1,4)  =  a(t,4)
  !  -C a(t+1,i-1)  + (1 + C) a(t+1,i)  =  a(t,i)
  !  ____                ____  __       __
  ! |                        | |             |
  ! | (1 + C)     0     -C   | | a(t+1,1)  | =  a(t,1)
  ! | -C       (1 + C)   0   | | a(t+1,2)  | =  a(t,i)
  ! |  0         -C   (1 + C)| | a(t+1,i-1)| =  a(t,i)
  ! |____               ____| |__       __|
  !
  !          A                        X       =    B
 !
  !# create the matrix
  !# loop over rows [ilo,ihi] and construct the matrix.  This will
  !# be almost bidiagonal, but with the upper right entry also
  !# nonzero.
  !
   AA(0,ihi  ) =  1.0_r8 + coef_C
   AA(0,ilo  ) = -coef_C
   DO i =ilo,ihi
      AA(i,i+1) = 0.0_r8
      AA(i,i  ) = 1.0_r8 + coef_C
      AA(i,i-1) = -coef_C
   END DO
   AA(nx,ihi  ) =   1.0_r8 + coef_C
   AA(nx,ilo  ) =  -coef_C
 END SUBROUTINE Init_Class_Fields
 END MODULE Class_Fields
```

```
MODULE Class_WritetoGrads
 USE Class_Fields, Only: Anew,A,iMax
 IMPLICIT NONE
 PRIVATE
  INTEGER, PUBLIC     , PARAMETER  :: r8=8
  INTEGER, PUBLIC     , PARAMETER  :: r4=4
 INTEGER             , PARAMETER  :: UnitData=1
 INTEGER             , PARAMETER  :: UnitCtl=2
 CHARACTER (LEN=400)              :: FileName
 LOGICAL                          :: CtrlWriteDataFile
 PUBLIC :: SchemeWriteCtl
 PUBLIC :: SchemeWriteData
 PUBLIC :: InitClass_WritetoGrads

CONTAINS

 SUBROUTINE InitClass_WritetoGrads()
   IMPLICIT NONE
  FileName=''
  FileName='ImplicitLinearAdvection1D'
  CtrlWriteDataFile=.TRUE.
 END SUBROUTINE InitClass_WritetoGrads

 FUNCTION SchemeWriteData(irec)  RESULT (ok)
   IMPLICIT NONE
   INTEGER   , INTENT (INOUT) :: irec
   INTEGER             :: ok
   INTEGER             :: lrec
   REAL (KIND=r4)      :: Yout(iMax)
   INQUIRE (IOLENGTH=lrec) Yout
   IF(CtrlWriteDataFile) OPEN (UnitData,FILE=TRIM(FileName)//'.bin', &
    FORM='UNFORMATTED', ACCESS='DIRECT',STATUS='UNKNOWN',&
    ACTION='WRITE',RECL=lrec)
   CtrlWriteDataFile=.FALSE.
   Yout=REAL(A(1:iMax),KIND=r4)
   irec=irec+1
   WRITE(UnitData,rec=irec)Yout
   ok=0
 END FUNCTION SchemeWriteData

 FUNCTION SchemeWriteCtl(nrec)  RESULT(ok)
   IMPLICIT NONE
   INTEGER, INTENT (IN) :: nrec
   INTEGER              :: ok

   OPEN (UnitCtl,FILE=TRIM(FileName)//'.ctl', &
   FORM='FORMATTED',ACCESS='SEQUENTIAL', &
   STATUS='UNKNOWN',ACTION='WRITE')
   WRITE (UnitCtl,'(A6,A       )')'dset ^',TRIM(FileName)//'.bin'
   WRITE (UnitCtl,'(A          )')'title  EDO'
   WRITE (UnitCtl,'(A          )')'undef  -9999.9'
   WRITE (UnitCtl,'(A6,I8,A18   )')'xdef  ',iMax,' linear 0.00 0.001'
   WRITE (UnitCtl,'(A          )')'ydef  1 linear  -1.27 1'
   WRITE (UnitCtl,'(A6,I6,A25   )')'tdef  ',nrec,' linear
00z01jan0001 1hr'
   WRITE (UnitCtl,'(A20        )')'zdef  1 levels 1000 '
   WRITE (UnitCtl,'(A          )')'vars 1'
   WRITE (UnitCtl,'(A          )')'A   0 99 resultado da edol yc'
   WRITE (UnitCtl,'(A          )')'endvars'
   CLOSE (UnitCtl,STATUS='KEEP')
   CLOSE (UnitData,STATUS='KEEP')
   ok=0
 END FUNCTION SchemeWriteCtl


END MODULE Class_WritetoGrads
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
PROGRAM Main
 USE  Class_Fields, Only :
Init_Class_Fields,B,A,AA,ilo,ihi,Anew
 USE  LinearSolve, OnLy : solve_tridiag
 USE  Class_WritetoGrads, OnLy :schemeWriteCtl,
schemeWriteData,initClass_WritetoGrads

 ! main program to check the Tridiagonal system solver
 INTEGER, PARAMETER :: r8 = selected_real_kind(15,
307)
 INTEGER, PARAMETER :: r4 = selected_real_kind(6, 37)
 INTEGER     , PARAMETER :: nn=200
 INTEGER     , PARAMETER :: nx=nn
 REAL (KIND=8) , PARAMETER :: xmin=0.0
 REAL (KIND=8) , PARAMETER :: xmax=1.0
 REAL (KIND=8) , PARAMETER :: C=0.5       ! # CFL
number
                                          !    [0.5, 1.0,
10.0]
! REAL(KIND=8) , PARAMETER :: u = 10.0
 REAL (KIND=8) , PARAMETER :: Dx=(xmax - xmin)/(nx-1)
! REAL(KIND=8) , PARAMETER :: Dt=C*dx/u
 INTEGER     , PARAMETER :: ninteraction=400
 INTEGER :: i,j

 CALL Init()
 CALL run()
 STOP

 CONTAINS

 SUBROUTINE Init()
  IMPLICIT NONE
    CALL Init_Class_Fields(nx,dx, C)
    CALL initClass_WritetoGrads()
 END SUBROUTINE Init
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
 SUBROUTINE Run()
  IMPLICIT NONE
  REAL (KIND=r8) :: a_sub_diagonal   (0:nx)
  REAL (KIND=r8) :: b_pri_diagonal   (0:nx)
  REAL (KIND=r8) :: c_sup_diagonal   (0:nx)
  INTEGER :: test,irec

  DO i =ilo,ihi
!         cc(i)= 0
!         b(i)= 1.0 + C
!         a(i)= - C

     a_sub_diagonal(i) =AA(i,i-1)   ! -C        !AA(i,i-1)
     b_pri_diagonal(i) =AA(i,i  )       ! 1.0_r8 + C !AA(i,i  )
     c_sup_diagonal(i) =AA(i,i+1)  ! 0.0_r8     !AA(i,i+1)
  END DO

  DO i =ilo,ihi
    WRITE(*,"(3F6.2)")a_sub_diagonal(i),b_pri_diagonal(i), &
                                c_sup_diagonal(i)
  END DO
   irec=0
```

```
 DO i=1,ninteraction

! create the RHS -- this holds all entries except for a[0]

        B    (ilo:ihi)  = A(ilo:ihi)
                      ! tridag(a,b,c,d,nn)
                     ! PRINT*,A
                     ! PRINT*," ! tridag(a,b,c,d,nn)tridag(a,b,c,d,nn)"
                      Anew (ilo:ihi)  = 0.0_r8
        test=SchemeWriteData(irec)

      CALL solve_tridiag( a_sub_diagonal(ilo:ihi), &
                              b_pri_diagonal(ilo:ihi), &
                               c_sup_diagonal(ilo:ihi), &
                               B                          (ilo:ihi), &
                              Anew                        (ilo:ihi), &
                               nx-1                       )
        Anew(ihi+1)= Anew(ihi)
       A(ilo:ihi+1) = Anew(ilo:ihi+1)

       A(ilo+2) = A(ihi  )
       A(ilo+1) = A(ihi-1)
       A(ilo  ) = A(ihi-2)

 END DO    !   t += dt
  test=SchemeWriteCtl(ninteraction)

 END SUBROUTINE Run
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
 SUBROUTINE Finalize()

 END SUBROUTINE Finalize
END PROGRAM Main
```

Exercício de difusão

- Resolver numericamente o problema de difusão que tem sido discutido na seção anterior usando o esquema de diferença finitas para a derivada temporal. Use uma resolução espacial de $\Delta x = 10^{-2}$ m e Coeficiente de difusão $K = 2{,}9 \times 10^{-5}$. Integrar para pelo menos

  Para 6 horas (cerca de 25000 segundos), e mostrar as soluções para T = 1 Hora, T = 2 Horas, T = 3 Horas, T = 4 Horas, T = 5 Horas, e T = 6 Horas. Comparar a solução com o Uma análise Fourier (retenção 1000 componentes). Escolha o passo de tempo sendo de tal ordem que o sistema seja estável. Deixe o primeiro setup em temperatura de 1m haste longa dada.

```
x=0.0
 DO i=1,iMax
   IF(x >= 0.0 .and. x < 0.5) THEN
     PHI_C(i)= 273.15 + 2*X
   ELSE IF(x > 0.5 .and. x < 1.0) THEN
     PHI_C(i)= 273.15 + 2.0 - 2*X
   END IF
    x =  (i)* DX
 END DO
```

Ambas as extremidades são mantidos na mesma temperatura

$T_0 = 273.15K$.