Dinâmica 29/09/2023 a 29/09/2023
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:

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.

- ✓ **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

PDEs Parabólicas: Esquemas Explícitos

Resumo: Solução de EDPs Parabólicas por Esquemas Explícitos

Vantagens:

- cálculos muito fáceis,
- simplesmente fornece um passo à frente n+1

Desvantagem:

– baixa precisão, O (Δt) em relação ao tempo
– sujeito a instabilidade; deve usar "pequenos" Δt'
- requer muitos passos !!!

Esquema implícito: FTBS

PDEs Parabólicas: Esquemas Implícitos Esquemas Implícitos para PDEs Parabólicas •
Expresso T_i^{n+1} termos de T_j^{n+1} , T_i^n, e possivelmente também T_j^n (em que j = i – 1 e i+1)

• Representa o domínio espacial e temporal. Para cada novo tempo, escreve m (nº de nós interiores) equações e simultaneamente resolve para m valores desconhecidos (sistema com bandas).

Esquema implícito: FTBS

The 1-D Heat Equation: $$\frac{\partial T}{\partial t} = k \frac{\partial^2 T}{\partial x^2}$$

Simple Implicit Method. *Substituting:*

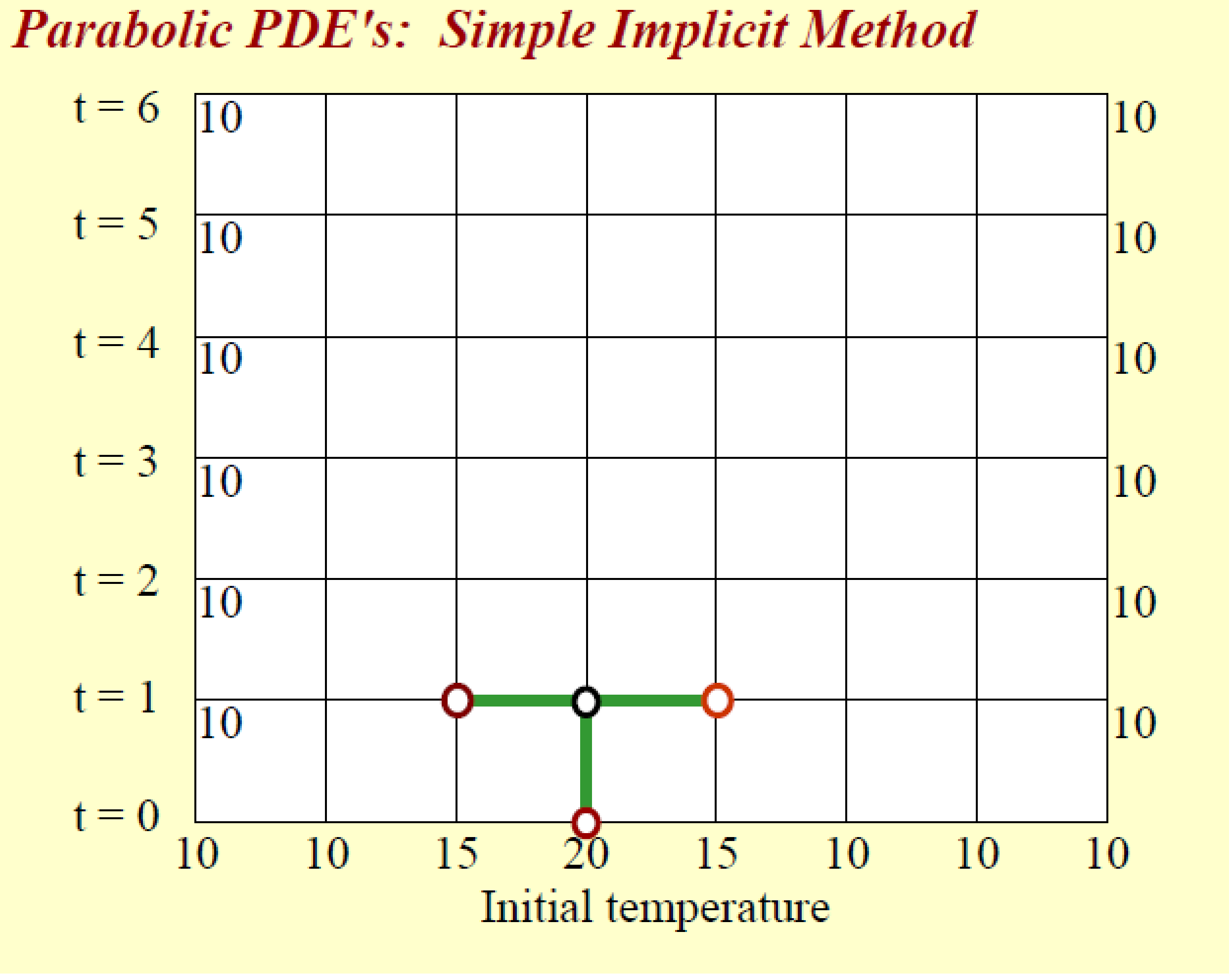
$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i-1}^{m+1} - 2T_i^{m+1} + T_{i+1}^{m+1}}{(\Delta x)^2} + O(\Delta x)^2 \qquad \text{Centered FDD}$$

$$\frac{\partial T}{\partial t} = \frac{T_i^{m+1} - T_i^{m}}{\Delta t} + O(\Delta t) \qquad \text{Backward FDD}$$

results in: $-\lambda T_{i-1}^{m+1} + (1+2\lambda) T_i^{m+1} - \lambda T_{i+1}^{m+1} = T_i^m$ with $\lambda = k \frac{\Delta t}{(\Delta x)^2}$

1. Requer I.C. para o caso em que m = 0: ou seja, T_i^m é dado para todo i.
2. Requer B.C.s para escrever n expressões.

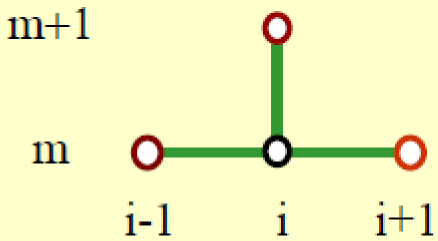
Esquema implícito: FTBS



1. Requer I.C. para o caso em que m = 0: ou seja, T_i^m é dado para todo i.
2. Requer B.C.s para escrever n expressões.

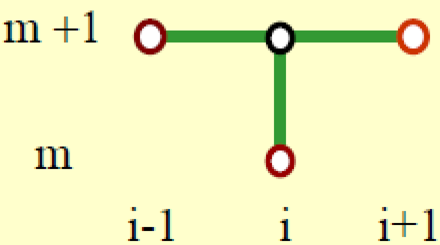
Esquema implícito: FTBS

Explicit Method



$$T_i^{m+1} = T_i^m + \lambda\left(T_{i+1}^m - 2T_i^m + T_{i-1}^m\right)$$

Simple Implicit Method



$$T_i^m = -\lambda T_{i-1}^{m+1} + (1+2\lambda)T_i^{m+1} - \lambda T_{i+1}^{m+1}$$

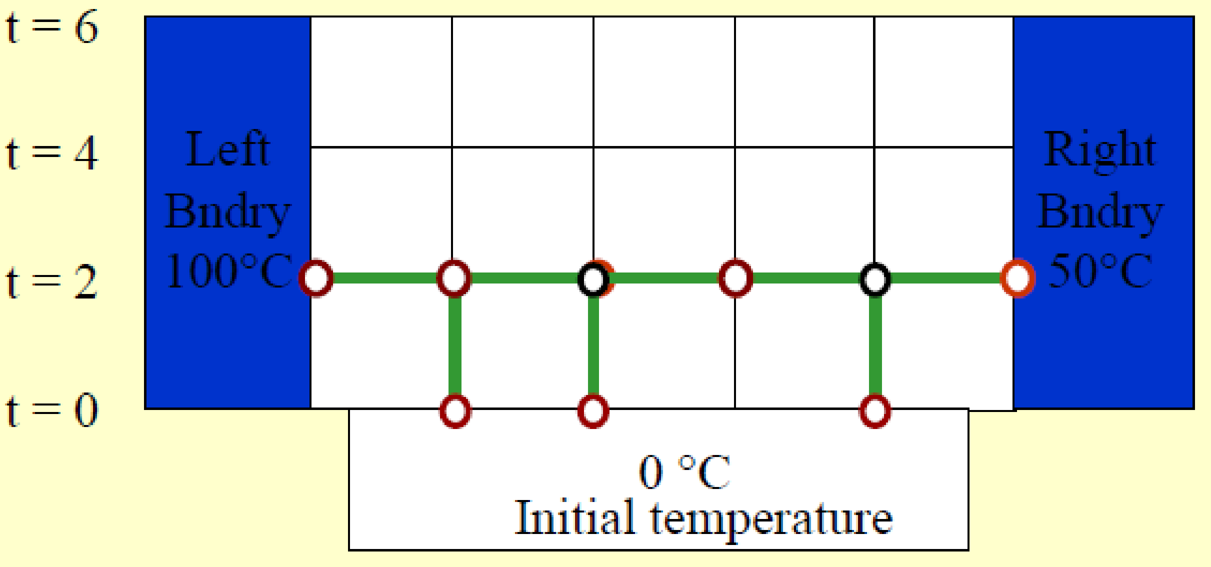
with $\lambda = k\dfrac{\Delta t}{(\Delta x)^2}$ for both

1. Requer I.C. para o caso em que m = 0: ou seja, T_i^m é dado para todo i.
2. Requer B.C.s para escrever n expressões.

Esquema implícito: FTBS

$$-\lambda T_{i-1}^{m+1} + (1+2\lambda)T_i^{m+1} - \lambda T_{i+1}^{m+1} = T_i^m \quad \text{with} \quad \lambda = k\frac{\Delta t}{(\Delta x)^2}$$

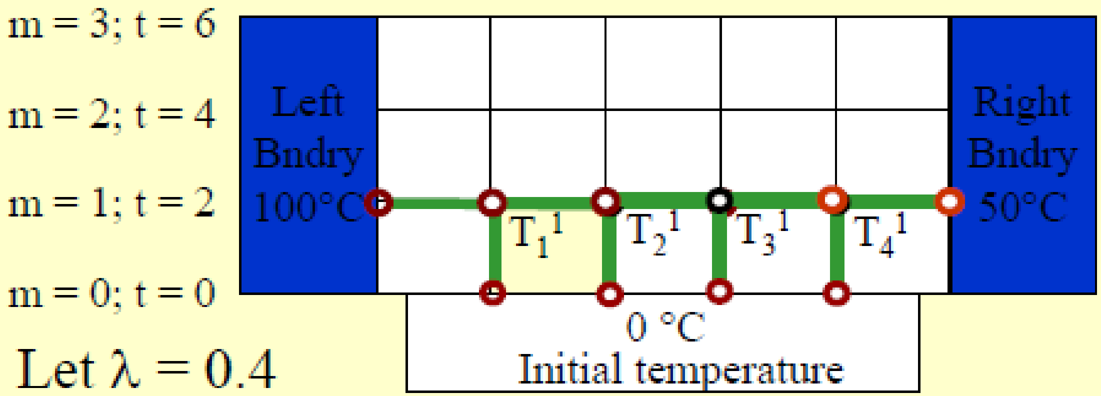
Parabolic PDE's: Simple Implicit Method



At the Left boundary: $(1+2\lambda)T_1^{m+1} - \lambda T_2^{m+1} = T_1^m + \lambda\, T_0^{m+1}$

Away from boundary: $-\lambda T_{i-1}^{m+1} + (1+2\lambda)T_i^{m+1} - \lambda T_{i+1}^{m+1} = T_i^m$

At the Right boundary: $(1+2\lambda)T_i^{m+1} - \lambda T_{i-1}^{m+1} = T_i^m + \lambda\, T_{i+1}^{m+1}$



Let $\lambda = 0.4$

$$\begin{bmatrix} 1.8 & -0.4 & 0 & 0 \\ -0.4 & 1.8 & -0.4 & 0 \\ 0 & -0.4 & 1.8 & -0.4 \\ 0 & 0 & -0.4 & 1.8 \end{bmatrix} \begin{Bmatrix} T_1^1 \\ T_2^1 \\ T_3^1 \\ T_4^1 \end{Bmatrix} = \begin{Bmatrix} 40 \\ 0 \\ 0 \\ 20 \end{Bmatrix}$$

$$\begin{Bmatrix} T_1^1 \\ T_2^1 \\ T_3^1 \\ T_4^1 \end{Bmatrix} = \begin{Bmatrix} 27.6 \\ 6.14 \\ 4.03 \\ 12.0 \end{Bmatrix}$$

At the Left boundary: $(1+2\lambda)T_1^1 - \lambda T_2^1 = T_1^0 + \lambda T_0^1$

$1.8\, T_1^1 - 0.4\, T_2^1 = 0 + 0.8*100 =$ 40

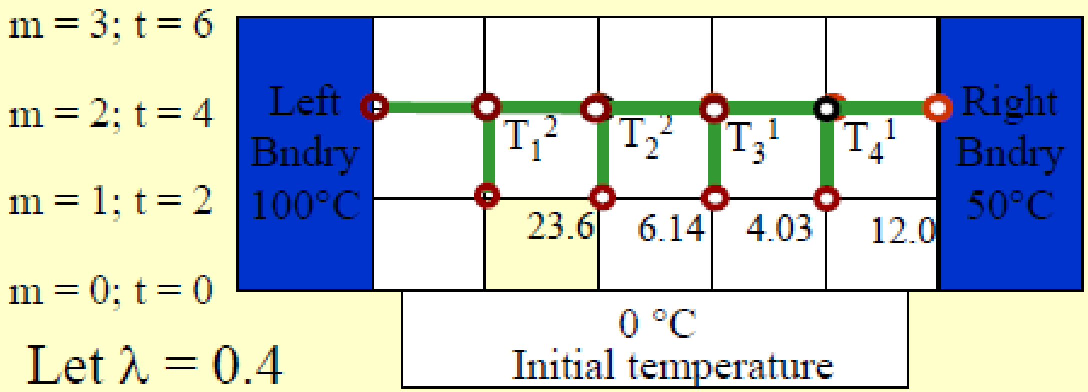
Away from boundary: $-\lambda T_{i-1}^1 + (1+2\lambda)T_i^1 - \lambda T_{i+1}^1 = T_i^0$

$-0.4\, T_1^1 + 1.8\, T_2^1 - 0.4\, T_3^1 = 0 =$ 0

$-0.4\, T_2^1 + 1.8\, T_3^1 - 0.4\, T_4^1 = 0 =$ 0

At the Right boundary: $(1+2\lambda)T_3^1 - \lambda T_2^1 = T_3^0 + \lambda T_4^1$

$1.8\, T_i^{m+1} - 0.4\, T_{i-1}^{m+1} = 0 + 0.4*50) =$ 20



Let $\lambda = 0.4$

$$\begin{bmatrix} 1.8 & -0.4 & 0 & 0 \\ -0.4 & 1.8 & -0.4 & 0 \\ 0 & -0.4 & 1.8 & -0.4 \\ 0 & 0 & -0.4 & 1.8 \end{bmatrix} \begin{Bmatrix} T_1^2 \\ T_2^2 \\ T_3^2 \\ T_4^2 \end{Bmatrix} = \begin{Bmatrix} 78.5 \\ 6.14 \\ 4.03 \\ 32.0 \end{Bmatrix}$$

$$\begin{Bmatrix} T_1^2 \\ T_2^2 \\ T_3^2 \\ T_4^2 \end{Bmatrix} = \begin{Bmatrix} 38.5 \\ 14.1 \\ 9.83 \\ 20.0 \end{Bmatrix}$$

At the Left boundary: $(1+2\lambda)T_1^2 - \lambda T_2^2 = T_1^1 + \lambda T_0^2$

$$1.8\ T_1^2 - 0.4\ T_2^2 = 23.6 + 0.4*100 = \qquad 78.5$$

Away from boundary: $-\lambda T_{i-1}^2 + (1+2\lambda)T_i^2 - \lambda T_{i+1}^2 = T_i^1$

$$-0.4*T_1^2 + 1.8*T_2^2 - 0.4*T_3^2 = 6.14 = \qquad 6.14$$

$$-0.4*T_2^2 + 1.8*T_3^2 - 0.4*T_4^2 = 4.03 = \qquad 4.03$$

At the Right boundary: $(1+2\lambda)T_3^2 - \lambda T_4^2 = T_3^1 + \lambda T_4^2$

$$1.8*T_3^2 - 0.4*T_4^2 = 12.0 + 0.4*50 = \qquad 32.0$$

Analise de estabilidade

Esquema implícito: FTBS

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

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

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

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

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

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

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

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

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

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.

$$\boxed{\phi_j^{n+1} = \frac{1}{(1+C)}\left(\phi_j^n + C\phi_{j-1}^{n+1}\right)} \qquad \textbf{(27)}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Então $2C(1 + C)$ **>0**

Portanto

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

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

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

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

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

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

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

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

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

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

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 $0 \leq X \leq 1000$M. Deixe $\Delta x = 0{,}5$ 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.**

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

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

```fortran
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
```

```fortran
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.
```

```fortran
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) |
 !                                  |                            |
 !                                    -                             -
```

```fortran
!
 !  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.

Esquemas Implícitos para PDEs Parabólicas

O Esquema Crank-Nicholson é centrado no tempo e no espaço.

Método Crank-Nicolson (CN) (Método Implícito) Fornece precisão de 2ª ordem no espaço e no tempo. Média da 2ª derivada no espaço para t^{m+1} e t^{m+1} .

Tem boas propriedades de estabilidade e precisão.

Tem precisão de segunda ordem e incondicionalmente estável.

Duas formas do esquema de Crank-Nicholson para o esquema de advecção são comumente usadas:

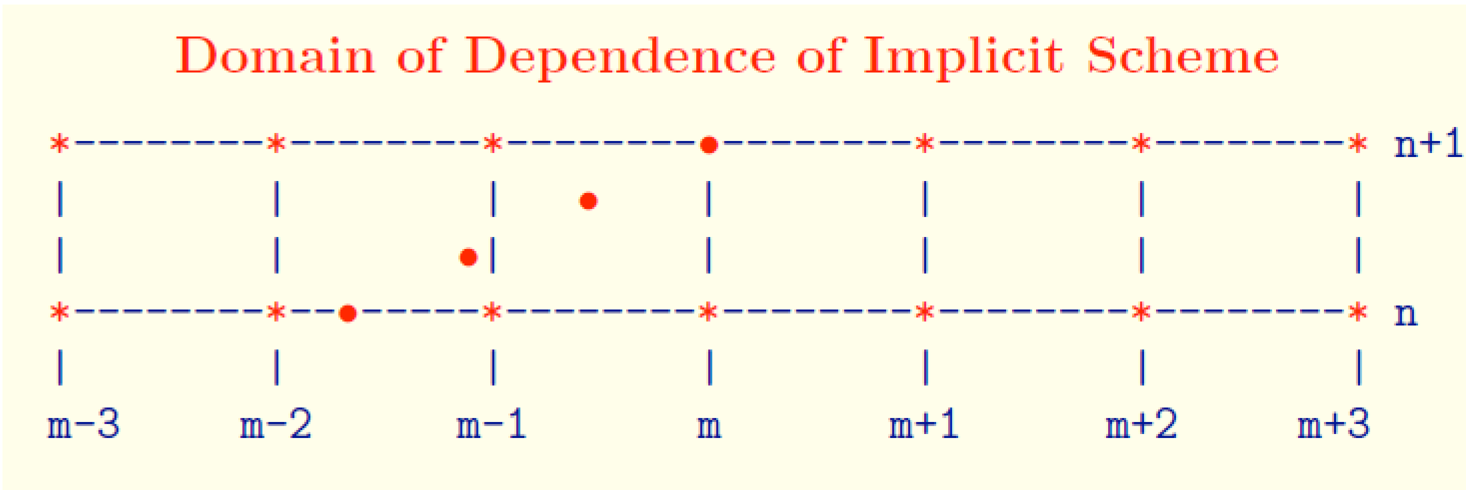
O Esquema C-N de quatro pontos:

$$\frac{1}{2}\left[\frac{U_m^{n+1}-U_m^n}{\Delta t}+\frac{U_{m+1}^{n+1}-U_{m-1}^n}{\Delta t}\right]+\frac{c}{2}\left[\frac{U_{m+1}^{n+1}-U_m^{n+1}}{\Delta x}+\frac{U_{m+1}^n-U_m^n}{\Delta x}\right]=0$$

O Esquema C-N de seis pontos:

$$\frac{U_m^{n+1}-U_m^n}{\Delta t}+\frac{c}{2}\left[\frac{U_{m+1}^{n+1}-U_{m-1}^{n+1}}{2\Delta x}+\frac{U_{m+1}^n-U_m^n}{2\Delta x}\right]=0$$

PDEs parabólicas: Método Crank-Nicolson



A linha com bolinhas (●) representa uma trajetória de parcela.

O valor no ponto $m\Delta x$ no tempo $(n+1)\Delta t$ depende de todos os pontos indicados por asteriscos vermelhos (*).

Assim, o domínio computacional de dependência envolve o domínio físico de dependência.

Esta é uma **condição necessária para um esquema estável**.

Todos os esquemas implícitos também têm uma desvantagem significativa.

Como U_m^{n+1} aparece nos lados esquerdo e direito, a solução para U_m^{n+1} requer a solução de um sistema de equações.

Se envolver apenas ***sistemas tridiagonais***, isso não é um obstáculo, pois existem métodos rápidos para resolvê-los.

Existem também métodos, como passos fracionários (com cada direção espacial resolvida sucessivamente), onde uma dimensão do espaço é considerada de cada vez.

Esses esquemas chamados ADI (alternating direction implicit) permitem grandes passos de tempo sem um grande custo computacional adicional.

Esquemas Implícitos para PDEs Parabólicas

Método Crank-Nicolson (CN) (Método Implícito) Fornece precisão de 2ª ordem no espaço e no tempo. Média da 2ª derivada no espaço para t^{m+1} e t^{m+1} .

$$\frac{\partial^2 T}{\partial x^2} = \frac{1}{2}\left[\frac{T_{i-1}^{m} - 2T_i^{m} + T_{i+1}^{m}}{(\Delta x)^2} + \frac{T_{i-1}^{m+1} - 2T_i^{m+1} + T_{i+1}^{m+1}}{(\Delta x)^2}\right] + O(\Delta x)^2$$

$$\frac{\partial T}{\partial t} = \frac{T_i^{m+1} - T_i^{m}}{\Delta t} + O(\Delta t^2) \quad \text{(central difference in time now)}$$

$$-\lambda T_{i-1}^{m+1} + 2(1+\lambda)T_i^{m+1} - \lambda T_{i+1}^{m+1} = \lambda T_{i-1}^{m} + 2(1-\lambda)T_i^{m} - \lambda T_{i+1}^{m}$$

Requer I.C. para o caso em que m = 0: T_i^0 valor dado, f(x)
Requer BC's para escrever a expressão para T_0^{m+1} & T_{i+1}^{m+1}

PDEs parabólicas: Método Crank-Nicolson

Exercicio:
Método Crank-Nicolson (CN).

Requer I.C. para o caso em que m = 0: T_i^0 valor dado, f(x)
Requer BC's para escrever a expressão para T_0^{m+1} & T_{i+1}^{m+1}

$$\frac{\partial^2 T}{\partial x^2} = \frac{1}{2}\left[\frac{T_{i-1}^{m} - 2T_i^{m} + T_{i+1}^{m}}{(\Delta x)^2} + \frac{T_{i-1}^{m+1} - 2T_i^{m+1} + T_{i+1}^{m+1}}{(\Delta x)^2}\right] + O(\Delta x)^2$$

$$\frac{\partial T}{\partial t} = \frac{T_i^{m+1} - T_i^{m}}{\Delta t} + O(\Delta t^2) \quad \text{(central difference in time now)}$$

$$-\lambda T_{i-1}^{m+1} + 2(1+\lambda)T_i^{m+1} - \lambda T_{i+1}^{m+1} = \lambda T_{i-1}^{m} + 2(1-\lambda)T_i^{m} - \lambda T_{i+1}^{m}$$

$$2T_i^{n+1} - 2T_i^{n} = \frac{\Delta t}{\Delta x \Delta x}\left(T_{i-1}^{n} - 2T_i^{n} + T_{i+1}^{n} + T_{i-1}^{n+1} - 2T_i^{n+1} + T_{i+1}^{n+1}\right)$$

$$2T_i^{n+1} - \frac{\Delta t}{\Delta x \Delta x}\left(T_{i-1}^{n+1} - 2T_i^{n+1} + T_{i+1}^{n+1}\right) = \frac{\Delta t}{\Delta x \Delta x}\left(T_{i-1}^{n} - 2T_i^{n} + T_{i+1}^{n}\right) + 2T_i^{n}$$

$$2T_i^{n+1} + 2\frac{\Delta t}{\Delta x \Delta x}T_i^{n+1} - \frac{\Delta t}{\Delta x \Delta x}T_{i-1}^{n+1} - \frac{\Delta t}{\Delta x \Delta x}T_{i+1}^{n+1} = 2T_i^{n} - 2\frac{\Delta t}{\Delta x \Delta x}T_i^{n} + \frac{\Delta t}{\Delta x \Delta x}T_{i-1}^{n} + \frac{\Delta t}{\Delta x \Delta x}T_{i+}^{n}$$

$$-\frac{\Delta t}{\Delta x \Delta x}T_{i-1}^{n+1} + \left(2 + 2\frac{\Delta t}{\Delta x \Delta x}\right)T_i^{n+1} - \frac{\Delta t}{\Delta x \Delta x}T_{i+1}^{n+1} = \frac{\Delta t}{\Delta x \Delta x}T_{i-1}^{n} + \left(2 - 2\frac{\Delta t}{\Delta x \Delta x}\right)T_i^{n} + \frac{\Delta t}{\Delta x \Delta x}T_{i+1}^{n}$$

PDEs parabólicas: Método Crank-Nicolson

$$-\frac{\Delta t}{\Delta x \Delta x} T_{i-1}^{n+1} + \left(2 + 2\frac{\Delta t}{\Delta x \Delta x}\right) T_i^{n+1} - \frac{\Delta t}{\Delta x \Delta x} T_{i+1}^{n+1} = \frac{\Delta t}{\Delta x \Delta x} T_{i-1}^{n} + \left(2 - 2\frac{\Delta t}{\Delta x \Delta x}\right) T_i^{n} + \frac{\Delta t}{\Delta x \Delta x} T_{i+1}^{n}$$

$$-\lambda T_{i-1}^{n+1} + 2(1+\lambda) T_i^{n+1} - \lambda T_{i+1}^{n+1} = \lambda T_{i-1}^{n} + 2(1-\lambda) T_i^{n} + \lambda T_{i+1}^{n}$$

Exercicio:
Método Crank-Nicolson (CN).

$$-\lambda T_{i-1}^{n+1} + 2(1+\lambda) T_i^{n+1} - \lambda T_{i+1}^{n+1} = \lambda T_{i-1}^{n} + 2(1-\lambda) T_i^{n} + \lambda T_{i+1}^{n}$$

$$i = 1 \Rightarrow -\lambda T_0^{n+1} + 2(1+\lambda) T_1^{n+1} - \lambda T_2^{n+1} = \lambda T_0^{n} + 2(1-\lambda) T_1^{n} + \lambda T_2^{n}$$

$$i = 2 \Rightarrow -\lambda T_1^{n+1} + 2(1+\lambda) T_2^{n+1} - \lambda T_3^{n+1} = \lambda T_1^{n} + 2(1-\lambda) T_2^{n} + \lambda T_3^{n}$$

$$i = 3 \Rightarrow -\lambda T_2^{n+1} + 2(1+\lambda) T_3^{n+1} - \lambda T_4^{n+1} = \lambda T_2^{n} + 2(1-\lambda) T_3^{n} + \lambda T_4^{n}$$

$$i = 4 \Rightarrow -\lambda T_3^{n+1} + 2(1+\lambda) T_4^{n+1} - \lambda T_5^{n+1} = \lambda T_3^{n} + 2(1-\lambda) T_4^{n} + \lambda T_5^{n}$$

$$i = 5 \Rightarrow -\lambda T_4^{n+1} + 2(1+\lambda) T_5^{n+1} - \lambda T_6^{n+1} = \lambda T_4^{n} + 2(1-\lambda) T_5^{n} + \lambda T_6^{n}$$

$$[A][X] = [B]$$

PDEs parabólicas: Método Crank-Nicolson

Exercicio:
Método Crank-Nicolson (CN).

$$-\lambda T_{i-1}^{n+1} + 2(1+\lambda)T_i^{n+1} - \lambda T_{i+1}^{n+1} = \lambda T_{i-1}^{n} + 2(1-\lambda)T_i^{n} + \lambda T_{i+1}^{n}$$

$$i = 1 \Rightarrow -\lambda T_0^{n+1} + 2(1+\lambda)T_1^{n+1} - \lambda T_2^{n+1} = \lambda T_0^{n} + 2(1-\lambda)T_1^{n} + \lambda T_2^{n}$$

$$i = 2 \Rightarrow -\lambda T_1^{n+1} + 2(1+\lambda)T_2^{n+1} - \lambda T_3^{n+1} = \lambda T_1^{n} + 2(1-\lambda)T_2^{n} + \lambda T_3^{n}$$

$$i = 3 \Rightarrow -\lambda T_2^{n+1} + 2(1+\lambda)T_3^{n+1} - \lambda T_4^{n+1} = \lambda T_2^{n} + 2(1-\lambda)T_3^{n} + \lambda T_4^{n}$$

$$i = 4 \Rightarrow -\lambda T_3^{n+1} + 2(1+\lambda)T_4^{n+1} - \lambda T_5^{n+1} = \lambda T_3^{n} + 2(1-\lambda)T_4^{n} + \lambda T_5^{n}$$

$$i = 5 \Rightarrow -\lambda T_4^{n+1} + 2(1+\lambda)T_5^{n+1} - \lambda T_6^{n+1} = \lambda T_4^{n} + 2(1-\lambda)T_5^{n} + \lambda T_6^{n}$$

$$\begin{bmatrix} 2(1+\lambda) & -\lambda & 0 & 0 & 0 & -\lambda \\ -\lambda & 2(1+\lambda) & -\lambda & 0 & 0 & 0 \\ 0 & -\lambda & 2(1+\lambda) & -\lambda & 0 & 0 \\ 0 & 0 & -\lambda & 2(1+\lambda) & -\lambda & 0 \\ 0 & & 0 & -\lambda & 2(1+\lambda) & -\lambda \\ -\lambda & 0 & 0 & 0 & -\lambda & 2(1+\lambda) \end{bmatrix} \begin{bmatrix} T_0^{n+1} \\ T_1^{n+1} \\ T_2^{n+1} \\ T_3^{n+1} \\ T_4^{n+1} \\ T_{i+1}^{n+1} \end{bmatrix} = \begin{bmatrix} \lambda T_{-1}^{n} + 2(1-\lambda)T_0^{n} + \lambda T_1^{n} \\ \lambda T_0^{n} + 2(1-\lambda)T_1^{n} + \lambda T_2^{n} \\ \lambda T_1^{n} + 2(1-\lambda)T_2^{n} + \lambda T_3^{n} \\ \lambda T_2^{n} + 2(1-\lambda)T_3^{n} + \lambda T_4^{n} \\ \lambda T_3^{n} + 2(1-\lambda)T_4^{n} + \lambda T_5^{n} \\ \lambda T_4^{n} + 2(1-\lambda)T_5^{n} + \lambda T_6^{n} \\ \lambda T_{i-1}^{n} + 2(1-\lambda)T_i^{n} + \lambda T_{i+1}^{n} \end{bmatrix}$$

PDEs parabólicas: Método Crank-Nicolson

Exercicio:
Método Crank-Nicolson (CN).

$$\begin{bmatrix} 2(1+\lambda) & -\lambda & 0 & 0 & 0 & -\lambda \\ -\lambda & 2(1+\lambda) & -\lambda & 0 & 0 & 0 \\ 0 & -\lambda & 2(1+\lambda) & -\lambda & 0 & 0 \\ 0 & 0 & -\lambda & 2(1+\lambda) & -\lambda & 0 \\ 0 & & 0 & -\lambda & 2(1+\lambda) & -\lambda \\ -\lambda & 0 & 0 & 0 & -\lambda & 2(1+\lambda) \end{bmatrix} \begin{bmatrix} T_0^{n+1} \\ T_1^{n+1} \\ T_2^{n+1} \\ T_3^{n+1} \\ T_4^{n+1} \\ T_{i+1}^{n+1} \end{bmatrix} = \begin{bmatrix} \lambda T_{-1}^n + 2(1-\lambda)T_0^n + \lambda T_1^n \\ \lambda T_0^n + 2(1-\lambda)T_,^n + \lambda T_2^n \\ \lambda T_1^n + 2(1-\lambda)T_2^n + \lambda T_3^n \\ \lambda T_2^n + 2(1-\lambda)T_3^n + \lambda T_4^n \\ \lambda T_3^n + 2(1-\lambda)T_4^n + \lambda T_5^n \\ \lambda T_4^n + 2(1-\lambda)T_5^n + \lambda T_6^n \\ \lambda T_{i-1}^n + 2(1-\lambda)T_i^n + \lambda T_{i+1}^n \end{bmatrix}$$

$$\begin{bmatrix} 2(1+\lambda) & -\lambda & 0 & 0 & 0 & -\lambda \\ -\lambda & 2(1+\lambda) & -\lambda & 0 & 0 & 0 \\ 0 & -\lambda & 2(1+\lambda) & -\lambda & 0 & 0 \\ 0 & 0 & -\lambda & 2(1+\lambda) & -\lambda & 0 \\ 0 & & 0 & -\lambda & 2(1+\lambda) & -\lambda \\ -\lambda & 0 & 0 & 0 & -\lambda & 2(1+\lambda) \end{bmatrix} \begin{bmatrix} T_0^{n+1} \\ T_1^{n+1} \\ T_2^{n+1} \\ T_3^{n+1} \\ T_4^{n+1} \\ T_{i+1}^{n+1} \end{bmatrix} = \begin{bmatrix} 2(1-\lambda) & \lambda & 0 & 0 & 0 & \lambda \\ \lambda & 2(1-\lambda) & \lambda & 0 & 0 & 0 \\ 0 & \lambda & 2(1-\lambda) & \lambda & 0 & 0 \\ 0 & 0 & \lambda & 2(1-\lambda) & \lambda & 0 \\ 0 & & 0 & \lambda & 2(1-\lambda) & \lambda \\ \lambda & 0 & 0 & 0 & \lambda & 2(1-\lambda) \end{bmatrix} \begin{bmatrix} T_0^n \\ T_1^n \\ T_2^n \\ T_3^n \\ T_4^n \\ T_{i+1}^n \end{bmatrix}$$

Simbolicamente, a equação pode ser escrita

$$M_1 U_i^{n+1} = M_2 U_i^n$$

Exercicio:
Método Crank-Nicolson (CN).

Simbolicamente, a equação pode ser escrita

$$M_1 U_i^{n+1} = M_2 U_i^n$$

A solução formal disso é trivial:

$$U_i^{n+1} = {M_1}^{-1} M_2 U_i^n$$

No entanto, isso requer a inversão de uma matriz M × M. Existem maneiras muito melhores de resolver isso.

A matriz M_1 é tri-diagonal periódica. Existem muito métodos numéricos eficientes de inverter um sistema com tal matriz.

Exercicio:
Método Crank-Nicolson (CN).

O problema não periódico, com dados U_0^n e U_M^n, resulta em uma matriz ligeiramente diferente, mas também tri-diagonal.

Se os termos não lineares são tratados implicitamente, devemos resolver um sistema algébrico não linear a cada passo de tempo.

Isso normalmente é impraticável.

A possibilidade de usar um passo de tempo com um número de Courant muito maior que 1 em um esquema implícito não garante que obteremos resultados precisos e econômico computacionalmente.

O esquema implícito mantém a estabilidade retardando as soluções, de modo que as ondas satisfaçam a condição CFL.

Exercicio:
Método Crank-Nicolson (CN).

Por esta razão, ***esquemas implícitos são úteis para aqueles modos que são muito rápidos***, mas de pouca importância meteorológica.

Em Estudos futuros, consideraremos esquemas nos quais ***os termos da onda gravitacional são implícitos*** enquanto os termos restantes são explícitos.

Esses esquemas semi-implícitos são de importância crucial na NWP moderna.

PDEs parabólicas: Método Crank-Nicolson

Exercicio:
Método Crank-Nicolson (CN).

$$\begin{bmatrix} 2(1+\lambda) & -\lambda & 0 & 0 & 0 & -\lambda \\ -\lambda & 2(1+\lambda) & -\lambda & 0 & 0 & 0 \\ 0 & -\lambda & 2(1+\lambda) & -\lambda & 0 & 0 \\ 0 & 0 & -\lambda & 2(1+\lambda) & -\lambda & 0 \\ 0 & & 0 & -\lambda & 2(1+\lambda) & -\lambda \\ -\lambda & 0 & 0 & 0 & -\lambda & 2(1+\lambda) \end{bmatrix} \begin{bmatrix} T_0^{n+1} \\ T_1^{n+1} \\ T_2^{n+1} \\ T_3^{n+1} \\ T_4^{n+1} \\ T_{i+1}^{n+1} \end{bmatrix} = \begin{bmatrix} 2(1-\lambda) & \lambda & 0 & 0 & 0 & \lambda \\ \lambda & 2(1-\lambda) & \lambda & 0 & 0 & 0 \\ 0 & \lambda & 2(1-\lambda) & \lambda & 0 & 0 \\ 0 & 0 & \lambda & 2(1-\lambda) & \lambda & 0 \\ 0 & & 0 & \lambda & 2(1-\lambda) & \lambda \\ \lambda & 0 & 0 & 0 & \lambda & 2(1-\lambda) \end{bmatrix} \begin{bmatrix} T_0^{n} \\ T_1^{n} \\ T_2^{n} \\ T_3^{n} \\ T_4^{n} \\ T_{i+1}^{n} \end{bmatrix}$$

$$M_1 U_i^{n+1} = M_2 U_i^n$$

$$U_i^{n+1} = {M_1}^{-1} M_2 U_i^n$$