
Rosalinda de Fainchtein

GRANT NAG5–2652
MARCH 1996

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

Prepared for
Goddard Space Flight Center
under Grant NAG5–2652

1996
Abstract

This report documents the code AMR1D, which is currently posted on the World Wide Web [http://sdc.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 call AMR ROUTINES
- Advance the solution with the FLOW-SOLVER ROUTINES
- Apply Boundary Conditions
- If NSTEP = INTEGER x NWRITE write output

3 THE FLOW-SOLVER ALGORITHM

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

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

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

\[
dt = COUR \times \min[RLEN/v_m]
\]

where,

\[
v_m = v_s + |v|,
\]

\( v_s \) 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,I: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 ==> refine
            =0 ==> do not refine
LDERE(ie) =1 ==> de-refine
            =0 ==> do not de-refine
IPACT(ip) =1 ==> point ip is active
            =0 ==> 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 AMRID

written by: Rosalinda de Fainchtein

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)

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)

-----------Get initial conditions and parameters

call init(gamma,cour,ntime,di,ctore,ctode,epsil,nrmax,
& nbuff,ntref,npoin,melem,mpoint,melem,period,
& intmat, mrhist, lrefe, ldere, ipact,
& xp, rho, p, v, rhov, rhoE,
& ibdry_l, ibdry_r )
gamma=gamma-1.

c --------Calculate element lengths, inverse mass matrix, and
coefficients for evaluating x derivatives on the elements.
c
    call geom(mpoin, melem, npoin, nelem, intmat, xp, ce, rmatm,
        &    rlen, period, mrhist, ipact )

c -------Refine a few times before starting

do iloop=1,5

c --------Refinement Routines

c
-----Compute the error function at each element and mark elements
that require refinement or de-refinement

c
    call grerror(nelem, npoin, melem, mpoin, rho,
        &    tempea, tempec, temppa, temppb, temppc,
        &    epsil, ctore, cnode, nbuff, errorp,
        &    erreor, mrhist, ipact,
        &    lrefe, ldere, ce, rlen, intmat,
        &    ietemp, iptemp, ibdry_l, ibdry_r )

c
-----Coarsen elements marked for derefinement

c
    call unref(melem, mpoin, nelem, npoin, intmat,
        &    mrhist, ldere, ipact )

c
-----Refine elements marked for refinement

c
    call refine(melem, mpoin, nelem, npoin, intmat, nrmax,
        &    mrhist, xp, rho, rhov, rhoE, lrefe, ipact)
Re-compute element lengths, inverse mass matrix and coefficients for computing x derivatives within each element

call geom(mpoin, melem, npoin, nelem, intmat, xp, ce, rmatm,
   & rlen, period, mrhist, ipact )

-----End of Refinement Routines

end do

--------------------BEGIN THE MAIN LOOP OVER TIME STEPS------------------------

do 1000 i=1,ntime

------Refinement Routines

if ( mod(i,ntref) .eq. 0 ) then

------Compute the error function at each element and mark elements that require refinement or de-refinement

call grerror(nelem, npoin, melem, mpoin, rho,
   & tempea, tempec, temppa, temppb, temppc,
   & epsil, ctope, ctope, nbuff, errorp,
   & errore, mrhist, ipact,
   & lrefe, ldere, ce, rlen, intmat,
   & ietemp,iptemp )

------Coarsen elements marked for derefinement

call unref(melem, mpoin, nelem, npoin, intmat,
   & mrhist, ldere,ipact )

------Refine elements marked for refinement
call refine(melem, mpoin, nelem, npoin, intmat, nrmax, mrhist, xp, rho, rhov, rhoE, lrefe, ipact)

--- Re-compute element lengths, inverse mass matrix and coefficients for computing x derivatives within each element

call geom(mpoin, melem, npoin, nelem, intmat, xp, ce, rmatm, rlen, period, mrhist, ipact )
end if

----- End of Refinement Routines

Compute fluxes at the points and allowed time step.
call fluxld(mpoin, melem, npoin, nelem, gamma, cour, intmat, rho, rhov, rhoE, p, v, rlen, vmax, flux, dt, mrhist, ipact )

------ Compute changes in conserved variables on elements, and scatter-add them to the point array dup
call deltau(mpoin, melem, npoin, nelem, intmat, du, dup, flux, dt, ce, rlen, cour, di, rho, rhov, rhoE, diff, vmax, mrhist )

------ Hard-Wall boundary conditions
if (period .eq. 0. ) then
dup(ibdry_1,2)=0.
dup(ibdry_r,2)=0.
end if

------ Advance the solution.
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)= gammal * ( rhoE(ip) - 0.5*rhov(ip)*v(ip) )
end if
end do

...... Output.

if ( mod(i, 10) .eq. 0 ) then
    write(50,*)' it,dt = ',i,dt
    write(50,*)' it npoin(active) nelem(active)'
endif

How many active points (neac) ?

npac=0
do ip=1,mpoin
    if (ipact(ip) .eq. 1 ) npac=npac+1
end do

How many active elements (neac) ?

neac=0
do ie=1,melem
    if (mrhist(ie,5) .eq. 1 ) then
        neac=neac+1
        intmat(ie,1)=mrhist(ie,4)
    end if
end do
write(50,*),' i,${\nu}_{p}$,neac, npoin, nelem
write(50,*),' xp rho v , rhoE p l ',
& write(50,6) xp(ip),rho(ip),v(ip),rhoE(ip),p(ip),l(ip)
end do
end if

1000 continue

---END OF MAIN TIME STEP LOOP-------------------------------

5 format(8(2x,f6.3))
6 format(5(2x,e12.5),1x,i1)
stop
end

 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 )

 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)

parameter(pi=3.141592654)

-------Set constants, etc.

gamma=1.4
gamma1=gamma-1.
cour =0.5
ntime=3000
di =0.5
ctore=.25
ctode=.1

! Courant number
! Number of time steps
! Diffusion coefficient
! Threshold of error fcn. to refine
! 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
"Set the initial conditions"
c
npoin=128
nelem=127

do ie=1,npoin-1
  intmat(ie,1)=ie
  intmat(ie,2)=ie+1
end do

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

do ie=1,melem
  do ir=1,5
    mrhist(ie,ir)=0
  end do
  lrefe(ie)=0
  ldere(ie)=0
end do

do ie=1,nelem
  mrhist(ie,5)=1
end do
do ip=1,npoin
  ipact(ip)=1
end do

do ip=npoin+1,mpoin
  ipact(ip)=0
end do

------Initial Configuration for Sod's Shock Tube Problem

do ip=1,npoin/2
  rho(ip)= 8.
  p(ip)=10.
end do

do ip=npoin/2,npoin
  rho(ip)= 1.
  p(ip)= 1.
end do

do ip=1,npoin
  xp(ip) =float(ip-1)*6.4/float(npoin)
  v(ip) =0.0
  rhoE(ip)=p(ip)/gamma1 + 0.5*rho(ip)*v(ip)*v(ip)
end do

------Write out the initial conditions

it=0
write(50,'(A,1X,4F15.6)')'initial conditions'
write(50,'(A,1X,3A,1X,4F15.6)')'it npoin nelem'
write(50,'(A,1X,3A,1X,4F15.6)')it,npoin,nelem
write(50,'(A,1X,3A,1X,4F15.6)')xp rho v,
& rhoE p ,
do i=1,npoin
  write(50,6) xp(i),rho(i),v(i),rhoE(i),p(i)
end do

6 format(5(2x,e12.5))

return

d end

******************************************************************************

subroutine geom(mpoin, melem, npoin, nelem, intmat, xp,
& ce, rmatm, rlen, period, mrhist, ipact )

-------COMPUTE THE GEOMETRY ARRAYS FOR A 1D FE FLOW SIMULATION.

integer mpoin, melem, npoin, nelem, intmat(melem,2)
real xp(mpoin), ce(melem,2), rmatm(mpoin), rlen(melem)
integer mrhist(melem,5), ipact(mpoin)

-------Initialize the mass matrix

do ip=1,npoin
   rmatm(ip)=0.
end do

-------Initialize element arrays

do ie=1,nelem
   rlen(ie)=0.
end do

-------Compute the length of the elements (rlen)

do ie=1,nelem
   if ( mrhist(ie,5) .eq. 1 )
      & rlen(ie)= xp(intmat(ie,2)) - xp(intmat(ie,1))
end do
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
------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
------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
------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
  subroutine grerror(nelem, npoin, melem, mpoint, 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
  THIS SUBROUTINE COMPUTES THE H-2 SEMINORM ERROR ESTIMATOR
  
  [ errorp = temppa / ( temppb + epsil*temppc) ]
  (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(mpoint),temppb(mpoint),temppc(mpoint),errorp(mpoint)
  real vref(mpoint)

  nbuff=3

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
  ------Initialize point arrays

20
do ip=1,mpoin
    temppa(ip)=0.
    temppb(ip)=0.
    temppc(ip)=0.
    errorp(ip)=0.
end do

---Sums at the elements.

doa=1,nelem
    if (mrhist(a,5).eq.1) then
      do in=1,2
        tempea(a) = tempea(a) + ce(a,in)*vref(intmat(a,in))
        tempec(a) = tempec(a) + abs(ce(a,in)) *
                    abs(vref(intmat(a,in)))
      end do
    end if
end do

---Scatter to the points.

    do ia=1,nelem
      if (mrhist(ia,5).eq.1) then
        do in=1,2
          temppa(intmat(ia,in)) = temppa(intmat(ia,in)) +
                    rlen(ia)*ce(a,in)*tempea(a)
          & temppb(intmat(ia,in)) = temppb(intmat(ia,in)) +
                    rlen(ia) * abs(ce(a,in)) *
                    abs(tempea(ia))
          & temppc(intmat(ia,in)) = temppc(intmat(ia,in)) +
                    rlen(ia) * abs(ce(a,in)) *
                    tempec(a)
          end do
        end if
      end do

---Boundary terms
temppa(ibdry_l) = 0.
temppa(ibdry_r) = 0.

The error estimate at the points

\[
\text{do ip=1,npoin}
\quad \text{if ( ipact(ip) .eq. 1 )}
\quad \begin{align*}
& \text{errorp(ip) = abs(temppa(ip)) /} \\
& \text{( temppb(ip) + epsil*temppc(ip) )}
\end{align*}
\text{end do}
\]

Compute the error estimate at the elements (max. of node values).

\[
\text{do ie=1,nelem}
\quad \text{if (mrhist(ie,5) .eq. 1) }
\quad \begin{align*}
& \text{errore(ie)=amax1( errorp(intmat(ie,1)),} \\
& \text{errorp(intmat(ie,2)))}
\end{align*}
\text{end do}
\]

Mark the elements for refinement or de-refinement.

\[
\text{do ie=1,nelem}
\quad \text{if (mrhist(ie,5).eq.1) then}
\quad \begin{align*}
& \quad \text{if (errore(ie).gt.ctore) lrefe(ie)=1} \\
& \quad \text{if (errore(ie).lt.ctode) ldere(ie)=1}
\end{align*}
\quad \text{end if}
\quad \text{end if}
\]

Buffer Layers

\[
\text{do ib=1,nbuff}
\quad \text{do ip=1,npoin}
\quad \text{iptemp(ip)=0}
\quad \text{end do}
\]

\[
\text{do ie=1,nelem}
\]
if (mrhist(ie,5).eq.1) then
  do in=1,2
    if ( lrefe(ie) .eq. 1)
      iptemp(intmat(ie,in)) = i
    end do
  end do
end do

do ie=l,nelem
if (mrhist(ie,5).eq.1) then
  do in=1,2
    if (iptemp(intmat(ie,in)) .eq. 1)
      irefe(ie) = 1
    end do
  end do
end do

---Space Continuity on levels of refinement (only necessary on 1D)---

do iloop=1,5
  do ie=l,nelem
    ietemp(ie) = 0
    if (mrhist(ie,5).eq.1)
      ietemp(ie) = mrhist(ie,4) + lrefe(ie)
    end do
  end do
end do

do ip=1,npoin
  iptemp(ip)=0
end do

do ie=l,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 do
end if
end do

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
    end do
    if ( iptemp(intmat(ie,in))-mrhist(ie,4) .ge. 1)
      ldere(ie) = 0
  end do
  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
end if
end do
end do

return
end

*************************************************************************

subroutine unref(melem, mpoin, nelem, npoin, intmat, mrhist, ldere, ipact )

--------THIS ROUTINE DE-REFINES THOSE ELEMENTS MARKED FOR DE-REFINEMENT

integer intmat(melem,2)
integer mrhist(melem,5),ldere(melem)
integer ipact(mpoin)
c ------AMR arrays
    =========

    mhist(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 )

    ldere(ie) =1 ==> de-refine
    =0 ==> do not de-refine

    ------Loop over the elements
    do ie=1,nelem
       if ( ldere(ie).eq.1 .and. mhist(ie,4).ne.0 ) then

    --------Identify elements involved
    iparent=mhist(ie,1)
    ichild1=mhist(iparent,2)
    ichild2=mhist(iparent,3)

    --------Derefine iff both children are marked for de-refinement (4 steps)
    if ( ldere(ichild1).eq.1 .and. ldere(ichild2).eq.1 )
       then

    ............(1) Identify and "deactivate" points not assoc. w/parent.
    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

    25
end do

(2) Disable children elements

mrhist(ichild1,5)=0
mrhist(ichild2,5)=0

(3) Enable parent element

mrhist(iparent,5)=1

(4) Unmark the children elements

ldere(ichild1)=0
ldere(ichild2)=0

end if ! (...if all children marked)

d end if ! (...if one child marked)

d end do ! (end loop over elements)

d return
d end

c*************************************************************
c subroutine refine(melem, mpoin, nelem, npoin, intmat, nrmax,
& mrhist, xp, rho, rhov, rhoE, lrefe, ipact )
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 ------AMR arrays

  ===========
c

  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 )

c
  lrefe(ie) =1 ======> refine
  =0 ======> do not refine

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
    end if
  end do

  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)) )

27
rhoE(npoin) = 0.5*( rhoE(intmat(ie,1)) +
   rhoE(intmat(ie,2)) )

------New Element Connectivity (Input new elements at the end of the list)

intmat(nelem+1,1) = intmat(ie,1)
intmat(nelem+1,2) = npoin
intmat(nelem+2,1) = npoin
intmat(nelem+2,2) = intmat(ie,2)

------Disable the parent elements and enable the children

mrhist(ie,5)=0
mrhist(nelem+1,5) = 1
mrhist(nelem+2,5) = 1

------Modify the rest of the mesh refinement history parameters

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

------Modify the current number of elements

nelem=nelem+2
end if
end do

return
end

*************************************************************************

subroutine fluxld(mpoin, melem, npoin, nelem, gamma, cour,
Compute the fluxes at the points and the time step.

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)

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) )
   end if
end do

TIME STEP: dt.

----------Compute the "maximum velocity" (sound speed + v) at the points.
do ip = 1, npoin
   if ( ipact(ip).eq.1 )
      vmax(ip) = sqrt(gamma*p(ip)/rho(ip)) + abs(v(ip))
   end if
end do

----------and the time step...

29
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
c
end

***********************************************************************
c
subroutine deltau(mpoin, melem, npoin,nelem,intmat, 
  & du,dup,flux,dt,ce,rlen,cour,di,rho,rhov,rhoE,diff, 
  & vmax,mrhist)
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=1.0e20 
csmall=-1.0e20
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
----------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)
  end do
  if (mrhist(ie,5).eq.1 .and. rlen(ie).gt.rlmax)
    rlmax=rlen(ie)
  end do
end do

----------Initialize dup
c
do im=1,3
do ip=1,npoint
  dup(ip,im)=0.
end do
end do

----------BEGIN LOOP OVER THE ELEMENTS-----------------------------
c
do 1000 ie=1,nelem
  if ( mrhist(ie,5) .eq. 1 ) then
    ----------Compute du at the elements
    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
end do

----------Compute the diffusion terms at the elements
vv = \max (v_{\text{max}}(\text{intmat}(ie,1)), v_{\text{max}}(\text{intmat}(ie,2)))

\text{do}\ \text{in}=1,2
\text{diff}(ie,1) = \text{diff}(ie,1) + ce(ie,in) * rho(\text{intmat}(ie,in))
& \text{diff}(ie,2) = \text{diff}(ie,2) + ce(ie,in) * rhov(\text{intmat}(ie,in))
& \text{diff}(ie,3) = \text{diff}(ie,3) + ce(ie,in) * rhoE(\text{intmat}(ie,in))
\text{end do}

dfact = 0.5 * di

\text{do}\ \text{im}=1,3
\text{diff}(ie,im) = dfact * \text{diff}(ie,im) * rlen(ie) * rlmin
\text{end do}

\text{----------Scatter to the points.}
\text{do}\ \text{im}=1,3
\text{do}\ \text{in}=1,2
\text{dup}(\text{intmat}(ie,in), im) = \text{dup}(\text{intmat}(ie,in), im) +
& 0.5 * rlen(ie) * du(ie, im) +
& ce(ie,in) * \text{diff}(ie,im)
\text{end do}
\text{end do}
\text{end if}

1000 \text{ continue}

\text{----------END OF THE ELEMENT LOOP-----------------------------}

\text{return}
\text{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


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.htm_ ]. 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].

### 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.htm_ ]. 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].

### Subject Terms
- Computational Techniques, Adaptive-Mesh-Refinement, Fluid Dynamics, Unstructured-Mesh