An Introduction to the Piecewise Parabolic Method (PPM) Library I. Interpolation and Hydrodynamics Routines: Part 3de

Paul Ralph Woodward,[†] *B Edgar,*
and S. E. Anderson,[†]

Laboratory for Computational Science and Engineering
Department of Astronomy, School of Physics and Astronomy
116 Church St. SE.
University of Minnesota
Minneapolis, Minnesota 55455

3.d Lagrangian Hydrodynamics

Now on to Lagrangian hydrodynamics! When written in Lagrangian coordinates, Euler's equations in conservation form are

$$\begin{aligned} &\frac{\partial}{\partial t}\tau + \frac{\partial}{\partial m}(\, r^{\alpha}\, u\,) = 0 \\ &\frac{\partial}{\partial t}u + r^{\alpha}\frac{\partial}{\partial m}\, p = f \\ &\frac{\partial}{\partial t}E + \frac{\partial}{\partial m}(r^{\alpha}\, up) = uf \end{aligned} \tag{3.33}$$

where τ is the specific volume,

$$\tau = \frac{1}{\rho}\,, \tag{3.34}$$

p is the pressure, u is the velocity, E is the total energy per unit volume, f is a body force, and m is the mass coordinate. The mass coordinate is related to the spatial coordinate by

$$m(r) = \int_{r_0}^{r} \rho(r) r^{\alpha}\, dr, \tag{3.35}$$

with $\alpha = 0, 1, 2$ for planar cylindrical or spherical symmetry, respectively. To these equations are added an equation of state (EOS) which specifies the relationship between the state variables. The two equations of state described in the Eulerian section (eqns. 3.3 and 3.5) are also used here. These are a gamma law EOS

$$p = (\gamma - 1)\rho\varepsilon \tag{3.3}$$

where γ is the ratio of specific heats and ε is the internal energy per unit volume, and a generalized equation of state

$$p = \Phi + g\,\rho\,\varepsilon \tag{3.5}$$

where Φ and g are two EOS constants.

The initial data consist of averages of the conserved quantities in each zone at time t^n,

$$U_j^n = \frac{1}{\Delta m_i} \int_{m_i - 1/2}^{m_i + 1/2} U(r, t^n)\, dm,$$

$$U = \begin{pmatrix} \tau \\ u \\ E \end{pmatrix}, \tag{3.36}$$

$$\Delta m = \int_{r_i - 1/2}^{r_i + 1/2} \rho(r) r^\alpha \, dr \qquad m_{i+1/2} = \sum_{j \le i} \Delta m_j .$$

These zone averages are updated by first finding a finite difference approximation to eqns (3.33) by integrating them over a space-time volume Δm by Δt. This results in

$$r_{i-1/2}^{n+1} = r_{i-1/2}^{n} + \Delta t \;\; \bar{u}_{i-1/2}$$

$$\bar{A}_{i-1/2} = \frac{(r_{i-1/2}^{n+1})^{\alpha+1} - (r_{i-1/2}^{n})^{\alpha+1}}{(\alpha+1)\bar{u}_{i-1/2}\,\Delta t}$$

$$\tau^{n+1}{}_{i-1/2} = \frac{(r_{i+1/2}^{n+1})^{\alpha+1} - (r_{i-1/2}^{n+1})^{\alpha+1}}{(\alpha+1)\,\Delta m_i} \tag{3.37}$$

$$u_{i-1/2}^{n+1} = u_{i-1/2}^{n} + 1/2\,(\bar{A}_{i-1/2} + \bar{A}_{i+1/2})\,\frac{\Delta t}{\Delta m_i}\,(\bar{p}_{i-1/2} - \bar{p}_{i+1/2}) + \frac{\Delta t}{2}\,(f_i^{n+1} + f_i^{n+1})$$

$$E_i^{n+1} = E_i^n + \frac{\Delta t}{\Delta m_i}\,(\bar{A}_{i-1/2}\,\bar{u}_{i-1/2}\,\bar{p}_{i-1/2} - \bar{A}_{i-1/2}\,\bar{u}_{i-1/2}\,\bar{p}_{i-1/2}) + \frac{\Delta t}{2}\,(u_i^n f_i^n + u_i^{n+1} f_i^{n+1})$$

Here the bars indicate time averaged quantities. As before, these time-averaged fluxes are found by applying a Riemann solver to resolve the discontinuities at each zone interface.

At this point, the zone averages are known in Lagrangian coordinates. It is now necessary to construct a remap step, similar to the process described in the example `ADVECT_1` from the interpolation section. This remap step maps (regrids) the hydrodynamical quantities from the Lagrangian Grid to the Eulerian grid. At the completion of the remap step, a diffusion step similar to the diffusion step of the Direct Eulerian scheme described above is required. This adds additional numerical diffusion to those zones which contains shock structures likely to produce numerical noise.

Using modules from the PPMLIB library, the above procedure may be divided into the following steps:

1) Piecewise parabolic distributions of pressure and velocity are computed in each zone using interpolation routines described in the previous chapter. This produces a global distribution of each variable which is parabolic within each zone, but may have discontinuities at the zone boundaries.

2) Determine effective left and right states for a Riemann problem at each zone interface by averaging the above interpolations over the two domains of dependence for each zone interface. This generally results in discontinuities at zone boundaries.

3) The time averaged zone interface values of pressure ($\bar{p}_{j-1/2}$) and velocity ($\bar{u}_{j-1/2}$) are calculated using a Riemann solver to resolve the hydrodynamic discontinuities at the zone interfaces produced by step 2.

4) Calculate the updated zone averages on the Lagrangian grid by by differencing the Lagrangian fluxes using the difference equations (3.37).

5) Determine new piecewise parabolic interpolations of density, pressure, and velocity on the Lagrangian grid.

6) Integrate the interpolations found in step 5 over regions of overlap between the Lagrangian and Eulerian grids. This results in a set of advective fluxes which may be used to map quantities from the Lagrangian grid to the Eulerian grid.

7) Determine the preliminary zone averages on the Eulerian grid by conservatively differencing the advective fluxes found in step 6.

8) Determine zone centered Eulerian difusion velocities in those zones which are likely to generate numerical noise, such as strong shocks or shocks moving slowly with respect to the numerical grid.

9) Determine zone interface difusion fluxes of mass, momentum, and total energy from eqns. 3.11-12.

10) Calculate the final updated zone averages by conservatively differencing the diffusion velocities using eqns 3.14

In the following discussion, the above steps will be detailed under the simplifying assumptions of planar symmetry ($\alpha = 0$) and of no body forces.

Step 1. Interpolation of distributions

For a Lagrangian calculation, it is necessary to interpolate a global description of pressure and velocity. These interpolations are found indirectly by first interpolating Riemann invariants as described in the Direct Eulerian section. These interpolations may be found by the routines **PPM98_INTERP0_UP** for uniform grids or **PPM98_INTERP_UP** for nonuniform grids which first interpolate Riemann invariants from which the velocity and pressure interpolations are derived. This allows the monotonicity constraints to be applied directly to the Riemann invariants. The reader should note that only interpolations of velocity and pressure are necessary. An interpolation of the density is not needed at this point.

Step 2. Calculating the Effective Left and Right States

In Lagrangian hydrodynamics there are two domains of dependence (those portions of the zones adjacent to a zone interface which may interact during the timestep) over which spatial averages of pressure and velocity must be found. These averages form the effective left and right states for the Riemann problems. These domains of dependence may be approximated by tracing back the paths of all sound waves which may impinge upon the zone interface from the left and from the right during the timestep. This is illustrated in figure 21. The zone averaged Lagrangian sound speed

$$C_i = \sqrt{\gamma P_i \rho_i} \tag{3.38}$$

may be used to estimate the extent of the two domains of dependence. The required states consist of averages of velocity and pressure over these domains of dependence unsing eqns. 2.27-28. These effective left and right states may be found by using **PPM98_LAG0_DOMAIN_AVG** for uniform grids or **PPM98_LAG_DOMAIN_AVG** for non-uniform grids, both of which incorporate the interpolations of step 1.

Step 3. Solving the Riemann Problems

The determination of an effective left and right state for a zone boundary usually results in a discontinuity at that interface. As with the Direct Eulerian calculations, this discontinuity is resolved by using a Riemann solver. This calculates the solution to Riemann's shock tube problem, which is the break up of an arbitrary fluid discontinuity. In Lagrangian coordinates, these solutions may be found by **PPM98_L_RIEMANN_GAMMA** for gamma-law equations of state, eqn (3.3) or **PPM98_L_RIEMANN** for the general equation of state eqn (3.5). It should be noted that time averaged values of density are not returned by these modules, as this is not required in Lagrangian computations. With this calculation of the Lagrangian fluxes, it is now possible to update the zone averages on the Lagrangian grid.

Step 4. Calculating U_j^{n+1}

The Lagrangian fluxes found by solving Riemann problems at the zone interfaces are combined with eqns (3.37) to determine the new zone averages on the Lagrangian grid. With the assumptions of planar symmetry and no body forces, eqns (3.37) become

$$
\begin{aligned}
r^{n+1}_{i-1/2} &= r^{n}_{i-1/2} + \Delta t \;\; \bar{U}_{i-1/2} \\
\tau^{n+1}{}_{i-1/2} &= \frac{r^{n+1}_{i+1/2} - r^{n+1}_{i-1/2}}{\Delta m_i} \\
u^{n+1}_{i-1/2} &= u^{n}_{i-1/2} + \frac{\Delta t}{\Delta m_i} \left(\bar{p}_{i-1/2} - \bar{p}_{i+1/2} \right) \\
E^{n+1}_{i} &= E^{n}_{i} + \frac{\Delta t}{\Delta m_i} \left(\bar{u}_{i-1/2} \, \bar{p}_{i-1/2} - \bar{u}_{i+1/2} \, \bar{p}_{i+1/2} \right)
\end{aligned}
\tag{3.39}
$$

Step 5. Interpolation of distributions on Lagrangian Grid

At this point, the new zone averages of density, pressure, and velocity are known on the Lagrangian grid. Thus it is now necessary to construct new piecewise parabolic interpolations. As before these are done indirectly through the Riemann invariants. Interpolations of density as well as pressure and velocity are now required. Note that at this point the Lagrangian grid is, in general, non-uniform. The pressure and velocity interpolations are found from **PPM98_INTERP_UP** and the density interpolations are found using **PPM98_INTERP_RHO**.

Step 6. Determining Remap Fluxes

The fluxes necessary to remap from the Lagrangian to the Eulerian grid may be found by first determining the regions of overlap between the two grids. The interpolations of density, pressure, and velocity may then be interpolated over these regions to produce advective fluxes. These fluxes may be determined by **PPM98_MAP_FLUX_GAMMA_1D** for gamma law gases or **PPM98_MAP_FLUX_1D** for non gamma law gases. Both of these routines incorporate the interpolation of density, pressure, and velocity from the previous step. Both assume non-uniform grids.

Step 7. Determining Preliminary Zone-averages

The preliminary updated zone-averages of the conserved quantities on the Eulerian grid may now be found by conservatively differencing the advective fluxes found in step 6.

Step 8. Calculating Numerical dissipation

It is necessary in a Lagrangian calculation, as in the Direct Eulerian calculations, to add additional numerical diffusion to those zones which may contain shock structures likely to produce noise. The conserved quantities are known now on the Eulerian grid, so the diffusion procedure is basically the same as that of the Direct Eulerian calculation. First, zone centered Eulerian diffusion velocities are found in the manner described in the Direct Eulerian description.

Step 9. Calculating Diffusion Fluxes Zone interface centered diffusion fluxes are found using the zone centered difusion velocities and eqns (3.12b).

$$
\begin{aligned}
f^{\rho}_{j-1/2} &= \upsilon_{j-1/2} \left(\rho^{n+1}_{i} - \rho^{n+1}_{i-1} \right) \\
f^{u}_{j-1/2} &= \mu^{L}_{j-1/2} \, u^{n+1}_{i} - \mu^{R}_{j-1/2} \, u^{n+1}_{i-1}) \\
f^{E}_{j-1/2} &= \mu^{L}_{j-1/2} \, E^{n+1}_{i} - \mu^{R}_{j-1/2} \, E^{n+1}_{i-1})
\end{aligned}
\tag{3.12b}
$$

These fluxes are found by **PPM98_DIFUSEFLUX_HYDRO_CART**, which also returns the diffusive volume and mass fluxes of eqns (3.13a-c). These latter fluxes will be useful when there are other quantities in the calculations, such as the passively advected quantities described later.

Step 10. Updating Zone Averages The final updated zone averages may be found using the diffusive fluxes and eqns (3.14):

$$\rho'^{n+1}_{j-1/2} = \rho^{n+1}_{j-1/2} + \frac{\left(A_{j-1/2}\, f^{\rho}_{j-1/2} - A_{j+1/2}\, f^{\rho}_{j+1/2}\right)}{\Delta V_j}$$

$$(\rho u)'^{n+1}_{j-1/2} = (\rho u)^{n+1}_{j-1/2} + \frac{\left(A_{j-1/2}\, f^{u}_{j-1/2} - A_{j+1/2}\, f^{u}_{j+1/2}\right)}{\Delta V_j} \tag{3.14}$$

$$(\rho E)'^{n+1}_{j-1/2} = (\rho E)^{n+1}_{j-1/2} + \frac{\left(A_{j-1/2}\, f^{E}_{j-1/2} - A_{j+1/2}\, f^{E}_{j+1/2}\right)}{\Delta V_j}$$

e Lagrangian + Remap Tutorial

The hydrodynamical routines of PPMLIB will be demonstrated in this tutorial using templates written in a Lagrangian + remap style. This set of templates has been developed to illustrate how complete hydrodynamic routines may be constructed using PPMLIB modules. While they may serve as useful hydrodynamics routines in and of themselves, they are intended to serve as templates or frameworks from which the user starts, customizing as necessary to suit the requirements of the user's problem. The current Lagrangian + remap templates are:

1) Do_PPPM_LR0_1DC_GAMMA: This is the basic routine. It computes one timestep of a one-dimensional problem assuming a uniform grid and planar symmetry. A gamma law equation of state is also assumed.

2) Do_PPPM_LR_1DC_GAMMA: This is the same as 1, except the assumption of a uniform grid is relaxed.

3) Do_PPPM_LR0_1DC: This is is the same as 1, but a gamma law equation of state is no longer assumed.

4) Do_PPPM_LR0_1DC_PAQ_GAMMA: Again, this begins with the basic program of 1, but now adds the concept of a passively advected quantity (PAQ). A passively advected quantity is something which is merely transported with the flow. It does not actively influence the hydrodynamics, or if it does, its effects can be separated out into another step. For example, multi-dimensions are treated using directional splitting, in which changes in the flow as computed in only one direction at a time. The transverse momenta are merely passively advected during each individual directional pass.

5) Do_PPPM_LR0_2DC_GAMMA: This uses the concept of a passively advected quantity introduced in the last template to compute two dimensional flows.

6) Do_PPPM_LR0_3DC_GAMMA: This extends the last program to three dimensions.

These will now be discussed in greater detail. It is not possible to reproduce the entire source code to each of these templates here, only the highlights will be stressed. The user is encouraged to study the source to the templates which can be found at the PPMLIB web site. A simple driver is provided for each of these templates, which is the program which performs the basic housekeeping chores which, at the very least, must include

i) Initiating the problem by setting the initial values of all zone averages and constants upon which the program depends.

ii) Impose boundary conditions by setting the fake zones to values appropriate to the problem boundary conditions.

iii) Execute the hydrodynamic calculation.

iv) Produce output as desired.

v) Terminate execution at end of the problem.

The drivers provided for this tutorial are the same as those provided for the Direct Eulerian tutorial discussed above. The only change is to increase the number of fake zones used to implement boundary conditions (NBDY) from five to nine. These addisional fake zones are necessitated by the second interpolation of zone averages during the remap step. Therefore, these drivers shall not be discussed in detail here, except for the ‘‘hydro loop’’ which lies at the heart of the drivers and runs the actual hydrodynamic calculation. This is the hydro loop from ‘‘Driverlr1dg0c.EXA,’’ the driver to demonstrate **Do_PPMLR0_1DC_GAMMA :**

```
c
1010   continue
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                      c
c     Step 1: Boundary Condition Step                                  c
c       Boundary conditions are set at the beginning of each timestep  c
c       by placing appropriate values of XL, RHO, P, and U in the fake c
c       zones on the left and right ends of the arrays.                c
c                                                                      c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c

c      Impose reflecting boundaries:
      k=1-nbdy
      call boundary_refl(xl(k), rho(k), u(k), p(k), n, nbdy)

c
c     clear old maximum Courant number:
      courmx = 0.0
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                      c
c     Step 2: Perform a single timestep Lag + remap hydro update       c
c                                                                      c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      nn = N
      nnbdy = NBDY
c
      k = 1-nbdy
c
      call do_ppmlr0_1dc_gamma (xl(k), rho(k), p(k), u(k), rhonu(k),
     ^   pnu(k), unu(k), gamma, dt, smlrho, smallp, smallu, smalle,
     ^     courmx, nn, nnbdy)
c
c   Increment counters:
       time = time + dt
       nsteps = nsteps+1
c
c    Rename for the next timestep (It is assumed that XL is unchanged)
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                      c
c    Step 3:   Replace old zone averages with new zone averages        c
c                  i preparation of the next timestep.                 c
c                                                                      c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      do i=1,n
      rho(i) = rhonu(i)
      u(i) = unu(i)
      p(i) = pnu(i)
      enddo
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                      c
c     Step 4: Output                                                   c
c                                                                      c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c  A:  print values of updated quantities to screen each timestep
      print *,' '
      print *,' '
      print *,' '
      print *,' Updated results are:'
      print *,' i   xl(i)        rho(i)       p(i)         u(i)    '
      do 9999 i=1,n
        write(*,666) i,xl(i),rho(i),p(i),u(i)
666    format (i3, 1h ,1p, 8(1x,e12.5))
9999  continue
      print *,' ******  NSTEP TIME COURMX  **** = ',nsteps,time,courmx
c
c  B:  Create disk data files:
```

```
c   create some output files for mongo , one every timestep,
c    one every 10 timesteps, one every 25 timesteps:
c
      ido10 =  nsteps - 10*(nsteps/10)
      ido25 =  nsteps - 25*(nsteps/25)
     write(101,666) i,xl(1), 0.0, 0.0, 0.0
     if (ido10 .eq. 0) write(110,666) i,xl(1), 0.0, 0.0, 0.0
     if (ido25 .eq. 0) write(125,666) i,xl(1), 0.0, 0.0, 0.0
      do i=1,n
      write(101,666) i,xl(i),rho(i),p(i),u(i)
     if (ido10 .eq. 0) write(110,666) i,xl(i),rho(i),p(i),u(i)
     if (ido25 .eq. 0) write(125,666) i,xl(i),rho(i),p(i),u(i)
      enddo
     write(101,666) i,xl(n+1), 0.0, 0.0, 0.0
     if (ido10 .eq. 0) write(110,666) i,xl(n+1), 0.0, 0.0, 0.0
     if (ido25 .eq. 0) write(125,666) i,xl(n+1), 0.0, 0.0, 0.0
c
c     Go back to 1010 if further timesteps are to be computed:
c
      if(nsteps .lt. MAXSTEPS) goto 1010
```

At the beginning of this loop, a call to a subroutine `boundary_refl` imposes reflecting boundary conditions by placing appropriate values in the `NBDY` fake zones appended to each end of the data arrays. Next comes the call to **Do_PPMLR0_DC_GAMMA** will will perform one timestep. Input consists of the arrays holding the locations of the lefthand zone interfaces (`xl`), and the zone averages of density (`rho`), pressure (`p`), and velocity (`u`). Scalar input includes the equation of state parameter (`gamma`), the timestep (`dt`), and trivial values for density (`smlrho`), pressure (`smallp`), velocity (`smallu`), and energy (`smalle`). The number of non-fake zones (`nn`) and the number of fake zones (`nnbdy`) are also sent. Returned arrays are the new zone averages of density (`rhonu`), pressure (`pnu`), and velocity (`unu`). Also returned is the Courant number (`courno`) or stability condition. This is the maximum Courant number encountered during the hydrodynamic calculation. For stability, this should always be less than 1.0, adjusting the timestep, if necessary. Note that because the initial zone averages are not overwritten during the calculation, this must be done explicitly by the driver in preparation for the next timestep. Finally, output is produced if desired. Now, highlights from the individual templates will be examined.

Do_PPMLR0_1DC_GAMMA

```
      subroutine do_ppmlr0_1dc_gamma
     ^               (xl,      rho,      p,      u,  rhonu,
     ^                pnu,    unu, eosgam,     dt, smlrho,
     ^              smallp, smallu, smalle, courmx,
     ^          n_real_zones_in, nbdy_set)
c
c
```

Input to this template consists of the locations of the lefthand zone interfaces (`xl`), zone averages of density (`rho`), pressure (`p`), and velocity (`u`). Other input consists of the EOS parameter (`eosgam`), the time step (`dt`), and trivial values for density (`smlrho`), pressure (`smallp`), velocity (`smallu`), and energy (`smalle`). The number of real zones in the calculation is `n_real_zones_in` and the number of fake zones appended to each end of the input arrays is `nbdy_set`. It returns updated zone averages of density (`rhonu`), pressure (`pnu`), and velocity (`unu`). Also returned is the maximum Courant number (stability parameter) `courmx`.

This template calculates one timestep of an Eulerian calculation performed as a Lagrangian step followed by a remap, assuming a gamma law gas, uniform grid and planar symmetry. It requires a minimum of 9 fake zones added to each end of the input arrays to account for the boundary zones. This is reflected in the dimensions of the arrays:

```
      dimension       xl(1-nbdy_set:n_real_zones_in + nbdy_set+1),
     ^               rho(1-nbdy_set:n_real_zones_in + nbdy_set),
     ^                 p(1-nbdy_set:n_real_zones_in + nbdy_set),
     ^                 u(1-nbdy_set:n_real_zones_in + nbdy_set),
     ^             rhonu(1-nbdy_set:n_real_zones_in + nbdy_set),
     ^               pnu(1-nbdy_set:n_real_zones_in + nbdy_set),
     ^               unu(1-nbdy_set:n_real_zones_in + nbdy_set)
```

Here, the input parameter nbdy_set must be at least 9 to allow sufficient boundary zones for two interpolations and a diffusion step. After the allocation of scratch arrays and the calculation of a few necessary quantities, the first PPMLIB module is called.

```
      call do_ppm98_lag0_flux_gamma
     ^  (      xl(K),      dx(K),    rho(K),      p(K),      u(K),    ceul(K),
     ^     unsmup(K),    pavl(K),   uavl(K), courno(K),    eosgam,     smallp,
     ^            dt,     nzones, nbdy)
```

This routine, **Do_PPM98_LAG0_FLUX_GAMMA** uses as input the locations of the left hand zone interfaces, (xl_, the zone widths (dx), zone averages of density (rho), pressure (p), velocity (u) Eulerian sound speed (ceul), and the timestep (dt). Note that the arrays have a offset of equal to 1-nbdy_set, pointing to the the first fake zone. The number of fake zones sent to **Do_PPM98_LAG0_FLUX_GAMMA**, nnbdy_set, is the number appropriate to the subroutine being called *NOT* the number of fake zones required by **Do_PPMLR0_1DC_GAMMA**. Thus the number of zones to process N is also different to reflect this. The results of this routine are the Lagrangian fluxes of pressure (pavl) and velocity (uavl) at each lefthand zone interface. Also returned are arrays containing the results of the unsmoothness from the interpolations and the Courant number or stability condition. Within the routine **Do_PPM98_LAG0_FLUX_GAMMA**, are convenient points at which the geometrical terms (non-planar symmetries) or external force terms may to added to the domain averages following the prescription in Colella and Woodward (1984). The source to **Do_PPM98_LAG0_FLUX_GAMMA** is provided. These lagrangian fluxes are then used to provide the new zone averages on the Lagrangian grid.

```
      call ppm98_lag_update_cart
     ^     (      xl(K),       dx(K),      dm(K),      u(K), e_total(K),
     ^          uavl(K),     pavl(K),  xllag(k), dxlag(K),  rholag(K),
     ^          ulag(K), e_totlag(K),    smallu,   smalle,          dt,
     ^          nzones)
```

The reader should have noted that array offsets have been adjusted to account for the loss of boundary zones that occurred during the process of determining the Lagrangian fluxes. This routine computes the zone averages on the Lagrangian grid of density (rholag) velocity (ulag), and total energy (e_totlag). It also returns the locations of the lefthand zone interfaces of the Lagrangian grid (xllag) and the individual widths of the Lagrangian zones (dxlag).

The remap portion of the routine now begins. Because the numerical grid is now definitely non-uniform, it is necessary to use the non-uniform grid versions of the interpolation routines. This means first computing the non-uniform grid factors.

```
c   precompute the interpolation factors
      call ppm98_interpolation_factors
     ^   (dxlag(K), dalfac(K), darfac(K),  facda(K), fcdazl(K),
     ^   facdal(K), fa6dal(K), fa6dar(K), frrdal(K), frrdar(K),
     ^   flldal(K), flldar(K), nzones,nbdy)
```

Note that the input is the widths of the Lagrangian grid, not the initial grid zone widths. These factors are used in **PPM98_MAP_FLUX_GAMMA_1D** which constructs interpolations of density, pressure, and velocity from the Riemann invariants. It then determines the regions of overlap between then two different grids and integrates these interpolations over these regions. These results are used to construct the fluxes of volume (`dvoll`), mass (`dmassl`), momentum (`dmoml`), and total energy (`denl`) necessary to map the conserved quantities from the Lagrangian grid defined by the left hand zone interface locations given by (`xllag`) to the Eulerian grid defined by the left hand zone interface locations (`xl`).

```
      call  ppm98_map_flux_gamma_1d
     ^    (  dalfac(K), darfac(K),  facda(K), fcdazl(K), facdal(K), fa6dal(K),
     ^       fa6dar(K), flldal(K), flldar(K), frrdal(K), frrdar(K),  xllag(k),
     ^           xl(K), rholag(k),   plag(k),   ulag(k), unsmad(k), unsmup(k),
     ^       stpadb(k),  dvoll(k), dmassl(k),  dmoml(k),   denl(k), dvolfl(k),
     ^      dmassfl(k), courno(k),        dt,    eosgam,   smalle, nzones, nbdy)
```

In addition to the interface fluxes, this routine also returns the fractional volume advected (`dvolfl`) and the fractional mass advected (`dmassfl`) which will be useful when there exist quantities in addition to the hydrodynamical quantities to remap. With the zone interface fluxes, the zone averages are updated explicitly.

```
c
      do i=kstart3,kend3
          dmnu(i) = dm(i) + dmassl(i)- dmassl(i+1)
          rhonu(i) = dmnu(i) * dxinv(i)
          unu(i) = (ulag(i)*dm(i) + dmoml(i)-dmoml(i+1))/dmnu(i)
          eenew = (e_totlag(i)*dm(i) + denl(i)-denl(i+1))/dmnu(i)
          eknew = 0.5*unu(i)*unu(i)
          einew = mymax1 (smalle, eenew - eknew)
          e_totnu(i) = einew+eknew
          pnu(i) = (eosgam - 1.0) * rhonu(i)*einew
      enddo
```

Now, the diffusion section begins. First, the zone centered Eulerian velocities are found by **PPM98_LAG_DIFUSE_1D**.

```
      call ppm98_lag_difuse_1d
     ^            (   dx(k-2), dx(k-1), dx(K), dx(k+1),  dx(k+2),
     ^                u(k-2), u(k-1), u(K), u(k+1),  u(k+2),
     ^                p(k-2), p(K), p(k+2), rho(k-2),   rho(K), rho(k+2),
     ^               ceul(k-2), ceul(K), ceul(k+2),
     ^       difuse_vel(K),   courno(K),   smallu,        nzones)
```

The difference between **PPM98_LAG_DIFUSE_1D** for Lagrangian calculations and **PPM98_EUL_DIFUSE_1D** for Direct Eulerian calculations is the inclusion of an additional Courant calculation to ensure that no shock wave travels move than one zone width per timestep. It should be noted that this input consists of zone averages at the initial time, not the preliminary zone average just determined. From these zone centered diffusion velocities (`difuse_vel`), diffusion fluxes are found:

```
      call ppm98_difuseflux_hydro_cart
     ^    (         dx(K), difuse_vel(K),
     ^        rhonu(K),         unu(K), e_totnu(K),
     ^        dvoll(K),      dmasll(K),  dmasrl(K),
     ^       dmassl(K),       dmoml(K),  detotl(K),
     ^       courno(K),             dt,     nzones,
     ^        nbdy_difuse_fluxes )
```

The fluxes that **PPM98_DIFUSEFLUX_HYDRO_CART** computes are the volume flux (`dvoll`) and the left and right mass fluxes of eqns (3.13a-c) (`dmasll` and `dmasrl`) as well as the net fluxes of mass (`dmassl`), momentum (`dmoml`), and total energy (`detotl`). After this, the final zone average update is done explicitly:

```
      do 9100   i = KSTARTDF2,KENDDF2
c
c           this saves a flop or two:
          if ((dvoll(i) + dvoll(i+1)) .ne. zero)   then
             dmnew(i) = dmnu(i) + dmassl(i) - dmassl(i+1)
             rhonu(i) = dmnew(i) * dvolinv(i)
             dmnewi = one / dmnew(i)
             unu(i) = (unu(i) * dmnu(i)  +  dmoml(i) - dmoml(i+1))  *  dmnewi
             e_totnu(i) = (e_totnu(i) * dmnu(i) +
     ^                                           detotl(i) - detotl(i+1)) * dmnewi
9099         continue
         endif
9101  continue
          usq = unu(i) * unu(i)
c  if ((unu(i) * unu(i)) .lt. ssmlu2)   unu(i) = 0.0
         if ( usq .lt. ssmlu2)   then
            unu(i) = 0.0
            usq = 0.0
         endif
         eknu = 0.5 * usq
         e_intnu = mymax1 (smalle, (e_totnu(i) - eknu))
         pnu(i) = gamma1 * rhonu(i) * e_intnu
9100  continue
```

Here, in the interests of efficiency, where there are no left or right zone interface diffusive fluxes (that is `dvoll(i)` and `dvoll(i+1)` both are zero) there is no change to the zone average and the arithmetical operations are skipped. A special check in the velocity is also included to prevent underflows.

After the maximum Courant number has been determined, the calculation is complete and control returns to the calling routine. The next example shows how this program may be changed to accommodate a non-uniform grid.

Do_PPMLR_1DC_GAMMA

The template which relaxes the requirement of a uniform grid is **Do_PPMLR_1DC_GAMMA**. There is only one significant change. The call to **Do_PPM98_LAG0_FLUX_GAMMA** is replaced by calls to two routines:

```
c   precompute the nonuniform grid interpolation factors

      call ppm98_interpolation_factors
     ^   (   dx(K), dalfac(K), darfac(K),  facda(K), fcdazl(K),
     ^   facdal(K), fa6dal(K), fa6dar(K), frrdal(K), frrdar(K),
     ^   flldal(K), flldar(K), N,nnbdy_set)

      call  Do_PPM98_LAG_FLUX_GAMMA
     ^        (  xl(K),        dx(K),       dalfac(K),       darfac(K),
     ^       facda(K),   fcdazl(K),       facdal(K),       fa6dal(K),
     ^      fa6dar(K),   flldal(K),       flldar(K),       frrdal(K),
     ^      frrdar(K),      rho(K),            p(K),            u(K),
     ^        ceul(K),   unsmup(K),         pavl(K),         uavl(K),
     ^      courno(K),      eosgam,          smallp,              dt,
     ^         nzones, nbdy)
```

As noted earlier, The routine **PPM98_INTERPOLATION_FACTORS**, is used to collect together the various geometrical factors which arise when interpolations on non-uniform grids are performed. By placing them into one routine, efficiency is gained when more than one interpolation is performed on the same nonuniform grid. As is seen here, it is called with the same array offsets as

Do_PPM98_LAG_FLUX_GAMMA, the nonuniform grid version of **Do_PPM98_LAG0_FLUX_GAMMA**. The calculation then proceeds as before.

Do_PPPM_LR0_1DC

```
      subroutine do_ppmlr0_1dc
     ^         (      xl,    rho,       p,       u,  e_total,    ceul, rhonu,
     ^           e_totnu,    unu,      dt, smlrho,    smallu, smalle, courmx,
     ^           n_real_zones_in, nbdy_set)
c
```

This template relaxes the requirement of a gamma law gas. It uses the generalized equation of state of eqn (3.5)

$$p_i = \Phi_i + g_i \, \rho_i \, \varepsilon_i . \tag{3.32}$$

where p_i is the zone averaged pressure, ρ_i is the zone averaged density, and ε_i is the zone averaged internal energy. The two equation of state constants, Φ_i and g_i, must be determined for each zone from the initial data. Input to this template consists of the locations of the lefthand zone interfaces (`xl`), zone averages of density (`rho`), pressure (`p`), velocity `u` (`p`), total energy (`e_total`) and (`p`), Eulerian sound speed (`ceul`). Other input consists of the time step (`dt`), and trivial values for density (`smlrho`), velocity (`smallu`), and energy (`smalle`). The number of real zones in the calculation is `n_real_zones_in` and the number of fake zones appended to each end of the input arrays is `nbdy_set`. It returns updated zone averages of density (`rhonu`), total energy (`e_totnu`), and velocity (`unu`). It should be noted that new updated zone-averages of pressure are *not* returned. Also returned is the maximum Courant number (stability parameter) `courmx`.

There are several significant changes from **Do_PPMLR0_1DC_GAMMA**. First, additional input is required in the form of the zone averages of the total energy `e_total` and the Eulerian sound speed `ceul`. These are necessary in order to be able to calculate the generalized equation of state parameter.

```
          cesq = ceul(i) * ceul(i)
          ek = 0.5 * ( u(i)*u(i) )
          ei = e_total(i) - ek
          gamma1(i) = cesq(i) / (ei + p(i)/rho(i) )
          p00(i) = p(i)  -  gamma1(i) * rho(i) * ei
```

Here `gamma1(i)` and `p00(i)` correspond to g_i and Φ_i in eqn. (3.32), resp. These generalized equation of state parameters are used in the general EOS Lagrangian flux routine **Do_PPM98_LAG0_FLUX**.

```
      call do_ppm98_lag0_flux
     ^  (      xl(K),     dx(K),    rho(K),      p(K),      u(K),   ceul(K),
     ^        p00(K), gamma1(K), unsmup(K),   pavl(K),   uavl(K), courno(K),
     ^             dt,    nzones, nbdy)
```

Once gas from neighboring zones has mixed, the general equation of state parameters calculated above are no longer valid. Thus it is not possible to calculate a zone averaged pressure to return to the calling program. It is assumed that the calling program has access to the actual equation of state, and thus able to calculate a new zone averaged pressure and Eulerian sound speeds for each zone in preparation for the next timestep.

Do_PPM_LR0_1DC_PAQ_GAMMA

```
 subroutine do_ppmlr0_1dc_paq_gamma
^         (      xl,     rho,       p,       u,     paq,   rhonu,     pnu,
^              unu,   paqnu, eosgam,      dt, smlrho, smallp, smallu,
^            smalle, courmx, nbyvol, nbymass, n_real_zones_in, nbdy_set)
```

This template introduces the concept of passively advected quantities (PAQ) into the calculation. A passively advected quantity is some quantity which does not effect the hydrodynamics and is merely transported along with the flow (e.g. smoke or dye) or whose effects can be separated (e.g. transverse momenta). There are two types of passively advected quantities: mass weighted and volume weighted, both of which are allowed.

This template has as input the locations of the lefthand zone interfaces (`xl`), zone averages of density (`rho`), pressure (`p`), velocity (`u`) and an array which contains the zone averages of *all* of the passively advected quantities (`paq`) to be advected. Other input consists of the equation of state parameter (`eosgam`), the time step (`dt`), and trivial values for density (`smlrho`), pressure (`smallp`), velocity (`smallu`), and energy (`smalle`).

Two possible passively advected quantities exist: volume weighted and mass weighted. The number of of volume weighted passively advected quantities is `nbyvol` and the number of mass weighted passively advected quantities is `nbymass`. The number of real zones in the calculation is `n_real_zones_in` and the number of fake zones appended to each end of the input arrays is `nbdy_set`. Returned quantities are the updated zone averages of density (`rhonu`), pressure (`pnu`), velocity (`unu`) and the passively advected quantities (`paqnu`).

The initial zone averages of all passively advected quantities are passed as one two-dimensional array. The fastest running index is the grid index, the second index is the "paq" index. In this way, each PAQ is contiguous in memory. Thus one may think of them as arrays of arrays of zone-averaged data. Furthermore, it is assumed that the volume weighted passively advected quantities (if any) precede the mass weighted passively advected quantities (if any). Specifically, the assumed dimensions are

`[1-NBDY_SET:N_REAL_ZONES_IN+NBDY_SET, 1:(NBYVOL+NBYMASS)]`

where `NBYVOL` is the number of volume weighted passively advected quantities and `NBYMASS` is the number of mass weighted passively advected quantities. Similarly, the updated PAQ zone averages are returned in a single array, `paqnu`. The user must adhere to this convention, or undesirable behaviour may result. The hydrodynamics of this template as essentially those of `Do_PPPM_LR0_1DC_GAMMA`. Therefore the discussion here will focus on the passively advected quantities.

The zone interface fluxes of the volume weighted passively advected quantities are found using the fraction volume volumes determined by **PPM98_MAP_FLUX_GAMMA_1D**. These may be combined with the (as yet not constructed) piecewise parabolic PAQ interpolations to determine fluxes of the passively advected quantities using eqn (2.27-28). The actual preliminary PAQ update is performed by a call to **PPM98_ADVECT_PASSIVE0** as in the example `ADVECT_2` from the previous section in interpolation.

```
        call ppm98_advect_passive0
^              (dvolfl(K),    dvoll(K),    dvol(K),
^              dvolinv(K), paq(K,ipaq), smallda(K),
^              unsmuta(K),   steepa(K),     dal(K),
^           paqnu(K,ipaq),      nzones,       nbdy)
```

The program then proceeds to update the mass weighted PAQs in the same way as the volume weighted PAQs, also by a call to **PPM98_ADVECT_PASSIVE0**, with the volume terms replaced by appropriate mass terms.

```
           call ppm98_advect_passive0
```

```
^             (dmasfl(K),   dmassl(K),      dm(K),
^             dmnuinv(K), paq(K,ipaq), smallda(K),
^             unsmuta(K),   steepa(K),     dal(K),
^          paqnu(K,ipaq),      nzones,        nbdy )
```

Diffusion must also be applied to the passively advected quantities where the hydrodynamical quantities have had additional diffusion applied. Again, this is divided into volume weighted and mass weighted operations. First, the volume diffusion fluxes are supplied by a call to **PPM98_DIFUSEFLUX_PASSIVE**.

```
c   PPM98_DIFUSEFLUX_PASSIVE used the diffusion volumes DVOLL
c     computed above  to compute lefthand zone interface volume
c      weighted interface diffusion fluxes DAL.

          call ppm98_difuseflux_passive
^            (paqnu(K,ipaq),  dvoll(K),  dvoll(K), dal(K),
^                   nzones,      nbdy)
```

Here, input consists of the preliminary updated PAQ zone averages `paqnu` and the diffusive volume flux weighting factors `dvoll`. **PPM98_DIFUSEFLUX_PASSIVE** allows for differing left and right flux weighting factors, as will be needed to the mass weighted PAQs (cf ens. 3.12a-c). Here, because this is a volume weighted quantity, both weighting vectors are identical. The final zone average update is computed explicitly:

```
          do 6800   i = KSTARTDF,KENDDF
              ghoti = paqnu(i,ipaq)
             if ((dvoll(i) + dvoll(i+1)) .ne. zero)   then
               paqnu(i,ipaq) = paqnu(i,ipaq) + (dal(i) - dal(i+1)) * dvolinv(i)
             endif
6800      continue
       enddo
```

The mass weighted PAQs are next. Their fluxes are also found by a call to **PPM98_DIFUSEFLUX_PASSIVE** using the left and right mass flux weighting factors `dmasll` and `dmasrl` returned by **fBPPM98_DIFUSEFLUX_HYDRO_CART**.

```
               call ppm98_difuseflux_passive
^              (paqnu(K,ipaq), dmasll(K), dmasrl(K), dal(K), nzones,  nbdy)

          KSTARTDF = KSTARTDF1 + nbdy
```

The final updated zone averages of the mass weighted passively advected diagnostics are then determined.

```
               do 7800   i = KSTARTDF,KENDDF
              ghoti = paqnu(i,ipaq)
                  if ((dvoll(i) + dvoll(i+1)) .ne. zero)   then
                     dmnui = one / dmnu(i)
                     paqnu(i,ipaq) = (paqnu(i,ipaq) * dmnew(i) +
^                                        dal(i) - dal(i+1)) * dmnui
               endif
7800           continue
c
           enddo
```

As before, only those zones which have diffusive fluxes are modified.

Do_PPM_LR0_2DC_GAMMA

This template uses the concept of passively advected diagnostics introduced in the previous template to construct an algorithm suitable for two-dimensional hydrodynamical calculations. Recall that multi-dimensions are computed using the technique of directional splitting. In this technique, only changes in the direction of a pass (the longitudinal direction) are computed. Changes in the orthogonal or transverse directions are computed in subsequent passes, transverse momenta being passively advected during the longitudinal computation.

```
       subroutine do_ppmlr0_2dc_gamma
     ^    (  xl, rhozbb,    rho, rhoztt,   pzbb,      p, pztt,
     ^     uzbb,      u,   uztt,  utzbb,   utzb,     ut, utzt,
     ^    utztt,  rhonu,    pnu,    unu,   utnu, eosgam,   dt,
     ^   smlrho, smallp, smallu, smalle, courmx, n_real_zones_in, nbdy_set)
```

This template requires considerably more information than previous templates. In addition to the usual arrays of zone averages of density (`rho`), pressure (`p`), and longitudinal velocity (`u`). Zone averages of the transverse velocity (`ut`) are also required, The diffusion calculation, which needs to determine the orientation of any shocks which may be present requires zone averages of these quantities two rows above (suffix `ztt`) and two rows below (suffix `zbb`) the row being updated, as well as zone averages of the transverse velocity one row above (suffix `zt`) and one row below `zb`).

The reader should note carefully that because of the split of the velocity into longitudinal and transverse terms, the kinetic energy has also been split.

```
       gamm1i = 1.0 / ( eosgam - 1.0)
       do i=KSTART2 ,KEND2
          ekl = 0.5 * u(I)*u(I)
          eint = gamm1i * p(i) / rho(i)
          e_total(i) = eint + ekl
       enddo
```

Note that the array `e_total` contains on the internal energy (`eint`) and the longitudinal portion of the kinetic energy (`ekl`). Thus the call to **PPM98_LAG_UPDATE_CART** will only partially update the total energy. The transverse velocity (hand hence kinetic energy portion) remain unchanged during the Lagrangian hydrodynamical calculation step. this term would only cancel during the calculation of the zone averaged pressure.

Note that in the calculation of the advective fluxes used to remap the total energy, the kinetic energy terms due to the transverse velocity will not be included. These terms will not be determined until the transverse momenta advective fluxes have been determined. This requires first constructing the advected fractions over which the piecewise parabolic interpolations of the transverse velocity will be integrated. Historically, `PPM` has used volume fractions for this purpose rather than mass fractions as would be expected. Volume fractions are used here to maintain compatibility with these earlier versions. The actual advection will be performed using mass weightings however, as the call to **PPM98_ADVECT_PASSIVE0** `shows.`

```
             call ppm98_advect_passive0
     ^              (dvolfl(K),    dmassl(K),       dm(K),
     ^              dmnuinv(K),        ut(K), smalldu(K),
     ^              unsmutut(K),  steeput(K),  dmomtl(K),
     ^                  utnu(K),      nzones,      nbdy )
```

In addition to providing the preliminary updates of the transverse velocity `utnu`, this also provides the transverse momenta fluxes `dmomtl` which are used to remap the transverse kinetic energy terms in the remapped total energy zone averages found previously.

```
c     Update the transverse kinetic energy and add it to the new total E

      do i=KSTART3 ,KEND3+1
         if (dmassl(i) .eq. 0.0) then
            dektl(i) = 0.0
         else
            dektl(i) = 0.5 * dmomtl(I)*dmomtl(I) / dmassl(i)
         endif
      enddo
      do i=KSTART3 ,KEND3
         ekt = 0.5 * ut(I)*ut(I)
         ektnu = ( ekt * dm(i) + dektl(i) - dektl(i+1))*dmnuinv(i)
         e_totnu(i) =  e_totnu(i) + ektnu
      enddo
```

The diffusion calculation is computationally expensive, so only those zones which are in unsmooth portions of the data (and hence in which it is possible to harbor a discontinuity such as a shock) are checked, that is, wherever the unsmoothness calculation is non-zero. However, in multi dimensions, it is necessary to combine the unsmoothness calculations of the longitudinal and transverse velocities to ensure that those zones in which may roost shock fronts parallel to the direction of the pass will also be checked.

```
      do i=KSTART3 ,KEND3
         unsmooth(i) =  unsmutut(i) + unsmup(i)
      enddo
```

The zone centered Eulerian diffusion velocities are then found by a call to **PPM98_LAG_DIFUSE0_GAMMA_2D**

```
      call ifelse(ifdouble,1,d_,) ppm98_lag_difuse0_gamma_2d
     ^            (uzbb(k),    u(k-2),   u(k-1),    u(K),     u(k+1),  u(k+2),
     ^             uztt(k),  utzbb(k),  utzb(k), ut(k-2),    ut(K),   ut(k+2),
     ^             utzt(k),  utztt(k),  pzbb(k),  p(k-2),     p(K),    p(k+2),
     ^             pztt(k), rhozbb(k), rho(k-2),  rho(K), rho(k+2), rhoztt(k),
     ^             ceul(K), unsmooth(k), difuse_vel(K),  shockd(k), courno(K),
     ^              eosgam, smallu,  dtbydx,    nzones)
```

This routine uses information above and below the row being updated in order to be able to determine the position and orientation with respect to the grid of any shocks which may be present. Using these zone centered velocities, the hydrodynamic quantities are updated by **PPM98_DIFUSEFLUX_HYDRO_CART**. The reader should note that at this point the total energy does contain both the longitudinal and transverse kinetic energy terms. Therefore, the total energy diffusive fluxes returned by this subroutine are all that are needed to update the total energy zone averages. The transverse momentum diffusive fluxes are found by a call to **PPM98_DIFUSEFLUX_PASSIVE**. These diffusive fluxes are then used for the final updates of the zone averages.

```
      do 9100   i = KSTARTDF2,KENDDF2
c
c           this saves a flop or two:
          if ((dvoll(i) + dvoll(i+1)) .ne. zero)   then
             dmnew(i) = dmnu(i) + dmassl(i) - dmassl(i+1)
             rhonu(i) = dmnew(i) * dvolinv(i)
             dmnewi = one / dmnew(i)
             unu(i) = (unu(i) * dmnu(i)   +  dmoml(i) - dmoml(i+1))  *  dmnewi
             utnu(i) = (utnu(i) * dmnu(i) + dmomtl(i) - dmomtl(i+1))  *dmnewi
             e_totnu(i) = (e_totnu(i) * dmnu(i) +
     ^                                         detotl(i) - detotl(i+1)) * dmnewi
9099         continue
```

```
        endif
9101  continue
         usq = unu(i) * unu(i)
c  if ((unu(i) * unu(i)) .lt. ssmlu2)   unu(i) = 0.0
        if ( usq .lt. ssmlu2)   then
           unu(i) = 0.0
           usq = 0.0
        endif
         utsq = utnu(i) * utnu(i)
        if ( utsq .lt. ssmlu2)   then
           utnu(i) = 0.0
           utsq = 0.0
        endif
        eknu = 0.5 * (usq + utsq )
        e_intnu = mymax1 (smalle, (e_totnu(i) - eknu))
        pnu(i) = gamma1 * rhonu(i) * e_intnu
9100  continue
```

Do_PPM_LR0_3DC_GAMMA

This template extends the previous example to three dimensions.

```
      subroutine do_ppm_lr0_3dc_gamma (xl, rhozbb, rho, rhoztt, rhozmm, rhozpp,
     ^      pzbb, p, pztt, pzmm, pzpp,  u1zbb, u1, u1ztt, u1zmm, u1zpp, u2zbb,
     ^      u2zb, u2, u2zt, u2ztt, u2zmm, u2zpp, u3zmm, u3zm, u3,
     ^      u3zp, u3zpp, u3zbb, u3ztt, rhonu,pnu, u1nu, u2nu,
     ^      u3nu, eosgam, dt, smlrho, smallp, smallu, smalle, courmx,
     ^      n_real_zones_in, nbdy_set)
c
```

Now in addition to the information above and below the row being updated as in the previous example, information is needed two planes in front of the row being updated (suffix `zpp`) and two planes behind the row being updated (suffix `zmm`) for density, pressure, longitudinal velocity and both transverse velocities. Additionally, zone averages one plane in front (suffix `zp`) and one plane behind (suffix `zm`) for the second transverse velocity are required. It is assumed that the directions of the velocities form a righthand coordinate system, with `u1` (the longitudinal velocity) pointing in the direction of increasing x, the direction of the first transverse velocity (`u2`) pointing in the direction from ‘zbb’’ to ‘‘ztt’’, and the second transverse (`u3`) pointing in the direction from ‘‘zmm’’ to ‘‘zpp.’’

The calculation proceeds as with the previous example, with the notable exception that the zone centered Eulerian diffusion velocities are now found from **PPM98_LAG_DIFUSE0_GAMMA_3D**.

```
      call ifelse(ifdouble,1,d_,) ppm98_lag_difuse0_gamma_3d (
     ^    u1zbb(K),   u1(k-2),   u1(k-1),      u1(K),    u1(k+1),    u1(k+2),
     ^    u1ztt(k),  u1zmm(k),  u1zpp(k),  u2zbb(K),    u2zb(k),    u2(K-2),
     ^       u2(k),   u2(k+2),   u2zt(k),  u2ztt(K),  u2zmm(k),  u2zpp(k),
     ^    u3zbb(K),   u3(k-2),     u3(K),    u3(k+2),  u3ztt(k),  u3zmm(K),
     ^     u3zm(k),   u3zp(k),  u3zpp(k),    pzbb(K),     p(k-2),       p(K),
     ^      p(k+2),    pztt(k),   pzmm(K),    pzpp(k),  rhozbb(K),  rho(k-2),
     ^      rho(K),  rho(k+2), rhoztt(k), rhozmm(K), rhozpp(k),    ceul(k),
     ^ unsmooth(K), difuse_vel(K),  shockd(K), courno(k), eosgam, dtbydx,
     ^   smallu, nzones)
```

This routine uses information in three dimensions in order to be able to determine the position and orientation with respect to the grid of any shocks which may be present. Using these zone centered velocities, the hydrodynamic quantities are updated by **PPM98_DIFUSEFLUX_HYDRO_CART**, with the remaining calculation following the proceeding template **fBfBDo_PPM_LR0_2DC_GAMMA .**

f. hydrodynamics Summary

This section concludes with a listing of the names of available PPMLIB routines related to hydrodynamics, with a brief description of their function. Additional templates (Do_ppm's) listed here but not discussed above build upon previously detailed techniques. Their dissections are left as fertile exercises for eager readers.

Template Routines:

A. Direct Eulerial Hydrodynamics

do_ppm_de0_1dc_gamma - Computes one dimensional direct eulerain hydrodynamics on a uniform Cartesian grid with numerial difusion. Assumes a gamma law equation of state. Driver is "Driverdirecte1dg0c.EXA."

do_ppmde0_1dc_paq_gamma - one dimensional direct eulerain hydrodynamics on a uniform Cartesian grid with numerial difusion. Assumes a gamma law equation of state and provides for both mass and volume weighted passively advected quantities. Driver is "Driverdirecte1dgpaq0c.EXA."

do_ppm_de0_1dc - one dimensional direct eulerain hydrodynamics on a uniform Cartesian grid with numerial difusion. Does NOT assume a gamma law equation of state. Driver is "Driverdirecte1d0c.EXA."

do_ppm_de_1dc_gamma - Computes one dimensional direct eulerain hydrodynamics on a uniform Cartesian grid with numerial difusion. Assumes a gamma law equation of state. Driver is "Driverdirecte1dgc.EXA."

do_ppm_de_1dc_paq_gamma - one dimensional direct eulerain hydrodynamics on a uniform Cartesian grid with numerial difusion. Assumes a gamma law equation of state and provides for both mass and volume weighted passively advected quantities. Driver is "Driverdirecte1dgpaqc.EXA."

do_ppm_de_1dc - one dimensional direct eulerain hydrodynamics on a non-uniform Cartesian grid with numerial difusion. Does NOT assume a gamma law equation of state. Driver is "Driverdirecte1dc.EXA."

do_ppm_de0_2dc_gamma - two dimensional direct eulerain hydrodynamics using directional splitting on a uniform Cartesian grid with numerial difusion. Assumes a gamma law equation of state. Driver is "Driverdirect2dg0c.EXA."

do_ppmde0_2dc_paq_gamma - two dimensional direct eulerain hydrodynamics using directional splitting on a uniform Cartesian grid with numerial difusion. Contains provisions for passively advected diagnostics. Assumes a gamma law equation of state. Driver is "Driverdirect2dg0cpaq.EXA."

do_ppmde0_2dc - two dimensional direct eulerain hydrodynamics using directional splitting on a uniform Cartesian grid with numerial difusion. Does not assume a gamma law equation of state. Driver is "Driverdirect2d0C.EXA."

do_ppmde0_2dc_paq - two dimensional direct eulerain hydrodynamics using directional splitting on a uniform Cartesian grid with numerial difusion. Contains provisions for passively advected diagnostics. Does not assume a gamma law equation of state. Driver is "Driverdirect2d0cpaq.EXA."

do_ppm_de0_3dc_gamma - three dimensional direct eulerain hydrodynamics using directional splitting on auniform Cartesian grid with numerial difusion. Assumes a gamma law equation of state. Driver is "Driverdirect3dg0c.EXA."

do_ppmde0_3dc - three dimensional direct eulerain hydrodynamics using directional splitting on auniform Cartesian grid with numerial difusion. Does not assume a gamma law equation of state. Driver is "Driverdirect3d0c.EXA."

B. Lagrangian step + remap step

do_ppmlr0_1dc_gamma - one dimensional two stage (Lagrangian + remap) hydrodynamics on a uniform Cartesian grid with numerial difusion. Assumes a gamma law equation of state. Driver is "Driverlr1dg0c.EXA."

do_ppmlr_1dc_gamma - one dimensional two stage (Lagrangian + remap) hydrodynamics on a non-uniform Cartesian grid with numerial difusion. Assumes a gamma law equation of state. Driver is "Driverlr1dgc.EXA."

do_ppmlr0_1dc_paq_gamma - one dimensional two stage (Lagrangian + remap) hydrodynamics on a uniform Cartesian grid with numerial difusion. Assumes a gamma law equation of state and provides for both mass and volume weighted passively advected quantities. Driver is "Driverlr1dgpaq0c.EXA."

do_ppmlr_1dc_paq_gamma - one dimensional two stage (Lagrangian + remap) hydrodynamics on a non-uniform Cartesian grid with numerial difusion. Assumes a gamma law equation of state and provides for both mass and volume weighted passively advected quantities. Driver is "Driverlr1dgpaqc.EXA."

do_ppmlr0_1dc - one dimensional direct eulerain hydrodynamics on a uniform Cartesian grid with numerial difusion. Does NOT assume a gamma law equation of state. Driver is "Driverlr1d0c.EXA."

do_ppmlr_1dc - one dimensional direct eulerain hydrodynamics on a non-uniform Cartesian grid with numerial difusion. Does NOT assume a gamma law equation of state. Driver is "Driverlr1dc.EXA."

do_ppmlr0_2dc_gamma - two dimensional two stage (Lagrangian + remap) hydrodynamics using directional splitting on a uniform Cartesian grid with numerial difusion. Assumes a gamma law equation of state. Driver is “Driverlagremap2dg0c.EXA.”

do_ppmlr0_2dc_paq_gamma - two dimensional two stage (Lagrangian + remap) hydrodynamics using directional splitting on a uniform Cartesian grid with numerial difusion. Provision is made for passively advected diagnostics. Does not assume a gamma law equation of state. Driver is “Driverlagremap2dg0cpaq.EXA.”

do_ppmlr0_2dc - two dimensional two stage (Lagrangian + remap) hydrodynamics using directional splitting on a uniform Cartesian grid with numerial difusion. Does not assume a gamma law equation of state. Driver is “Driverlagremap2d0c.EXA.”

do_ppmlr0_2dc_paq - two dimensional two stage (Lagrangian + remap) hydrodynamics using directional splitting on a uniform Cartesian grid with numerial difusion. Provision is made for passively advected diagnostics. Does not assume a gamma law equation of state. Driver is “Driverlagremap2d0cpaq.EXA.”

do_ppmlr0_3dc_gamma - three dimensional two stage (Lagrangian + remap) hydrodynamics using directional splitting on a uniform Cartesian grid with numerial difusion. Assumes a gamma law equation of state. Driver is “Driverlr3dg0c.EXA.”

do_ppmlr0_3dc - three dimensional two stage (Lagrangian + remap) hydrodynamics using directional splitting on a uniform Cartesian grid with numerial difusion. Does not assume a gamma law equation of state. Driver is "Driverlr3d0c.EXA."

Other related templates

do_ppm_lag0_flux_gamma - computes the one-dimenaional hydrodynamical Lagrangian fluxes on a uniform Cartesian grid. It assumes a gamma law equation of state.

do_ppm_lag_flux_gamma - computes the one-dimenaional hydrodynamical Lagrangian fluxes on a non-uniform Cartesian grid. It assumes a gamma law equation of state.

do_ppm_lag0_flux - computes the one-dimenaional hydrodynamical Lagrangian fluxes on a uniform Cartesian grid. It does NOT assume a gamma law equation of state.

do_ppm_lag_flux - computes the one-dimenaional hydrodynamical Lagrangian fluxes on a non-uniform Cartesian grid. It does NOT assume a gamma law equation of state.

Hydrodynamic Modules

ppm98_deul_update_cart updates zone-averaged hydrodynamic quantities given zone interface fluxes. 1D.

ppm98_difuseflux_hydro_cart constructd zone interface centered fluxes from zone-averaged difusion velocities and zone averaged of the hydrodynamic quantities.

ppm98_e_riemann solves Riemann's shock tube problem to provide time-averaged values of density, pressure, and velocity. A generalized equation of state is used.

ppm98_e_riemann_gamma solves Riemann's shock tube problem to provide time-averaged values of density, pressure, and velocity. A generalized equation of state is assumed.

ppm98_eul_difuse0_2d Constructs zone centered Eulerian difusion velocities. Assumes 2 dimensions and a uniform Cartesian grid. No equation of state dependence is assumed.

ppm98_eul_difuse0_3d Constructs zone centered Eulerian difusion velocities. Assumes 3 dimensions and a uniform Cartesian grid. No equation of state dependence is assumed.

ppm98_eul_difuse0_gamma_2d Constructs zone centered Eulerian difusion velocities. Assumes 2 dimensions and a uniform Cartesian grid. A gamma law equation of state is assumed.

ppm98_eul_difuse0_gamma_3d Constructs zone centered Eulerian difusion velocities. Assumes 3 dimensions and a uniform Cartesian grid. A gamma law equation of state is assumed.

ppm98_eul_difuse_1d Constructs zone centered Eulerian difusion velocities. Assumes 1 dimension and a Cartesian grid, which need not be uniform. No equation of state dependence is assumed.

ppm98_eul_difuse_2d Constructs zone centered Eulerian difusion velocities. Assumes 2 dimensions and a Cartesian grid, which need not be uniform. No equation of state dependence is assumed.

ppm98_eul_difuse_3d Constructs zone centered Eulerian difusion velocities. Assumes 3 dimensions and a Cartesian grid, which need not be uniform. No equation of state dependence is assumed.

ppm98_eul_difuse_gamma_2d Constructs zone centered Eulerian difusion velocities. Assumes 2 dimensions and a Cartesian grid, which need not be uniform. A gamma law equation of state is assumed.

ppm98_eul_difuse_gamma_3d Constructs zone centered Eulerian difusion velocities. Assumes 3 dimensions and a Cartesian grid, which need not be uniform. A gamma law equation of state is assumed.

ppm98_euler0_domain_avg Interpolates density, pressure and longitudinal velocity. Determines spatial averages of pressure and longitudinal velocity over the + and - Eulerian domains of dependence. Assumes a uniform grid for efficiency.

ppm98_euler_domain_avg Interpolates density, pressure and longitudinal velocity. Determines spatial averages of pressure and longitudinal velocity over the + and - Eulerian domains of dependence.

ppm98_interp0_rho Computed piecewise parabolic interpolations of density based on zone-averaged data. With monotonicity constraints and contact-discontinuity detection. For use on uniform grids.

ppm98_interp0_up Computed piecewise parabolic interpolations of pressure and velocity based on zone-averaged data. With monotonicity constraints and discontinuity detection. For use on uniform grids.

ppm98_interp_rho Computed piecewise parabolic interpolations of density based on zone-averaged data. With monotonicity constraints and contact-discontinuity detection. For use on non-uniform grids.

ppm98_interp_up Computed piecewise parabolic interpolations of pressure and velocity based on zone-averaged data. With monotonicity constraints and discontinuity detection. For use on non-uniform grids.

ppm98_l_riemann solves Riemann's shock tube problem to provide time-averaged values of pressure and velocity. A generalized equation of state is used.

ppm98_l_riemann_gamma solves Riemann's shock tube problem to provide time-averaged values of pressure and velocity. A gamma law equation of state is assumed.

ppm98_lag0_domain_avg Determines piecewise parabolic interpolations for pressure and longitudinal velocity, and finds averages of pressure and velocity in the plus and minus Lagrangian domains of dependence for each lefthand zone interface. For uniform grids.

ppm98_lag_domain_avg Determines piecewise parabolic interpolations for pressure and longitudinal velocity, and finds averages of pressure and velocity in the plus and minus Lagrangian domains of dependence for each lefthand zone interface. For non-uniform grids.

ppm98_lag_difuse0_2d Constructs zone centered Eulerian difusion velocities. Assumes 2 dimensions and a uniform Cartesian grid. No equation of state dependence is assumed. A special Courant (stability) calculation is performed to ensure that no wave traverses more than one Lagrangian zone.

ppm98_lag_difuse0_3d Constructs zone centered Eulerian difusion velocities. Assumes 3 dimensions and a uniform Cartesian grid. No equation of state dependence is assumed. A special Courant (stability) calculation is performed to ensure that no wave traverses more than one Lagrangian zone.

ppm98_lag_difuse0_gamma_2d Constructs zone centered Eulerian difusion velocities. Assumes 2 dimensions and a uniform Cartesian grid. Assumes a gamma law equation of state. A special Courant (stability) calculation is performed to ensure that no wave traverses more than one Lagrangian zone.

ppm98_lag_difuse0_gamma_3d Constructs zone centered Eulerian difusion velocities. Assumes 3 dimensions and a uniform Cartesian grid. Assumes a gamma law equation of state. A special Courant (stability) calculation is performed to ensure that no wave traverses more than one Lagrangian zone.

ppm98_lag_difuse_1d Constructs zone centered Eulerian difusion velocities. Assumes 1 dimensions and a Cartesian grid which need not be uniform. No equation of state dependence is assumed. A special Courant (stability) calculation is performed to ensure that no wave traverses more than one Lagrangian zone.

ppm98_lag_update_cart performs a Lagrangian update of the zone-averaged quantities given Lagrangian zone interface fluxes (pressure and velocity) in Cartesian coordinates. 1D.

ppm98_map_flux_1d Computes the Conservative zone interface fluxes (mass, momentum and energy) necessary to map hydrodynamic quantities from one grid to another. based on zone averaged data. Assumes a generalized equation of state. 1D

ppm98_map_flux_gamma_1d Computes the Conservative zone interface fluxes (mass, momentum and energy) necessary to map hydrodynamic quantities from one grid to another. based on zone averaged data. Assumes a gamma law equation of state. 1D

ppm98_map_flux_gamma_2d Computes the Conservative zone interface fluxes (mass, momentum and energy) necessary to map hydrodynamic quantities from one grid to another. based on zone averaged data. Assumes a gamma law equation of state. 2D

ppm98_map_flux_gamma_3d Computes the Conservative zone interface fluxes (mass, momentum and energy) necessary to map hydrodynamic quantities from one grid to another. based on zone averaged data. Assumes a gamma law equation of state. 3D

ppm98_map_update_1d Updates the zone averaged conserved hydrodynamic quantities given the advective fluxes. 1D

ppm98_map_update_2d Updates the zone averaged conserved hydrodynamic quantities given the advective fluxes. 2D

ppm98_map_update_3d Updates the zone averaged conserved hydrodynamic quantities given the advective fluxes. 3D

Modules for Passively advected Quantities

ppm98_advect_passive performs an entire remap of a passively advected quantity on a nonuniform Cartesian grid.

ppm98_advect_passive0 performs an entire remap of a passively advected quantity on a uniform Cartesian grid.

ppm98_difuseflux_passive constructd zone interface centered passive fluxes from zone interface cenetered difusion fluxes.

ppm98_passive_flux determines piecewise parabolic interpolations zone-averages of a passive quantity and determines the fluxes necessary to map it from one grid to another. For non-uniform grids.

ppm98_passive_flux0 determines piecewise parabolic interpolations zone-averages of a passive quantity and determines the fluxes necessary to map it from one grid to another. For uniform grids.