



Dinâmica 23/09/2021 a 23/09/2021

Modelo Numérico da Atmosfera

MET-576-4

Modelagem Numérica da Atmosfera

Dr. Paulo Yoshio Kubota

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

3 Meses
24 Aulas (2 horas cada)



Dinâmica 23/09/2021 a 23/09/2021

Modelo Numérico da Atmosfera

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

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 ϕ 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 $\phi = A^n e^{ikj\Delta x}$ para a análise de **estabilidade de Von Neumann**, que veremos em seguida

$$\begin{aligned} \phi_j^{n+1} &= \phi_j^{n-1} - u \frac{\Delta t}{\Delta x} (\phi_{j+1}^n - \phi_{j-1}^n) \\ A^{n+1} e^{ikj\Delta x} &= A^{n-1} e^{ikj\Delta x} - u \frac{\Delta t}{\Delta x} (A^n e^{ik(j+1)\Delta x} - A^n e^{ik(j-1)\Delta x}) \end{aligned}$$

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 Δx , tal que $(C\sin(k\Delta x))^2 > 1$

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

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

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

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

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

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

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

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

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

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

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

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

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

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 **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 **C = 1**. Em seguida, é conveniente escrever **A** na forma

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

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

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

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

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

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

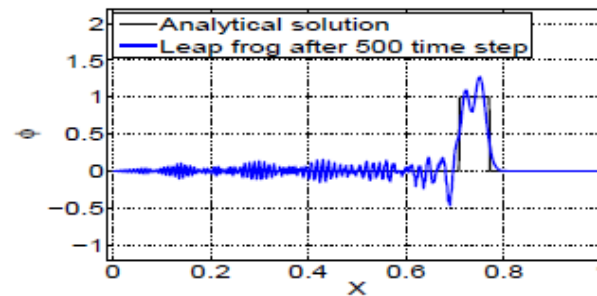
O modo Físico é proporcional à solução exata $e^{ik(x-ut)}$

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 $-Un\Delta t$).



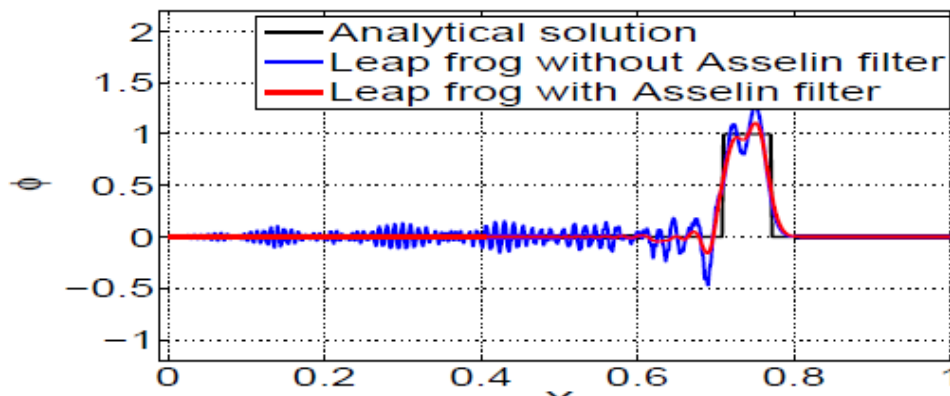
Modo computacional e o filtro RA

- A solução ϕ_j^{n+1} , Depende, ϕ_j^{n-1} Mas não no ϕ_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 Φ^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 ϕ^{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, α Devem ser mantidos pequenos (na parcela acima $\alpha = 0,05$)

filtro RAW - Williams (2009), MWR

Método RAW, que devemos aplicar a filtragem não somente em Φ^N mas também em Φ^{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

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

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

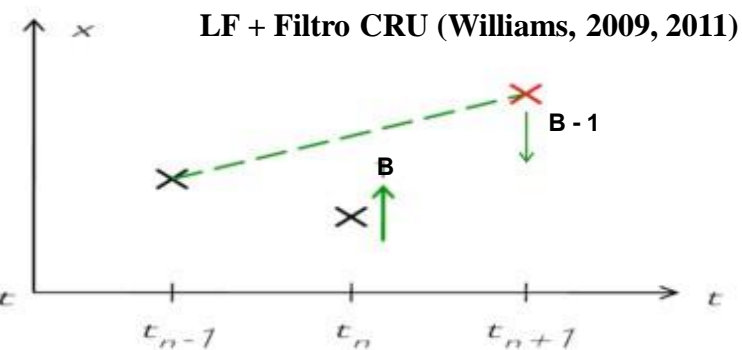
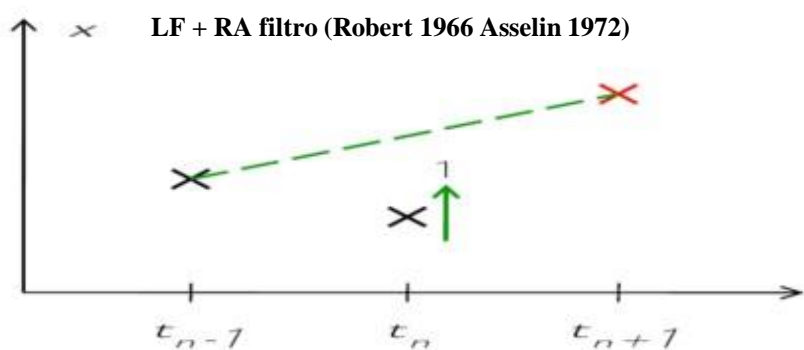
Depois da implementação do filtro e depois atualizar

$$\begin{array}{ccc} \hat{\phi}^{n-1} & \Rightarrow & \hat{\phi}^n \\ \hat{\phi}^n & \Rightarrow & \hat{\phi}^{n+1} \end{array}$$

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

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

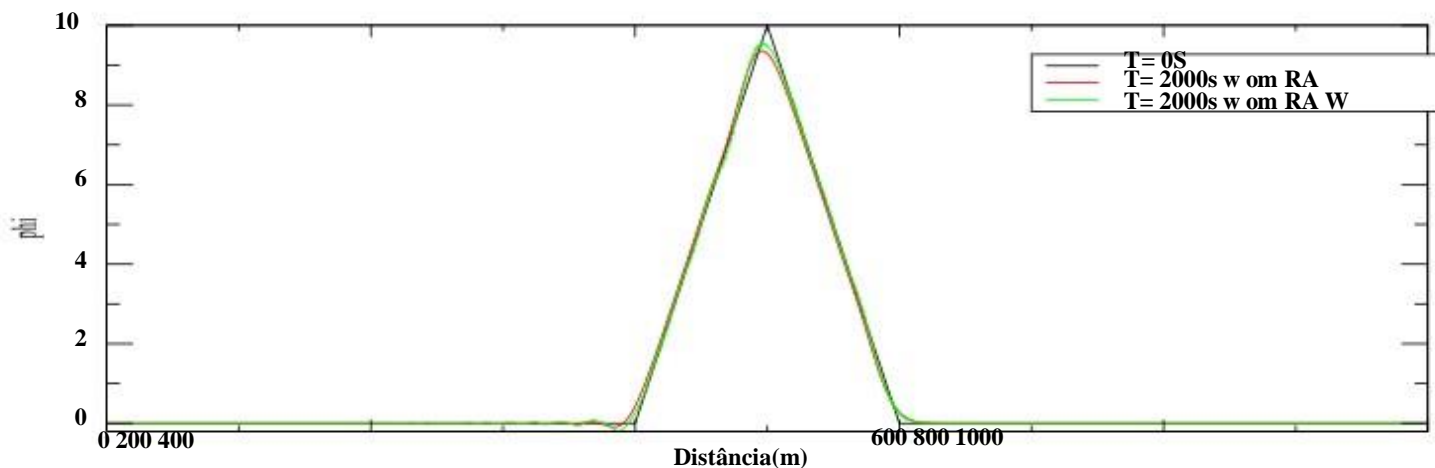
O filtro RA vs RAW



- RA filtro aplica-se em
- Reduz curvatura mas não conserva a curvatura média
- Precisão da amplitude é 1.º ordem

- Filtro-RWA aplica-se em x^n e x^{n+1}
- Reduz e ao mesmo tempo, conserva a curvatura média (para $B = 1/2$)
- Precisão da amplitude é $\approx 3.ª$ ordem

Salto com Veto RA_and_RAW_filter social e $dt=0,5$, $\alpha=0,05$, $\beta=0,6$



Exercício

- 🔴 Resolver a equação advecção 1D numericamente no domínio $0 \leq X \leq 1000M$. Deixe $\Delta x = 1 M$, e assumas as condições limite periódicas. Suponha que a velocidade de advecção $U = 1 M/s$. Deixe o estado inicial ser um triângulo

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

Escolha o intervalo de tempo, que o sistema seja estável. Integre 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($\text{Beta} = 0,60$ e $\text{Beta} = 1,0$))

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 :: DescI(:)
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 (Desc(-1:iMax+2))
END SUBROUTINE Init_Class_Fields

```

[illegible]

```

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
  !
  ! 
$$\frac{F(j,n+1) - F(j,n)}{dt} + u \frac{F(j+1,n) - F(j,n)}{dx} = 0$$

  !
  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

```

```

!-----
-----
FUNCTION Filter_RA() 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)
  END DO
  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
  ok=0
END FUNCTION SchemeUpdate
!-----
-----
END MODULE Class_NumericalMethod

```


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

!

$$\frac{F(j,n+1) - F(j,n)}{dt} + u \frac{F(j+1,n) - F(j,n)}{dx} = 0$$

!

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

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

!-----

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

PROGRAM Main

USE Class_Fields, **Only** :Init_Class_Fields

USE Class_NumericalMethod, **Only**:

InitNumericalScheme, &

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