NASA Contractor Report 4718

A User's Guide to AMR1D: An Instructional Adaptive Mesh Refinement Code for Unstructured Grids

Rosalinda de Fainchtein

GRANT NAG5–2652
MARCH 1996

NASA Contractor Report 4718

A User's Guide to AMR1D: An Instructional Adaptive Mesh Refinement Code for Unstructured Grids

Rosalinda de Fainchtein
George Mason University
Fairfax, VA 22030-4444

Prepared for
Goddard Space Flight Center
under Grant NAG5–2652

National Aeronautics
and Space Administration

**Scientific and Technical
Information Branch**

1996

This publication is available from the NASA Center for AeroSpace Information,
800 Elkridge Landing Road, Linthicum Heights, MD 21090-2934, (301) 621-0390.

Abstract

This report documents the code AMR1D, which is currently posted on the World Wide Web [http://sdcd.gsfc.nasa.gov/ESS/exchange/contrib/de-fainchtein/adaptive_mesh_refinement.html]. AMR1D is a one dimensional finite element fluid-dynamics solver, capable of adaptive mesh refinement (AMR). It was written as an instructional tool for AMR on unstructured mesh codes. It is meant to illustrate the minimum requirements for AMR on more than one dimension. For that purpose, it uses the same type of data structure that would be necessary on a two dimensional AMR code (loosely following the algorithm described by Löhner [1]).

1 INTRODUCTION

The basic idea of AMR is to provide high resolution to a simulation, only WHEN and WHERE it is needed. For that purpose, a mechanism to evaluate the need for refinement "on the fly" is necessary. AMR1D uses the "Error Indicator" function of Löhner [1]. This "Error Indicator" is an upper bound estimate of the L^2 norm [2] of the discretization error, which has been normalized and filtered for noise.

Unstructured codes are especially well suited for adaptive mesh refinement. They allow for the total disengagement of the mesh refinement process from the solver algorithm. When AMR executes it delivers a (modified) unstructured mesh, with all the appropriate mesh variables, as well as values for all the unknowns at each mesh point. Because the fluid solver is already equipped to handle unstructured meshes, this mesh information is sufficient for the fluid solver to advance the solution. The fluid solver requires no additional information pertaining to the refinement history of the mesh.

AMR1D is designed to return the original mesh when and where high resolution is no longer needed, through adaptive DE-REFINEMENT. Thus, when a portion of the mesh is de-refined, AMR1D simply undoes a previous mesh refinement. A "history array" that keeps track of the refinement history of the mesh is maintained for that purpose.

AMR1D allows for multiple mesh refinements. The maximum number of refinements that a given element may undergo is determined by a user-defined parameter. However, a given element may only be refined or de-refined once each time AMR is invoked.

Once the elements requiring refinement are selected, a layer of elements around them are also selected for refinement. This layer acts as a buffer to maintain accuracy at high resolution regions.

The contents of this report are organized as follows: Section 2 provides the general algorithm of AMR1D. Section 3 explains briefly the flow-solver algorithm used. Section 4 explains in some detail the AMR algorithm, and the subroutines that compose it. Section 5 is a description of the variables and parameters used in the code. Section 6 contains the Fortran code for AMR1D.

Figure 1 shows results produced by running AMR1D with initial conditions set for the Sod Shock Tube Problem [3].

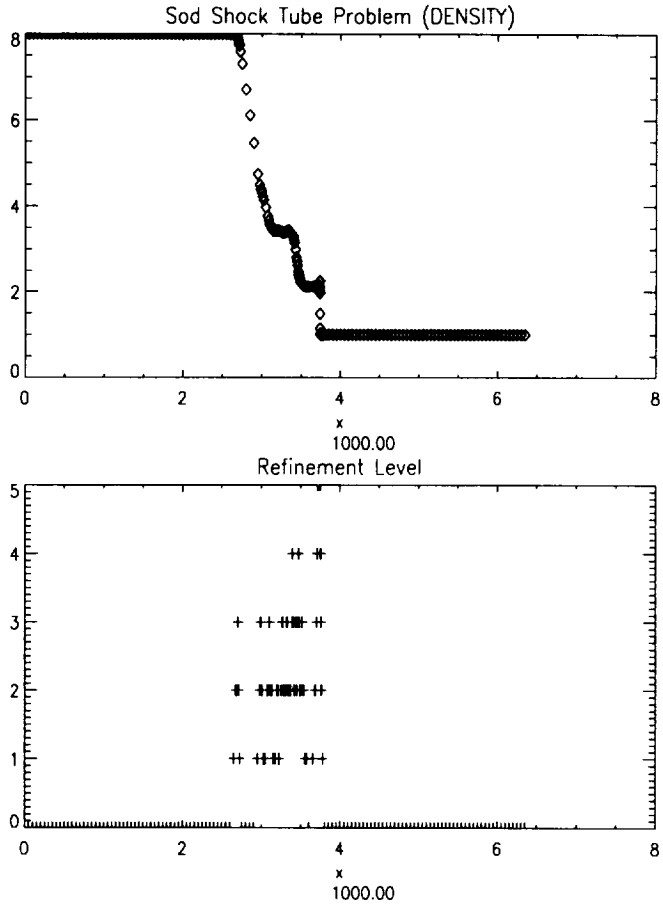


Figure 1: *Results produced by AMR1D for the Sod Shock Tube problem. The first plot shows the density. Below it, the refinement level of each element is plotted.*

2 THE GENERAL ALGORITHM

AMR1D consists of a loop over the time steps, preceded by some initialization routines.

INITIALIZATION

- **call INIT**
  Define all simulation and AMR parameters, as well as initial conditions.
- **call GEOM**
  Compute the mesh geometry and connectivity array values (e.g. element lengths).
- **call AMR ROUTINES**
  Refine a few times before beginning the simulation.

MAIN LOOP OVER TIME STEPS

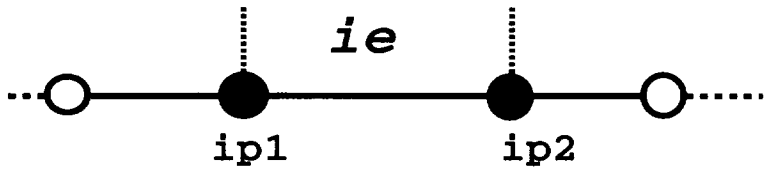
- If NSTEP = INTEGER x NREF $\Longrightarrow$ call AMR ROUTINES
- Advance the solution with the FLOW-SOLVER ROUTINES
- Apply Boundary Conditions
- If NSTEP = INTEGER x NWRITE $\Longrightarrow$ write output

3 THE FLOW-SOLVER ALGORITHM

This is a conservative algorithm. It solves a set of three equations of the form

$$\partial u/\partial t + \partial f/\partial x = 0 \tag{1}$$

by advancing in time the three conserved variables: mass density (u=rho), linear momentum (u=rhov), and specific total energy (u=rhoE). The appropriate fluxes (f) at the points are computed in subroutine FLUX1D. They are the fluxes corresponding to the Euler equations.




```
intmat(ie,1)=ip1
intmat(ie,2)=ip2
```

Figure 2: *Schematics of the grid connectivity. The two nodes of element ie are the points numbered* **ip1** *and* **ip2**, *respectively.*

The time step is also computed in FLUX1D using a Courant condition:

$$dt = COUR * min[RLEN/v_m] \tag{2}$$

where,

$$v_m = v_s + |v|, \tag{3}$$

v_s is the local speed of sound and v is the local velocity. The minimum is computed over each element, COUR is a user-defined parameter (between 0 and 1), and RLEN is the length of the element.

The increments (du) for each time step are computed in subroutine DELTAU, using a Galerkin finite element scheme. The elements are Lagrangian [4] and have two nodes (points) and piecewise linear interpolation functions. Figure 2 illustrates the grid connectivity between elements and points. The scheme is first order in time and space. First order numerical diffusion is added for stability, in a manner similar to that used in the Rusanov scheme [5, 6].

The flow-solver algorithm is modeled after UNST, a simple two dimensional unstructured fluid solver written by S.Zalesak [1]

4 THE AMR ALGORITHM

The refinement algorithm consists of three steps, performed by calling three subroutines:

[1] Private Communication

- **subroutine GRERROR**
- **subroutine UNREF**
- **subroutine REFINE**

Each element can only be refined or de-refined once, during a given pass through the refinement routines. After refinement, subroutine GEOM is called to re-compute all the geometry arrays for the "new" mesh.

SUBROUTINE GRERROR

Subroutine GRERROR determines which elements should be refined or de-refined. The Error Indicator function (ERRORE) is computed and assigned to each element (ie). An element with a value of ERRORE larger than the user defined threshold for refinement (CTORE), is marked for refinement: LREFE(ie)=1. Similarly, an element with a value of ERRORE smaller than a user defined threshold for de-refinement (CTODE), is marked for de-refinement: LDERE(ie)=1.

Next, a user-defined number of buffer layers of refined elements (NBUFF) are added around each element originally marked for refinement.

Finally, a check is performed to avoid "jumps" in the level of refinement on neighboring elements. This step is only necessary on one-dimensional grids. Connectivity rules on two and three dimensional grids automatically avoid discontinuity on the level of refinement of neighboring elements.

SUBROUTINE UNREF

Subroutine UNREF de-refines elements marked for de-refinement in GRERROR. Mesh de-refinement follows two rules:

- Elements are de-refined, as opposed to coarsened. That is, a pair of sibling elements marked for de-refinement are not arbitrarily coarsened, but are instead replaced by the original "parent" element. (Note that the initial mesh is the coarsest mesh allowed). This rule makes it necessary to maintain a refinement history element array: MRHIST.
- In order for de-refinement to take place, both siblings must be marked for de-refinement.

The elements marked for de-refinement in subroutine GRERROR are checked for consistency with these rules before being de-refined. The actual de-refinement proceeds as follows,

1. The de-refined elements are de-activated: MRHIST(ie_child,4)=0
2. The parent element is activated: MRHIST(ie_parent,4)=1
3. The "extra" point (jp) previously created to produce the sibling elements out of the parent element (c.f. SUBROUTINE REFINE) is de-activated: IPACT(jp)=0

SUBROUTINE REFINE

Element refinement consists of dividing the parent element into two children elements of equal length, sharing a newly created point.

Subroutine REFINE refines some or all of the elements marked for refinement in subroutine GRERROR. AMR1D allows for only a (user-defined) given maximum number of refinements of each original element(NMXRE). Thus, subroutine REFINE checks every element marked for refinement in GRERROR, and refines only those whose level of refinement will not exceed NMXRE.

The refinement process is fairly straightforward:

1. Add (activate) a new point at the end of the point list. (The point will be at the center of the element ie marked for refinement).
2. By linear interpolation, compute the new coordinate and unknowns: density, momentum and specific total energy.
3. Modify the connectivity array (INTMAT) to add the two new elements.
4. Disable the parent element and enable the children elements.
5. Update the refinement history element array (MRHIST).
6. Update NPOIN and NELEM.

5 DEFINITIONS

FLOW SOLVER VARIABLES

NPOIN = the number of points
NELEM = the number of elements
INTMAT = the "element list"
(INTMAT(ie,1) and INTMAT(ie,2) are the first and second point numbers that comprise element ie)
XP(ip) = the coordinate of point ip
RHO(ip) = the density at point ip
RHOV(ip) = the momentum at point ip
RHOE(ip) = the specific total (internal plus kinetic) energy at point ip
CE(ie,in) = coefficients for evaluating x first derivatives on an element
RLEN(ie) = the length of element ie
RMATM(ip) = the ip diagonal element of the inverse lumped mass matrix
(= 2./ [sum of lengths of elements sharing point ip])
FLUX(ip,1:3) = flux functions f for point ip
P(ip) = pressure at point ip.
V(ip) = velocity at point ip
PERIOD = mesh period length (when PERIOD=0, boundaries are "hard walls").

ADAPTIVE MESH REFINEMENT VARIABLES

ERRORE(ie) = "Error indicator"
MRHIST(ie,1) = parent element number. (default=0)
2) = first child element number. (default=0)
3) = second child element number. (default=0)
4) = level of refinement. (default=0)
5) = element active? (1 = yes)

LREFE(ie) =1 $\Longrightarrow$ refine
=0 $\Longrightarrow$ do not refine
LDERE(ie) =1 $\Longrightarrow$ de-refine
=0 $\Longrightarrow$ do not de-refine
IPACT(ip) =1 $\Longrightarrow$ point ip is active
=0 $\Longrightarrow$ point ip is not active (deleted)

CTORE = threshold for refinement
CTODE = threshold for de-reninement
EPSIL = noise tolerance parameter
NBUFF = No. of buffer layers

6 THE FORTRAN CODE

```
        program AMR1D
c
c    written by:   Rosalinda de Fainchtein
c
c=============================================================================
c
        parameter(mpoin=40000,melem=80000,period=0.0)
        integer intmat(melem,2)
        real xp(mpoin),ce(melem,2),rmatm(mpoin),rlen(melem)
        real rho(mpoin),rhov(mpoin),rhoE(mpoin)
        real p(mpoin),v(mpoin),flux(mpoin,3)
        real diff(melem,3)
        real vmax(mpoin)
        real du(melem,3), dup(mpoin,3)
c
        integer mrhist(melem,5),lrefe(melem),ldere(melem)
        integer ipact(mpoin)
        integer iptemp(mpoin),ietemp(melem)
        integer l(mpoin)
        real tempea(melem),tempec(melem),errore(melem)
        real temppa(mpoin),temppb(mpoin),temppc(mpoin),errorp(mpoin)
c
c  ---------Get initial conditions and parameters
c
             call init(gamma,cour,ntime,di,ctore,ctode,epsil,nrmax,
     &                 nbuff,ntref,npoin,nelem,mpoin,melem,period,
     &                 intmat, mrhist, lrefe, ldere, ipact,
     &                 xp, rho, p, v, rhov, rhoE,
     &                 ibdry_l, ibdry_r )
```

```
        gamma1=gamma-1.
c
c  ---------Calculate element lengths, inverse mass matrix, and
c           coefficients for evaluating x derivatives on the elements.
c
        call geom(mpoin, melem, npoin, nelem, intmat, xp, ce, rmatm,
     &             rlen,  period, mrhist, ipact )
c
c ------Refine a few times before starting
c
       do iloop=1,5

c
c -------Refinement Routines
c        -------------------
c
c
c  -----Compute the error function at each element and mark elements
c       that require refinement or de-refinement
c
        call grerror(nelem, npoin, melem, mpoin, rho,
     &                     tempea, tempec, temppa, temppb, temppc,
     &                     epsil, ctore, ctode, nbuff,errorp,
     &                     errore, mrhist, ipact,
     &                     lrefe, ldere, ce, rlen, intmat,
     &                     ietemp,iptemp,ibdry_l,ibdry_r )
c
c  -----Coarsen elements marked for derefinement
c
        call unref(melem, mpoin, nelem, npoin, intmat,
     &              mrhist, ldere,ipact )
c
c  -----Refine elements marked for refinement
c
        call refine(melem, mpoin, nelem, npoin, intmat, nrmax,
     &               mrhist, xp, rho, rhov, rhoE, lrefe, ipact)
c
```

```
c  -----Re-compute element lengths, inverse mass matrix and coefficients
c       for computing x derivatives within each element
c
         call geom(mpoin, melem, npoin, nelem, intmat, xp, ce, rmatm,
     &             rlen,  period, mrhist, ipact )
c
c ------End of Refinement Routines
c       -------------------------
c
         end do
c
c --------------BEGIN THE MAIN LOOP OVER TIME STEPS-----------------------
c               ==================================
c
         do 1000 i=1,ntime
c
c -------Refinement Routines
c        -------------------
c
         if ( mod(i,ntref) .eq. 0 ) then
c
c  -----Compute the error function at each element and mark elements
c       that require refinement or de-refinement
c
         call grerror(nelem, npoin, melem, mpoin, rho,
     &                     tempea, tempec, temppa, temppb, temppc,
     &                     epsil, ctore, ctode, nbuff, errorp,
     &                     errore, mrhist, ipact,
     &                     lrefe, ldere, ce, rlen, intmat,
     &                     ietemp,iptemp )
c
c  -----Coarsen elements marked for derefinement
c
         call unref(melem, mpoin, nelem, npoin, intmat,
     &              mrhist, ldere,ipact )
c
c  -----Refine elements marked for refinement
```

```
c
         call refine(melem, mpoin, nelem, npoin, intmat, nrmax,
     &               mrhist, xp, rho, rhov, rhoE, lrefe, ipact)
c
c  -----Re-compute element lengths, inverse mass matrix and coefficients
c       for computing x derivatives within each element
c
         call geom(mpoin, melem, npoin, nelem, intmat, xp, ce, rmatm,
     &            rlen,  period, mrhist, ipact )
         end if
c
c ------End of Refinement Routines
c       -------------------------
c
c
c -------Compute fluxes at the points and allowed time step.
c
         call flux1d(mpoin, melem, npoin, nelem, gamma, cour,
     1               intmat, rho, rhov, rhoE, p, v,
     2               rlen, vmax, flux, dt, mrhist, ipact )
c
c  ------Compute changes in conserved variables on elements, and
c       scatter-add them to the point array dup
c
        call deltau(mpoin, melem, npoin, nelem,intmat,du,dup,flux,dt,
     &              ce,rlen, cour,di,rho,rhov,rhoE,diff,vmax, mrhist )
c
c  -------Hard-Wall boundary conditions
c
        if (period .eq. 0. ) then
          dup(ibdry_l,2)=0.
          dup(ibdry_r,2)=0.
        end if
c
c  -------Advance the solution.
c
        do ip=1,npoin
```

```
          if ( ipact(ip) .eq. 1 ) then
           rho(ip) =  rho(ip)  - dup(ip,1)*rmatm(ip)*dt
           rhov(ip) = rhov(ip) - dup(ip,2)*rmatm(ip)*dt
           rhoE(ip) = rhoE(ip) - dup(ip,3)*rmatm(ip)*dt
           p(ip)= gamma1 * ( rhoE(ip) - 0.5*rhov(ip)*v(ip) )
          end if
         end do
c
c ------Output.
c        =======
c
         if ( mod(i, 10) .eq. 0 ) then
         write(50,*)'  it,dt = ',i,dt
         write(50,*)'           it   npoin(active)   nelem(active)'
c
c  -------How many active points (neac) ?
c
         npac=0
         do ip=1,mpoin
          if ( ipact(ip) .eq. 1 )   npac=npac+1
         end do
c
c  -------How many active elements (neac) ?
c
         neac=0
         do ie=1,melem
          if ( mrhist(ie,5) .eq. 1 ) then
           neac=neac+1
           l(intmat(ie,1))=mrhist(ie,4)
          end if
         end do
         write(50,*) i,npac ,neac, npoin, nelem
         write(50,*)'      xp            rho            v ',
     &             '           rhoE           p  l '
         do ip=1,npoin
          if ( ipact(ip) .eq. 1 )
     &   write(50,6) xp(ip),rho(ip),v(ip),rhoE(ip),p(ip),l(ip)
```

```
        end do
        end if
c
 1000   continue
c
c  ---END OF MAIN TIME STEP LOOP-----------------------------------------------
c
   5    format(8(2x,f6.3))
   6    format(5(2x,e12.5),1x,i1)
        stop
        end
c
c*******************************************************************************
c
        subroutine init(gamma,cour,ntime,di,ctore,ctode,epsil,nrmax,
     &                  nbuff,ntref,npoin, nelem, mpoin, melem,
     &                  period, intmat, mrhist, lrefe, ldere, ipact,
     &                  xp, rho, p, v, rhov, rhoE,
     &                  ibdry_l, ibdry_r )
c
         real xp(mpoin),rho(mpoin),p(mpoin),v(mpoin),rhov(mpoin)
         real rhoE(mpoin)
         integer intmat(melem,2),mrhist(melem,5),lrefe(melem)
         integer ldere(melem),ipact(mpoin)
c
c
         parameter(pi=3.141592654)
c
c  ---------Set constants, etc.
c
         gamma=1.4
         gamma1=gamma-1.
         cour =0.5                          !  Courant number
         ntime=3000                         !  Number of time steps
         di   =0.5                          !  Diffusion coefficient
         ctore=.25                          !  Threshold of error fcn. to refine
         ctode=.1                           !  Threshold  to de-refine
```

```
        epsil=0.005                          !  Refinement filter coefficient
        nbuff=3                              !  No. of buffer layers
        ntref=2                              !  No. of time steps per refinement
        nrmax=3                              !  Max. No. of Refinement Levels.
c
c  --------Set the initial conditions
c
        npoin=128
        nelem=127
c
        do ie=1,npoin-1
          intmat(ie,1)=ie
          intmat(ie,2)=ie+1
        end do
c
c ----boundary conditions.
c
        if (period .ne. 0.) then
          intmat(nelem,1)=nelem
          intmat(nelem,2)=1
        else
          ibdry_l=1
          ibdry_r=npoin
        end if
c
        do ie=1,melem
         do ir=1,5
          mrhist(ie,ir)=0
         end do
         lrefe(ie)=0
         ldere(ie)=0
        end do
c
        do ie=1,nelem
         mrhist(ie,5)=1
        end do
c
```

```
      do ip=1,npoin
       ipact(ip)=1
      end do
c
      do ip=npoin+1,mpoin
       ipact(ip)=0
      end do
c
c  -------Initial Configuration for Sod's Shock Tube Problem
c
      do ip=1,npoin/2
        rho(ip)= 8.
          p(ip)=10.
      end do
c
      do ip=npoin/2,npoin
        rho(ip)= 1.
          p(ip)= 1.
      end do
c
      do ip=1,npoin
        xp(ip) =float(ip-1)*6.4/float(npoin)
        v(ip)  =0.0
        rhov(ip)=rho(ip)*v(ip)
        rhoE(ip)=p(ip)/gamma1 + 0.5*rho(ip)*v(ip)*v(ip)
      end do
c
c  --------Write out the initial conditions
c
      it=0
      write(50,*)'initial conditions'
      write(50,*)'           it      npoin        nelem'
      write(50,*) it,npoin,nelem
      write(50,*)'      xp            rho             v ',
     &            '           rhoE            p '
      do i=1,npoin
       write(50,6) xp(i),rho(i),v(i),rhoE(i),p(i)
```

```
        end do
c
   6    format(5(2x,e12.5))
c
       return
c
       end
c
c*********************************************************************
c
       subroutine geom(mpoin, melem, npoin, nelem, intmat, xp,
     &                 ce, rmatm,  rlen, period, mrhist, ipact )
c
c ------COMPUTE THE GEOMETRY ARRAYS FOR A 1D FE FLOW SIMULATION.
c
       integer mpoin,melem,npoin,nelem,intmat(melem,2)
       real xp(mpoin),ce(melem,2),rmatm(mpoin),rlen(melem)
       integer mrhist(melem,5),ipact(mpoin)
c
c -------Initialize the mass matrix
c
       do ip=1,npoin
         rmatm(ip)=0.
       end do
c
c --------Initialize element arrays
c
          do ie=1,nelem
           rlen(ie)=0.
          end do
c
c --------Compute the length of the elements (rlen)
c
       do ie=1,nelem
         if ( mrhist(ie,5) .eq. 1 )
     &   rlen(ie)= xp(intmat(ie,2)) - xp(intmat(ie,1))
       end do
```

```
c
c ------Periodic b.c.?
c
       if ( period .ne. 0. ) then
         do ie=1,nelem
           if ( abs(rlen(ie)) .gt. period/2. )
     &          rlen(ie)=period + rlen(ie)
         end do
       end if
c
c --------The basis functions' gradient
c
       do ie=1,nelem
         if ( mrhist(ie,5) .eq. 1 ) then
          ce(ie,2)=1./rlen(ie)
          ce(ie,1)=-1.*ce(ie,2)
         end if
       end do
c
c ---------The inverse lumped mass matrix.
c
       do ie=1,nelem
        if ( mrhist(ie,5) .eq. 1 ) then
        do in=1,2
          rmatm(intmat(ie,in)) = rmatm(intmat(ie,in)) + rlen(ie)
        end do
        end if
       end do
c
c ---------Invert the sum
c
       do ip=1,npoin
        if ( ipact(ip).eq.1 )
     &   rmatm(ip) = 2./rmatm(ip)
       end do
c
       return
```

```
      end
c
c**********************************************************************
c
        subroutine grerror(nelem, npoin, melem, mpoin, vref,
     &                     tempea, tempec, temppa, temppb, temppc,
     &                     epsil, ctore, ctode, nbuff, errorp,
     &                     errore, mrhist, ipact,
     &                     lrefe, ldere, ce, rlen, intmat,
     &                     ietemp,iptemp,ibdry_l,ibdry_r )
c
c ---------THIS SUBROUTINE COMPUTES THE H-2 SEMINORM ERROR ESTIMATOR
c             [ errorp = temppa/ ( temppb + epsil*temppc) ]
c          (Ref: Lohner)
c
        integer intmat(melem,2),mrhist(melem,5),ipact(mpoin)
        integer lrefe(melem),ldere(melem)
        integer iptemp(mpoin),ietemp(melem)
        real ce(melem,2),rlen(melem)
        real tempea(melem),tempec(melem),errore(melem)
        real temppa(mpoin),temppb(mpoin),temppc(mpoin),errorp(mpoin)
        real vref(mpoin)
c
c       nbuff=3
c
c --------Initialize element arrays
c
        do ie=1,melem
          tempea(ie)=0.
          tempec(ie)=0.
          errore(ie)=0.
          lrefe(ie) =0.
          ldere(ie) =0.
        end do
c
c --------Initialize point arrays
c
```

```
          do ip=1,mpoin
            temppa(ip)=0.
            temppb(ip)=0.
            temppc(ip)=0.
            errorp(ip)=0.
          end do
c
c --------Sums at the elements.
c
          do ie=1,nelem
           if (mrhist(ie,5) .eq. 1) then
           do in=1,2
             tempea(ie) = tempea(ie) + ce(ie,in)*vref(intmat(ie,in))
             tempec(ie) = tempec(ie) + abs( ce(ie,in) ) *
     &                                 abs( vref(intmat(ie,in)) )
           end do
           end if
          end do
c
c -------Scatter to the points.
c
          do ie=1,nelem
           if (mrhist(ie,5) .eq. 1) then
           do in=1,2
             temppa(intmat(ie,in)) = temppa(intmat(ie,in)) +
     &                               rlen(ie)*ce(ie,in)*tempea(ie)
             temppb(intmat(ie,in)) = temppb(intmat(ie,in)) +
     &                               rlen(ie) * abs(ce(ie,in)) *
     &                               abs(tempea(ie))
             temppc(intmat(ie,in)) = temppc(intmat(ie,in)) +
     &                               rlen(ie) * abs(ce(ie,in)) *
     &                               tempec(ie)
           end do
           end if
          end do
c
c -------Boundary terms
```

```
c
          temppa(ibdry_l) = 0.
          temppa(ibdry_r) = 0.
c
c --------The error estimate at the points
c
         do ip=1,npoin
           if ( ipact(ip) .eq. 1 )
     &     errorp(ip) = abs(temppa(ip)) /
     &                  ( temppb(ip) + epsil*temppc(ip) )
         end do
c
c --------Compute the error estimate at the elements (max. of node values).
c
         do ie=1,nelem
          if (mrhist(ie,5) .eq. 1)
     &      errore(ie)=amax1( errorp(intmat(ie,1)) ,
     &                        errorp(intmat(ie,2)) )
         end do
c
c --------Mark the elements for refinement or de-refinement.
c
         do ie=1,nelem
          if (mrhist(ie,5).eq.1) then
           if (errore(ie).gt.ctore) lrefe(ie)=1
           if (errore(ie).lt.ctode) ldere(ie)=1
          end if
         end do
c
c -------Buffer Layers
c
        do ib=1,nbuff
         do ip=1,npoin
          iptemp(ip)=0
         end do
c
         do ie=1,nelem
```

```
          if (mrhist(ie,5).eq.1) then
           do in=1,2
            if ( lrefe(ie) .eq. 1)
     &         iptemp(intmat(ie,in)) = 1
           end do
          end if
         end do
c
         do ie=1,nelem
          if (mrhist(ie,5).eq.1) then
           do in=1,2
            if ( iptemp(intmat(ie,in)) .eq. 1)
     &         lrefe(ie) = 1
           end do
          end if
         end do
        end do
c
c -------Space Continuity on levels of refinement (only necessary on 1D)
c
        do iloop=1,5
         do ie=1,nelem
          ietemp(ie) = 0
          if (mrhist(ie,5).eq.1)
     &    ietemp(ie) = mrhist(ie,4) + lrefe(ie)
         end do
c
         do ip=1,npoin
          iptemp(ip)=0
         end do
c
         do ie=1,nelem
          if (mrhist(ie,5).eq.1) then
           do in=1,2
            if ( ietemp(ie) .gt. iptemp(intmat(ie,in)) )
     &         iptemp(intmat(ie,in)) = ietemp(ie)
           end do
```

```
          end if
         end do
c
         do ie=1,nelem
          if (mrhist(ie,5).eq.1) then
           do in=1,2
            if ( iptemp(intmat(ie,in))-mrhist(ie,4) .gt. 1)
     &         lrefe(ie) = 1
            if ( iptemp(intmat(ie,in))-mrhist(ie,4) .ge. 1)
     &         ldere(ie) = 0
           end do
c
           ietemp(ie) = mrhist(ie,4) + lrefe(ie)
           do in=1,2
            if ( ietemp(ie) .gt. iptemp(intmat(ie,in)) )
     &         iptemp(intmat(ie,in)) = ietemp(ie)
           end do
c
          end if
         end do
        end do
c
         return
         end
c
c*********************************************************************
c
        subroutine unref(melem, mpoin, nelem, npoin, intmat,
     &                   mrhist, ldere,ipact )
c
c  --------THIS ROUTINE DE-REFINES THOSE ELEMENTS MARKED FOR DE-REFINEMENT
c
         integer intmat(melem,2)
         integer mrhist(melem,5),ldere(melem)
         integer ipact(mpoin)
c
c
```

```
c  ------AMR arrays
c        ==========
c
c           mrhist(ie,1) == parent element number.  (default=0)
c                     2) == first child element number.  (default=0)
c                     3) == second child element number.  (default=0)
c                     4) == level of refinement.   (default=0)
c                     5) == element active?        ( 1=yes )
c
c           ldere(ie) =1  ======> de-refine
c                     =0  ======> do not de-refine
c
c  -----Loop over the elements
c
         do ie=1,nelem
          if ( ldere(ie).eq.1 .and. mrhist(ie,4).ne.0 ) then
c
c ----------Identify elements involved
c
          iparent=mrhist(ie,1)
          ichild1=mrhist(iparent,2)
          ichild2=mrhist(iparent,3)
c
c  ---------Derefine iff both children are marked for de-refinement (4 steps)
c
          if ( ldere(ichild1).eq.1 .and. ldere(ichild2).eq.1 )
     &                                                    then
c
c ...........(1) Identify and ``deactivate" points not assoc. w/parent.
c
           do in=1,2
            ip1=intmat(ichild1,in)
            ip2=intmat(ichild2,in)
            if (ip1.ne.intmat(iparent,1) .and.
     &          ip1.ne.intmat(iparent,2) )   ipact(ip1)=0
            if (ip2.ne.intmat(iparent,1) .and.
     &          ip2.ne.intmat(iparent,2) )   ipact(ip2)=0
```

```
           end do
c
c ..........(2) Disable children elements.............................
c
           mrhist(ichild1,5)=0
           mrhist(ichild2,5)=0
c
c ..........(3) Enable parent element.................................
c
           mrhist(iparent,5)=1
c
c ..........(4) Unmark the children elements..........................
c
           ldere(ichild1)=0
           ldere(ichild2)=0
c
          end if                ! (...if all children marked)
c
         end if                 ! (...if one child marked)
c
        end do                  ! (end loop over elements)
c
        return
        end
c
c*************************************************************************
c
        subroutine refine(melem, mpoin, nelem, npoin, intmat, nrmax,
     &                    mrhist, xp, rho, rhov, rhoE, lrefe, ipact )
c
c  ------THIS ROUTINE REFINES THOSE ELEMENTS MARKED FOR REFINEMENT
c
         integer intmat(melem,2)
         real xp(mpoin)
         real rho(mpoin),rhov(mpoin),rhoE(mpoin)
c
         integer mrhist(melem,5),lrefe(melem)
```

```
      integer ipact(mpoin)
c
c  ------AMR arrays
c      ==========
c
c         mrhist(ie,1) == parent element number.  (default=0)
c                     2) == first child element number.  (default=0)
c                     3) == second child element number.  (default=0)
c                     4) == level of refinement.   (default=0)
c                     5) == element active?        ( 1=yes )
c
c         lrefe(ie) =1  ======> refine
c                   =0  ======> do not refine
c
c  -----Loop over the elements
c
      nrmax1=nrmax-1
      do ie=1,nelem
       if ( lrefe(ie).eq.1 .and. mrhist(ie,4).le.nrmax1 ) then
        npoin=npoin+1                        !new point
        ipact(npoin)=1
        if (npoin .gt. mpoin) then
         write(*,*)'Please increase mpoin.  Needed:',npoin
         stop
        end if
c
        nelem2=nelem+2
        if (nelem2 .gt. melem) then
         write(*,*)'Please increase melem.  Needed:',nelem2
         stop
        end if
c
        xp(npoin)   = 0.5*( xp(intmat(ie,1)) + xp(intmat(ie,2)) )
        rho(npoin)  = 0.5*(  rho(intmat(ie,1))  +
     &                        rho(intmat(ie,2)) )
        rhov(npoin) = 0.5*( rhov(intmat(ie,1))  +
     &                       rhov(intmat(ie,2)) )
```

```
           rhoE(npoin) = 0.5*( rhoE(intmat(ie,1))  +
     &                         rhoE(intmat(ie,2)) )
c
c ------New Element Connectivity (Input new elements at the end of the list)
c
           intmat(nelem+1,1) = intmat(ie,1)
           intmat(nelem+1,2) = npoin
           intmat(nelem+2,1) = npoin
           intmat(nelem+2,2) = intmat(ie,2)
c
c ------Disable the parent elements and enable the children
c
           mrhist(ie,5)=0
           mrhist(nelem+1,5) = 1
           mrhist(nelem+2,5) = 1
c
c ------Modify the rest of the mesh refinement history parameters
c
           mrhist(ie,2) = nelem+1
           mrhist(ie,3) = nelem+2
           mrhist(nelem+1,1) = ie
           mrhist(nelem+2,1) = ie
           mrhist(nelem+1,4) = mrhist(ie,4) + 1
           mrhist(nelem+2,4) = mrhist(ie,4) + 1
c
c ------Modify the current number of elements
c
            nelem=nelem+2
          end if
         end do
c
         return
         end
c
c**********************************************************************
c
       subroutine flux1d(mpoin, melem, npoin, nelem, gamma, cour,
```

```
     &                    intmat, rho, rhov, rhoE, p, v,
     &                    rlen, vmax, flux, dt, mrhist, ipact )
c
c --------Compute the fluxes at the points and the time step.
c         ==================================================
c
       integer mpoin,melem,npoin,nelem,intmat(melem,2)
       real rho(mpoin),rhov(mpoin),rhoE(mpoin)
       real p(mpoin),v(mpoin),flux(mpoin,3)
       real rlen(melem),vmax(mpoin)
       integer mrhist(melem,5),ipact(mpoin)
c
       cbig=1.e30
       gamma1=gamma-1.
       do ip=1,npoin
        if ( ipact(ip).eq.1 ) then
         v(ip)=rhov(ip)/rho(ip)
         p(ip)= gamma1 * ( rhoE(ip) - 0.5*rhov(ip)*v(ip) )
c
         flux(ip,1) = rhov(ip)
         flux(ip,2) = rhov(ip)*v(ip) + p(ip)
         flux(ip,3) = rhoE(ip)*v(ip) + p(ip)*v(ip)
        end if
       end do
c
c --------TIME STEP: dt.
c          ==============
c
c  -------Compute the ``maximum velocity" (sound speed + v) at the points.
c
       do ip=1,npoin
        if ( ipact(ip).eq.1 )
     &   vmax(ip) = sqrt(gamma*p(ip)/rho(ip)) + abs(v(ip))
       end do
c
c  -------and the time step...
c
```

```
        dt= cbig
        do ie=1,nelem
         if ( mrhist(ie,5) .eq. 1 ) then
         do in=1,2
           dt = amin1( dt, rlen(ie)/vmax(intmat(ie,in)) )
         end do
         end if
        end do
c
        dt = cour * dt
c
        return
        end
c
c*************************************************************************
c
            subroutine deltau(mpoin, melem, npoin,nelem,intmat,
     &             du,dup,flux,dt,ce,rlen,cour,di,rho,rhov,rhoE,diff,
     &             vmax,mrhist)
c
c ---------Compute the du's and dup's
c
            integer mpoin,melem,npoin,nelem, intmat(melem,2)
            real du(melem,3), dup(mpoin,3),flux(mpoin,3),dt
            real ce(melem,2),rlen(melem)
            real rho(mpoin),rhov(mpoin),rhoE(mpoin),diff(melem,3)
            real vmax(mpoin)
            integer mrhist(melem,5)
c
        cbig=1.e20
        csmall=-1.e20
c
c  --------Initialize du
c
        do im=1,3
         do ie=1,nelem
           du(ie,im)   = 0
```

```
          diff(ie,im) = 0.
         end do
        end do
c
c ----------The minimum delta x.
c
        rlmin=cbig
        rlmax=csmall
         do ie=1,nelem
          if (mrhist(ie,5).eq.1 .and. rlen(ie).lt.rlmin)
     &       rlmin=rlen(ie)
          if (mrhist(ie,5).eq.1 .and. rlen(ie).gt.rlmax)
     &       rlmax=rlen(ie)
         end do
c
c  ---------Initialize dup
c
        do im=1,3
         do ip=1,npoin
          dup(ip,im)=0.
         end do
        end do
c
c  ---------BEGIN LOOP OVER THE ELEMENTS---------------------------------
c
        do 1000 ie=1,nelem
         if ( mrhist(ie,5) .eq. 1 ) then
c
c  ---------Compute du at the elements
c
          do im=1,3
           do in=1,2
            du(ie,im) = du(ie,im) + ce(ie,in)*flux(intmat(ie,in),im)
           end do
          end do
c
c  --------Compute the diffusion terms at the elements
```

```
c
        vv=amax1(vmax(intmat(ie,1)),vmax(intmat(ie,2)))
        do in=1,2
         diff(ie,1) = diff(ie,1) + ce(ie,in)*rho(intmat(ie,in))
     &                           *vv
         diff(ie,2) = diff(ie,2) + ce(ie,in)*rhov(intmat(ie,in))
     &                           *vv
         diff(ie,3) = diff(ie,3) + ce(ie,in)*rhoE(intmat(ie,in))
     &                           *vv
        end do
c
        dfact = 0.5*di
c
        do im=1,3
         diff(ie,im) = dfact * diff(ie,im)*rlen(ie)*rlmin
        end do
c
c  --------Scatter to the points.
c
        do im=1,3
         do in=1,2
           dup(intmat(ie,in),im) = dup(intmat(ie,in),im)  +
     &                           0.5*rlen(ie)*du(ie,im) +
     &                           ce(ie,in)*diff(ie,im)
         end do
        end do
c
        end if
c
 1000  continue
c
c -----END OF THE ELEMENT LOOP-------------------------------
c
       return
       end
```

7 ACKNOWLEDGEMENTS

The author would like to acknowledge the encouragement and useful suggestions provided by Steven T. Zalesak, during the development of this code. Thanks are also due to Peter MacNeice, Steven T. Zalesak, and Daniel S. Spicer, who took the time to review this manuscript and suggest improvements.

8 THE FORTRAN CODE

References

[1] Löhner, R., 1987, "An Adaptive Finite Element Scheme for Transient Problems in CFD", *Comp. Meth. Appl. Mech. Eng.*, **61**, pp. 323-338.

[2] Becker, Eric B., Carey, Graham F., and Oden, J. Tinsley, 1981, *Finite Elements. An Introduction*, Vol. I, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

[3] Sod, Gary A., 1978, "A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws", *J. Comp. Phys.*, **27**, pp. 1-31.

[4] Hirsch, C., 1988, *Numerical Computation of Internal and External Flows*, Vol. I, J.Wiley and Sons, Inc., New York.

[5] Roache, Patrick J., 1982, *Computational Fluid Dynamics*, Hermosa Publishers, Albuquerque, NM.

[6] Rusanov V.V., 1961, *The Calculation of the Interaction of Non-Stationary Shock Waves and Obstacles*, Zh. vych. mat. 1: No. 2, pp.267-279, USSR. [English Translation in: Rusanov V.V., 1962, J. Comp. Math. Math. Phys., No.2, USSR].

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1996	Contractor Report

4. TITLE AND SUBTITLE

A User's Guide to AMR1D: An Instructional Adaptive Mesh Refinement Code for Unstructured Grids

5. FUNDING NUMBERS

Code 934
NAG5–2652

6. AUTHOR(S)

Rosalinda de Fainchtein

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

George Mason University
Fairfax, VA 22030-4444

8. PERFORMING ORGANIZATION REPORT NUMBER

96B00053

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Goddard Space Flight Center
National Aeronautics and Space Administration
Washington , DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

CR-4718

11. SUPPLEMENTARY NOTES

Rosalinda de Fainchtein: George Mason University, Fairfax, Virginia

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified-Unlimited
Subject Catagory: 75
This report is available from the NASA Center for AeroSpace Information, 800 Elkridge Landing Road, Linthicum Heights, MD 21090; (301) 621-0390

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report documents the code AMR1D, which is currently posted on the World Wide Web [http://sdcd.gsfc.nasa.gov/ESS/exchange/contrib/de-fainchtein/adaptive_mesh_refinement.html]. AMR1D is a one-dimensional finite element fluid-dynamics solver, capable of adaptive mesh refinement (AMR). It was written as an instructional tool for AMR on unstructured mesh codes. It is meant to illustrate the minimum requirements for AMR on more than one dimension. For that purpose, it uses the same type of data structure that would be necessary on a two-dimensional AMR code (loosely following the algorithm described by Löhner [1].

14. SUBJECT TERMS

Computational Techniques, Adaptive-Mesh-Refinement, Fluid Dynamics, Unstructured-Mesh

15. NUMBER OF PAGES

33

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

National Aeronautics and
Space Administration
Code JTT
Washington, D.C.
20546-0001