



# Mini-Curso do MCGA-CPTEC/INPE

•*Paulo Yoshio Kubota*  
•*CPTEC, C. Paulista, Brasil*  
•*17 agosto, 2015*



## **Universidade Estatual da Amazônia-UEA**

### **Minicurso: MCGA do CPTEC/INPE: Teoria, Desenvolvimento;(em preparação)**

**Data 14/09 (14h-17) –**

**Palestra: MCGA do CPTEC/INPE: Teoria,  
Desenvolvimento e Aplicações - Parte I;**

**Data 15,16,17,18/09 (9h-12) –**

**Palestra: MCGA do CPTEC/INPE: Teoria,  
Desenvolvimento e Aplicações - Parte II;**

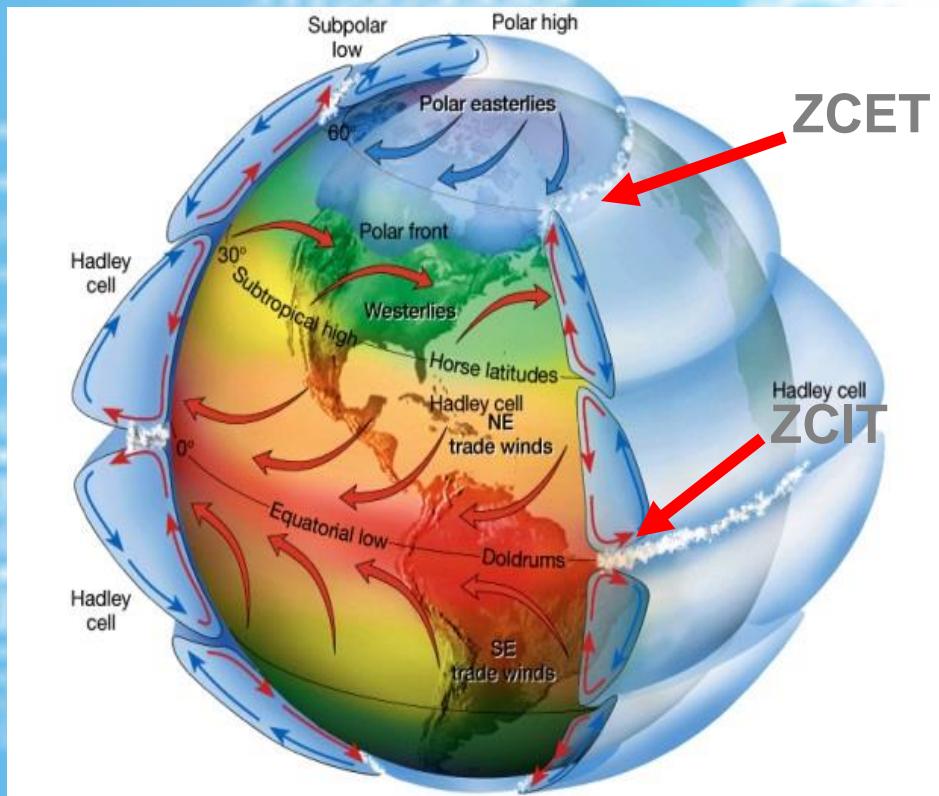
# Teoria

## O que a dinâmica Atmosférica deve Resolver?

MESOESCALA					
	1 MÊS	1 DIA	1 HORA	1 MINUTO	1 SEG.
10,000 Km	ondas longas	marés			
2,000 Km	ondas baroclinicas				
200 Km	frentes e furacões				
20 Km	jatos baixos ondas inertias				
2 Km		tempestades			
200 m	tornados convecção profunda ondas de gravidade				
20 m	poeiras				
	plumas rugosidade turbulência				MICRO $\gamma$ ESCALA
					MICRO $\beta$ ESCALA
					MICRO $\alpha$ ESCALA
					DEFINIÇÃO
	ESCALA CLIMATOLÓGICA	ESCALA SINÓTICA	MESO ESCALA	MICRO ESCALA	

# Teoria

## O que a dinâmica Atmosférica deve Resolver? R: Circulações de Grande Escala



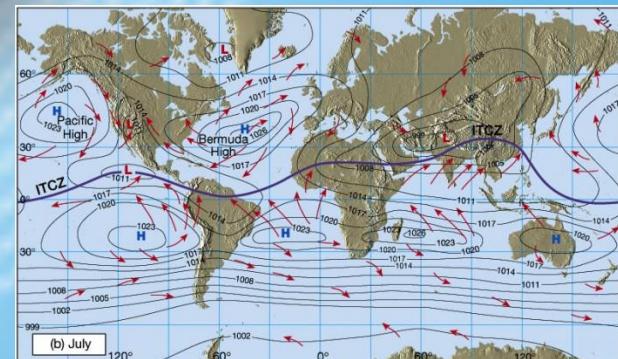
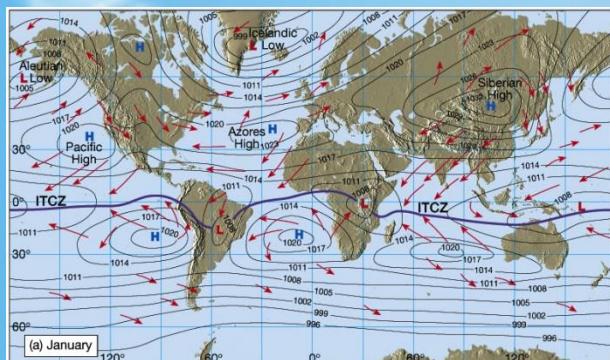
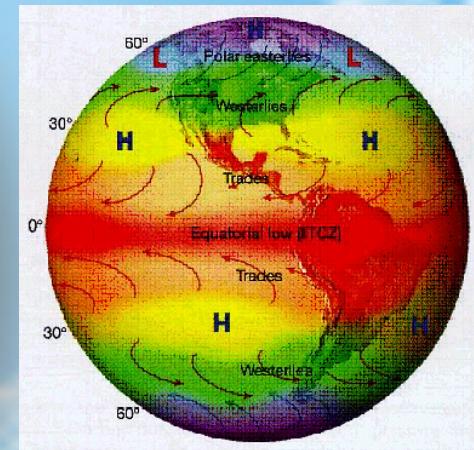
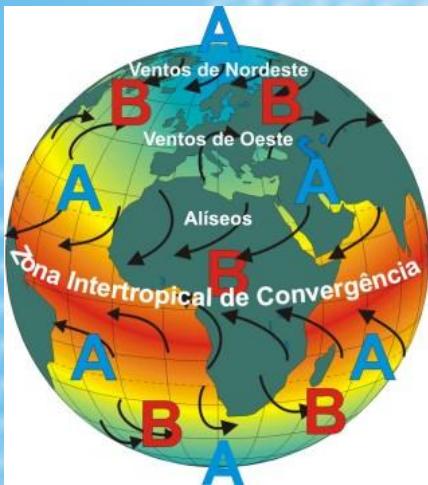
Interações entre as células de circulação de grande escala:  
Polar , Ferrel, Hadley e Walker



Impacto no clima?

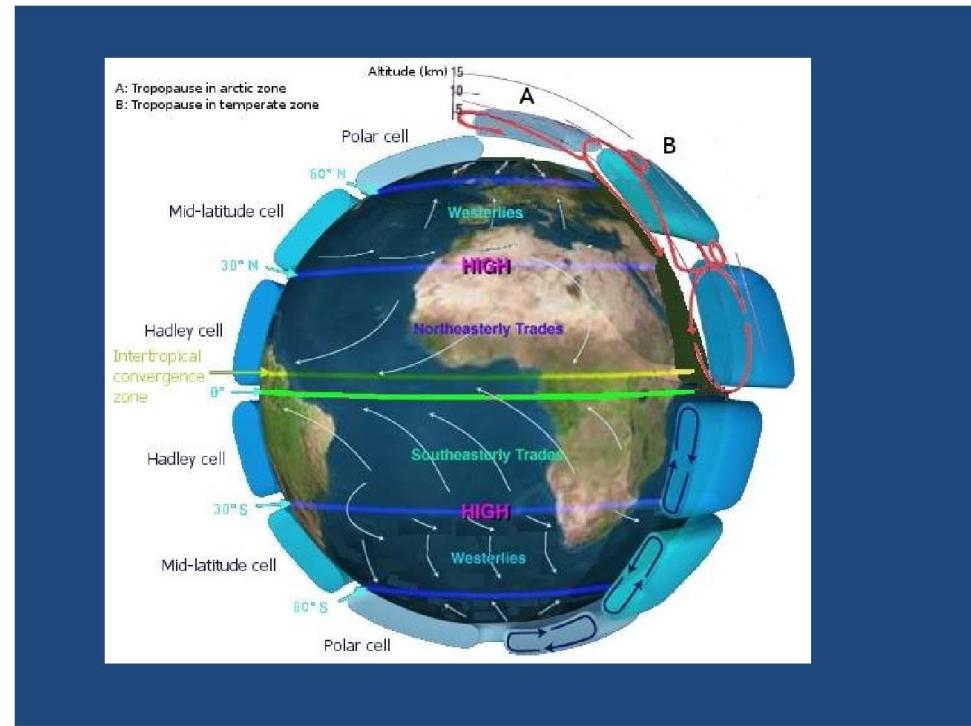
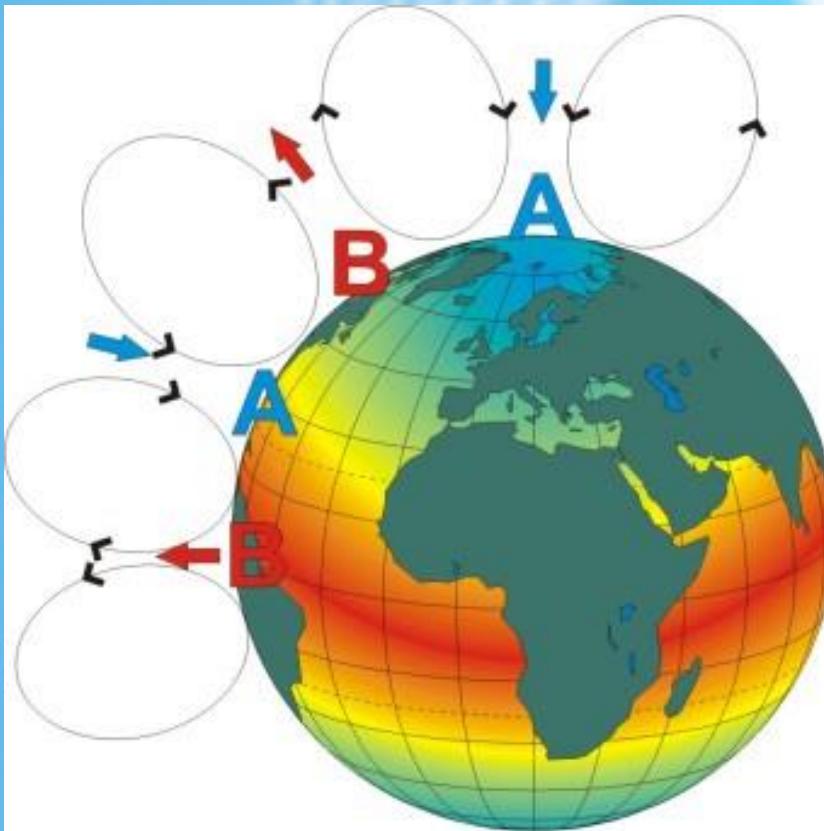
## Teoria

O que a dinâmica Atmosférica deve Resolver?  
R:Distribuição da Pressão Atmosférica com a Latitude



## Teoria

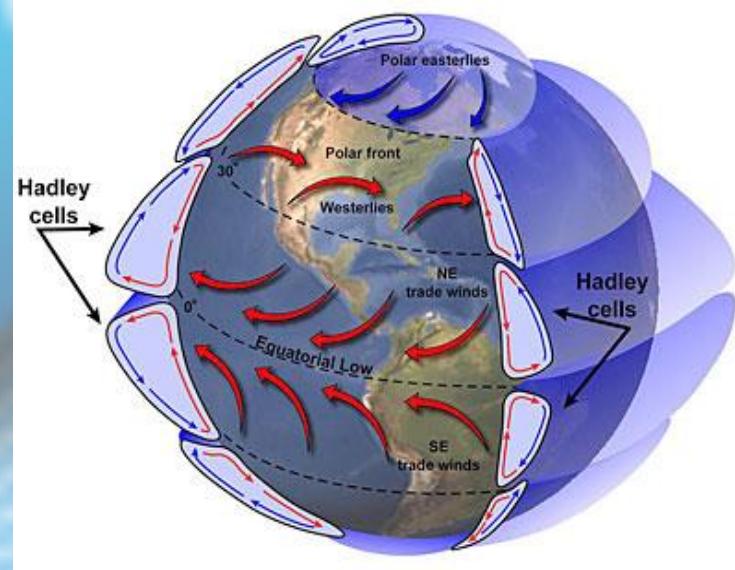
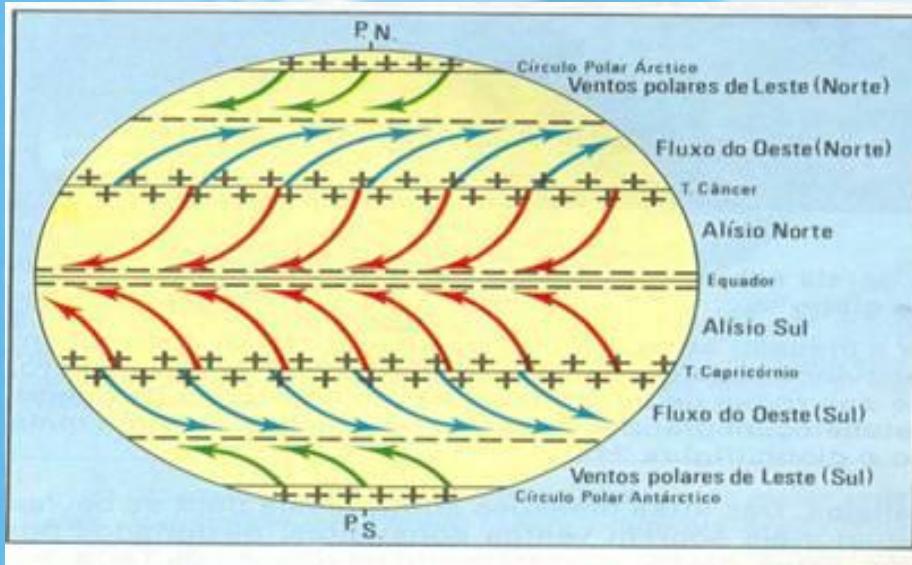
# O que a dinâmica Atmosférica deve Resolver? R:Células de Hadley, Ferrel e Polar



## Teoria

# O que a dinâmica Atmosférica deve Resolver?

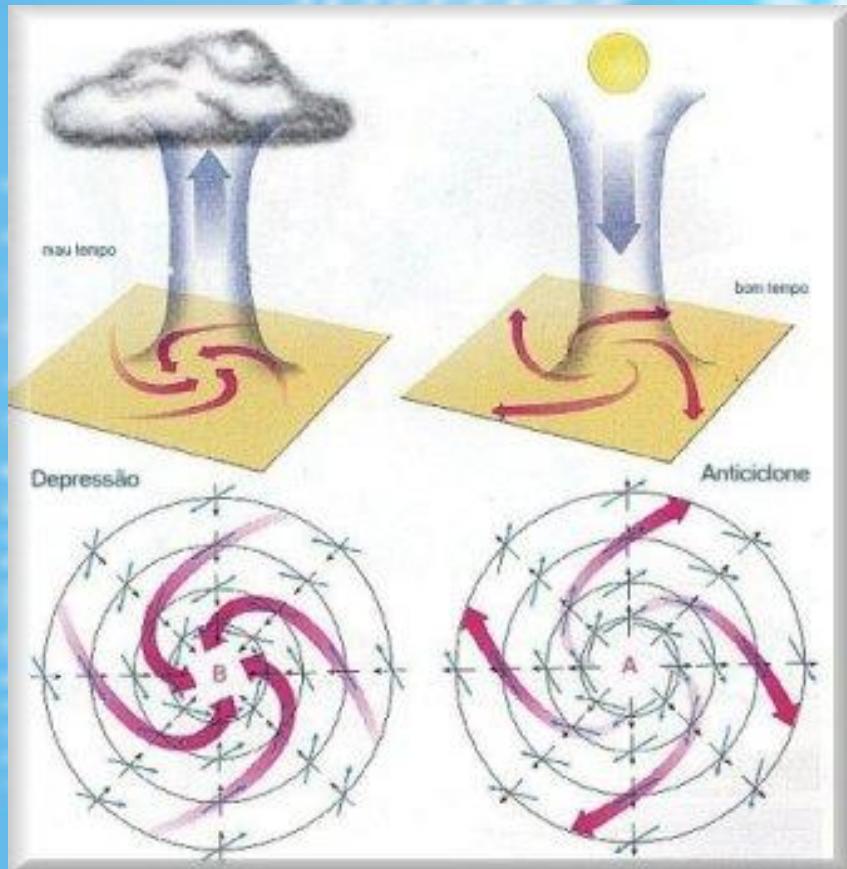
## R: Ventos: Alíseos; Oeste; Leste



- Rotação da Terra transferência de momentum.

# Teoria

## Circulação do Ar nos Centros de Alta e Baixa Pressão Estados de Tempo



- Centro de Baixas Pressões ou Depressão  
Convergência e Ascendência do ar
- Efeito: Instabilidade Atmosférica
- Centro de Altas Pressões ou Anticiclone  
Subsidiência e Divergência do ar
- Efeito: Estabilidade Atmosférica

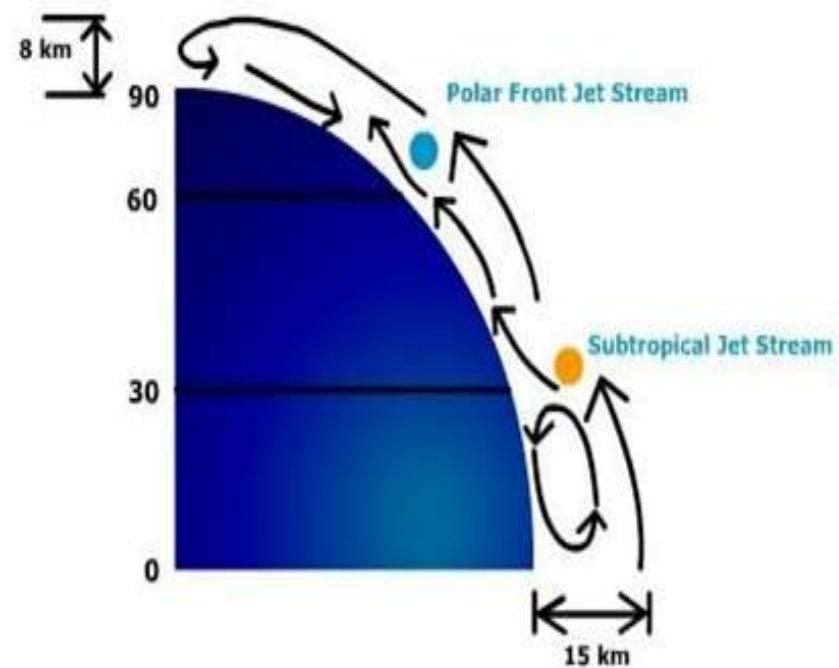
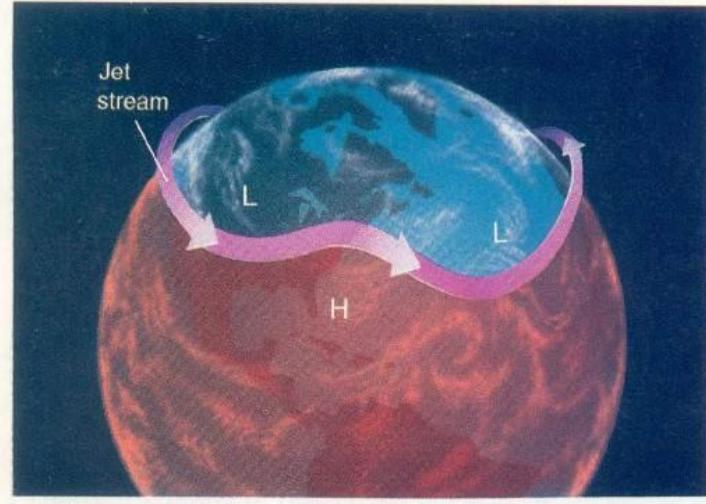
Tornados e furacões

## Teoria

# O que a dinâmica Atmosférica deve Resolver?

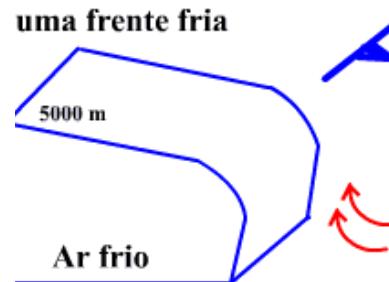
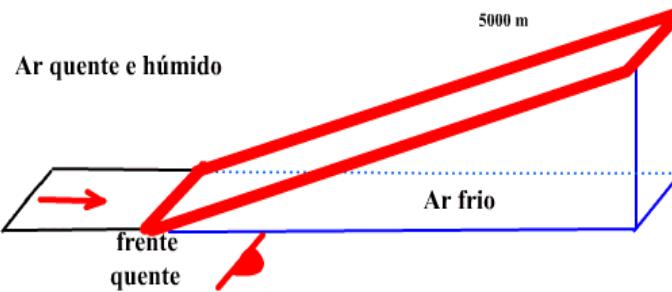
## R:CORRENTES DE JATO

Ventos de oeste e correntes de jato

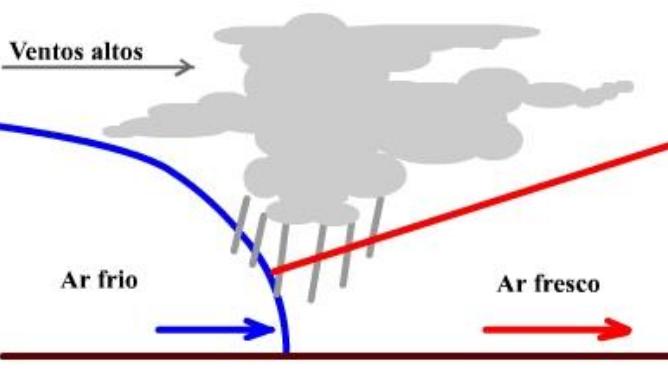
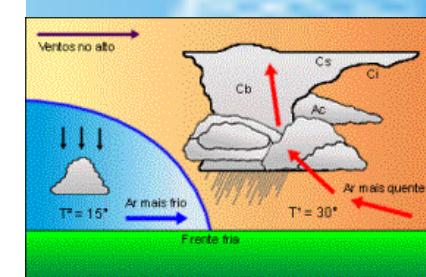


## Teoria

# O que a dinâmica Atmosférica deve Resolver? R:Frentes: Quente, Fria, Oclusa, Estacionária

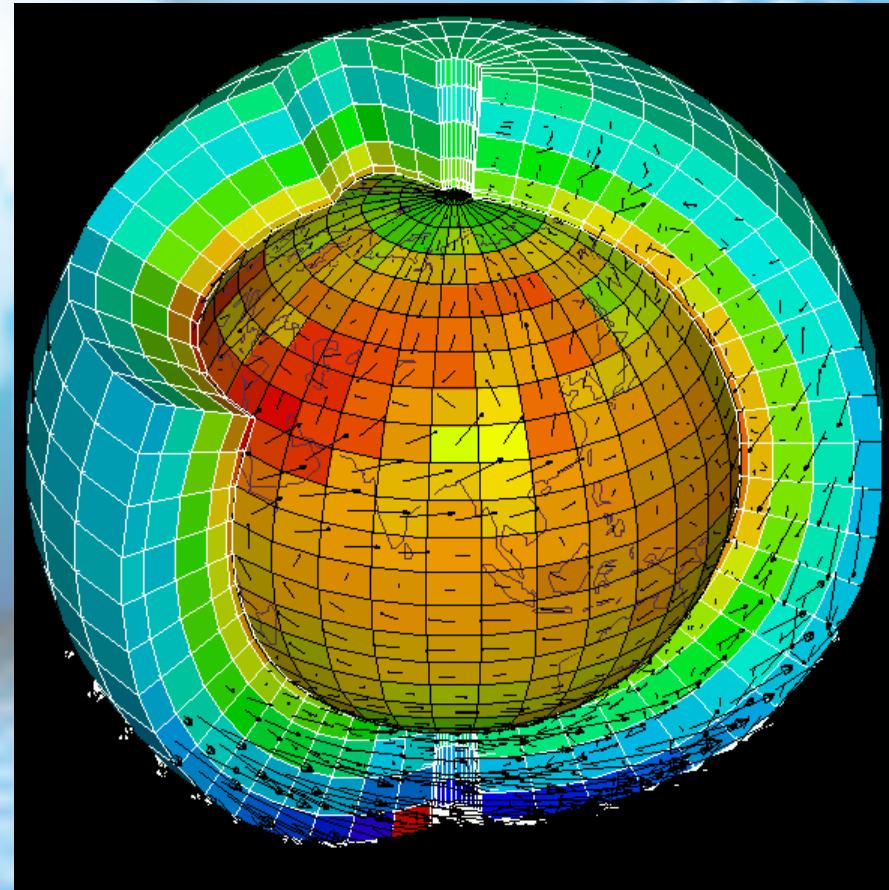
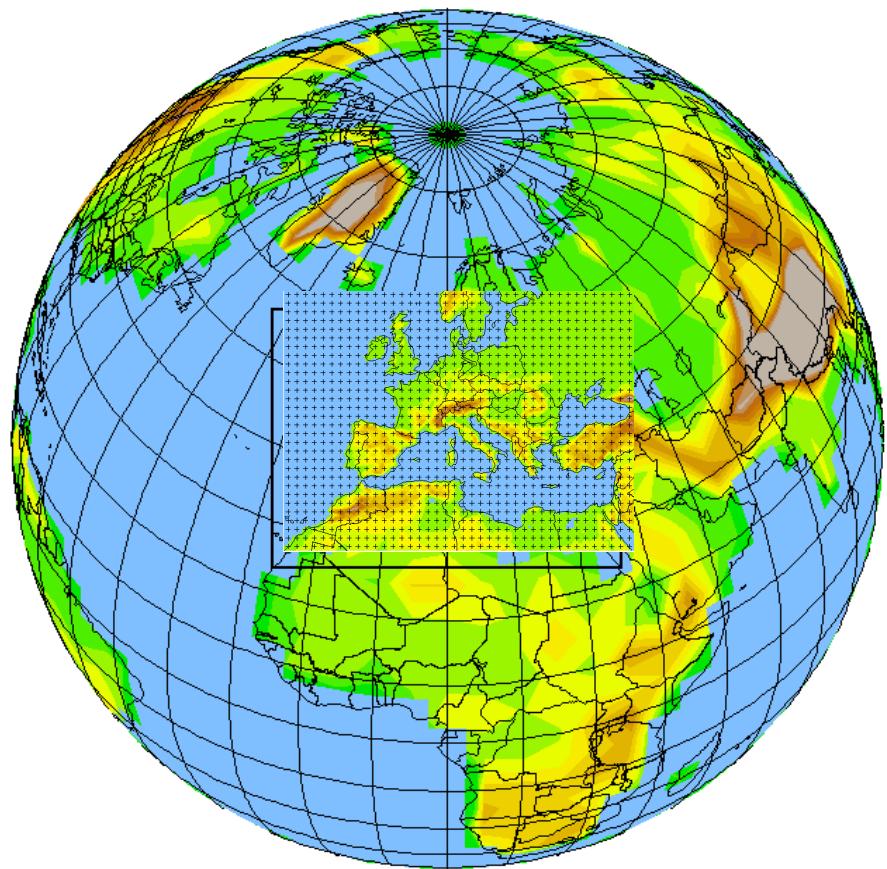


Ar quente e húmido



## Teoria

O que a dinâmica Atmosférica deve Resolver?  
A circulação Atmosférica em ponto de grade



# Teoria O “Núcleo dinâmico” de um Modelo Climático Coordenada Cartesiana

Conservation  
of momentum

Conservation  
of energy

Conservation  
of mass

Conservation of water  
(or chemical tracer)

Equation of state

$$\frac{\partial \bar{V}}{\partial t} + \bar{V} \cdot \nabla \bar{V} = -\frac{\nabla p}{\rho} - 2\bar{\Omega} \times \bar{V} + \bar{g} + F_V$$

$$C_p \left( \frac{\partial T}{\partial t} + \bar{V} \cdot \nabla T \right) = \frac{1}{\rho} \frac{dp}{dt} + Q + F_T$$

$$\frac{\partial \rho}{\partial t} + \bar{V} \cdot \nabla \rho = -\rho \nabla \cdot \bar{V}$$

$$\frac{\partial q}{\partial t} + \bar{V} \cdot \nabla q = \frac{S_q}{\rho} + F_q$$

$$p = \rho R T$$

Gravity

Coriolis force

Pressure gradient  
force

Adiabatic heating

Compressibility

## Teoria

# O “Núcleo dinâmico” de um Modelo Climático Coordenada Esfericas

### Equação de momentum

$$\frac{\partial U}{\partial t} + \frac{1}{a \cos^2 \varphi} (U \frac{\partial U}{\partial \lambda} + V \cos \varphi \frac{\partial U}{\partial \varphi}) + \dot{\sigma} \frac{\partial U}{\partial \sigma} - fV + \frac{1}{a} \left( \frac{\partial \Phi}{\partial \lambda} + RT \frac{\partial \ln ps}{\partial \lambda} \right) = F_u$$

$$\frac{\partial V}{\partial t} + \frac{1}{a \cos^2 \varphi} (U \frac{\partial V}{\partial \lambda} + V \cos \varphi \frac{\partial V}{\partial \varphi}) + \dot{\sigma} \frac{\partial V}{\partial \sigma} + fU + \frac{\cos \varphi}{a} \left( \frac{\partial \Phi}{\partial \varphi} + RT \frac{\partial \ln ps}{\partial \varphi} \right) + \frac{\sin \varphi}{a \cos^2 \varphi} (U^2 + V^2) = F_v$$

### Equação de Termodinâmica

$$\frac{\partial T}{\partial t} + \frac{1}{a \cos^2 \varphi} (U \frac{\partial T}{\partial \lambda} + V \cos \varphi \frac{\partial T}{\partial \varphi}) + \dot{\sigma} \frac{\partial T}{\partial \sigma} - \theta \dot{\sigma} \frac{\partial \Pi}{\partial \sigma} = \kappa T \left( \frac{\partial}{\partial t} + \vec{V} \cdot \nabla \right) \ln ps + F_T$$

### Equação de Umidade

$$\frac{\partial q}{\partial t} + \frac{1}{a \cos^2 \varphi} (U \frac{\partial q}{\partial \lambda} + V \cos \varphi \frac{\partial q}{\partial \varphi}) + \dot{\sigma} \frac{\partial q}{\partial \sigma} = F_q$$

Variables:

$$U = u \cos \varphi, V = v \cos \varphi, \vec{V} = (U, V)$$

$T$  - virtual temperature,  $\theta$  - potential temperature,  $T = \Pi \theta$   
 $ps$  - surface pressure,  $\sigma = p/ps$  - vertical coordinate,  $\dot{\sigma}$  - vertical velocity,  
 $f$  - Coriolis,  $q$  - specific humidity,  $D$  - horizontal divergence.  $F_U$ ,  $F_V$ ,  $F_T$  and  
 $F_q$  are forcing terms due to the physical parameterization processes.

# Teoria

## Dinâmica

### Equação de Pressão de Superfície

$$\frac{\partial \ln ps}{\partial t} + \int_0^1 (\vec{V} \cdot \nabla \ln ps) d\sigma + \int_0^1 D d\sigma = 0$$

### Equação de Hidrostática

$$\frac{\partial \phi}{\partial \sigma} + \frac{RT}{\sigma} = 0$$

### Equação de Velocidade Vertical

$$\sigma \frac{\partial ps}{\partial t} + \int_0^\sigma \nabla \cdot (ps \vec{V}) d\sigma = -ps \dot{\sigma}$$

Variables:

$$U = u \cos \varphi, V = v \cos \varphi, \vec{V} = (U, V)$$

$T$  - virtual temperature,  $\theta$  - potential temperature,  $T = \Pi \theta$   
 $ps$  - surface pressure,  $\sigma = p/ps$  - vertical coordinate,  $\dot{\sigma}$  - vertical velocity,  
 $f$  - Coriolis,  $q$  - specific humidity,  $D$  - horizontal divergence.  $F_U$ ,  $F_V$ ,  $F_T$  and  
 $F_q$  are forcing terms due to the physical parameterization processes.

# Dinâmica

## As Variáveis Prognósticas

D – Campo de Divergência

$\xi$  - Campo de Vorticidade

As velocidades U e V serão derivadas de  $\xi$  e D.

T – temperatura Virtual

Q – Umidade específica

Ln ps – log da pressão de superfície

Será armazenado no espaço espectral . Em cada nível k, nós armazenaremos os coeficientes  $F_n^m$  de uma expansão tal como:

$$F(\lambda, \varphi) = \sum_{m=-M}^{M} \sum_{n=|m|}^{M} F_n^m P_n^m(\sin \varphi) e^{im\lambda}$$

Onde, é utilizado um truncamento triangular para cada campo prognostico

# Dinâmica

## Duas Formas de Discretização das equações primitivas no espaço espectral

### euleriana

Time step segue o critério de estabilidade de CFL.

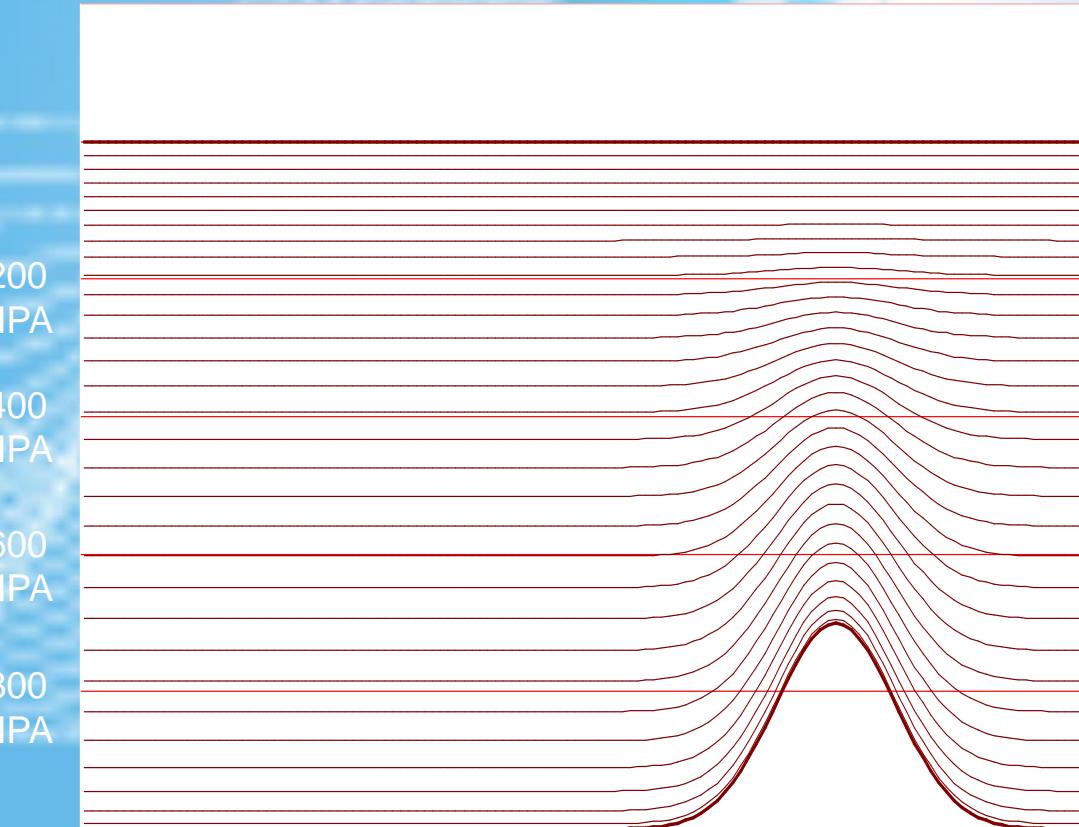
- ✓ Utilizado para baixas resoluções > (T299L64) ~ 40 km

### Semi-Lagrangiana

- ✓ Suporta até 5 vezes o time step da discretização euleriana.
- ✓ Utilizado para altas resoluções < (T299L64) ~ 40 km

As parametrizações físicas responsáveis pelas tendências não lineares estão no espaço físico em um grade regular gaussiana

## O "núcleo" dinâmicos de um modelo Discretização Vertical



Coordenada Vertical  
O terreno seguinte  
 $(\Sigma)$   
Altura (z)  
Pressão (p)  
Hybrid



# Métodos Numéricos para Modelagem do Sistema Terrestre

# Revisão de Método Numérico

- **Introdução**
- **Equações diferenciais Ordinárias**
- **Advecção Linear**
  - Análise de estabilidade para os esquemas downstream, upstream e leapfrog
  - Modo computacional e filtragem no tempo
  - Tópicos Avançados: introdução ao semi-Lagrangiano, esquema Implícito, Sistemas não-lineares, advecção
- **Divulgação problema**
  - Análise de estabilidade
  - Problema de difusão pura 1D
- **Problema combinado advecção - difusão**
- **Equação da onda de gravidade linear 1D**

# **Leitura recomendada**

**Durran D. R. (1999) Numerical Methods for Wave Equations in Geophysical Fluid Dynamics. New York, Springer-Verlag.**

**Haltiner, G.J. and Williams, R.T. (1980) Numerical prediction and dynamic meteorology. 2nd ed**

**ECMWF Training Lecture notes**

(<http://www.ecmwf.int/newsevents/training/rcourse-notes/NUMERICAL-METHODS/>)

# Introdução

**As diferenças entre soluções analíticas e métodos numéricos**

- **Soluções numéricas são os seguintes:**
  - **Valor aproximado**
  - **Sempre particular para um determinado conjunto de parâmetros**
- **Soluções analíticas são as seguintes:**
  - **Exatamente**
  - **Global**

**As principais fontes de erros para soluções numéricas são :**

- **Round-off**
- **Truncamento**
- **Instabilidade**

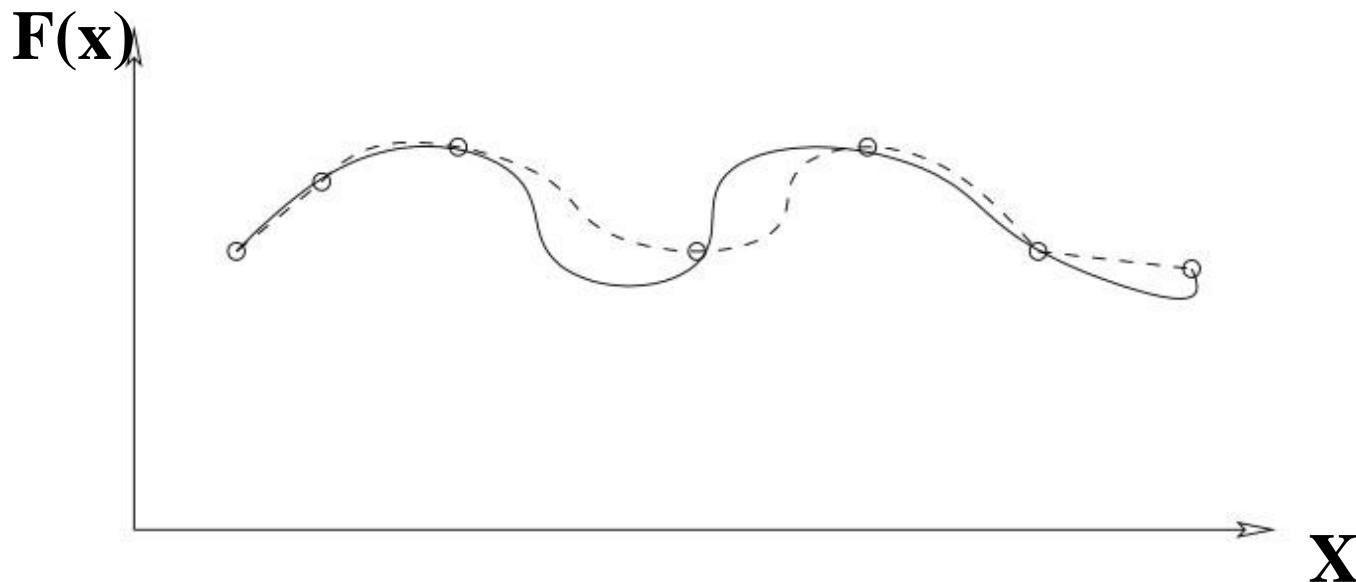
# erro Round-off

- Em geral, isso é **menos importante do que as outras fontes de erro**  
**No entanto, torna-se importante quando**
- Adicionar os números pequenos em um numero grandes,  
especialmente quando se está acumulando somas
  - e.g.  $\hat{x} = \frac{1}{N} \sum_{i=1}^N x_i; N = 10^8$ ; let  $x_i = 1$  and  $i = 1, \dots, 10^8$ 
    - A soma do primeiro número **9.999.999** é **0.9999999e7**.
    - Adicionar o próximo está **0.1000000e1**. (**= 0.0000001e7**) dá **0.1000000e8**
    - O próximo  $x_i$  é convertido em **0.0000000e8** e este  $x_i$  Não Alterar a soma.
    - O mesmo será verdade para todos os  $x_i$  e o último Resultado é  **$10^7$**  e a média será  **$10^7/10^8 = 0,1$** .
  - Devido a esse erro, é uma boa prática a utilização de Variáveis de precisão dupla para acumular somas.

```
PROGRAM Main
IMPLICIT NONE
INTEGER, PARAMETER :: N=100000000
REAL (KIND=4) :: X
INTEGER :: i
X=0.0
DO i=1, N
    X=X+1.0
    IF (i > X) THEN
        PRINT*,X,i
        exit
    END IF
END DO
PRINT*,X,X/N
END PROGRAM Main
```

# Erro de truncamento

- Truncamento é produzido quando funções contínuas são representadas por uma série de valores de ponto.
- Em geral, quando mais valores são utilizados para aproximar a função mais precisa a aproximação será.



- A resolução necessária para reduzir o erro de truncamento para um nível aceitável depende da função a ser aproximados.

# Resolver Numericamente as Equações Diferenciais

## Equação do momentum

$$\frac{\partial \vec{V}}{\partial t} + \vec{V} \cdot \nabla \vec{V} = -\frac{1}{\rho} \nabla P + \vec{g} + \mu \nabla^2 u$$

Simplificando a equação para uma dimensão local e considerando somente aceleração convectiva

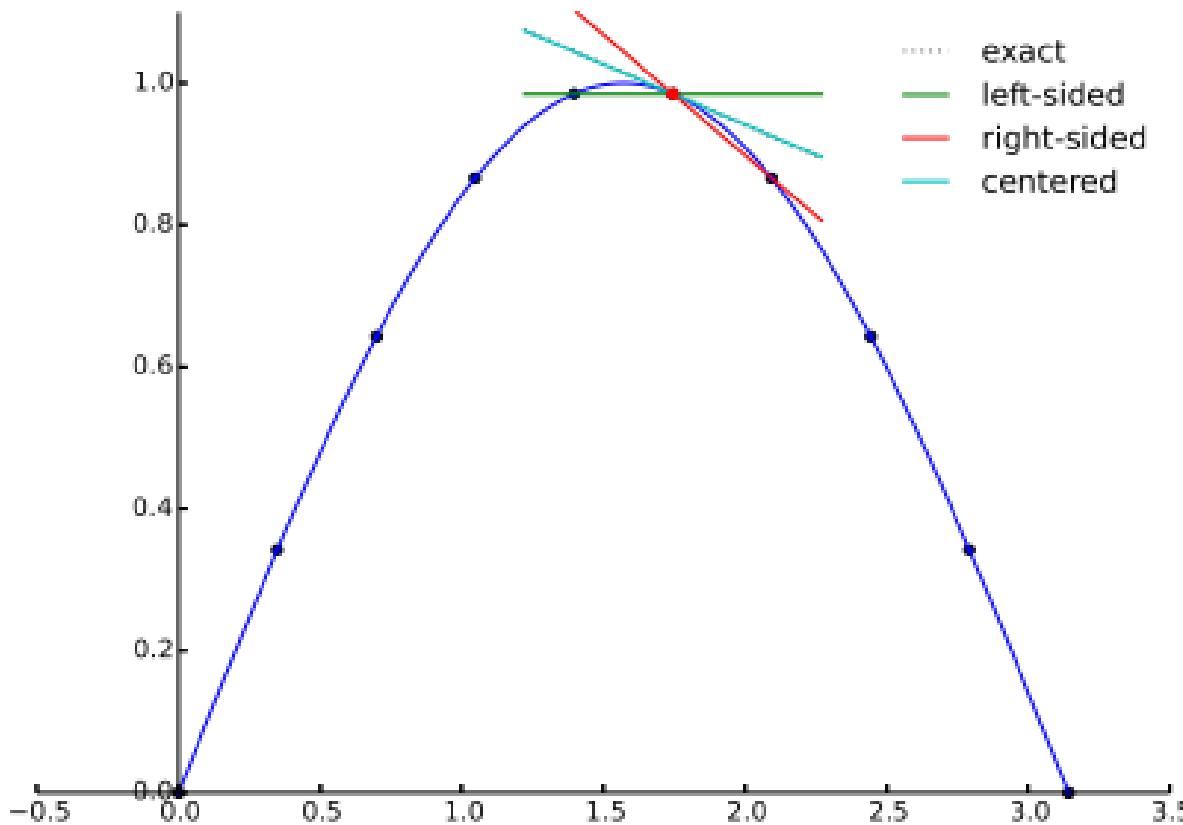
$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = 0$$

Como representar as derivadas espaciais e temporais como função discretas

# Diferença numérica aproximação de diferenciação

## Diferenciação

Considere-se uma coleção de pontos espaçados, marcado com um índice  $i$ , com o espaçamento físico entre elas é definido como  $\Delta x$ . Nós podemos expressar a primeira derivada de uma quantidade  $a$  em um ponto  $i$  na como:



- exact
- left-sided
- right-sided
- centered

$$\left. \frac{da}{dx} \right|_i \approx \frac{a_i - a_{i-1}}{\Delta x}$$

$$\left. \frac{da}{dx} \right|_i \approx \frac{a_{i+1} - a_i}{\Delta x}$$

$$\left. \frac{da}{dx} \right|_i \approx \frac{a_{i+1} - a_{i-1}}{2\Delta x}$$

# Diferença numérica aproximação de diferenciação

Aproximações por diferença finita para derivadas são baseadas na série de Taylor Truncada.

$$f(x_{i+1}) = f(x + \Delta x) = f(x_i) + \Delta x f'(x_i) + \frac{(\Delta x)^2}{2} f''(x_i) + \frac{(\Delta x)^3}{6} f'''(x_i) + O(\Delta x^4) \quad (1)$$

$$f(x_{i-1}) = f(x - \Delta x) = f(x_i) - \Delta x f'(x_i) + \frac{(\Delta x)^2}{2} f''(x_i) - \frac{(\Delta x)^3}{6} f'''(x_i) + O(\Delta x^4) \quad (2)$$

A reorganização das Eq. 1. Dá a Aproximação por diferença forward

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{\Delta x} + O(\Delta x), \quad (3)$$

Enquanto Eq. 2 Dá a Aproximação por diferença backward

$$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{\Delta x} + O(\Delta x). \quad (4)$$

Estes são chamados aproximações de primeira ordem porque o Erro Truncamento é proporcional a  $\Delta x$  a potência de 1.

# Diferença finita ...

Para Aproximação de diferença centralizada segunda-ordem: subtraindo Eq. 2 da Eq. 1. E a reorganização leva a:

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2\Delta x} + O(\Delta x^2). \quad (5)$$

Quando  $\Delta x$  Aproxima-se de zero,  $\Delta x^2$  Aproxima-se de zero muito mais rápido em relação a  $\Delta x$ .

Portanto, uma aproximação de segunda ordem irá convergir mais rápido do que uma aproximação primeira-ordem.

Uma aproximação para a segunda derivada pode ser encontrado adicionando a Eq. 2 na Eq. 1.

$$f''(x_i) = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{\Delta x^2} + O(\Delta x^2). \quad (6)$$

Esta fórmula tem precisão de segunda ordem.

# Equações diferenciais Ordinárias

Considere uma equação diferencial ordinária linear

$$\frac{dy}{dt} = -\lambda y \quad (7)$$

Onde  $\lambda > 0$ , Com condição inicial ( $t = 0$ )  $\rightarrow y(0) = y_0$

Como sabe-se que a solução exata é  $y = y_0 e^{-\lambda t}$

A derivada temporal da Eq. 7º pode ser aproximadas utilizando a diferenciação forward

$$\frac{y^{n+1} - y^n}{\Delta t} = -\lambda y^n \quad (8)$$

Onde  $y^n$  e significa que  $y(t_n)$  e  $t_N = N\Delta t$ ,  $n = 0, 1, 2$

Assim

$$y^{n+1} = (1 - \lambda\Delta t)y^n \quad (9)$$

Ou

$$y^{n+1} = (1 - \lambda\Delta t)^n y^0.$$

# Exatidão e Estabilidade

- Se o erro de truncamento do esquema de diferença finita se aproxima de zero, quando  $\Delta t \rightarrow 0$ , então, o esquema é coerente.
- Para um problema de valor inicial, a coerência não é suficiente para garantir que um esquema numérico vá convergir para as soluções corretas quando  $\Delta t \rightarrow 0$  ( $e\Delta x \rightarrow 0$ ).
- O teorema equivalência de Lax, diz que para os estados consistentes e método linear, a estabilidade é a condição necessária e suficiente para a convergência.
- Para resolver um problema de valor inicial o erro roundoff e de truncamento, irá acumular com cada passo de tempo, e a solução aproximada às vezes vai Divergir da solução correta. Se o passo de tempo  $\Delta t$  é pequeno o erro a cada intervalo de tempo deve ser pequeno e deve se acumular mais lentamente.
- No entanto, algumas vezes, a solução numérica irá rapidamente divergir da solução correta, caso em que o método numérico é dito ser Instável.

# Análise de Estabilidade edo linear

Assumir que a solução para eq. 7º tem uma forma  $y^n \approx A^n$  e substitua no Esquema Forward ou seja Eqn. numérica 9º

$$\begin{aligned}A^{n+1} &= A^n(1 - \lambda\Delta t) \\ \Rightarrow A^n A &= A^n(1 - \lambda\Delta t) \\ \Rightarrow A &= (1 - \lambda\Delta t)\end{aligned}$$

$$|1 - \lambda\Delta t| \leq 1$$

$$\begin{cases} (1 - \lambda\Delta t) \leq 1 & \text{se } \Delta t \geq 0 \\ (1 - \lambda\Delta t) \geq -1 & \text{se } \Delta t \leq 2/\lambda \end{cases}$$

$A^n$  é chamado de fator de ampliação

O sistema é estável, se  $|A| \leq 1$

$$|1 - \lambda\Delta t| \leq 1$$

$$\therefore \Delta t \leq 2/\lambda$$

N.B. Para uma equação não linear, a análise de estabilidade da forma linearizada da equação não linear é uma condição necessária, mas não é suficiente.

**MODULE** Class\_NumericalScheme**IMPLICIT NONE****PRIVATE****INTEGER, PUBLIC , PARAMETER :: r8=8**  
**INTEGER, PUBLIC , PARAMETER :: r4=4****REAL (KIND=r8) :: dt**  
**REAL (KIND=r8) :: lambda**  
**INTEGER , PARAMETER :: UnitData=1**  
**INTEGER , PARAMETER :: UnitCtl=2**  
**CHARACTER (LEN=400) :: FileName**  
**LOGICAL :: CtrlWriteDataFile**  
**PUBLIC :: InitNumericalScheme**  
**PUBLIC :: SchemeForward**  
**PUBLIC :: SchemeUpdate**  
**PUBLIC :: SchemeWriteData**  
**PUBLIC :: SchemeWriteCtl**  
**PUBLIC :: AnaliticFunction****CONTAINS****SUBROUTINE** InitNumericalScheme(dt\_in,lambda\_in)**IMPLICIT NONE****REAL (KIND=r8) :: dt\_in**  
**REAL (KIND=r8) :: lambda\_in**

FileName="

dt=dt\_in

lambda=lambda\_in

FileName='EDOL'

CtrlWriteDataFile=.TRUE.

**END SUBROUTINE** InitNumericalScheme**FUNCTION** SchemeForward(yc) **RESULT**(yp)**IMPLICIT NONE****! Utilizando a diferenciacao forward**  
!  
**y(n+1) - y(n)**  
----- = lambda \* y(n) ; onde lambda > 0  
!  
dt

!

**REAL (KIND=r8), INTENT (IN ) :: yc**  
**REAL (KIND=r8) :: yp**

yp = yc -lambda\*dt\*yc

**END FUNCTION** SchemeForward**FUNCTION** SchemeUpdate(y\_in) **RESULT** (y\_out)**IMPLICIT NONE****REAL (KIND=r8), INTENT(IN) :: y\_in**  
**REAL (KIND=r8) :: y\_out**  
y\_out=y\_in**END FUNCTION** SchemeUpdate**FUNCTION** AnaliticFunction(y0,tn) **RESULT** (y\_out)**IMPLICIT NONE****REAL (KIND=r8), INTENT (IN) :: y0**  
**INTEGER , INTENT (IN) :: tn****REAL (KIND=r8) :: y\_out**  
y\_out=y0\*exp(-lambda\*tn\*dt)**END FUNCTION** AnaliticFunction

```

FUNCTION SchemeWriteData(irec,y_in,ya) RESULT(ok)
IMPLICIT NONE
INTEGER , INTENT (INOUT) :: irec
REAL (KIND=r8), INTENT (IN) :: y_in
REAL (KIND=r8), INTENT (IN) :: ya
INTEGER :: ok
INTEGER :: irec
REAL (KIND=r4) :: Yout
INQUIRE (IOLENGTH=irec) Yout

IF(CtrlWriteDataFile)OPEN(UnitData,FILE=TRIM(FileName)//'.
bin',& FORM='UNFORMATTED', ACCESS='DIRECT',
STATUS='UNKNOWN', &
ACTION='WRITE',RECL=irec)
CtrlWriteDataFile=.FALSE.
Yout=REAL(y_in,KIND=r4)
irec=irec+1
WRITE(UnitData,rec=irec) Yout
irec=irec+1
WRITE(UnitData,rec=irec) REAL (ya,KIND=r4)
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='FORMATTE
D',&

ACCESS='SEQUENTIAL',STATUS='UNKNOWN',ACTION='WRI
TE')
WRITE (UnitCtl,'(A6,A)')'dset ^',TRIM(FileName)//'.bin'
WRITE (UnitCtl,'(A )')'title EDO'
WRITE (UnitCtl,'(A )')'undef -9999.9'
WRITE (UnitCtl,'(A )')'xdef 1 linear -48.00 1'
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 2'
WRITE (UnitCtl,'(A )')'yc 0 99 resultado da edol yc'
WRITE (UnitCtl,'(A )')'ya 0 99 funcao analitica'
WRITE (UnitCtl,'(A )')'endvars'
CLOSE (UnitCtl,STATUS='KEEP')
CLOSE (UnitData,STATUS='KEEP')

ok=0
END FUNCTION SchemeWriteCtl

END MODULE Class_NumericalScheme

```

**PROGRAM MAIN**

USE Class\_NumericalScheme, Only :r8, InitNumericalScheme, SchemeForward,  
SchemeUpdate,&  
                          SchemeWriteCtl, SchemeWriteData, AnaliticFunction

**IMPLICIT NONE**

REAL (KIND=r8) :: yn  
REAL (KIND=r8) :: yc  
REAL (KIND=r8) :: yp  
REAL (KIND=r8) :: ya

REAL (KIND=r8), PARAMETER :: lambda=0.1 ![1/s]

REAL (KIND=r8), PARAMETER :: y0=1.0\_r8

INTEGER , PARAMETER :: nrec=200

REAL(KIND=r8) :: dt=1.5/lambda

INTEGER :: test

INTEGER :: irec

INTEGER :: tn

**CALL** Init()

yc=y0

irec=0

**DO** tn=0,nrec

    yp=SchemeForward(yc)

    ya=AnaliticFunction(y0,tn)

    test=SchemeWriteData(irec,yc,ya)

    yn=SchemeUpdate(yc)

    yc=SchemeUpdate(yp)

**END DO**

test=SchemeWriteCtl(nrec)

**CONTAINS**

**SUBROUTINE** Init()

**IMPLICIT NONE**

**CALL** InitNumericalScheme(dt,lambda)

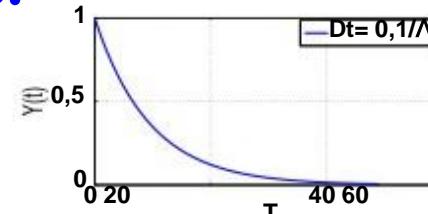
**END SUBROUTINE** Init

**END PROGRAM** MAIN

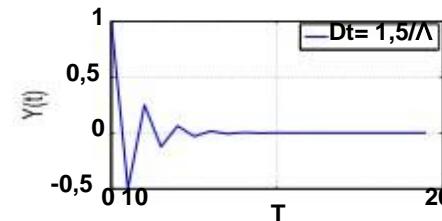
Há três possibilidades para a solução numérica ou seja

$$y^{n+1} = (1 - \lambda\Delta t)y^n \text{ Dependendo da escolha do } \Delta t.$$

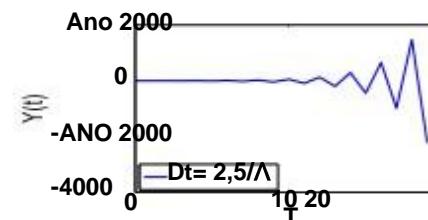
(I) se  $\Delta t < 1/\lambda$  então,  $0 < (1 - \lambda\Delta t) < 1$ . A solução numérica  
É uma função decrescente de tempo.



(II) se  $1/\lambda < \Delta t < 2/\lambda$ . A solução numérica diminui de Magnitude mas oscila no sinal. O sistema é estável, mas não Muito precisos.



(III) se  $\Delta t > 2/\lambda$  então,  $(1 - \lambda\Delta t) < -1$ . A solução oscila no Sinal e aumenta em magnitude com o passar do tempo. O método é Instável para as grandes passo de tempo.



• A restrição de  $\Delta t$  para garantir a estabilidade pode por vezes ser removido pela escolha do método numérico. ex. usando o esquema backward .

# Exercício sobre edo

1. Integre a equação diferencial 7º numericamente usando o a fórmula de diferença forward  $\lambda = 10^{-1} \text{ s}^{-1}$ ,  $y_0 = 1$

Comparar a solução com a solução exata para os três casos

I)  $\Delta t < 1/\lambda$ , II)  $1/\lambda < \Delta t < 2/\lambda$ , e iii)  $\Delta t > 2/\lambda$ .

2. Mostrar que o esquema de diferença backward é estável para A equação diferencial 7.

$$\frac{dy}{dt} = -\lambda y$$

$$\frac{y^n - y^{n-1}}{\Delta t} = -\lambda y^n$$

$$A^n - A^{n-1} = -\lambda \Delta t A^n$$

$$A^n - A^n A^{-1} = -\lambda \Delta t A^n$$

$$1 - A^{-1} = -\lambda \Delta t$$

$$A^{-1} = 1 + \lambda \Delta t$$

$$A = \frac{1}{1 + \lambda \Delta t}$$

O sistema é estável, se  $|A| \leq 1$

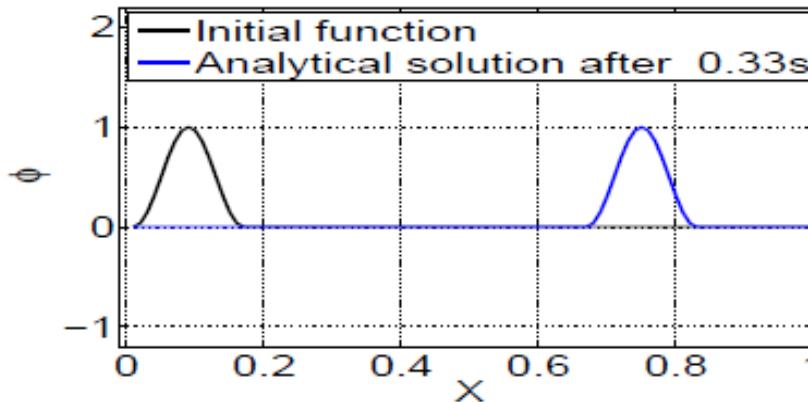
$$|A| = \left| \frac{1}{1 + \lambda \Delta t} \right| \leq 1$$

# A equação de advecção Linear

Considere

$$\frac{\partial \phi}{\partial t} + u \frac{\partial \phi}{\partial x} = 0 , \quad (10)$$

Onde  $u$  é uma velocidade constante, o domínio é  $0 \leq x \leq 1$ . Com Condições fronteira periódicas  $\phi(0, t) = \phi(1, t)$ , E a primeira Condição é dada como  $\phi(x, 0) = F(x)$ . Por exemplo.  $F(x) = \sin^2(x)$ .



A solução analítica é

$$\phi(x, t) = F(x - ut) \quad (11)$$

A função inicial é advectada com velocidade  $u$ . A forma é preservada.

**MODULE** Class\_Fields**IMPLICIT NONE****PRIVATE****INTEGER, PUBLIC , PARAMETER :: r8=8**  
**INTEGER, PUBLIC , PARAMETER :: r4=4**  
**REAL (KIND=r8),PUBLIC ,ALLOCATABLE :: PHI\_P(:)**  
**REAL (KIND=r8),PUBLIC ,ALLOCATABLE :: PHI\_A(:)**  
**REAL (KIND=r8),PUBLIC ,ALLOCATABLE :: PHI\_C(:)**  
**REAL (KIND=r8),PUBLIC ,ALLOCATABLE :: PHI\_M(:)**  
**REAL (KIND=r8),PUBLIC :: Uvel**  
**INTEGER ,PUBLIC :: iMax****PUBLIC :: Init\_Class\_Fields****CONTAINS**

---

**SUBROUTINE** Init\_Class\_Fields(xdim,Uvel0)**IMPLICIT NONE****INTEGER , INTENT (IN ) :: xdim**  
**REAL (KIND=r8), INTENT (IN ):: Uvel0**  
iMax=xdim  
Uvel=Uvel0  
**ALLOCATE (PHI\_A(-1:iMax+2))**  
**ALLOCATE (PHI\_P(-1:iMax+2))**  
**ALLOCATE (PHI\_C(-1:iMax+2))**  
**ALLOCATE (PHI\_M(-1:iMax+2))**  
**END SUBROUTINE** Init\_Class\_Fields

---

**END MODULE** Class\_Fields**MODULE** Class\_NumericalMethod**USE** Class\_Fields, Only :**PHI\_A,PHI\_P,PHI\_C,PHI\_M,Uvel,iMax****IMPLICIT NONE****PRIVATE****INTEGER, PUBLIC , PARAMETER :: r8=8**  
**INTEGER, PUBLIC , PARAMETER :: r4=4**  
**REAL (KIND=r8) :: Dt**  
**REAL (KIND=r8) :: Dx****PUBLIC :: InitNumericalScheme****PUBLIC :: SchemeForward****PUBLIC :: SchemeUpdate****PUBLIC :: SchemeUpStream****PUBLIC :: AnalyticFunction****CONTAINS**

---

**SUBROUTINE** InitNumericalScheme(dt\_in,dx\_in)**IMPLICIT NONE****REAL (KIND=r8), INTENT (IN ) :: dt\_in**  
**REAL (KIND=r8), INTENT (IN ) :: dx\_in**  
**INTEGER :: i**  
**REAL (KIND=r8) :: x**

Dt=dt\_in

Dx=dx\_in

x=0.0

**DO** i=1,iMaxPHI\_C(i)= sin(x)\*sin(x)  
x = (6\*i)\* DX**END DO**

PHI\_M=PHI\_C

PHI\_P=PHI\_C

**END SUBROUTINE** InitNumericalScheme

```

FUNCTION AnaliticFunction(tn) RESULT(ok)
IMPLICIT NONE
INTEGER, INTENT (IN) :: tn
INTEGER :: j,ok
REAL (KIND=r8) :: x
ok=1
x=0.0
DO j=1,iMax
    PHI_A(j)=(sin(x - Uvel*tn*(dt))**2)
    x = (6*j)* DX
END DO
ok=0
END FUNCTION AnaliticFunction

```

```

!-----
----_
FUNCTION SchemeForward() RESULT(ok)
IMPLICIT NONE
! Utilizando a diferenciacao forward
!
!  $F(j,n+1) - F(j,n) \quad F(j+1,n) - F(j,n)$ 
!----- + u ----- = 0
!      dt           dx
!
INTEGER :: ok
INTEGER :: j
DO j=1,iMax
    PHI_P(j) = PHI_C(j) - (Uvel*Dx)*(PHI_C(j+1)-PHI_C(j))
END DO
CALL UpdateBoundaryLayer()
END FUNCTION SchemeForward

```

```

!-----
FUNCTION SchemeUpStream() RESULT(ok)
IMPLICIT NONE
! Utilizando a diferenciacao forward no tempo e
! backward no espaco (upstream)
!
!  $F(j,n+1) - F(j,n) \quad F(j,n) - F(j-1,n)$ 
!----- + u ----- = 0
!      dt           dx
!
INTEGER :: ok
INTEGER :: j
DO j=1,iMax
    PHI_P(j) = PHI_C(j) - (Uvel*Dx)*(PHI_C(j)-PHI_C(j-1))
END DO
CALL UpdateBoundaryLayer()
END FUNCTION SchemeUpStream

```

```

!-----
SUBROUTINE UpdateBoundaryLayer()
IMPLICIT NONE
PHI_P(0 ) = PHI_P(iMax )
PHI_P(-1 ) = PHI_P(iMax-1)
PHI_P(imax+1 ) = PHI_P(1)
PHI_P(iMax+2 ) = PHI_P(2)
END SUBROUTINE UpdateBoundaryLayer

```

```

!-----
FUNCTION SchemeUpdate() RESULT(ok)
IMPLICIT NONE
INTEGER :: ok
PHI_M=PHI_C
PHI_C=PHI_P
ok=0
END FUNCTION SchemeUpdate

```

```

!-----
END MODULE Class_NumericalMethod
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

```

```
!!!!!!!!

MODULE Class_WritetoGrads
USE Class_Fields, Only:
PHI_A, PHI_P, PHI_C, PHI_M, Uvel, 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='AdvecLinearConceitual1D'
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',&
n',&
  FORM='UNFORMATTED', ACCESS='DIRECT',
  STATUS='UNKNOWN', &
  ACTION='WRITE', RECL=lrec)
ok=1
CtrlWriteDataFile=.FALSE.
Yout=REAL(PHI_C(1:iMax),KIND=r4)
irec=irec+1
```

```
Yout=REAL(PHI_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
ok=1

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 2'
WRITE (UnitCtl,'(A      )')'phic 0 99 resultado da edol yc'
WRITE (UnitCtl,'(A      )')'phia 0 99 solucao analitica ya'
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
USE Class_NumericalMethod, Only : InitNumericalScheme, &
    SchemeForward, SchemeUpdate, SchemeUpStream,&
    AnaliticFunction
USE Class_WritetoGrads, Only :InitClass_WritetoGrads, &
    SchemeWriteData, SchemeWriteCtl

IMPLICIT NONE
INTEGER , PARAMETER :: r8=8
INTEGER , PARAMETER :: r4=4
INTEGER , PARAMETER :: xdim=100
REAL (KIND=r8) , PARAMETER :: LX=1.0
REAL (KIND=r8) , PARAMETER :: Uvel0=10.0!m/s
REAL (KIND=r8) , PARAMETER :: dx=LX/xdim !m
REAL (KIND=r8) , PARAMETER :: dt=0.1*dx/Uvel0 !s      !
c*Dx/Dt < 1
!
dx/Uvel0
INTEGER , PARAMETER :: ninteraction=20000
CALL Init()
CALL run()

```

### CONTAINS

```

SUBROUTINE Init()
IMPLICIT NONE
CALL Init_Class_Fields(xdim,Uvel0)
CALL InitNumericalScheme(dt,dx)
CALL InitClass_WritetoGrads
END SUBROUTINE Init

```

```

SUBROUTINE Run()
IMPLICIT NONE
INTEGER :: test,tn,irec
irec=0
DO tn=0,ninteraction
    test=SchemeUpStream()
    test=AnaliticFunction(tn)
    test=SchemeWriteData(irec)
    test=SchemeUpdate()
END DO
test=SchemeWriteCtl(ninteraction)
END SUBROUTINE Run

SUBROUTINE Finalize()
IMPLICIT NONE

END SUBROUTINE Finalize

END PROGRAM Main

```

# Discretization de advecção : 1. FTFS

Valor aproximado a derivada temporal e espacial utilizando a formula de diferença finita forward. p. ex.

$$\frac{\phi_j^{n+1} - \phi_j^n}{\Delta t} + u \frac{\phi_{j+1}^n - \phi_j^n}{\Delta x} = 0 . \quad (12)$$

Resolver a equação advecção 10 numericamente no domínio  $0 \leq X \leq 1000$ m. Deixe  $\Delta x = 1$  m, e assuma as condições limite periódicos. Suponha que a velocidade de advecção  $u= 1$  m/s.

Faça o estado inicial ser um triângulo

$$\phi(x, 0) = \begin{cases} 0 & \text{for } x < 400 \\ 0.1(x - 400) & \text{for } 400 \leq x \leq 500 \\ 20 - 0.1(x - 400) & \text{for } 500 \leq x \leq 600 \\ 0 & \text{for } x > 600 \end{cases} .$$

Escolha FTFS e integre para 20s usando o seguinte esquema e mostre as soluções para  $T = 0$ s e  $T = 20$ s, O que é Você percebe?

## **MODULE Class\_Fields**

**IMPLICIT NONE**

**PRIVATE**

**INTEGER, PUBLIC , PARAMETER :: r8=8**  
**INTEGER, PUBLIC , PARAMETER :: r4=4**  
**REAL (KIND=r8), PUBLIC ,ALLOCATABLE :: PHI\_P(:)**  
**REAL (KIND=r8), PUBLIC ,ALLOCATABLE :: PHI\_C(:)**  
**REAL (KIND=r8), PUBLIC ,ALLOCATABLE :: PHI\_M(:)**  
**REAL (KIND=r8), PUBLIC :: Uvel**  
**INTEGER , PUBLIC :: iMax**  
**PUBLIC :: Init\_Class\_Fields**

**CONTAINS**

---

### **SUBROUTINE Init\_Class\_Fields(xdim,Uvel0)**

**IMPLICIT NONE**

**INTEGER , INTENT (IN ) :: xdim**

**REAL (KIND=r8), INTENT (IN ):: Uvel0**

**iMax=xdim**

**Uvel=Uvel0**

**ALLOCATE (PHI\_P(-1:iMax+2))**

**ALLOCATE (PHI\_C(-1:iMax+2))**

**ALLOCATE (PHI\_M(-1:iMax+2))**

**END SUBROUTINE Init\_Class\_Fields**

---

**END MODULE Class\_Fields**

```
!!!!!!!!

MODULE Class_NumericalMethod
  USE Class_Fields, Only : PHI_P,PHI_C,PHI_M,Uvel,iMax
  IMPLICIT NONE
  PRIVATE
  INTEGER, PUBLIC    , PARAMETER :: r8=8
  INTEGER, PUBLIC    , PARAMETER :: r4=4
  REAL (KIND=r8) :: Dt
  REAL (KIND=r8) :: Dx

  PUBLIC :: InitNumericalScheme
  PUBLIC :: SchemeForward
  PUBLIC :: SchemeUpdate
  PUBLIC :: SchemeUpStream
  CONTAINS
  !-----
```

---

```
SUBROUTINE InitNumericalScheme(dt_in,dx_in)
  IMPLICIT NONE
  REAL (KIND=r8), INTENT (IN ) :: dt_in
  REAL (KIND=r8), INTENT (IN ) :: dx_in
  INTEGER :: i
  Dt=dt_in
  Dx=dx_in
  DO i=-1,iMax+2
    IF (i*dx < 400.0) THEN
      PHI_C(i)= 0.0
    ELSE IF ( i*dx >= 400.0 .AND. i*dx <= 500.0 ) THEN
      PHI_C(i)= 0.1*(i*dx -400.0)
    ELSE IF( i*dx >= 500.0 .AND. i*dx <= 600.0 ) THEN
      PHI_C(i)= 20.0 - 0.1*(i*dx -400.0)
    ELSE IF( i*dx > 600.0 ) THEN
      PHI_C(i)= 0.0
    END IF
  END DO
  PHI_M=PHI_C
  PHI_P=PHI_C
END SUBROUTINE InitNumericalScheme
!-----
```

```
FUNCTION SchemeForward() RESULT (ok)
  IMPLICIT NONE
  ! Utilizando a diferenciacao forward
  !
  ! 
$$\frac{F(j,n+1) - F(j,n)}{dt} + u \frac{F(j+1,n) - F(j,n)}{dx} = 0$$

  !
  INTEGER :: ok
  INTEGER :: j
  DO j=1,iMax
    PHI_P(j) = PHI_C(j) - (Uvel*Dt/Dx)*(PHI_C(j+1)-PHI_C(j))
  END DO
  CALL UpdateBoundaryLayer()
END FUNCTION SchemeForward
!
```

---

```
FUNCTION SchemeUpStream() RESULT (ok)
  IMPLICIT NONE
  ! Utilizando a diferenciacao forward no tempo e
  ! backward no espaco (upstream)
  !
  ! 
$$\frac{F(j,n+1) - F(j,n)}{dt} + u \frac{F(j,n) - F(j-1,n)}{dx} = 0$$

  !
  INTEGER :: ok
  INTEGER :: j
  DO j=1,iMax
    PHI_P(j) = PHI_C(j) - (Uvel*Dt/Dx)*(PHI_C(j)-PHI_C(j-1))
  END DO
  CALL UpdateBoundaryLayer()
END FUNCTION SchemeUpStream
```

```
!-----  
---  
SUBROUTINE UpdateBoundaryLayer()  
  IMPLICIT NONE  
  PHI_P(0)    = PHI_P(iMax )  
  PHI_P(-1)   = PHI_P(iMax-1)  
  PHI_P(imax+1)= PHI_P(1)  
  PHI_P(iMax+2)= PHI_P(2)  
END SUBROUTINE UpdateBoundaryLayer  
!  
---  
FUNCTION SchemeUpdate() RESULT (ok)  
  IMPLICIT NONE  
  INTEGER :: ok  
  PHI_M=PHI_C  
  PHI_C=PHI_P  
  ok=0  
END FUNCTION SchemeUpdate  
!  
---  
!  
---  
END MODULE Class_NumericalMethod  
!!!!!!!!!!!!!!
```

```

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
MODULE Class_WritetoGrads
USE Class_Fields, Only: PHI_P,PHI_C,PHI_M,Uvel,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='AdvecLinear1D'
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',
n', &
     FORM='UNFORMATTED', ACCESS='DIRECT', &
     STATUS='UNKNOWN', ACTION='WRITE',RECL =lrec)

```

```

CtrlWriteDataFile=.FALSE.
Yout= REAL(PHI_C(1:iMax),KIND=r4)
irec=irec+1
WRITE(UnitData,rec=irec)Yout
ok=0

```

```

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      )')'phic 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  
USE Class_NumericalMethod, Only: nitNumericalScheme, &  
    SchemeForward,SchemeUpdate,SchemeUpStream  
USE Class_WritetoGrads, Only :InitClass_WritetoGrads, &  
    SchemeWriteData,SchemeWriteCtl  
IMPLICIT NONE  
INTEGER , PARAMETER :: r8=8  
INTEGER , PARAMETER :: r4=4  
INTEGER , PARAMETER :: xdim=1000  
REAL (KIND=r8) , PARAMETER :: Uvel0=10.0!m/s  
REAL (KIND=r8) , PARAMETER :: dt=0.1 !s  
REAL (KIND=r8) , PARAMETER :: dx=10.0 !m  
INTEGER , PARAMETER :: ninteraction=20000  
CALL Init()  
CALL run()
```

CONTAINS

```
SUBROUTINE Init()  
IMPLICIT NONE  
CALL Init_Class_Fields(xdim,Uvel0)  
CALL InitNumericalScheme(dt,dx)  
CALL InitClass_WritetoGrads  
END SUBROUTINE Init
```

```
SUBROUTINE Run()  
IMPLICIT NONE  
INTEGER :: test,it,irec  
irec=0  
DO it=1,ninteraction  
    test=SchemeUpStream()  
    test=SchemeWriteData(irec)  
    test=SchemeUpdate()  
END DO  
test=SchemeWriteCtl(ninteraction)  
END SUBROUTINE Run  
  
SUBROUTINE Finalize()  
IMPLICIT NONE  
END SUBROUTINE Finalize  
  
END PROGRAM Main
```

# Análise de estabilidade (Von Neumann)

- Em um domínio qualquer a função periódica pode ser decomposta em componentes de Fourier. Se a evolução da função é regida por uma equação linear com coeficientes constantes, então, seu comportamento pode ser determinado, através da observação do comportamento de cada componente Fourier.
- Da mesma forma, a estabilidade de um método numérico linear pode ser encontrado ao considerar um único componente de Fourier, vendo se ela cresce com o tempo.
- Análise de estabilidade de Von Neumann nos dá uma equação para o Fator de ampliação  $\phi_j^n = \phi(x_j, t_n) = A^n e^{ikj\Delta x}$ .
- Método de Von Neumann requer normalmente que o Fator de amplificação seja delimitada, de tal forma que  $|A| \leq 1$
- Análise de estabilidade de Von Neumann é uma Condição necessária e suficiente para a estabilidade de uma equação linear de diferença finita com Coeficiente constante. Para equações não-lineares, porém, ele é um Condição necessária, mas não é suficiente.

# Análise de Estabilidade da FTFS

**Substituindo**  $\phi(x_j, t_n) = A^n e^{ikj\Delta x}$  **Na eqn 25º temos.**

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

$$A^{n+1}e^{ikj\Delta x} = A^n e^{ikj\Delta x} - u \frac{\Delta t}{\Delta x} (A^n e^{ik(j+1)\Delta x} - A^n e^{ikj\Delta x})$$

$$A^n A^1 e^{ikj\Delta x} = A^n e^{ikj\Delta x} - u \frac{\Delta t}{\Delta x} (A^n e^{ikj\Delta x} e^{ik(1)\Delta x} - A^n e^{ikj\Delta x})$$

$$A^n A e^{ikj\Delta x} = A^n e^{ikj\Delta x} - u \frac{\Delta t}{\Delta x} A^n e^{ikj\Delta x} (e^{ik\Delta x} - 1)$$

# Análise de Estabilidade da FTFS

Defina  $C = u \frac{\Delta t}{\Delta x}$ .  $C$  é chamado de número de Courant. Cancele  $A^n$  e  $e^{ikj\Delta x}$  De ambos os lados

$$\cancel{A^n A e^{ikj\Delta x}} = \cancel{A^n e^{ikj\Delta x}} - u \frac{\Delta t}{\Delta x} \cancel{A^n e^{ikj\Delta x}} (e^{ik\Delta x} - 1)$$

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

A solução numérica será estável se  $|A| \leq 1$ .  $|A|^2$  É dado por  $A$  vezes o complexo conjugado ou seja

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

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

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

# Análise de Estabilidade da FTFS

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

$$\begin{aligned}|A|^2 &= 1 - Ce^{-ik\Delta x} + C - Ce^{ik\Delta x} + C^2 e^{ik\Delta x - ik\Delta x} - C^2 e^{ik\Delta x} + C - C^2 e^{-ik\Delta x} \\&\quad + C^2\end{aligned}$$

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

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

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

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

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

# FTFS regime

$$|A|^2 = 1 + 2C((1 + C)(1 - \cos(k\Delta x))) \leq 1 \quad (13)$$

- Desde  $(1 - \cos(k\Delta x)) > 0$  Para todos os números de ondas exceto o trivial caso isto é  $k=0$ , o reduz a desigualdade acima

$$C(1 + C) \leq 0$$

⇒ Nunca pode ser satisfeita Uma vez que :  $c > 0$  Porque  $u$  é +

- Isto implica FTFS (com  $u+$ ) é incondicionalmente Instável!

# Regime Upstream

**Esquema Upstream(Forward no tempo, Backward no espaço)**

aproxima a derivada espacial com Diferença backward e a derivada temporal com diferença forward Fórmula. p. ex.

$$\frac{\phi_j^{n+1} - \phi_j^n}{\Delta t} + u \frac{\phi_j^n - \phi_{j-1}^n}{\Delta x} = 0 . \quad (14)$$

Análise de Estabilidade (Von Neumann)

Podemos seguir o mesmo procedimento como no FTFS para fazer o Análise de estabilidade ou seja deixar a solução ter a forma de

$$\phi(x_j, t_n) = \phi_j^n = A^n e^{ikj\Delta x}$$

Von Neumann método requer normalmente que o Fator de ampliação a seja delimitada, tal que  $|A| \leq 1$

$$\frac{A^{n+1} e^{ikj\Delta x} - A^n e^{ikj\Delta x}}{\Delta t} = -u \left( \frac{A^n e^{ikj\Delta x} - A^n e^{ik(j-1)\Delta x}}{\Delta x} \right) \quad (15)$$

# Regime Upstream

$$\frac{A^{n+1}e^{ikj\Delta x} - A^n e^{ikj\Delta x}}{\Delta t} = -u \frac{A^n e^{ikj\Delta x} - A^n e^{ik(j-1)\Delta x}}{\Delta x} = 0$$

$$A^{n+1}e^{ikj\Delta x} = A^n e^{ikj\Delta x} - u \frac{\Delta t}{\Delta x} (A^n e^{ikj\Delta x} - A^n e^{ik(j-1)\Delta x})$$

$$A^n A e^{ikj\Delta x} = A^n e^{ikj\Delta x} - u \frac{\Delta t}{\Delta x} (A^n e^{ikj\Delta x} - A^n e^{ikj\Delta x} e^{-ik\Delta x})$$

$$A^n A e^{ikj\Delta x} = A^n e^{ikj\Delta x} - u \frac{\Delta t}{\Delta x} A^n e^{ikj\Delta x} (1 - e^{-ik\Delta x})$$

Definir  $c = u \frac{\Delta t}{\Delta x}$ . , C é chamado o *número Courant*. Cancelar os termos de  $A^n e^{ikj\Delta x}$ .

~~$$A^n A e^{ikj\Delta x} = A^n e^{ikj\Delta x} - u \frac{\Delta t}{\Delta x} A^n e^{ikj\Delta x} (1 - e^{-ik\Delta x})$$~~

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

# Regime Upstream

A solução numérica será estável se  $|A| \leq 1$ .  $|A|^2$  é dado pela  $A$  vezes o seu complexo ou seja conjugado

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

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

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

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

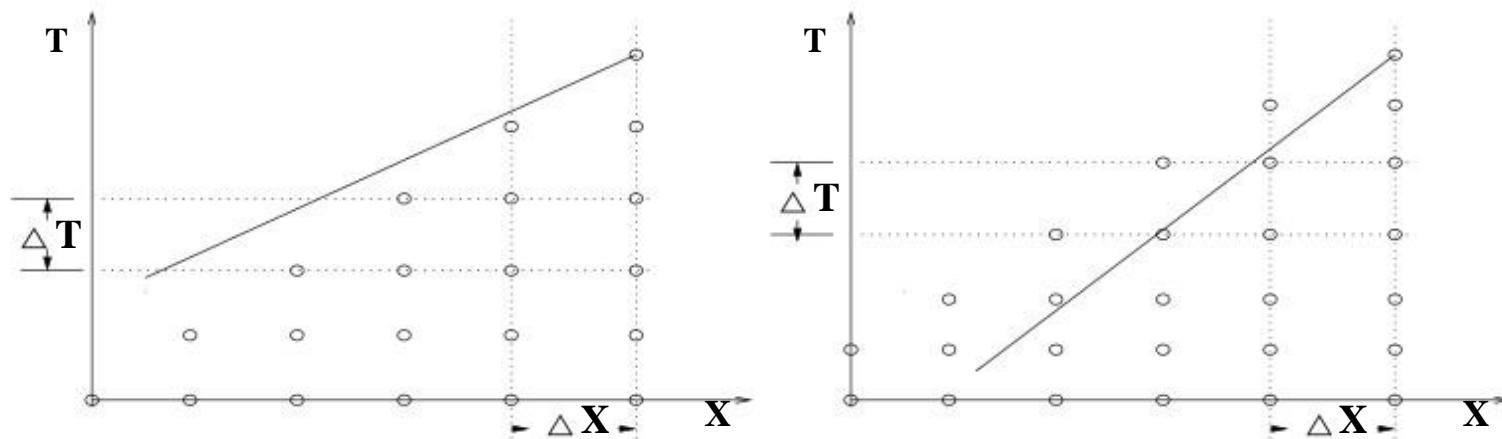
Desde  $(1 - \cos(k\Delta x)) > 0$ . Para todos os números de onda, com exceção do caso trivial ou seja,  $k = 0$ , a desigualdade acima reduz a.

$$C(1 - C) \geq 0$$

$$C \geq 0 \text{ e } C \leq 1; \quad 0 \leq C \leq 1$$

# O Critério de Courant-Friedrich-Lowy

- O critério de CFL afirma que uma condição necessária para a estabilidade é que o domínio de dependência da solução numérica deve incluir o domínio da dependência da equação diferencial parcial original.
- A condição de CFL exige que a cada passo de tempo a solução em um Ponto  $X_j$ , a solução numérica não pode se propagar mais do que um ponto grade por passo tempo e a velocidade  $\frac{\Delta x}{\Delta t}$  não deve exceder o permitido pela aproximação FTBS.”



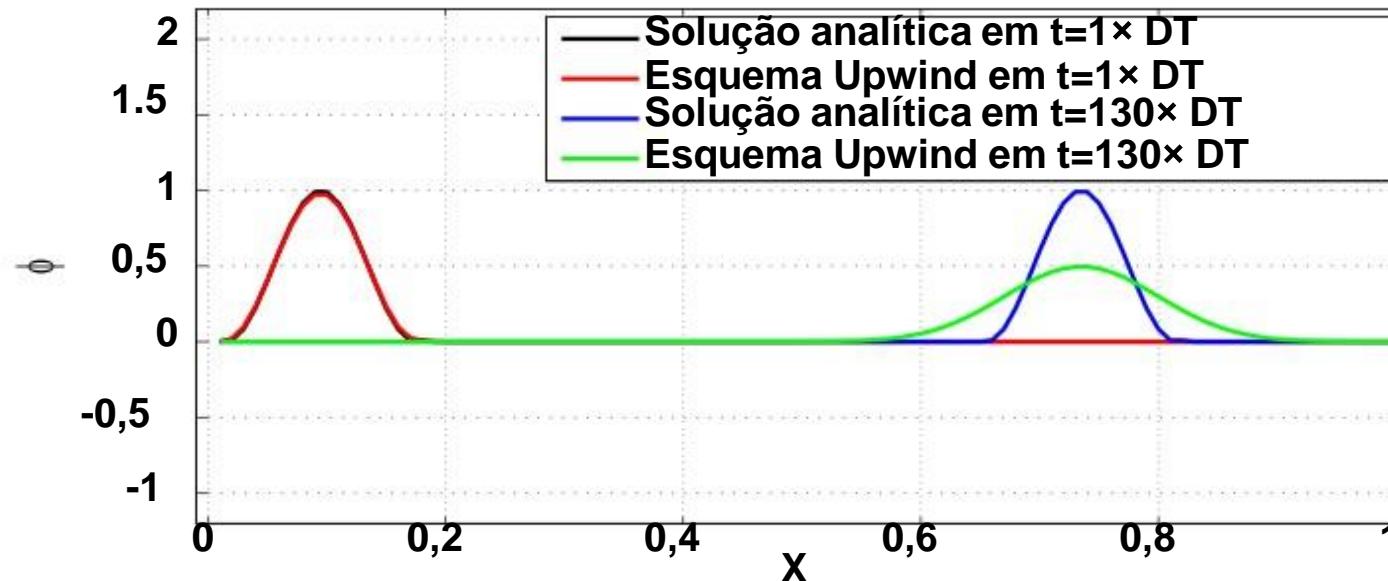
No painel direito do diagrama acima, é evidente que o domínio numérico de dependência contém o domínio físico desde que a dependência

$$1/|U| \geq \Delta t/\Delta x \text{, i.e., } 0 \leq |U|\Delta t/\Delta x \leq 1$$

# Notas: esquema upstream (upwind)

- FTBS regime é instável se  $U < 0$ . Nesse caso, é preciso usar o esquema forward no tempo e no espaço (FTFS).
- No caso geral quando  $U$  Pode alterar sinal devemos utilizar FTBS quando  $U > 0$  e FTFS quando  $U < 0$ , tal que sempre usamos as informações do lado a upstream do ponto cujo novo valor estamos tentando calcular.
- O problema deste esquema é a Excessiva difusão numérica, é o erro de truncamento dominante no esquema upwind primeiro ordem.

Barlavento advecção regime após 1st. e 130TH Tempo passo



# Exercício: esquema upstream

Resolver a equação advecção 10 numericamente no domínio  $0 \leq X \leq 1000M$ . Deixe  $\Delta x = 1 M$ , e assuma as condições de limite periódicos

. Suponha que a velocidade de advecção  $U = 1 M/s$ .

Deixe

$$\phi(x, 0) = \begin{cases} 0 & \text{for } x < 400 \\ 0.1(x - 400) & \text{for } 400 \leq x \leq 500 \\ 20 - 0.1(x - 400) & \text{for } 500 \leq x \leq 600 \\ 0 & \text{for } x > 600 \end{cases}$$

Escolha um esquema upstream e integre forward e mostre as Soluções para  $T = 0S$   $T = 300s$ ,  $T = 600s$ ,  $T = 900s$ ,  $T = 1000s$ ,  $T = 1500s$ ,  $T = 2000s$

E explicar as características da solução.

# O Esquema CTCS (Leapfrog)

Um outro método para resolver o problema de advecção (i.e.  $\partial\Phi/\partial t + u\partial\Phi/\partial x = 0$ ). É usar o esquema centrada no tempo, Centrado no espaço (CTCS). i.e.

$$\frac{\phi_j^{n+1} - \phi_j^{n-1}}{2\Delta t} + u \frac{\phi_{j+1}^n - \phi_{j-1}^n}{2\Delta x} = 0 . \quad (17)$$

Trata-se de uma fórmula de três-nível, uma vez que ela envolve valores de  $\emptyset$  em três tempos  $t_{n+1}, t_n, t_{n-1}$ .

O esquema CTCS é de precisão de segunda ordem no espaço e no tempo. análise de estabilidade Von Neumann

Define-se  $\emptyset = A^n e^{ikj\Delta x}$  para a análise de estabilidade de Von Neumann, que veremos em seguida

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

$$A^{n+1} e^{ikj\Delta x} = A^{n-1} e^{ikj\Delta x} - u \frac{\Delta t}{\Delta x} (A^n e^{ik(j+1)\Delta x} - A^n e^{ik(j-1)\Delta x})$$

# O Esquema CTCS (Leapfrog)

$$A^{n+1} e^{ikj\Delta x} = A^{n-1} e^{ikj\Delta x} - u \frac{\Delta t}{\Delta x} (A^n e^{ik(j+1)\Delta x} - A^n e^{ik(j-1)\Delta x})$$

$$A^n A e^{ikj\Delta x} = \frac{A^n}{A} e^{ikj\Delta x} - u \frac{\Delta t}{\Delta x} (A^n e^{ikj\Delta x} e^{ik\Delta x} - A^n e^{ikj\Delta x} e^{-ik\Delta x})$$

$$A^n A e^{ikj\Delta x} = \frac{A^n}{A} e^{ikj\Delta x} - C (A^n e^{ikj\Delta x} e^{ik\Delta x} - A^n e^{ikj\Delta x} e^{-ik\Delta x})$$

$$A^n A e^{ikj\Delta x} = \frac{A^n}{A} e^{ikj\Delta x} - C A^n e^{ikj\Delta x} (e^{ik\Delta x} - e^{-ik\Delta x})$$

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

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

$$A^2 = 1 - C 2i (\sin(k\Delta x)) A$$

$$A^2 = 1 - C2i(\sin(k\Delta x))A$$

$$A^2 + C2i(\sin(k\Delta x))A - 1 = 0$$

$$A^2 - C2i(\sin(k\Delta x))A - 1 = 0$$

$$\begin{cases} ax^2 + bx + c = 0 \\ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \end{cases}$$

$$A = \frac{C2i(\sin(k\Delta x)) \pm \sqrt{(-C2i(\sin(k\Delta x)))^2 - 4(1)(-1)}}{2}$$

$$A = \frac{C2i(\sin(k\Delta x)) \pm \sqrt{((-1)^2 C^2 4 (\sqrt{-1}^2) (\sin^2(k\Delta x)))^2 + 4}}{2}$$

$$A = \frac{C2i(\sin(k\Delta x)) \pm \sqrt{-4C^2 \sin^2(k\Delta x) + 4}}{2}$$

$$A = Ci(\sin(k\Delta x)) \pm \sqrt{-C^2 \sin^2(k\Delta x) + 1}$$

$$A = Ci(\sin(k\Delta x)) \pm \sqrt{1 - C^2 \sin^2(k\Delta x)}$$

**Há dois casos a considerar  $|C| > 1$ , em seguida, para algum  $\Delta x$ , tal que  $(C \sin(k\Delta x))^2 > 1$**

$$A = Ci(\sin(k\Delta x)) \pm i\sqrt{C^2 \sin^2(k\Delta x) - 1}$$

$$\begin{aligned} |A|^2 &= \left( Ci(\sin(k\Delta x)) \pm i\sqrt{C^2 \sin^2(k\Delta x) - 1} \right) \left( -Ci(\sin(k\Delta x)) \right. \\ &\quad \left. \pm -i\sqrt{C^2 \sin^2(k\Delta x) - 1} \right) \end{aligned}$$

$$\begin{aligned} |A|^2 &= \left( Ci(\sin(k\Delta x)) \pm i\sqrt{C^2 \sin^2(k\Delta x) - 1} \right) \left( -Ci(\sin(k\Delta x)) \right. \\ &\quad \left. \pm -i\sqrt{C^2 \sin^2(k\Delta x) - 1} \right) \end{aligned}$$

$$\begin{aligned}|A|^2 &= i \left( C(\sin(k\Delta x)) \right. \\&\quad \left. \pm \sqrt{C^2 \sin^2(k\Delta x) - 1} \right) \left( -i \left( C(\sin(k\Delta x)) \pm \sqrt{C^2 \sin^2(k\Delta x) - 1} \right) \right) \\|A_{\pm}|^2 &= \left( C(\sin(k\Delta x)) \pm \sqrt{C^2 \sin^2(k\Delta x) - 1} \right)^2\end{aligned}$$

**Existe pelo menos uma raiz que  $|A_{\pm}| > 1$ , Portanto, a solução é instável durante  $|C| > 0$**

**2.  $|c| \leq 1$  em seguida,  $C(\sin(k\Delta x)) \leq 1$ . As duas raízes são:**

$$A = Ci(\sin(k\Delta x)) \pm i\sqrt{C^2 \sin^2(k\Delta x) - 1}$$

$$A = Ci(\sin(k\Delta x)) \pm \sqrt{1 - C^2 \sin^2(k\Delta x)}$$

$$|A_+|^2 = \left( Ci(\sin(k\Delta x)) + \sqrt{1 - C^2 \sin^2(k\Delta x)} \right) \left( -Ci(\sin(k\Delta x)) + \sqrt{1 - C^2 \sin^2(k\Delta x)} \right)$$

$$\begin{aligned} |A_+|^2 &= \left( -i^2 C^2 \sin^2(k\Delta x) + Ci(\sin(k\Delta x)) * \left( \sqrt{1 - C^2 \sin^2(k\Delta x)} \right) - Ci(\sin(k\Delta x)) \right. \\ &\quad \left. * \left( \sqrt{1 - C^2 \sin^2(k\Delta x)} \right) + 1 - C^2 \sin^2(k\Delta x) \right) \end{aligned}$$

$$|A_+|^2 = (C^2 \sin^2(k\Delta x) + 1 - C^2 \sin^2(k\Delta x))$$

$$|A_+|^2 = 1$$

**Da mesma forma para  $|A_-|^2$**

$$|A_+|^2 = (C^2 \sin^2(k\Delta x) + 1 - C^2 \sin^2(k\Delta x))$$

$$|A_+|^2 = 1$$

**$\therefore$  A condição de estabilidade é  $\left|C = \frac{u\Delta t}{\Delta x}\right| \leq 1.$**

# Modo computacional de CTCS

O esquema CTCS dá dois valores para  $\mathbf{A}$

$$A_p = -ic \sin k\Delta x + (1 - (c \sin k\Delta x)^2)^{1/2} \quad (18)$$

$$A_c = -ic \sin k\Delta x - (1 - (c \sin k\Delta x)^2)^{1/2}, \quad (19)$$

Portanto, a forma geral da solução numérica é

$$\phi_j^n = [P(A_p)^n + C(A_c)^n]e^{ikj\Delta x} \quad (20)$$

Vamos escolher  $\mathbf{C} = 1$ . Em seguida, é conveniente escrever  $\mathbf{A}$  na forma

$$A_p = e^{-i\alpha}, A_c = -e^{i\alpha} \quad (21)$$

Onde  $\alpha = k\Delta x = uk\Delta t$

$$\phi_j^n = Pe^{ik(j\Delta x - un\Delta t)} + (-1)^n Ce^{ik(j\Delta x + un\Delta t)}.$$

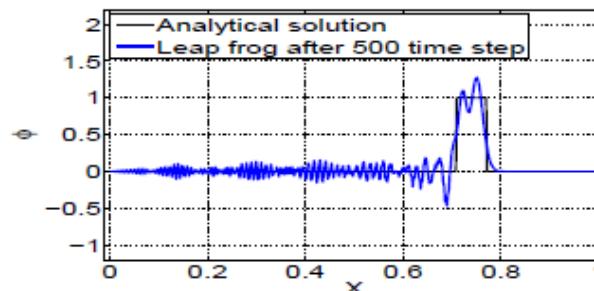
Nos casos em que  $P$  e  $C$  sejam constantes complexas, determinados pela primeira Condições ou seja,  $t=0$  ( $n=0$ ).

$$\phi_j^0 = e^{ikj\Delta x} = (P + C)e^{ikj\Delta x} \Rightarrow (P + C) = 1$$

$$\Rightarrow \phi_j^n = \underbrace{(1 - C)e^{ik(j\Delta x - un\Delta t)}}_{Physical\ mode} + \underbrace{(-1)^n C e^{ik(j\Delta x + un\Delta t)}}_{Computational\ mode} \quad (22)$$

O modo Físico é proporcional à solução exata  $e^{ik(x-ut)}$   
 O modo computacional não correspondem a qualquer solução da equação diferencial original ; ela é um artefato do Método numérico.

Duas das características do modo computacional  
 Ela oscila no tempo de um passo tempo para o próximo passo de tempo (por causa do fator  $(-1)^n$ )  
 Se propaga na direção oposta à verdadeira solução (Por causa do termo  $+un\Delta t$  na exponencial ao invés de  $-Un\Delta t$ ).



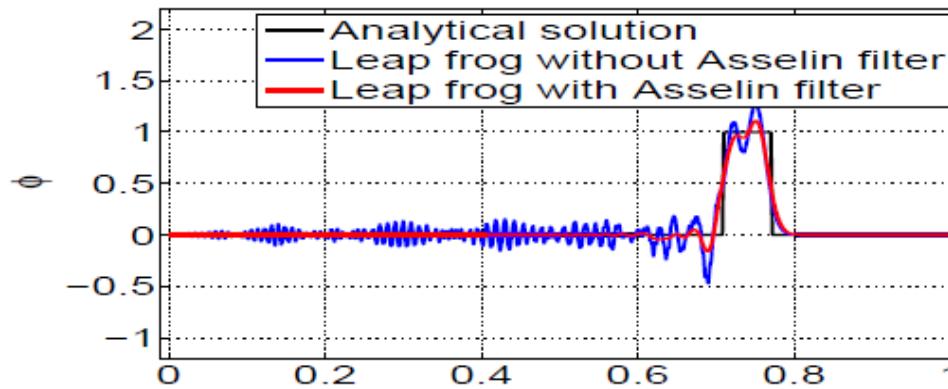
# Modo computacional e o filtro RA

- A solução  $\phi_j^{n+1}$ , Depende,  $\phi_j^{n-1}$  Mas não no  $\phi_j^n$
- Exceto no momento inicial, a solução é encontrada em dois conjuntos de pontos que não são associados.
- Em qualquer ponto J a solução oscila entre os duas soluções Desatrelada.
- Para manter a amplitude do modo computacional pequenas é Necessário soluções acopladas sobre os dois conjuntos alternando de Pontos de grade.
- A maneira mais comum de se fazer isto no modelo Atmosférica é através do Uso do filtro de Robert-Asselin (RA), que fortemente descarrega as Oscilação  $\phi^{n+1} = \phi^{n-1} - [\text{other terms}]$
- Calcular o deslocamento do filtro ou seja  
$$d = \alpha * (\phi^{n-1} - 2\phi^n + \phi^{n+1})$$
- Aplicar o filtro para  $\Phi^N$  Ou seja  
$$\hat{\phi}^n = \phi^n + d = (1 - 2\alpha)\phi^n + \alpha(\phi^{n+1} + \phi^{n-1})$$

Então, atualize  $\hat{\phi}^{n+1}$  por  $\hat{\phi}^n$  e  $\hat{\phi}^n$  por  $\phi^{n+1}$  Para a próxima iteração  
Depois de aplicar o filtro, o esquema de advecção fica parecido com

$$\phi^{n+1} = \hat{\phi}^{n-1} - [\text{other terms}]$$

A figura abaixo mostra advecção de uma onda quadrada (com  $C=0,7$ ) após 500 tempo passo com (linha vermelha) e sem (azul). Linha) apresentando Asselin filtro de tempo.



Note que este filtro também induz a um amortecimento artificial do modo físico, alfa Devem ser mantidos pequenos (na parcela acima alfa = 0,05 )

# filtro RAW - Williams (2009), MWR

Método RAW, que devemos aplicar a filtragem não somente em  $\Phi^N$  mas também em  $\Phi^{n+1}$

Tal como antes, use primeiro o esquema leapfrog como antes

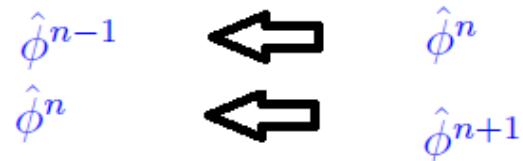
$$\phi^{n+1} = \phi^{n-1} - [\text{other terms}]$$

Em seguida, aplica-se o filtro

$$\hat{\phi}^n = \phi^n + d\beta$$

$$\hat{\phi}^{n+1} = \phi^{n+1} + d(\beta - 1)$$

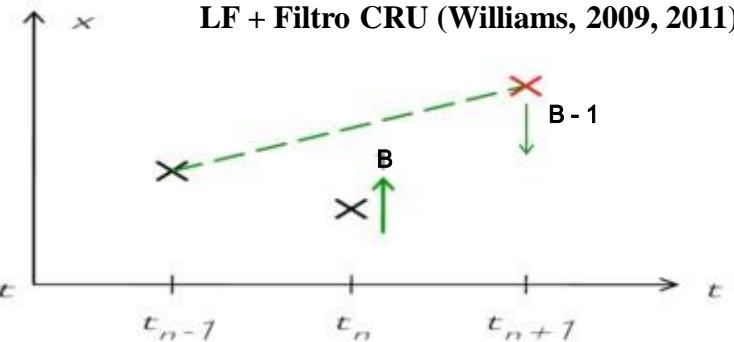
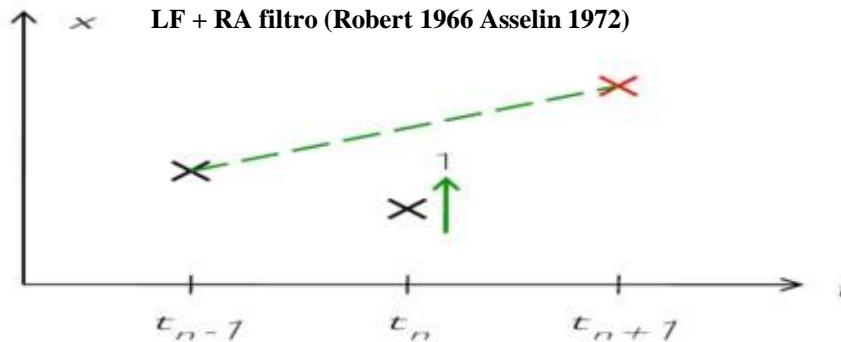
Depois da implementação do filtro e depois atualizar



$$\phi^{n+1} = \hat{\phi}^{n-1} - [\text{other terms}]$$

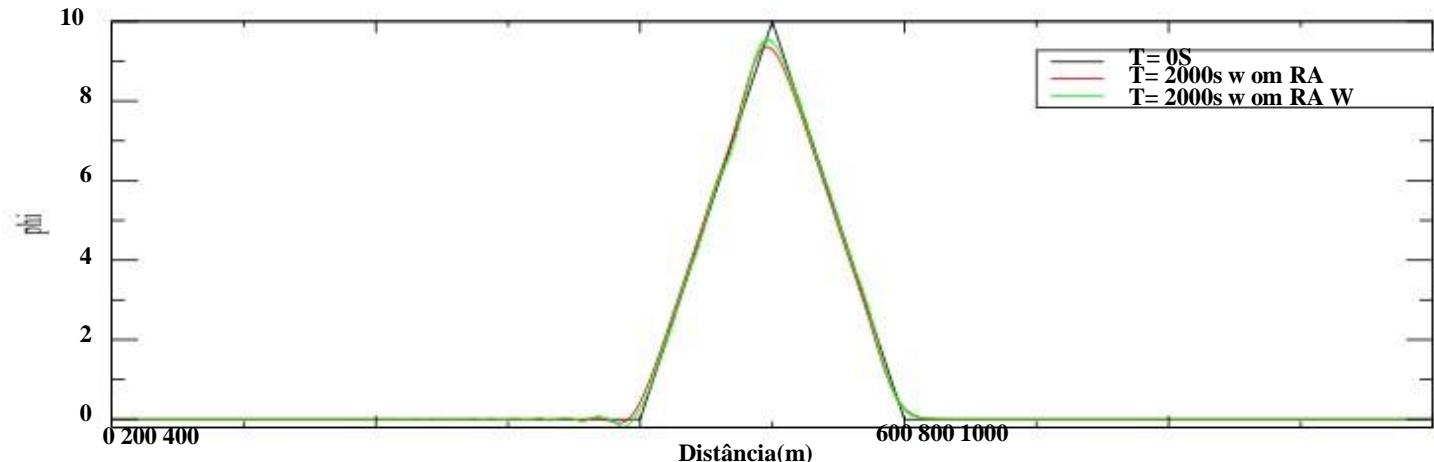
Devido à estabilidade razão , o valor beta é  $0,5 < \text{beta} \leq 1$

# O filtro RA vs RAW



- ➊ RA filtro aplica-se em
- ➋ Reduz curvatura mas não conserva a curvatura media
- ➌ Precisão da amplitude é 1.<sup>o</sup> ordem
- ➊ Filtro-RWA aplica-se em  $x^n$  e  $x^{n+1}$
- ➋ Reduz e ao mesmo tempo, conserva a curvatura média (para  $B = 1/2$ )
- ➌ Precisão da amplitude é  $\approx 3.$ <sup>a</sup> ordem

Salto com Vcto RA \_and\_RA W \_filter social e dt=0,5, a alfa=0,05, beta=0,6



# Exercício

- Resolver a equação advecção 10 numericamente no domínio  $0 \leq X \leq 1000M$ . Deixe  $\Delta x = 1 M$ , e assuma As condições limite periódicos . Suponha que a velocity de advecção  $U = 1 M/s$ . Deixe o estado inicial ser um triângulo

$$\Phi(x,0) = \begin{cases} 0 & \text{Para } X < 400 \\ 0.1(X - 400) & \text{Para } 400 \leq X \leq 500 \\ 20 - 0.1(x - 400) & \text{Para } 500 \leq X \leq 600 \\ 0 & \text{Para } X > 600 \end{cases}$$

Escolha o intervalo de tempo, que o sistema seja estável. Integrar Para 2000seg usando os seguintes esquemas, e mostrar as soluções para  $T = 0S$   $T = 200s$ ,  $T = 400S$   $T = 600S$   $T = 800S$   $T = 1000S$   $T = 1200S$   $T = 1400S$   $T = 1600S$   $T = 1800S$   $T = 2000S$ .

1. Leap frog com a RA esquema filtro de tempo (use  $\alpha = 0,25$ )
2. Leap frog com matérias-primas regime filtro de tempo( $\beta = 0,60$  e  $\gamma = 1,0$ ))

# Filtro RA

## MODULE Class\_Fields

```
IMPLICIT NONE
PRIVATE
INTEGER, PUBLIC , PARAMETER :: r8=8
INTEGER, PUBLIC , PARAMETER :: r4=4
REAL (KIND=r8),PUBLIC ,ALLOCATABLE :: PHI_P(:)
REAL (KIND=r8),PUBLIC ,ALLOCATABLE :: PHI_C(:)
REAL (KIND=r8),PUBLIC ,ALLOCATABLE :: PHI_M(:)
REAL (KIND=r8),PUBLIC ,ALLOCATABLE :: Deslc(:)
REAL (KIND=r8),PUBLIC          :: Uvel
INTEGER      ,PUBLIC          :: iMax
REAL (KIND=r8),PUBLIC          :: alfa
PUBLIC :: Init_Class_Fields
```

## CONTAINS

```
SUBROUTINE Init_Class_Fields(xdim,Uvel0,alfa_in)
IMPLICIT NONE
INTEGER      , INTENT (IN ) :: xdim
REAL (KIND=r8), INTENT (IN ):: Uvel0
REAL (KIND=r8), INTENT (IN ):: alfa_in
iMax=xdim
Uvel=Uvel0
alfa=alfa_in
ALLOCATE (PHI_P(-1:iMax+2))
ALLOCATE (PHI_C(-1:iMax+2))
ALLOCATE (PHI_M(-1:iMax+2))
ALLOCATE (Deslc(-1:iMax+2))
END SUBROUTINE Init_Class_Fields
```

```
END MODULE Class_Fields
```

## MODULE Class\_NumericalMethod

```
USE Class_Fields, Only : PHI_P,PHI_C,PHI_M,Deslc,Uvel,iMax,alfa
IMPLICIT NONE
PRIVATE
INTEGER, PUBLIC , PARAMETER :: r8=8
INTEGER, PUBLIC , PARAMETER :: r4=4
REAL (KIND=r8) :: Dt
REAL (KIND=r8) :: Dx
PUBLIC :: InitNumericalScheme
PUBLIC :: SchemeForward
PUBLIC :: SchemeUpdate
PUBLIC :: SchemeUpStream
PUBLIC :: Filter_RA
```

## CONTAINS

```
!-----
SUBROUTINE InitNumericalScheme(dt_in,dx_in)
IMPLICIT NONE
REAL (KIND=r8), INTENT (IN ) :: dt_in
REAL (KIND=r8), INTENT (IN ) :: dx_in
INTEGER :: i
Dt=dt_in
Dx=dx_in
DO i=-1,iMax+2
IF (i*dx < 400.0) THEN
    PHI_C(i)= 0.0
ELSE IF ( i*dx >= 400.0 .AND. i*dx <= 500.0 ) THEN
    PHI_C(i)= 0.1*(i*dx -400.0)
ELSE IF( i*dx >= 500.0 .AND. i*dx <= 600.0 )THEN
    PHI_C(i)= 20.0 - 0.1*(i*dx -400.0)
ELSE IF( i*dx > 600.0 )THEN
    PHI_C(i)= 0.0
END IF
END DO
PHI_M=PHI_C
PHI_P=PHI_C
END SUBROUTINE InitNumericalScheme
```

```

!-----
----  

FUNCTION SchemeForward() RESULT(ok)  

IMPLICIT NONE  

! Utilizando a diferenciacao forward  

!  

!  $F(j,n+1) - F(j,n) - F(j+1,n) + F(j,n)$   

!----- + u ----- = 0  

! dt dx  

!  

INTEGER :: ok  

INTEGER :: j  

DO j=1,iMax  

     $\text{PHI\_P}(j) = \text{PHI\_C}(j) - (\text{Uvel} * \text{Dt}/\text{Dx}) * (\text{PHI\_C}(j+1) - \text{PHI\_C}(j))$   

END DO  

CALL UpdateBoundaryLayer()  

END FUNCTION SchemeForward
!
```

```

!-----  

----  

FUNCTION SchemeUpStream() RESULT (ok)  

IMPLICIT NONE  

! Utilizando a diferenciacao forward no tempo e  

! backward no espaco (upstream)  

!  

!  $F(j,n+1) - F(j,n) - F(j,n) + F(j-1,n)$   

!----- + u ----- = 0  

! dt dx  

!  

INTEGER :: ok  

INTEGER :: j  

DO j=1,iMax  

     $\text{PHI\_P}(j) = \text{PHI\_C}(j) - (\text{Uvel} * \text{Dt}/\text{Dx}) * (\text{PHI\_C}(j) - \text{PHI\_C}(j-1))$   

END DO  

CALL UpdateBoundaryLayer()  

END FUNCTION SchemeUpStream

```

```

!-----  

----  

FUNCTION Filter_RA() RESULT (ok)  

IMPLICIT NONE  

INTEGER :: i  

INTEGER :: ok  

DO i=-1,iMax+2  

     $\text{Deslc}(i) = \text{alfa} * (\text{PHI\_M}(i) - 2.0 * \text{PHI\_C}(i) + \text{PHI\_P}(i))$   

     $\text{PHI\_C}(i) = \text{PHI\_C}(i) + \text{Deslc}(i)$   

END DO  

 $\text{ok}=0$   

END FUNCTION Filter_RA
!
```

```

!-----  

----  

SUBROUTINE UpdateBoundaryLayer()  

IMPLICIT NONE  

PHI\_P(0) = PHI\_P(iMax)  

PHI\_P(-1) = PHI\_P(iMax-1)  

PHI\_P(imax+1) = PHI\_P(1)  

PHI\_P(iMax+2) = PHI\_P(2)  

END SUBROUTINE UpdateBoundaryLayer
!
```

```

!-----  

----  

FUNCTION SchemeUpdate() RESULT (ok)  

IMPLICIT NONE  

INTEGER :: ok  

PHI\_M=PHI\_C  

PHI\_C=PHI\_P  

 $\text{ok}=0$   

END FUNCTION SchemeUpdate
!
```

```

!-----  

----  

END MODULE Class_NumericalMethod

```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
MODULE Class_WritetoGrads
USE Class_Fields, Only: PHI_P,PHI_C,PHI_M,Uvel,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='AdvecLinear1D_RA'
  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=irec) Yout

IF(CtrlWriteDataFile)OPEN(UnitData,FILE=TRIM(FileName)//'.bin',&
  FORM='UNFORMATTED', ACCESS='DIRECT',
  STATUS='UNKNOWN', &
  ACTION='WRITE', RECL=irec)
CtrlWriteDataFile=.FALSE.
Yout=REAL(PHI_C(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        ')')'phic 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****USE Class\_NumericalMethod, Only:****InitNumericalScheme, &****SchemeForward,SchemeUpdate,SchemeUpStream,Filter\_****RA****USE Class\_WritetoGrads, Only :InitClass\_WritetoGrads,****&****SchemeWriteData,SchemeWriteCtl****IMPLICIT NONE****INTEGER , PARAMETER :: r8=8****INTEGER , PARAMETER :: r4=4****INTEGER , PARAMETER :: xdim=1000****REAL (KIND=r8) , PARAMETER :: Uvel0=10.0!m/s****REAL (KIND=r8) , PARAMETER :: dt=0.08 !s****REAL (KIND=r8) , PARAMETER :: dx=1.0 !m****INTEGER , PARAMETER :: ninteraction=400****REAL (KIND=r8) , PARAMETER :: alfa=0.05****CALL Init()****CALL run()****CONTAINS****SUBROUTINE Init()****IMPLICIT NONE****CALL Init\_Class\_Fields(xdim,Uvel0,alfa)****CALL InitNumericalScheme(dt,dx)****CALL InitClass\_WritetoGrads****END SUBROUTINE Init****SUBROUTINE Run()****IMPLICIT NONE****INTEGER :: test,it,irec****irec=0****DO it=1,ninteraction****PRINT\*,it****test=SchemeUpStream()****test=Filter\_RA()****test=SchemeWriteData(irec)****test=SchemeUpdate()****END DO****test=SchemeWriteCtl(ninteraction)****END SUBROUTINE Run****SUBROUTINE Finalize()****IMPLICIT NONE****END SUBROUTINE Finalize****END PROGRAM Main**

# O RAW

```
MODULE Class_Fields
```

```
IMPLICIT NONE
```

```
PRIVATE
```

```
INTEGER, PUBLIC , PARAMETER :: r8=8
```

```
INTEGER, PUBLIC , PARAMETER :: r4=4
```

```
REAL (KIND=r8),PUBLIC ,ALLOCATABLE :: PHI_P(:)
```

```
REAL (KIND=r8),PUBLIC ,ALLOCATABLE :: PHI_C(:)
```

```
REAL (KIND=r8),PUBLIC ,ALLOCATABLE :: PHI_M(:)
```

```
REAL (KIND=r8),PUBLIC ,ALLOCATABLE :: Deslc(:)
```

```
REAL (KIND=r8),PUBLIC :: Uvel
```

```
INTEGER ,PUBLIC :: iMax
```

```
REAL (KIND=r8),PUBLIC :: alfa
```

```
REAL (KIND=r8),PUBLIC :: beta
```

```
PUBLIC :: Init_Class_Fields
```

```
CONTAINS
```

```
SUBROUTINE Init_Class_Fields(xdim,Uvel0,alfa_in,beta_in)
```

```
IMPLICIT NONE
```

```
INTEGER , INTENT (IN ) :: xdim
```

```
REAL (KIND=r8), INTENT (IN ):: Uvel0
```

```
REAL (KIND=r8), INTENT (IN ):: alfa_in
```

```
REAL (KIND=r8), INTENT (IN ):: beta_in
```

```
iMax=xdim
```

```
Uvel=Uvel0
```

```
alfa=alfa_in
```

```
beta=beta_in
```

```
ALLOCATE (PHI_P(-1:iMax+2))
```

```
ALLOCATE (PHI_C(-1:iMax+2))
```

```
ALLOCATE (PHI_M(-1:iMax+2))
```

```
ALLOCATE (Deslc(-1:iMax+2))
```

```
END SUBROUTINE Init_Class_Fields
```

```
END MODULE Class_Fields
```

```
MODULE Class_NumericalMethod
```

```
USE Class_Fields, Only : PHI_P,PHI_C,PHI_M,Deslc,Uvel,iMax,alf
```

```
IMPLICIT NONE
```

```
PRIVATE
```

```
INTEGER, PUBLIC , PARAMETER :: r8=8
```

```
INTEGER, PUBLIC , PARAMETER :: r4=4
```

```
REAL (KIND=r8) :: Dt
```

```
REAL (KIND=r8) :: Dx
```

```
PUBLIC :: InitNumericalScheme
```

```
PUBLIC :: SchemeForward
```

```
PUBLIC :: SchemeUpdate
```

```
PUBLIC :: SchemeUpStream
```

```
PUBLIC :: Filter_RAW
```

```
CONTAINS
```

```
SUBROUTINE InitNumericalScheme(dt_in,dx_in)
```

```
IMPLICIT NONE
```

```
REAL (KIND=r8), INTENT (IN ) :: dt_in
```

```
REAL (KIND=r8), INTENT (IN ) :: dx_in
```

```
INTEGER :: i
```

```
Dt=dt_in
```

```
Dx=dx_in
```

```
DO i=-1,iMax+2
```

```
IF (i*dx < 400.0) THEN
```

```
PHI_C(i)= 0.0
```

```
ELSE IF ( i*dx >= 400.0 .AND. i*dx <= 500.0 ) THEN
```

```
PHI_C(i)= 0.1*(i*dx -400.0)
```

```
ELSE IF ( i*dx >= 500.0 .AND. i*dx <= 600.0 ) THEN
```

```
PHI_C(i)= 20.0 - 0.1*(i*dx -400.0)
```

```
ELSE IF ( i*dx > 600.0 ) THEN
```

```
PHI_C(i)= 0.0
```

```
END IF
```

```
END DO
```

```
PHI_M=PHI_C
```

```
PHI_P=PHI_C
```

```
END SUBROUTINE InitNumericalScheme
```

```
FUNCTION SchemeForward() RESULT(ok)
```

```
IMPLICIT NONE
```

```
! Utilizando a diferenciacao forward
```

```
!
```

```
! F(j,n+1) - F(j,n)      F(j+1,n) - F(j,n)
```

```
!----- + u ----- = 0
```

```
!     dt           dx
```

```
!
```

```
INTEGER :: ok
```

```
INTEGER :: j
```

```
DO j=1,iMax
```

```
    PHI_P(j) = PHI_C(j) - (Uvel*Dx/Dt)*(PHI_C(j+1)-PHI_C(j))
```

```
END DO
```

```
CALL UpdateBoundaryLayer()
```

```
END FUNCTION SchemeForward
```

---

```
FUNCTION SchemeUpStream() RESULT (ok)
```

```
IMPLICIT NONE
```

```
! Utilizando a diferenciacao forward no tempo e
```

```
! backward no espaco (upstream)
```

```
!
```

```
! F(j,n+1) - F(j,n)      F(j,n) - F(j-1,n)
```

```
!----- + u ----- = 0
```

```
!     dt           dx
```

```
!
```

```
INTEGER :: ok
```

```
INTEGER :: j
```

```
DO j=1,iMax
```

```
    PHI_P(j) = PHI_C(j) - (Uvel*Dx/Dt)*(PHI_C(j)-PHI_C(j-1))
```

```
END DO
```

```
CALL UpdateBoundaryLayer()
```

```
END FUNCTION SchemeUpStream
```

```
FUNCTION Filter_RAW() RESULT (ok)
```

```
IMPLICIT NONE
```

```
INTEGER :: i
```

```
INTEGER :: ok
```

```
DO i=-1,iMax+2
```

```
    Deslc(i) = alfa*(PHI_M(i) - 2.0*PHI_C(i) + PHI_P(i) )
```

```
    PHI_C(i) = PHI_C(i) + Deslc(i)
```

```
    PHI_P(i) = PHI_P(i) + Deslc(i)*(beta-1.0)
```

```
END DO
```

```
ok=0
```

```
END FUNCTION Filter_RAW
```

```
IMPLICIT NONE
```

```
PHI_P(0) = PHI_P(iMax )
```

```
PHI_P(-1) = PHI_P(iMax-1)
```

```
PHI_P(imax+1) = PHI_P(1)
```

```
PHI_P(iMax+2) = PHI_P(2)
```

```
END SUBROUTINE UpdateBoundaryLayer
```

```
IMPLICIT NONE
```

```
INTEGER :: ok
```

```
PHI_M=PHI_C
```

```
PHI_C=PHI_P
```

```
ok=0
```

```
END FUNCTION SchemeUpdate
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
MODULE Class_WritetoGrads
USE Class_Fields, Only: PHI_P,PHI_C,PHI_M,Uvel,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='AdvecLinear1D_RAW'
  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=irec) Yout

IF(CtrlWriteDataFile)OPEN(UnitData,FILE=TRIM(FileName)//'.bin',&
n',&
  FORM='UNFORMATTED', ACCESS='DIRECT',
  STATUS='UNKNOWN', &
  ACTION='WRITE', RECL=irec)
CtrlWriteDataFile=.FALSE.
Yout=REAL(PHI_C(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        ')')'phic 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****USE Class\_NumericalMethod, Only:**

InitNumericalScheme, &amp;

SchemeForward,SchemeUpdate,SchemeUpStream,Filter\_  
RAW**USE Class\_WritetoGrads, Only :InitClass\_WritetoGrads,**  
&

SchemeWriteData,SchemeWriteCtl

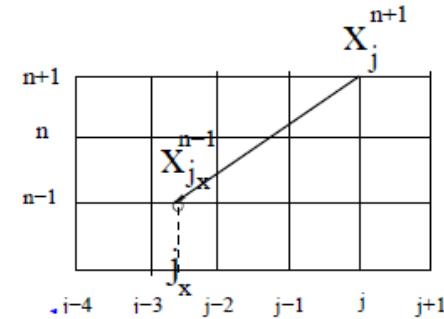
**IMPLICIT NONE****INTEGER , PARAMETER :: r8=8****INTEGER , PARAMETER :: r4=4****INTEGER , PARAMETER :: xdim=1000****REAL (KIND=r8) , PARAMETER :: Uvel0=10.0 !m/s****REAL (KIND=r8) , PARAMETER :: dt=0.08 !s****REAL (KIND=r8) , PARAMETER :: dx=1.0 !m****INTEGER , PARAMETER :: ninteraction=400****REAL (KIND=r8) , PARAMETER :: alfa=0.05****REAL (KIND=r8) , PARAMETER :: beta=0.05 !0.5 <****beta <= 1****CALL Init()****CALL run()****CONTAINS****SUBROUTINE Init()****IMPLICIT NONE****CALL Init\_Class\_Fields(xdim,Uvel0,alfa,beta)****CALL InitNumericalScheme(dt,dx)****CALL InitClass\_WritetoGrads****END SUBROUTINE Init****SUBROUTINE Run()****IMPLICIT NONE****INTEGER :: test,it,irec****irec=0****DO it=1,ninteraction****PRINT\*,it****test=SchemeUpStream()****test=Filter\_RAW()****test=SchemeWriteData(irec)****test=SchemeUpdate()****END DO****test=SchemeWriteCtl(ninteraction)****END SUBROUTINE Run****SUBROUTINE Finalize()****IMPLICIT NONE****END SUBROUTINE Finalize****END PROGRAM Main**

# Semi-Lagrangian

A equação de Advecção pode ser escrita na forma Euleriano ou Lagrangiana. i.e.

$$\frac{\partial \phi}{\partial t} = -u \frac{\partial \phi}{\partial x} \quad \text{Euleriano}$$

$$\frac{D\phi}{Dt} = 0 \quad \text{Lagrangiano}$$



Para um esquema semi-Lagrangiana de três vezes nível no tempo

$$\frac{D\phi}{Dt} \approx \frac{\phi_j^{n+1} - \phi_{j_x}^{n-1}}{2\Delta t} = 0$$

$$\Rightarrow \phi(x_j^{n+1}) = \phi(x_{j_x}^{n-1})$$

⇒ O valor futuro de  $\phi$  no ponto de chegada é igual ao valor anterior de  $\phi$  no ponto de partida.

O método Semi-Lagrangian requer dois cálculo importantes

O cálculo do ponto de partida

Interpolação do campos advectado do ponto de partida

# O cálculo do ponto de partida

Para uma velocidade constante caso (deixe  $\Delta t = 2\Delta T$ )

$$x_{jx}^{n-1} = x_j^{n+1} - u\Delta t$$

No caso da variável velocidade:  
Ponto repetitivo velocidade média

$$x_{jx}^{n-1} = x_j^{n+1} - u_m \Delta t$$

Série de Taylor aproximação (McGregor, 1993).

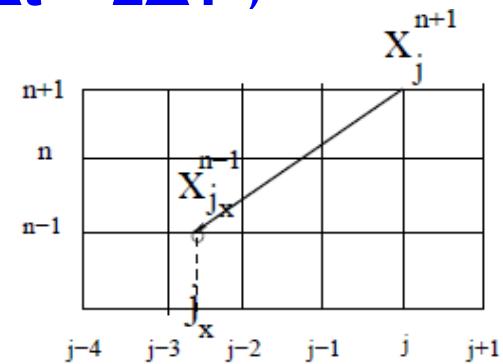
$$x_{jx}^{n-1} = x_j^{n+1} - \hat{u}\Delta t + \frac{\Delta t^2}{2} \hat{u} \frac{\partial \hat{u}}{\partial x} - \frac{\Delta t^3}{6} \left( \hat{u} \frac{\partial}{\partial x} \left( u \frac{\partial \hat{u}}{\partial x} \right) \right)$$

Para obter a função da ponderação para a interpolação

$$x_{jx}^{n-1} = x_{j-m} - \alpha$$

Onde,

$$\alpha = \frac{x_{j-m} - x_{jx}}{\Delta x}$$



# Interpolação do campo advectado no ponto de partida

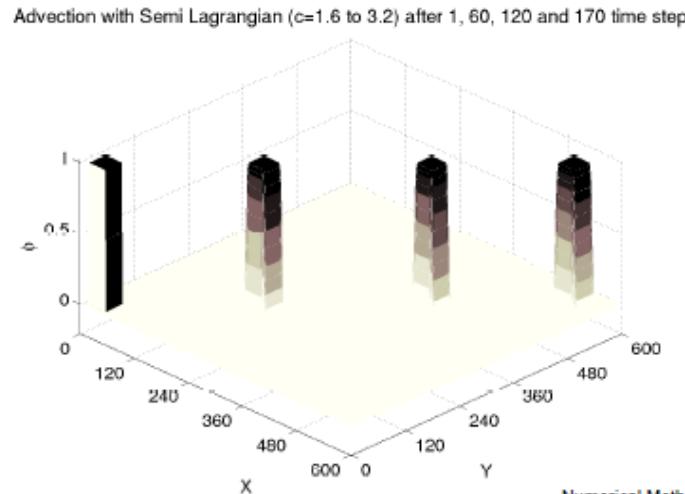
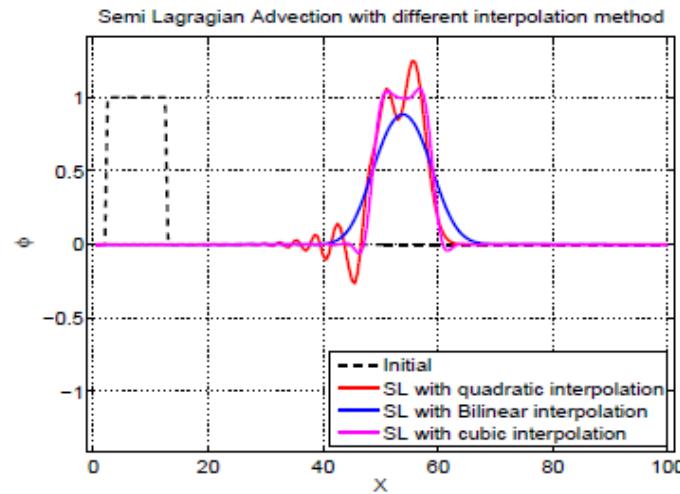
## Linear

$$\phi(x_{jx}^{n-1}) = \alpha\phi(x_{j-m-1}^{n-1}) + (1 - \alpha)\phi(x_{j-m}^{n-1})$$

## Metros Cúbicos

$$\begin{aligned}\phi(x_{jx}^{n-1}) = & -\frac{\alpha(1 - \alpha^2)}{6}\phi(x_{j-m-2}^{n-1}) + \frac{\alpha(1 + \alpha)(2 - \alpha)}{2}\phi(x_{j-m-1}^{n-1}) + \\ & \frac{(1 - \alpha^2)(2 - \alpha)}{2}\phi(x_{j-m}^{n-1}) - \frac{\alpha(1 - \alpha)(2 - \alpha)}{6}\phi(x_{j-m+1}^{n-1})\end{aligned}$$

## Precisão e monotonicity



Numerical Methods II – p. 35/56

# Análise de Estabilidade

Assumir um esquema de dois níveis no tempo e um de interpolação linear  $\emptyset$  no ponto de partida ou seja .

$$\emptyset_j^{n+1} = \emptyset_*^n = (1 - \alpha)\emptyset_{j-m}^n + \alpha\emptyset_{j-m-1}^n \quad (23)$$

(Note-se que quando  $m=0$ ,  $\alpha = \frac{u\Delta\tau}{\Delta x}$ , e 23 se torna Idêntico ao esquema diferencial **upstream**). assim como para outros esquemas de advecção vamos supor que uma solução na forma: $\emptyset_j^n = A^n e^{ikj\Delta x}$

A substituição de 23 nós, obter

$$|A| = [1 - 2\alpha(1 - \alpha)[1 - \cos(k\Delta x)]]^{0.5} \quad (24)$$

Portanto  $|A| \leq 1$  Enquanto  $\alpha(1 - \alpha) \geq 0$  Ou seja,  $0 \leq \alpha \leq 1$

O esquema é, portanto, estável se os pontos da interpolação são os dois são mais próximos do ponto de partida, mas é neutro somente

Se  $\alpha = 0$  Ou  $\alpha = 1$ , Ou seja, quando a interpolação não é necessária

# Esquema implícito: FTBS

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

$$\frac{\phi_j^{n+1} - \phi_j^n}{\Delta t} + u \frac{\phi_j^{n+1} - \phi_{j-1}^{n+1}}{\Delta x} = 0 \quad (25)$$

$$\phi_j^{n+1} - \phi_j^n = -u \frac{\Delta t}{\Delta x} (\phi_j^{n+1} - \phi_{j-1}^{n+1})$$

$$\phi_j^{n+1} = \phi_j^n - u \frac{\Delta t}{\Delta x} (\phi_j^{n+1} - \phi_{j-1}^{n+1})$$

$$\phi_j^{n+1} = \phi_j^n - C (\phi_j^{n+1} - \phi_{j-1}^{n+1})$$

$$\phi_j^{n+1} = \phi_j^n - C \phi_j^{n+1} + C \phi_{j-1}^{n+1}$$

$$\phi_j^{n+1} + C \phi_j^{n+1} = \phi_j^n + C \phi_{j-1}^{n+1}$$

$$(1 + C) \phi_j^{n+1} = \phi_j^n + C \phi_{j-1}^{n+1}$$

$$\phi_j^{n+1} = \frac{1}{(1 + C)} (\phi_j^n + C \phi_{j-1}^{n+1}) \quad (26)$$

**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.

$$\phi_j^{n+1} = \frac{1}{(1 + C)}(\phi_j^n + C\phi_{j-1}^{n+1}) \quad (27)$$

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

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

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

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

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

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

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

$$(1+C)A = (1 + CA e^{ik\Delta x})$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**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 escrita na forma:

Aqui o nosso método upwind torna-se :

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

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

$$\phi_j^{n+1} - \phi_j^n = -u \frac{\Delta t}{\Delta x} (\phi_j^{n+1} - \phi_{j-1}^{n+1})$$

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

$$\phi_j^{n+1} - \phi_j^n = -C(\phi_j^{n+1} - \phi_{j-1}^{n+1})$$

$$\phi_j^{n+1} - \phi_j^n = -C\phi_j^{n+1} + C\phi_{j-1}^{n+1}$$

$$\phi_j^{n+1} + C\phi_j^{n+1} - C\phi_{j-1}^{n+1} = \phi_j^n$$

$$-C\phi_{j-1}^{n+1} + (1 + C)\phi_j^{n+1} = \phi_j^n$$

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  $0 \leq X \leq 1000M$ . 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  $\phi(0,t) = 0$ .

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

Usando o esquema BTBS e mostrar soluções para  $T = 0S$   
 $T = 100^{\circ}-s$ ,  $T = 200S$   $T = 300S$  ,  $T = 800S$ .

1. Compare as soluções de  $26^{\circ}$  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)
!
```

```

! 
$$\frac{a(t+1,i) - a(t,i)}{Dt} = -\frac{a(t+1,i) - a(t+1,i-1)}{Dx}$$

!
! 
$$a(t+1,i) - a(t,i) = -U \frac{a(t+1,i) - a(t+1,i-1)}{Dx}$$

!
! 
$$a(t+1,i) - a(t,i) = -C \frac{a(t+1,i) - a(t+1,i-1)}{Dx}$$

!
```

```

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

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

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

!
! 
$$-C a(t+1, 0) + (1 + C) a(t+1,1) = a(t,1)$$

! 
$$-C a(t+1, 1) + (1 + C) a(t+1,2) = a(t,2)$$

! 
$$-C a(t+1, 2) + (1 + C) a(t+1,3) = a(t,3)$$

! 
$$-C a(t+1, 3) + (1 + C) a(t+1,4) = a(t,4)$$

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

!
! 
$$(1 + C) \begin{vmatrix} a(t+1,1) \\ a(t+1,2) \\ a(t+1,i-1) \end{vmatrix} = a(t,i)$$

! 
$$-C \begin{vmatrix} a(t+1,1) \\ a(t+1,2) \\ a(t+1,i-1) \end{vmatrix} = a(t,i)$$

!
! 
$$A \quad X = B$$

!
!# 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\_WritetoGradsUSE 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 :: irec

REAL (KIND=r4) :: Yout(iMax)

INQUIRE (IOLENGTH=irec) Yout

IF(CtrlWriteDataFile) **OPEN** (UnitData,FILE=TRIM(FileName) //' .bin', &

FORM='UNFORMATTED', ACCESS='DIRECT', STATUS='UNKNOWN', &amp;

ACTION='WRITE', RECL=irec)

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

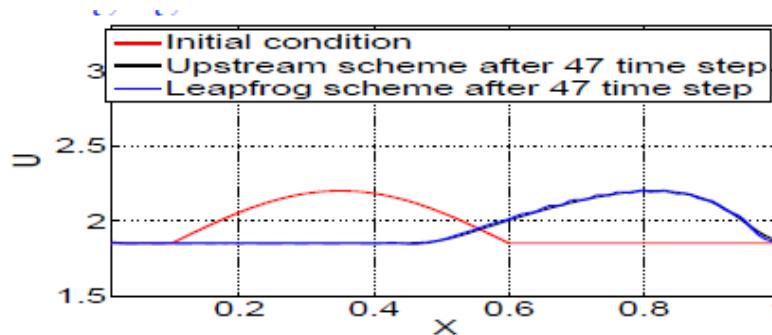
**END PROGRAM** Main

# A advecção regime não-linear

Equação de Advecção Linear pode ser usada para modelar a advecção de algumas quantidades (por exemplo, temperatura, umidade) pelo vento, mas A equação para a energia eólica propriamente dita não é linear ou seja

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = 0$$

A não-linearidade permite mais interesse na dinâmica. Mas pode causar problemas numéricos tanto por meio de truncamento e a estabilidade. Ele gera grande gradiente



A solução mais simples é a de incluir a difusão que vai  
Evitar a formação de grandes gradientes

# A equação de difusão

Considerar a difusão equação linear

$$\frac{\partial \phi}{\partial t} = K \frac{\partial^2 \phi}{\partial^2 x}, \quad \text{onde, } K = \text{constante}$$

Assumir o domínio é periódicos em  $x$ . Se o estado inicial ( $t = 0$ ) é

$$\phi(x, 0) = e^{ikx}$$

Em seguida, a solução exata pode ser encontrado da seguinte forma.  
Assuma a solução  $\phi(x, t) = \Phi(t)e^{ikx}$ . E substitua na equação de  
Difusão

$$\Phi(t) = \Phi_0 e^{-k^2 Kt}$$

$$t = 0$$

$$\Phi(0) = \Phi_0$$

$$\phi(x, t) = \Phi_0 e^{-k^2 Kt} e^{ikx}$$

Considere a amplitude  $\Phi_0 = 1$ , Portanto a solução da  
equação de difusão torna-se:

$$\phi(x, t) = e^{-k^2 Kt} e^{ikx}$$

$$\phi(x, t) = e^{-k^2 K t} e^{ikx}$$

## Substituindo na equação de difusão

$$\frac{\partial \phi}{\partial t} = K \frac{\partial^2 \phi}{\partial^2 x}, \quad \text{onde, } K = \text{constante}$$

$$\frac{\partial \phi(x, t)}{\partial t} = -k^2 K \Phi_0 e^{-k^2 K t} e^{ikx}$$

$$\frac{\partial \phi(x, t)}{\partial x} = ik \Phi_0 e^{-k^2 K t} e^{ikx}$$

$$\frac{\partial^2 \phi(x, t)}{\partial^2 x} = -k^2 \Phi_0 e^{-k^2 K t} e^{ikx}$$

$$\frac{\partial \phi}{\partial t} = K \frac{\partial^2 \phi}{\partial^2 x}$$

$$-k^2 K \Phi_0 e^{-k^2 K t} e^{ikx} = -k^2 K \Phi_0 e^{-k^2 K t} e^{ikx}$$

Assim a solução para a equação de difusão satisfaz a equação diferencial

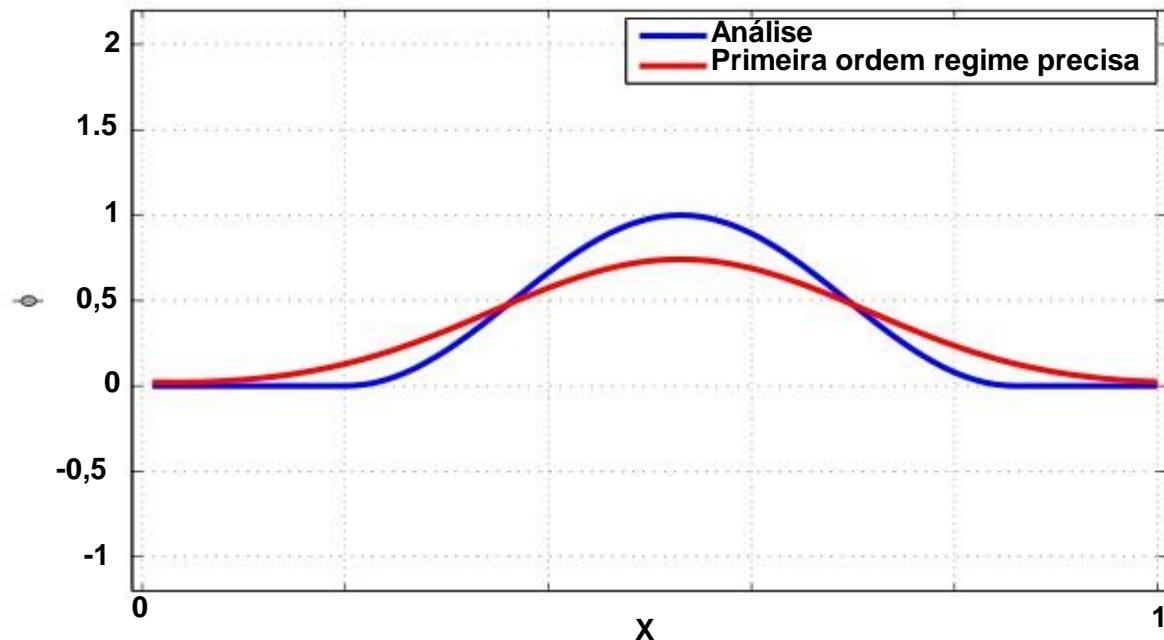
$$\frac{\partial \Phi(t) e^{ikx}}{\partial t} = -k^2 K \Phi(t) e^{ikx}$$

A solução é uma onda estacionária diminuindo a amplitude. O amortecimento é mais rápido para ondas curtas de que ondas longas.

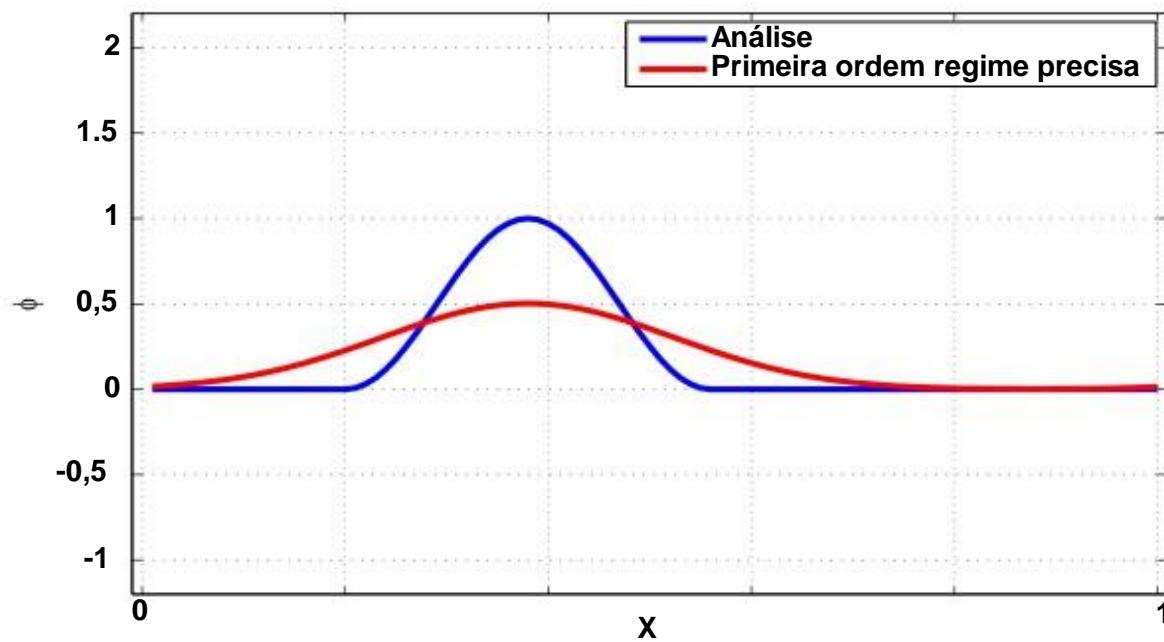
Solução analítica

$$\emptyset(x, t) = \Phi_0 e^{-k^2 K t} e^{ikx}$$

Solução apos 200 tempo passo



Solução apos 400 tempo passo



# Estabilidade

Considere uma Equação diferencial ordinária usando o esquema diferença centralizada de segunda ordem

$$\frac{\partial \phi}{\partial t} = K \frac{\partial^2 \phi}{\partial^2 x}, \quad \text{onde, } K = \text{constante}$$

$$\frac{\phi_j^{n+1} - \phi_j^{n-1}}{2\Delta t} = K \left( \frac{\phi_{j+1}^n - 2\phi_j^n + \phi_{j-1}^n}{\Delta x^2} \right)$$

Supõem-se que a solução é dada pela equação:  $\phi_j^n = A^n e^{ikj\Delta x}$ , substituindo o esquema numérico

$$\frac{A^{n+1} e^{ikj\Delta x} - A^{n-1} e^{ikj\Delta x}}{2\Delta t} = K \left( \frac{A^n e^{ik(j+1)\Delta x} - 2A^n e^{ik(j)\Delta x} + A^n e^{ik(j-1)\Delta x}}{\Delta x^2} \right)$$

$$A^{n+1} e^{ikj\Delta x} = A^{n-1} e^{ikj\Delta x} + \frac{2K\Delta t}{\Delta x^2} (A^n e^{ik(j+1)\Delta x} - 2A^n e^{ik(j)\Delta x} + A^n e^{ik(j-1)\Delta x})$$

$$A^n A e^{ikj\Delta x} = A^n A^{-1} e^{ikj\Delta x} + \frac{2K\Delta t}{\Delta x^2} (A^n e^{ikj\Delta x} e^{ik\Delta x} - 2A^n e^{ikj\Delta x} + A^n e^{ikj\Delta x} e^{-ik\Delta x})$$

$$A^n A e^{ikj\Delta x} = A^n A^{-1} e^{ikj\Delta x} + \frac{2K\Delta t}{\Delta x^2} (A^n e^{ikj\Delta x} e^{ik\Delta x} - 2A^n e^{ikj\Delta x} + A^n e^{ikj\Delta x} e^{-ik\Delta x})$$

**Eliminando os termos  $A^n e^{ikj\Delta x}$**

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

**Multiplica a equação por  $A$**

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

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

$$\cos(k\Delta x) = \frac{e^{ik\Delta x} + e^{-ik\Delta x}}{2}$$

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

$$A^2 - A \frac{2K\Delta t}{\Delta x^2} (-2 + 2\cos(k\Delta x)) - 1 = 0$$

$$A^2 - A \frac{2K\Delta t}{\Delta x^2} (-2 + 2 \cos(k\Delta x)) - 1 = 0$$

$$A^2 + A \frac{2K\Delta t}{\Delta x^2} (2 - 2 \cos(k\Delta x)) - 1 = 0$$

$$ax^2 + bx + c = 0$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$A = \frac{-\frac{2K\Delta t}{\Delta x^2} (2 - 2 \cos(k\Delta x)) \pm \sqrt{\left(\frac{2K\Delta t}{\Delta x^2} (2 - 2 \cos(k\Delta x))\right)^2 + 4}}{2}$$

$$A = \frac{-\frac{4K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x)) \pm \sqrt{\left(\frac{4K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x))\right)^2 + 4}}{2}$$

$$A = -\frac{4K\Delta t}{2\Delta x^2} (1 - \cos(k\Delta x)) \pm \sqrt{\frac{1}{4}\left(\frac{4K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x))\right)^2 + \frac{4}{4}}$$

$$A = -\frac{4K\Delta t}{2\Delta x^2} (1 - \cos(k\Delta x)) \pm \sqrt{\frac{1}{4} \left( \frac{4K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x)) \right)^2 + \frac{4}{4}}$$

$$A = -\frac{2K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x)) \pm \sqrt{\left( \frac{4K\Delta t}{2\Delta x^2} (1 - \cos(k\Delta x)) \right)^2 + 1}$$

$$A = -\frac{2K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x)) \pm \sqrt{\left( \frac{2K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x)) \right)^2 + 1}$$

$$A = -\frac{2K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x)) \pm \sqrt{\left( \frac{2K\Delta t}{\Delta x^2} \right)^2 (1 - \cos(k\Delta x))^2 + 1}$$

Ambas as raízes  $A_+$  e  $A_-$  são reais ( $(1 - \cos(k\Delta x)) > 0$ ) e o seu produto é  $A_+ A_- = -1$ .  $A_- \leq -1$

$$A_+ A_- = \left( -\frac{2K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x)) + \sqrt{\left( \frac{2K\Delta t}{\Delta x^2} \right)^2 (1 - \cos(k\Delta x))^2 + 1} \right) \left( -\frac{2K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x)) \right)$$

$$A_+ A_- = \left( -\frac{2K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x)) + \sqrt{\left(\frac{2K\Delta t}{\Delta x^2}\right)^2 (1 - \cos(k\Delta x))^2 + 1} \right) \left( -\frac{2K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x)) \right)$$

$$A_+ A_- = \frac{\left( -\frac{2K\Delta t}{\Delta x^2} \left( \frac{1}{\Delta x^2} \right) \cos(k\Delta x) + \sqrt{\left(\frac{2K\Delta t}{\Delta x^2}\right)^2 (1 - \cos(k\Delta x))^2 + 1} \right)}{+\left( \frac{2K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x)) \right) \sqrt{\left(\frac{2K\Delta t}{\Delta x^2}\right)^2 (1 - \cos(k\Delta x))^2 + 1}} \\ - \left( \frac{2K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x)) \right) \sqrt{\left(\frac{2K\Delta t}{\Delta x^2}\right)^2 (1 - \cos(k\Delta x))^2 + 1}$$

$$A_+ A_- = \frac{\left( \frac{2K\Delta t}{\Delta x^2} \left( \frac{2K\Delta t}{\Delta x^2} \right) \cos(k\Delta x) + \sqrt{\left(\frac{2K\Delta t}{\Delta x^2}\right)^2 (1 - \cos(k\Delta x))^2 + 1} \right)}{+\left( \frac{2K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x)) \right) \sqrt{\left(\frac{2K\Delta t}{\Delta x^2}\right)^2 (1 - \cos(k\Delta x))^2 + 1}} \\ - \left( \frac{2K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x)) \right) \sqrt{\left(\frac{2K\Delta t}{\Delta x^2}\right)^2 (1 - \cos(k\Delta x))^2 + 1}$$

$$\begin{aligned}
 A_+ A_- &= \left( \frac{2K\Delta t}{\Delta x^2} \right)^2 (1 - \cos(k\Delta x))^2 \\
 &\quad + \left( \frac{2K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x)) \right) \sqrt{\left( \frac{2K\Delta t}{\Delta x^2} \right)^2 (1 - \cos(k\Delta x))^2 + 1} \\
 &\quad - \left( \frac{2K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x)) \right) \sqrt{\left( \frac{2K\Delta t}{\Delta x^2} \right)^2 (1 - \cos(k\Delta x))^2 + 1} \\
 &\quad \cancel{(2K\Delta t)^2} \\
 A_+ A_- &= -1
 \end{aligned}$$

$$A_- \leq -1$$

$$A = -\frac{2K\Delta t}{\Delta x^2} (1 - \cos(k\Delta x)) - \sqrt{\left( \frac{2K\Delta t}{\Delta x^2} \right)^2 (1 - \cos(k\Delta x))^2 + 1}$$

$<0 \qquad <0$

Ambas as raízes  $A_+$  e  $A_-$  São reais e o seu produto é  $A_+ A_- = -1$ .

$A_- \leq -1$  (Usando  $(1 + y)^{1/2} \approx 1 + 1/2y$ ), Resultando em  $|A| \geq 1$ . Este Corresponde ao modo computacional e é incondicionalmente instável.

$A_+$  situa-se entre 0 e 1 E isso corresponde ao modo físico.

# Estabilidade

Agora, considere a diferença forward de primeira ordem para a derivada no tempo.

$$\frac{\partial \phi}{\partial t} = K \frac{\partial^2 \phi}{\partial^2 x}, \quad \text{onde, } K = \text{constante}$$

$$\frac{\phi_j^{n+1} - \phi_j^n}{\Delta t} = K \left( \frac{\phi_{j+1}^n - 2\phi_j^n + \phi_{j-1}^n}{\Delta x^2} \right)$$

Supõem-se que a solução é dada pela equação:  $\phi_j^n = A^n e^{ikj\Delta x}$ , substituindo o esquema numérico. A análise de estabilidade de Von Neumann leva à seguinte equação para o fator de ampliação

$$\frac{A^{n+1} e^{ikj\Delta x} - A^n e^{ikj\Delta x}}{\Delta t} = K \left( \frac{A^n e^{ik(j+1)\Delta x} - 2A^n e^{ik(j)\Delta x} + A^n e^{ik(j-1)\Delta x}}{\Delta x^2} \right)$$

$$A^{n+1} e^{ikj\Delta x} = A^n e^{ikj\Delta x} + \frac{K\Delta t}{\Delta x^2} (A^n e^{ik(j+1)\Delta x} - 2A^n e^{ik(j)\Delta x} + A^n e^{ik(j-1)\Delta x})$$

$$A^n A e^{ikj\Delta x} = A^n e^{ikj\Delta x} + \frac{K\Delta t}{\Delta x^2} (A^n e^{ikj\Delta x} e^{ik\Delta x} - 2A^n e^{ikj\Delta x} + A^n e^{ikj\Delta x} e^{-ik\Delta x})$$

$$A^n A e^{ikj\Delta x} = A^n e^{ikj\Delta x} + \frac{K\Delta t}{\Delta x^2} (A^n e^{ikj\Delta x} e^{ik\Delta x} - 2A^n e^{ikj\Delta x} + A^n e^{ikj\Delta x} e^{-ik\Delta x})$$

**Eliminado os termos:**  $A^n e^{ikj\Delta x}$

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

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

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

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

**análise de estabilidade de Von Neumann leva à seguinte equação para o fator de ampliação**

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

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

**Agora**  $0 \leq (1 + \cos(k\Delta x)) \leq 2$ , **Então**  $|A| \leq 1$  se  $\frac{2K\Delta t}{\Delta x^2} \leq 1$ ,  
**Ou seja, se**  $\Delta t \leq \frac{\Delta x^2}{2K}$ .

$$|A|^2 = \left(1 - \frac{2K\Delta t}{\Delta x^2} (1 + \cos(k\Delta x))\right) \left(1 - \frac{2K\Delta t}{\Delta x^2} (1 + \cos(k\Delta x))\right)$$

$$|A|^2 = \left(1 - \frac{2K\Delta t}{\Delta x^2} (1 + \cos(k\Delta x)) + \frac{2K\Delta t}{\Delta x^2} (1 + \cos(k\Delta x)) + \left(\frac{2K\Delta t}{\Delta x^2} (1 + \cos(k\Delta x))\right)^2\right)$$

$$|A|^2 = \left(1 + \left(\frac{2K\Delta t}{\Delta x^2} (1 + \cos(k\Delta x))\right)^2\right)$$

**Este regime é condicionalmente estável.** Note-se que devido à dependência do quadrado de  $\Delta x$ , um Aumento moderado na resolução podem exigir um muito menor no intervalo de tempo  $\Delta t$ . Ex. fazer a análise de estabilidade para o esquema de versões anteriores da terivada temporal

```

MODULE Difusion1D
IMPLICIT NONE
PRIVATE
PUBLIC :: Init,Run,Finalize
INTEGER :: NX
REAL :: X0
REAL :: DX
REAL :: DT
REAL :: kappa
REAL :: total_time
REAL, ALLOCATABLE :: X (:)
REAL, ALLOCATABLE :: T (:)
REAL, ALLOCATABLE :: Tp (:)
CHARACTER (LEN=2), ALLOCATABLE :: CT (:)

CONTAINS

SUBROUTINE Init(NX_IN,X0_IN,DX_IN,kappa_in,DT_IN,total_time_in)
  INTEGER, INTENT (IN ) :: NX_IN
  REAL, INTENT (IN ) :: X0_IN
  REAL, INTENT (IN ) :: DX_IN
  REAL, INTENT (IN ) :: kappa_in
  REAL, INTENT (IN ) :: DT_IN
  REAL, INTENT (IN ) :: total_time_in
  INTEGER :: i
  NX=NX_IN
  X0=X0_IN
  DX=DX_IN
  kappa=kappa_in
  DT=DT_IN
  total_time=total_time_in
  ALLOCATE (X (1:NX ));X=0.0
  ALLOCATE (T (0:NX+1));T=0.0
  ALLOCATE (Tp(0:NX+1));Tp=0.0
  ALLOCATE (CT(0:NX+1));CT=' '
  FORALL (i=1:NX)X(i) = X0 + (i-1)*DX
  T=273.16
  T(0:1)= 300.0
  CT='.'

```

**END SUBROUTINE** Init

```

SUBROUTINE Run()
!
! dT      d dT
!---- = k * -----
! dt      dx dx
!
! k=1
!
IMPLICIT NONE
INTEGER :: itr
INTEGER :: i

DO itr=1,(total_time/DT)
  DO i=1,NX
    Tp(i) = T(i) + DT*(kappa*((T(i+1) + T(i-1) - 2*T(i))/2*DX))
  END DO
  T(1:NX)=Tp(1:NX)
  WHERE (T /= 273.16)CT='X '
  PRINT *, 'CT' ,MAXVAL(T),MINVAL(T)
  WRITE(*,'(50A)')(CT(i),i=0,NX-1)

  END DO

END SUBROUTINE Run

SUBROUTINE Finalize()
IMPLICIT NONE
DEALLOCATE (X )
DEALLOCATE (T )
DEALLOCATE (Tp)
DEALLOCATE (CT)

END SUBROUTINE Finalize

END MODULE Difusion1D

```

!-----

## PROGRAM MAIN

```
USE Difusion1D, Only :Init,Run,Finalize
IMPLICIT NONE
INTEGER, PARAMETER :: NX =50
REAL , PARAMETER :: DX =
5.0/REAL(NX)!m
REAL , PARAMETER :: alfa =10.0 ! 0.1
-> 1
REAL , PARAMETER :: total_time
=300.0
REAL , PARAMETER :: kappa=2.0
REAL , PARAMETER :: DT=
(alfa*(DX**2))/kappa
REAL :: X0=1!m
PRINT*,total_time/DT
CALL
Init(NX,X0,DX,kappa,DT,total_time)
CALL Run()
CALL 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.

$$\phi(x) = \begin{cases} \frac{-T_0 + T_{\infty} + 2\phi_{\infty}}{273.15 + 20 - 20r} & r \leq x \leq 1 \\ 0.5 < x < 1 \end{cases}$$

Ambas as extremidades são mantidos na mesma temperatura  $T_0 = 273.15K$ .

```

MODULE Class_Fields
IMPLICIT NONE
PRIVATE
INTEGER, PUBLIC , PARAMETER :: r8=8
INTEGER, PUBLIC , PARAMETER :: r4=4
REAL(KIND=r8),PUBLIC ,ALLOCATABLE :: PHI_P(:)
REAL(KIND=r8),PUBLIC ,ALLOCATABLE :: PHI_C(:)
REAL(KIND=r8),PUBLIC ,ALLOCATABLE :: PHI_M(:)
REAL(KIND=r8),PUBLIC :: Coef_K
INTEGER ,PUBLIC :: iMax
PUBLIC :: Init_Class_Fields
CONTAINS
!-----

```

```

SUBROUTINE Init_Class_Fields(xdim,Coef_K0)
IMPLICIT NONE
INTEGER , INTENT(IN ) :: xdim
REAL(KIND=r8), INTENT(IN ):: Coef_K0
iMax=xdim
Coef_K=Coef_K0
ALLOCATE(PHI_P(-1:iMax+2))
ALLOCATE(PHI_C(-1:iMax+2))
ALLOCATE(PHI_M(-1:iMax+2))
END SUBROUTINE Init_Class_Fields
!
```

```

END MODULE Class_Fields

```

```

MODULE Class_NumericalMethod
USE Class_Fields, Only : PHI_P,PHI_C,PHI_M,Coef_K,iMax
IMPLICIT NONE
PRIVATE
INTEGER, PUBLIC , PARAMETER :: r8=8
INTEGER, PUBLIC , PARAMETER :: r4=4
REAL(KIND=r8) :: Dt
REAL(KIND=r8) :: Dx

PUBLIC :: InitNumericalScheme
PUBLIC :: SchemeUpdate
PUBLIC :: SchemeUpStream
CONTAINS
!-----

```

```

--- SUBROUTINE InitNumericalScheme(dt_in,dx_in)
IMPLICIT NONE
REAL(KIND=r8), INTENT(IN ) :: dt_in
REAL(KIND=r8), INTENT(IN ) :: dx_in
INTEGER :: i
REAL(KIND=r8) :: x
Dt=dt_in
Dx=dx_in
PHI_C = 273.15
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
PHI_M=PHI_C
PHI_P=PHI_C
END SUBROUTINE InitNumericalScheme

```

!-----

```
FUNCTION SchemeUpStream() RESULT(ok)
IMPLICIT NONE
! Utilizando a diferenciacao forward no tempo e
! backward no espaco (upstream)
!
! 
$$F(j,n+1) - F(j,n) - \frac{F(j+1,n) - 2F(j,n) + F(j-1,n)}{dt} = 0$$

!----- + u ----- = 0
!      dt          dx
!
INTEGER :: ok
INTEGER :: j
DO j=1,iMax
    PHI_P(j) = PHI_C(j) - (Coef_K*Dt/(Dx**2))* &
                (PHI_C(j+1) - 2.0*PHI_C(j) + PHI_C(j-1))
END DO
CALL UpdateBoundaryLayer()
END FUNCTION SchemeUpStream
!-----
```

---

```
SUBROUTINE UpdateBoundaryLayer()
IMPLICIT NONE
PHI_P(0) = 273.15 !PHI_P(iMax )
PHI_P(-1) = 273.15 !PHI_P(iMax-1)
PHI_P(imax+1) = 273.15 !PHI_P(1)
PHI_P(iMax+2) = 273.15 !PHI_P(2)
END SUBROUTINE UpdateBoundaryLayer
!-----
```

---

```
FUNCTION SchemeUpdate() RESULT(ok)
IMPLICIT NONE
INTEGER :: ok
PHI_M=PHI_C
PHI_C=PHI_P
ok=0
END FUNCTION SchemeUpdate
!-----
```

---

## MODULE Class\_WritetoGrads

```
USE Class_Fields, Only:PHI_P,PHI_C,PHI_M,Coef_K,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='Diffusion1D'
```

```
CtrlWriteDataFile=.TRUE.
```

```
END SUBROUTINE InitClass_WritetoGrads
```

```
FUNCTION SchemeWriteData(irec) RESULT(ok)
```

```
IMPLICIT NONE
```

```
INTEGER , INTENT(INOUT) :: irec
```

```
INTEGER :: ok
```

```
INTEGER :: irec
```

```
REAL(KIND=r4) :: Yout(iMax)
```

```
INQUIRE(IOLENGTH=irec) Yout
```

```
IF(CtrlWriteDataFile)OPEN(UnitData,FILE=TRIM(FileName)//'.bin', &
```

```
FORM='UNFORMATTED',ACCESS='DIRECT',STATUS='UNKNOWN')
```

```
ACTION='WRITE',RECL=irec)
```

```
ok=1
```

```
CtrlWriteDataFile=.FALSE.
```

```
Yout=REAL(PHI_C(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
ok=1

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 2'
  WRITE(UnitCtl,'(A      )')'phic 0 99 resultado da edol yc'
  WRITE(UnitCtl,'(A      )')'phia 0 99 solucao analitica ya'
  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
USE Class_NumericalMethod, Only:InitNumericalScheme,&
  SchemeUpdate,SchemeUpStream
USE Class_WritetoGrads, Only :InitClass_WritetoGrads, &
  SchemeWriteData,SchemeWriteCtl
IMPLICIT NONE
INTEGER , PARAMETER :: r8=8
INTEGER , PARAMETER :: r4=4
REAL(KIND=r8) , PARAMETER :: dx=0.01 !m
REAL(KIND=r8) , PARAMETER :: LX=1.0
INTEGER , PARAMETER :: xdim=LX/dx
REAL(KIND=r8) , PARAMETER :: Coef_K0=2.9e-5!m/s
REAL(KIND=r8) , PARAMETER :: dt=0.1*(dx**2)/Coef_K0 !
  ! c*Dt/Dx < 1 => dt < dx/Coef_K0
INTEGER , PARAMETER :: ninteraction=20000
CALL Init()
CALL run()
CONTAINS
SUBROUTINE Init()
IMPLICIT NONE
CALL Init_Class_Fields(xdim,Coef_K0)
CALL InitNumericalScheme(dt,dx)
CALL InitClass_WritetoGrads
END SUBROUTINE Init
SUBROUTINE Run()
IMPLICIT NONE
INTEGER :: test,tn,irec
irec=0
DO tn=0,ninteraction
  PRINT*,(tn)*dt
  test=SchemeUpStream()
  test=SchemeWriteData(irec)
  test=SchemeUpdate()
END DO
test=SchemeWriteCtl(ninteraction)
END SUBROUTINE Run
END PROGRAM Main

```

# Advecção-difusão

Considere um processo de advecção-difusão

$$\frac{\partial \phi}{\partial t} + u \frac{\partial}{\partial x} \phi = K \frac{\partial^2 \phi}{\partial x^2}, \quad \text{onde } K = \text{constante} \quad (52)$$

Temos visto que leapfrog é condicionalmente estável para advecção e a aproximação forward é condicionalmente estável para a difusão

Pode-se utilizar a combinação do esquema forward para difusão E leapfrog para advecção

Para a difusão, use passo tempo  $2\Delta t$ , por conseguinte

$$\frac{\phi_j^{n+1} - \phi_j^{n-1}}{2\Delta t} + u \frac{\phi_{j+1}^n - \phi_{j-1}^n}{2\Delta t} = K \frac{\phi_{j+1}^{n-1} - 2\phi_j^{n-1} + \phi_{j-1}^{n-1}}{\Delta x^2}$$

NB. O termo de difusão é calculado com base em um passo de tempo anterior em relação ao outros termos.

$$\frac{\emptyset_j^{n+1}-\emptyset_j^{n-1}}{2\Delta t}=-u\frac{\emptyset_{j+1}^n-\emptyset_{j-1}^n}{2\Delta t}+K\frac{\emptyset_{j+1}^{n-1}-2\emptyset_j^{n-1}+\emptyset_{j-1}^{n-1}}{\Delta x^2}$$

$$\emptyset_j^{n+1}=\emptyset_j^{n-1}-\frac{u\Delta t}{\Delta t}\big(\emptyset_{j+1}^n-\emptyset_{j-1}^n\big)+\frac{2K\Delta t}{\Delta x^2}\big(\emptyset_{j+1}^{n-1}-2\emptyset_j^{n-1}+\emptyset_{j-1}^{n-1}\big)$$

# Advecção-difusão: Exercício

Resolver a equação advecção-difusão numericamente, na 52. Domain  $0 \leq X \leq 1000M$ . Deixe  $\Delta x = 1 M$ , assumir Condições-limite periódica . Suponha que a velocity advecção  $U = 1$

M/s e o coeficiente de difusão  $K = 2,9 \times 10^{-5}M^2/S$ . Deixe o Condicionamento inicial ser um triângulo

$$\Phi(x,0) = \begin{cases} 0.1(X - 400) & \text{Para } X < 400 \\ 20 - 0.1(x - 400) & \text{Para } 400 \leq X \leq 500 \\ 0 & \text{Para } 500 \leq X \leq 600 \\ & \text{Para } X > 600 \end{cases}$$

Escolha o intervalo de tempo, que o sistema é estável. Integrar Para 2000s usando os seguintes esquemas, e mostrar as soluções para  $T = 0S$   $T = 400S$   $T = 1000S$   $T = 2000S$ .

1. Esquema Leap frog com filtro de tempo RAW (use  $A = 0,25$  E  $B = 0,70$ )

# Modelagem gravity waves

Considere as equações da água rasa one-dimensional , com A força de Coriolis negligenciadas, e linearizado sobre um estado de descanso  $\mathbf{U} = \mathbf{0}$ ,  $\Phi = \Phi = gH$

$$\frac{\partial u}{\partial t} + \frac{\partial \varphi}{\partial x} = 0 \quad \text{Dinâmica} \quad (53)$$

$$\frac{\partial \varphi}{\partial t} + \Phi \frac{\partial u}{\partial x} = 0 \quad \text{Missa} \quad (54)$$

Estas equações dão suporte para as soluções de forma

$$U(x, t) = \hat{u} e^{i(kx - \omega t)} \quad \Phi(x, t) = \varphi e^{i(kx - \omega t)}$$

Com a dispersão relação  $\Omega_2 = K_2 \Phi$ . Estas ondas são Não-dispersivas e todos eles têm a mesma velocidade de fase  $\Omega/k = \Phi$ . Por exemplo considere a solução para as condições iniciais de  $\Phi(x, 0) = f(x)$  e  $U(x, 0) = 0$ , e o domínio periódico (ou infinito). Em seguida, as soluções para  $\Phi$  é:  
 $U(x, t) = i_2 \Phi(x-) + i_2 \Phi(x+a)$

# CTCS as solução ondas de gravidade

Discretize utilizando equações da água rasa de segunda ordem com diferenças centrada no espaço e no tempo.

$$\frac{U_j^{n+1} - U_j^{n-1}}{2\Delta t} + \frac{\Phi_{j+1}^n - \Phi_{j-1}^n}{2\Delta x} = 0 \quad (55)$$

$$\frac{\Phi_j^{n+1} - \Phi_j^{n-1}}{2\Delta t} + \frac{U_{j+1}^n - U_{j-1}^n}{2\Delta x} = 0 \quad (56)$$

Dando

$$U_j^{n+1} = U_j^{n-1} - \frac{\Delta t}{\Delta x} (\Phi_{j+1}^n - \Phi_{j-1}^n) \quad (57)$$

$$\Phi_j^{n+1} = \Phi_j^{n-1} - \frac{\Delta t}{\Delta x} (U_{j+1}^n - U_{j-1}^n) \quad (58)$$

Como de costume, podemos analisar as propriedades do presente regime, a Von Neumann. Olhar para as soluções de forma

$$U_j^n = A_n e^{ikj\delta x} \quad \Phi_j^n = B A_n e^{ikj\delta x}$$

Onde ambas A e B podem ser constantes Complexas.

# Critério de Estabilidade para a CTC esquema

- Resultando em

$$A^2 = 1 - \frac{\Delta t}{\Delta x B A} (e^{ik\delta x} - e^{-ik\delta x}) \quad (59)$$

$$B A^2 = \frac{B - \Phi \Delta t}{\Delta x A} (e^{ik\delta x} - e^{-ik\delta x}) \quad (60)$$

- Eliminar  $B$  e reorganizar, e deixe  $C^2 = \Phi(\Delta t / \Delta x)^2$  ( $C$  é o Courant número Para este problema)

$$A_2)_2 + U M_2 (4 (-\operatorname{csin} k \Delta x)_2 - 2 + 1 = 0 \quad (61)$$

- Há quatro soluções para  $A$ . Dois para o modos físico Correspondente ao peso da propagação à esquerda e à direita das ondas. O Outros dois dar modos computacional.

$$A_2 = 1 - 2 (-\operatorname{csin} k \Delta x)_2 \pm [ -4 (-\operatorname{csin} k \Delta x)_2 (+ / 4 (-\operatorname{csin} k \Delta x)_4]^{1/2} \quad (62)$$

- Se  $|C| \leq 1$  Em seguida, a raiz quadrada é puramente imaginário e  $|U M A_2| = 1$ ; O regime é estável. Se  $|C| > 1$  Em seguida, para algum valor de  $K$  A raiz quadrada é real,  $|U M A_2| >$  Para, pelo menos, uma das raízes, E o sistema é instável.

# Gravidade onda: Exercício

Resolver a gravidade vaga as equações 55 e 56º numericamente no Domain  $0 \leq X \leq 1000M$ . Deixe  $\Delta x = 1 M$ , periódica e de assumir Condições-limite. Suponha que a altura média dos Sistema é tal que  $\Phi = gH = 1 M_2/S^2$ . Deixe o estado inicial Para  $\Phi$  Ser um triângulo

$$\Phi(x,0) = \begin{cases} 0 & \text{Para } X < 400 \\ 0.001 (X - 400) & \text{Para } 400 \leq X \leq 500 \\ 0,2 - 0,001 (x - 400) & \text{Para } 500 \leq X \leq 600 \\ 0 & \text{Para } X > 600 \end{cases}$$

A condição inicial para o velocity é  $U(x, 0) = 0$ . Escolha o Intervalo de tempo tal que o sistema é estável. Integrar para o ano 2000 E mostrar soluções ( $\Phi$  E  $U$  Para  $T = 0S$   $T = 200S$ ,  $T = 400S$   $T = 600S$   $T = 800S$   $T = 1000S$   $T = 1200S$   $T = 1400S$   $T = 1600S$   $T = 1800S$   $T = 2000S$ .

**Muito Obrigado  
pela Atenção**