README File: RIEMANN Code for Euler Flow at 2nd Order

Written by Dinshaw S. Balsara (dbalsara@nd.edu)

Introduction

This is the README Document for the 2nd order accurate RIEMANN code for Euler flow by Dinshaw S. Balsara. The name celebrates the seminal contributions by the mathematician Bernhard Riemann to the theory of hyperbolic PDEs (Riemann 1860). Though second order for the sake of simplicity, the code exemplifies many frontline algorithmic ideas. The same algorithmic ideas are readily extended in our higher order codes. For pedagogic reasons, it might help to start with a 2nd order code and move up to a higher order code that follows the same floor-plan. The algorithmic ideas will be detailed below. The code is organized as a set of few Fortran subroutines, each of which does a functional task.

The code is based on the philosophy that simplicity, taken to its limit, becomes elegance. Astrophysical codes are very easy to understand if one takes an algorithms-first approach, because most such codes are based on a very well-known algorithmic philosophy. Unfortunately, the trend in astrophysics is to start with a very large and complex code and treat it like a black-box which one tries to understand "from the outside-in". This can become challenging. The present code is best understood "from the inside-out"; i.e., as an instantiation of a certain algorithmic philosophy. To help in that process, the reader is invited to download and learn from a set of lectures that are freely available on the web. Please see http://www.nd.edu/~dbalsara/Numerical-PDE-Course. We do request that using the code, or any part thereof, should result in citations to the original papers (as is customary in good scientific practice).

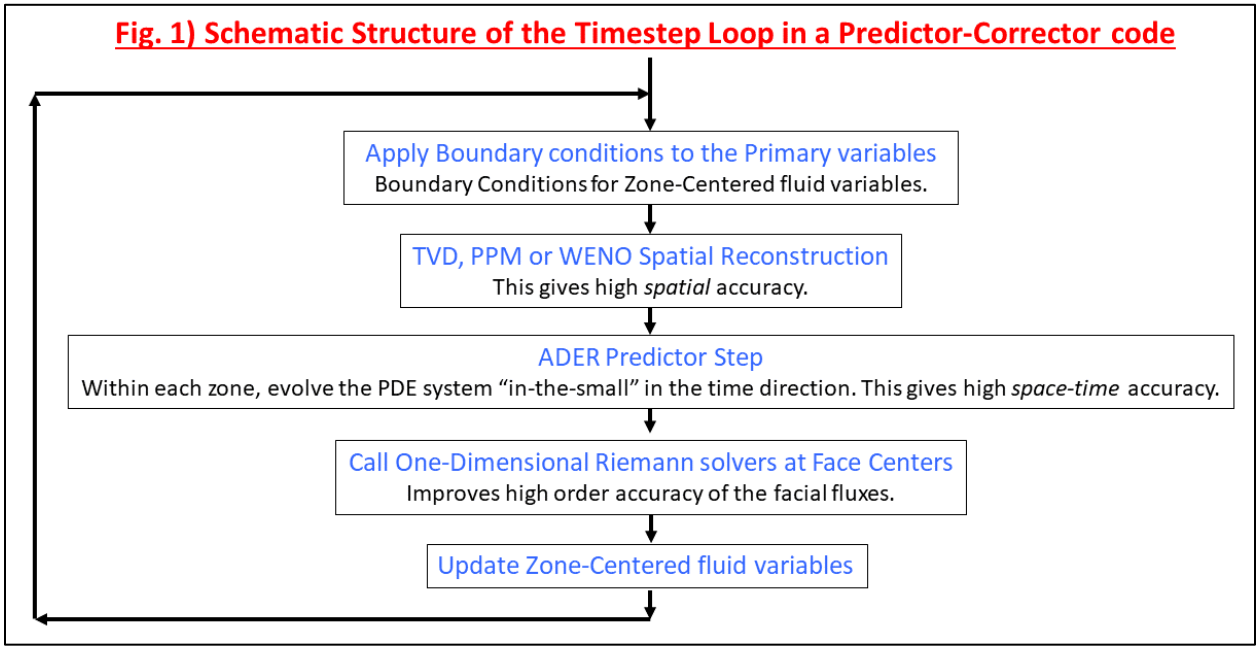
The RIEMANN code is self-documenting – In general, most types of information that you need will be documented right in the code and right where you need it. The headers of most subroutines describe in great detail the algorithms that are contained in that subroutine. Even so, the user needs some basic orientation to get started. For that reason, the next several sections describe the following pieces of information. Section I describes the algorithmic structure of the code. Section II deals with making a quick start by running your first simulation and visualizing

it. Section III describes the file structure and lists the subroutines in each of the files along with information on what they do. Section IV catalogues the variables that hold the main simulation data. Section V describes the structure of the main code and explains how one should set up an application and run it. Papers that have been referenced in this readme file are also listed in detail in the references section.

Good Luck, and Happy Computing!

I) Algorithmic Structure of the Code

The present code is a higher order Godunov code (Godunov 1959). The code is based on a ***predictor-corrector philosophy***. Such a philosophy for the time-evolution of hyperbolic systems was first initiated in van Leer (1977, 1979), Colella & Woodward (1984), Colella (1985) at second order. That algorithmic outlook has now been polished and made available at all orders via Harten *et al*. (1986), Jiang & Shu (1996), Balsara & Shu (2000), Dumbser *et al*. (2008), Balsara *et al*. (2009, 2013). Let us very, very briefly explore the algorithmic content that goes into such a code. As you read the subsequent paragraphs, please keep glancing at Fig. 1 from time to time.



The first step is to make a TVD (Harten 1983) or PPM (Colella & Woodward 1984, Colella & Sekora 2008, McCorquodale & Colella 2011) or WENO (Jiang & Shu 1996, Balsara & Shu

2000, Dumbser & Käser 2006, Balsara *et al*. 2009) ***spatial reconstruction***. (Please note that in order to have a meaningful reconstruction at the boundaries of the mesh, one has to apply boundary conditions before the reconstruction step. But the kinds of boundary conditions that one chooses are limited only by the computationalist's creativity.) In simple terms, spatial reconstruction just means endowing some reasonable sub-structure to the slab of fluid within a zone. This substructure is evaluated at a given zone by analyzing flow variables in its neighboring zones. TVD (Total Variation Diminishing) reconstructions yield second order accuracy. PPM (Piecewise Parabolic Method) usually includes several ingredients of third order accuracy. WENO (Weighted Essentially Non Oscillatory) reconstruction can provide arbitrary amounts of spatial accuracy, provided the stencil is suitably large. But the basic task of reconstruction (also known as limiting) is to make a non-oscillatory profile within each zone that is as high order accurate as possible. The non-oscillatory profile endows sub-structure to the slab of fluid within a zone without introducing new, spurious extrema. In this code, TVD and WENO reconstruction strategies have been used. PPM is used in the higher order code that is the analogue of the 2^{nd} order code described here.

Having worked hard to endow the slabs of fluid with a meaningful internal sub-structure, one desires to know how the PDE system will evolve within a zone for a small amount of time. This is the famous Cauchy problem for a hyperbolic PDE which says, in pedestrian terms, that given a smooth enough initial conditions for a PDE in space we can always evolve it for some interval in time. Observe from the formal structure of a hyperbolic PDE -- $\partial_t U + \partial_x F(U) = 0$ -- that if the spatial variation is known, the temporal evolution can be predicted, at least for a small amount of time. This "evolution-in-the-small" is what goes on in the predictor step. I.e., since we know the spatial gradients of a flow variable, via spatial reconstruction within a zone, we understand that this will also provoke a temporal evolution of the flow variables. This is the predictor step. Ideally, we want the time-accuracy of the predictor step to match the spatial accuracy of the reconstruction. At second order, this is easily done. At third and higher orders, this is a more sophisticated undertaking. A formal way of doing this, which always works at all orders, is given by the ***ADER timestepping strategy*** (Dumbser *et al*. 2008, Balsara *et al*. 2009, 2013). ADER stands for Arbitrary DERivatives in space and time. Basically, we are taking the spatially reconstructed solution within a zone and evolving it *within the same zone* in space-time.

For the sake of historical accuracy, it is worth mentioning that early ADER schemes were based on the generalized Riemann problem (Titarev & Toro 2002, 2005, Toro & Titarev 2002). But that construction proved to be unwieldy, giving rise to the more modern formulations described in the previous paragraphs.

Ultimately, the slabs of fluid need to interact with one another. This is needed if we have to work out the flux of fluid that flows from one zone into the neighboring zone. The numerically correct fluxes have to incorporate the physically consistent direction in which the flow occurs, and that task falls to the ***Riemann solver***. (Picking out the right direction from which the "wind" is blowing is known as upwinding; and this is what the Riemann solver does for us.) The one-dimensional Riemann solver sits at the zone boundary between two zones and uses the states from either side of the zone boundary. One can think of the one-dimensional Riemann solver as being just a machine that takes in two states (one from either side of a zone boundary) and produces a properly upwinded numerical flux. For very large systems, like Euler equations or MHD, it is very inefficient to use an exact Riemann solver. So one uses an approximate Riemann solver. Numerous good one-dimensional Riemann solvers have been designed for Euler equations as well as MHD. An incomplete list follows. There are linearized Riemann solvers for MHD (Brio & Wu 1988, Roe & Balsara 1996, Cargo & Gallice 1997, Balsara 1998a,b). Linearized Riemann solvers do not have a positivity property. I.e., the resolved state, from which the numerical flux is calculated, may not have positive density and pressure. HLL Riemann solvers have this positivity property. The HLL Riemann solver can be quite dissipative, and this has spawned several variants of the HLL Riemann solver that try to reduce the dissipation. Consequently, we also have HLLC Riemann solvers for Euler equations as well as MHD (Gurski 2005, Li 2006). The HLLC Riemann solver restores the contact discontinuity. Recent work by Dumbser & Balsara (2016) has resulted in an HLLEM Riemann solver which can restore all the waves in the MHD system while retaining many of the good positivity properties of the HLL Riemann solver. (We also refer to it as the HLLI Riemann solver because it is considerably different in scope and function from the original HLLEM Riemann solver. For example, the new HLLI Riemann solver can accommodate multiple **I**ntermediate waves, which accounts for the "I" in HLLI. The original HLLEM only accounted for one contact discontinuity.) The code works with linearized Riemann solvers, HLL, HLLC and also the HLLI Riemann solver. Recent versions of the code give preference to the HLLI Riemann solver

because of its superb stability, efficiency, versatility, accuracy with different wave families and good positivity properties.

Once the fluxes are obtained at the faces of the mesh, it is easy to update the zone-centered variables in conservative fashion. Since attention has been paid to the space-time accuracy, this will be a high order update in space and time. This completes the algorithmic description of the RIEMANN code for astrophysical Euler flow.

Any code should be capable of operating safely. I.e. variables that are intrinsically positive, like the density and pressure, should be kept positive by the code as much as possible. This is called the ***physical realizability property***, and a numerical code should respect physical realizability. Algorithms that ensure physical realizability have been described in Balsara & Spicer (1999b) and more recently in improved form in Balsara (2012b) and Balsara and Kim (2016). Those algorithms are used in the code.

II) Quick Start – Running your First Simulation and Visualizing it

The first step is always the hardest; so we have made it as simple as possible. Try to start in a Linux environment. The distribution already comes with a README file (this file). All the lines for compiling the code with various compilers are available in the file "sccompile.euler". Compile lines for Intel, PGI and GNU compilers are included. The GNU compilation is very fast and usually gives performance that is within 10% to 20% of the Intel compiler. (The code compiles serially or with OpenMP. It also has bindings with MPI-3 for up to PetaScale performance, though they may or may not be available with your particular distribution.) There is also a "Makefile" which works with the public domain GNU compiler. The MAIN code is in "Riemann_Euler_src". IDL-based plotting routines are available (type "ls *.pro" to see the .pro files associated with IDL). GnuPlot-based plotting routines are also available in the distribution (look for the files "onedplot.sh" and "twodplot.sh"; open them and modify them as you wish). In general, IDL runs faster than GnuPlot and generates better-looking images, but GnuPlot is freeware. We suggest copying the distribution over to a scratch folder before running your first simulation in that same scratch folder. That way, you retain an original copy of the distribution.

Like any code, the code wakes up so that it does a default start-up problem. In this case, it does a 3D hydrodynamic blast problem on a three-dimensional $96 \times 96 \times 96$ zone mesh. This should be easy to run on a sufficiently powerful single core Linux workstation. (It is possible that you are not set up to visualize 3D data. In that case, it is best to visualize a slice of data from the midplane of the 3D simulation. To do that, just open file "Riemann_Euler_src" and look for all the instances of "unit = 99". Just change "izz = 1, iz1" to "izz = iz1/2, iz1/2" in a few places where the image files are being written out. Now you will get the midplane slice of the 3D problem for visualization.)

It is also possible that you are running the code for the first time on a PC or a computer that is not so powerful. In that case, it might be appropriate to do a 2D version of the same problem. To do that, set "iz1 = 1" and "ioffz = 0" in the first "PARAMETER" statement in "Riemann_Euler_src". You have just turned the 3D problem into a small 2D problem.

Type "make" and you will have an executable called "xeuler". Type "nohup ./xeuler &" to run the executable. It will take a few minutes to run through. After it has run, modify "colorslice_jpg.pro" to visualize your images with IDL. Alternatively, modify "twodplot.sh" and visualize your images using gnuplot.

Visualizing with IDL:- Open the file "colorslice_jpg.pro" and do the following steps:-

1) Set "nx, ny" to the dimensions of your images. Set "nplanes" to the number of images in each family that you want to visualize. For example, if your images run from "rhoa0001" to "rhoa0015" then set "nplanes = 15".

2) "namearray" contains the names of all the image files that you might want to visualize. You might not want to visualize all the variables. Modify "namearray" and the extents of the loop "for index = 0, 4" to visualize only the variables of interest. Then close "colorslice_jpg.pro"

3) Type "module load idl", then type "idl" to get IDL started.

4) At the idl> prompt, type "colorslice_jpg". Then type "exit" to get out of IDL.

5) There is also a "onedplot.pro" file for one-dimensional plots. It works pretty much the same way.

Visualizing with GnuPlot:- Open the file "twodplot.sh" and do the following steps:-

1) Set “nx, ny” to the dimensions of your images. Set “xmin, xmax, ymin, ymax” to the physical extent of the simulation. Set “nimages” to the number of images in each family that you want to visualize. For example, if your images run from “rhoa0001” to “rhoa0015” then set “nimages = 15”.

2) “datafilename” contains a list of image file names. Retain the ones you want to visualize. Then close “twodplot.sh”.

3) Type “./twodplot.sh”. You should get several image files.

4) There is also a “onedplot.sh” file for one-dimensional plots. It works pretty much the same way.

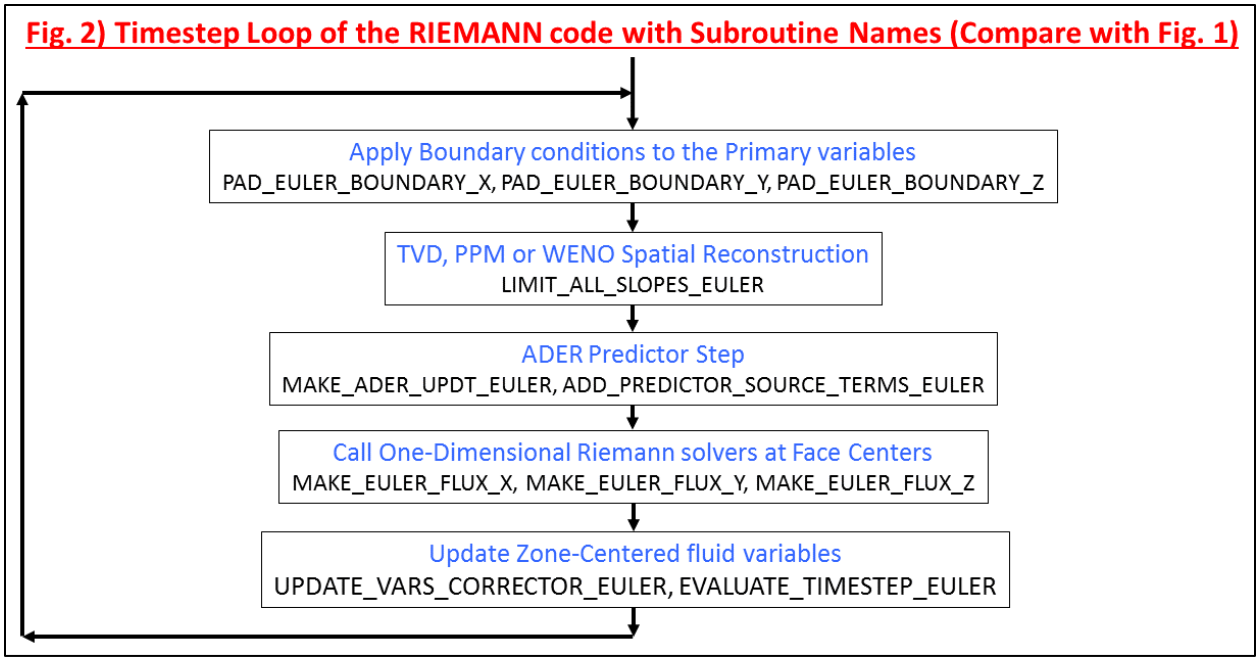
If you see several *.jpg or *.jpeg files, visualize them. You should see a blast problem. Congratulations! You have successfully run your first simulation with the code! (Please note that color images with GnuPlot don’t look quite as good as color images from IDL, so please use IDL if you at all can.)

III) File Structure – Listing of Subroutines in Each of the Files and What they Do

Fig 1 provides a schematic structure of any predictor-corrector code. But we have not described how those tasks are instantiated via subroutines in the RIEMANN code. This is done in Fig. 2, which should be cross-compared with Fig. 1. It clearly shows which named subroutine in the RIEMANN code carries out which algorithmic task in the predictor-corrector-based timestep loop.

In the remainder of this section, we describe the file names and the subroutines that they contain and/or the tasks that they do.

Fig. 2) Timestep Loop of the RIEMANN code with Subroutine Names (Compare with Fig. 1)



III.a) The "directives" file:- This is the file that you go to before starting any simulation. There are overall variables (known as "directives" even in computer science lingo) which control how the code behaves. In other words, the code undergoes conditional compilation based on your choice of directives. The directives are explained in detail in the directives file. To give several instances, this is where you choose whether the code makes a fresh start on a new simulation or restarts a pre-existing one. (Focus on the directive "RESTART" to see that the code compiles a little differently when it is making a fresh start v/s when it is restarting. A small file called "riemann.in" is written out by the code to help it know where to restart; so if you plan to restart a simulation, please save that file along with any checkpoint/dump files). This is also where you choose the kind of reconstruction you want to carry out. (Some forms of reconstruction may be inexpensive, but less sophisticated. Other forms or reconstruction may cost more, but are more sophisticated.) This is also where you choose whether you want to enforce positivity considerations on the code – i.e. whether you want the code to compulsively try and keep densities and pressures positive. This is also where you make your choice of Riemann solver (though we might default to HLLI all the time, we like it that much!). Any change to the directives file should be followed by a *complete recompilation of the code* if the requested changes are to take hold!

III.b) The "riemann.com" file:- This file again carries a lot of the information that is seen by practically every subroutine in the code. It should almost never be changed by the end-user.

III.c) The "Riemann_Euler_src" file:- This is probably the most important file. It contains several important subroutines. Please read the narrative below while following along via Figs. 1 and 2. That will give maximum insight as to what the subroutines do. The subroutines are:

A) "PROGRAM RIEMANN_EULER" which is the MAIN code. We will have a lot more to say about this program unit in Section V (i.e., after we have talked at length about the individual subroutines as well as the important variables in the code). Suffice it to say that for most applications, the user will be working with it and modifying it to her/his needs.

B) SUBROUTINES "PAD_EULER_BOUNDARY_X, PAD_EULER_BOUNDARY_Y, PAD_EULER_BOUNDARY_Z". This set of three subroutines apply the boundary conditions in the logical x, y, and z-directions in the structured mesh code. While the code is logically Cartesian, it can accommodate other orthogonal meshes like cylindrical and spherical. However, the default is Cartesian x,y,z. (The choice of coordinate system is made via the variable "igeom". "igeom = 1" for Cartesian. "igeom = 2" for cylindrical. "igeom = 3" for spherical. When "igeom = 2", the x-direction is "r", the y-direction is "phi", the z-direction is "z". When "igeom = 3", the x-direction is "r", the y-direction is "theta", the z-direction is "phi".) These subroutines also connect very strongly to the 3D arrays "x_indx_limits, y_indx_limits, z_indx_limits" which control the dynamically active portion of the mesh. The boundary conditions are also communicated to these subroutines via "bcarr_x, bcarr_y, bcarr_z" which specify what type of boundaries we want at the lower and upper x-boundaries, lower and upper y-boundaries and lower and upper z-boundaries respectively. Certain boundary conditions are built in. For example, "1" means inflow boundary condition (though the choice of inflow variables has to be explicitly specified in the PAD_EULER_BOUNDARY_? Routines). Likewise, "2" means continuative boundary condition. Similarly, "3" means reflective boundary condition. In serial setting "4" means periodic boundary conditions. When OpenMP parallelism is used exclusively, "4" can also be used for periodic boundary conditions. However, when MPI-based parallelism is used, please do not use "4" to specify the boundary conditions. The usage of the array sets "x_indx_limits, y_indx_limits, z_indx_limits" and "bcarr_x, bcarr_y, bcarr_z" to specify the boundary conditions is given in the main code.

C) SUBROUTINE "LIMIT_ALL_SLOPES_EULER". This is the main spatial reconstruction routine. Several choices of reconstruction have been built into this subroutine. The conserved fluid

variable “u” enter this subroutine without reconstruction and exit with appropriate spatial reconstruction. This subroutine performs several tasks and we list them in sequence:-

i) The first task in this subroutine consists of extracting the primitive variables “rhogr, prsgr, vlxgr, …etc.” from the conserved variable “u”. The primitive variables are used in many different ways throughout the code. They are also the variables that are used for visualization.

ii) From these, the divergence of the flow, “divvelgr” and a characteristic signal speed “msonicgr” are built at each zone. These two arrays are then used to design a “flattengr” variable within each zone which tells us how generously or cautiously the reconstruction can be carried out. (If there are no local shocks, we go for the best reconstruction that has been requested in the directives file. If local shocks exist, the flattener can be used to tone down the moments that are reconstructed.)

iii) The spatial limiting/reconstruction is then carried out for all zone-centered and face-centered variables. If characteristic variables are called for, the characteristic limiting is done via calling a 1D subroutine called “LIMIT_1D_EULER_CONS”. We start with “u (1: ix1, 1: iy1, 1: iz1, :, 1)” which has all the mean values of the conserved flow variables. At second order, the above variable is used to reconstruct the spatial gradients. “u (1: ix1, 1: iy1, 1: iz1, :, 2)” has the x-slopes of the flow variables; “u (1: ix1, 1: iy1, 1: iz1, :, 3)” has the y-slopes of the flow variables; “u (1: ix1, 1: iy1, 1: iz1, :, 4)” has the z-slopes of the flow variables.

iv) Global min and max values for various diagnostic quantities are also built here and written out if that is called for. (Under MPI parallelism, the global min and max values have to be obtained via global reduction, of course.)

D) SUBROUTINE “MAKE_ADER_UPDT_EULER”. This is the ADER space-time evolution step. The variables and their spatial gradients are stored in “u (1: ix1, 1: iy1, 1: iz1, :, 1:4)”. In other words, “u (1: ix1, 1: iy1, 1: iz1, :, 1)” has all the mean values of the flow variables; “u (1: ix1, 1: iy1, 1: iz1, :, 2)” has the x-slopes of the flow variables; “u (1: ix1, 1: iy1, 1: iz1, :, 3)” has the y-slopes of the flow variables; “u (1: ix1, 1: iy1, 1: iz1, :, 4)” has the z-slopes of the flow variables. We now wish to use the spatial gradients to obtain the time rate of change of those variables and store it in “u (1: ix1, 1: iy1, 1: iz1, :, 5)”. This information pertaining to the space-time evolution of the PDE is built in the present subroutine. Each zone only looks at its own gradients within the zone. Therefore, this is only an “in-the-small” evolution of the PDE. The

above narrative only describes the second order case because this is easiest to describe. For better than second order, we will even have to build the higher moments in time using the higher order spatial moments that are provided by the spatial reconstruction. Realistically, we use "expand_by_one_zone = 1" which builds all this space-time information for one layer of zones that go beyond the physical domain. This is useful in the subsequent two sets of subroutines where fluxes and electric fields are calculated.

E) SUBROUTINES "MAKE_EULER_FLUX_X, MAKE_EULER_FLUX_Y, MAKE_EULER_FLUX_Z". These subroutines just call a one-dimensional Riemann solver which is evaluated at the face centers of the mesh. This is done in a fully traditional Godunov scheme fashion. The one-dimensional Riemann solver that is called by this routine is called "EULER_RIEM_HLL". The same one-dimensional Riemann solver can be used at x-, y- and z-faces because we use cyclic rotations of the variables that are sent into and out of the interface of the one-dimensional Riemann solver.

F) SUBROUTINE "ADD_PREDICTOR_SOURCE_TERMS_EULER". Adds in geometric source terms to the time rate of change "u (1: ix1, 1: iy1, 1: iz1, :, 5)".

G) SUBROUTINE "UPDATE_VARS_CORRECTOR_EULER". Does the final update step using the fluxes from the corrector step.

H) SUBROUTINE "EVALUATE_TIMESTEP_EULER". Evaluates the new timestep.

III.d) The "Euler_RS_src" file:- It contains the 1D HLLI Riemann solver called "EULER_RIEM_HLLEM_DB" and other supporting subroutines.

III.e) The "Eos_Euler_src" file:- It contains the equation of state routines. They can be modified to use different equations of state.

III.f) The "Applications_Euler_src" file:- It contains numerous interesting hydrodynamics applications generator subroutines. The subroutine notes for each of the set-up subroutines contain sufficient information for setting up the application. All application-generators have subroutine names that begin with "INIT_" to show that they are application initialization routines. These initialization routines all (usually) have the same subroutine interface and are written so that they can also do patch-based initialization in an MPI setting. In some instances, an application might

need additional routines and those routines follow immediately after the "INIT_" routine. Do browse this file when thinking about how to set up newer applications.

IV) Variables that hold the Main Data – Setting up and Running a Simulation

The code can be run as a one-dimensional, two-dimensional or three-dimensional code. If we are simulating a 2D or 2.5D problem, we can turn off the third dimension in the following way. Imagine that the 2D or 2.5D problem has to be set up in the xy-plane. We can then set "iz1 = 1" (only one zone in the z-direction) and "ioffz = 0" (no offsets in the z-direction).

All the big static arrays are described in detail at the beginning of "PROGRAM RIEMANN_EULER". However, it would not hurt to give some supplemental information here.

A) "ix1, iy1, iz1, ioffx, ioffy, ioffz". These specify the extent of the logically Cartesian mesh. The range of dynamically active zone-centered variables is given by "(1: ix1, 1: iy1, 1: iz1)". The "ioffx, ioffy, ioffz" specify the offsets in each direction so that the real extent of zone-centered indexing, with ghost zones, is "(1 - ioffx: ix1 + ioffx, 1 - ioffy: iy1 + ioffy, 1 - ioffz: iz1 + ioffz)". To turn a dimension off, say to turn off the z-dimension, set "iz1 = 1" and "ioffz = 0". For dynamically active dimensions, "ioffx, ioffy, ioffz" are set to the same number and that number is determined by the size of the stencil. Second and third order, in three dimensions, can be safely accommodated by setting "ioffx = ioffy = ioffz = 4". For fourth order, we have to enlarge the stencil, so that we set "ioffx = ioffy = ioffz = 6". The second order stencils look like a cross-like shape. The higher order stencils include zones in other (diagonal) directions on the mesh.

B) "n_cc_components_euler". Number of flow components. 5 for Euler flow without species fractions. If some number of species fractions are used, this number increases. If we are doing a multispecies calculation, set "NFLUID_EULER" to the number of extra species.

C) "n_cc_modes_euler". This is the number of space-time modes. For 2^{nd} order code this is 5. I.e. we want the variable, its three spatial gradients and its linear evolution in time. For higher order codes, this number increases of course.

D) "u_euler" is the most important array. It stores the vector of conserved variables at each zone, as well as its moments in space and time. It is a five dimensional array. The first three indices of

this multidimensional array pertain to the x-, y- and z-directions. The fourth index pertains to the flow variables. The fourth index ranges from 1 to 5 in Euler flow. The vector of variables is (density, x-momentum density, y-momentum density, z-momentum density, energy density)^T . Of course, multispecies flow can be accommodated by increasing "n_cc_components_euler". The fifth index of "u" ranges from 1 to 5 at second order. At higher orders, the fifth index has an even larger range because there are more space-time moments to keep track of.

E) "entropy_euler_gr" is an entropy density. It is an advected variable and can be useful for pressure positivity.

F) "flux_euler_x, flux_euler_y, flux_euler_z" hold the face-centered fluid fluxes. They are very useful for updating the zone-centered flow variables.

G) "prs_flux_euler_x, prs_flux_euler_y, prs_flux_euler_z" hold the flux terms specifically stemming from the pressure. Never needed on Cartesian meshes. However, in cylindrical and spherical geometries, the pressure flux and advected flux need to be treated differently at a coordinate singularity. For that reason, in other geometries, we split off the pressure part from the momentum flux.

H) "entropy_flux_euler_x, entropy_flux_euler_y, entropy_flux_euler_z" holds the fluxes associated with the entropy density. This is useful for problems where pressure positivity might become an issue.

I) "rho_euler_gr, prs_euler_gr, …" are just the arrays associated with the primitive variables on the mesh. The nomenclature of these variables is self-explanatory. They can be used for imaging and as auxiliary (helping) variables.

J) "x_indx_limits, bcarr_x". These two arrays control the boundary conditions that get imposed in "PAD_EULER_BOUNDARY_X". They specify the lower and upper x-indices of the dynamically active part of the computational domain as well the boundary conditions that are to be imposed at the lower and upper x-boundaries of the domain. For zones with y- and z-indices given by (iyy, izz) the dynamically active region lies between "x_indx_limits (1, iyy, izz)" and "x_indx_limits (2, iyy, izz)". This allows us to set up non-cubical domains, if we wish. The boundary conditions at either end of this dynamically active region are specified by the values in "bcarr_x (1, iyy, izz)" and "bcarr_x (2, iyy, izz)". The integers that specify the different types of

boundary conditions are as follows:- “1” for inflow boundary condition; though the specific values have to be typed in. “2” for continuative boundary condition. “3” for reflective boundary condition. “4” for periodic boundary condition in a serial/OpenMP setting.

K) “y_indx_limits, bcarr_y”. These two arrays control the boundary conditions that get imposed in “PAD_EULER_BOUNDARY_Y”. They specify the lower and upper y-indices of the dynamically active part of the computational domain as well the boundary conditions that are to be imposed at the lower and upper y-boundaries of the domain. For zones with x- and z-indices given by (ixx, izz) the dynamically active region lies between “y_indx_limits (ixx, 1, izz)” and “y_indx_limits (ixx, 2, izz)”. This allows us to set up non-cubical domains, if we wish. The boundary conditions at either end of this dynamically active region are specified by the values in “bcarr_y (ixx, 1, izz)” and “bcarr_y (ixx, 2, izz)”.

L) “z_indx_limits, bcarr_z”. These two arrays control the boundary conditions that get imposed in “PAD_EULER_BOUNDARY_Z”. They specify the lower and upper z-indices of the dynamically active part of the computational domain as well the boundary conditions that are to be imposed at the lower and upper z-boundaries of the domain. For zones with x- and y-indices given by (ixx, iyy) the dynamically active region lies between “z_indx_limits (ixx, iyy, 1)” and “z_indx_limits (ixx, iyy, 2)”. This allows us to set up non-cubical domains, if we wish. The boundary conditions at either end of this dynamically active region are specified by the values in “bcarr_z (ixx, iyy, 1)” and “bcarr_z (ixx, iyy, 2)”.

M) “xb_save, dx_save, xc_save”. The 1D array “xb_save” specifies the boundaries of the zones in the x-direction. The 1D array “dx_save” specifies the zone size in the x-direction. The 1D array “xc_save” specifies the centers of the zones in the x-direction.

N) “yb_save, dy_save, yc_save, zb_save, dz_save, zc_save” Analogous arrays to the previous 1D arrays in the y- and z-directions.

V) Structure of the MAIN Code

Imitation is the sincerest form of flattery; and also the best way to learn about a numerical code. In the distribution, we have included several example applications. Please try to run several

of these before embarking on your own applications. The included applications will give you some idea as to how the code is to be set up and run. The narrative in this section maps out the mental decision-making that goes into setting up and running a simulation. In doing so, it also explains the structure of the code.

Compilation:- Compilation lines for various compilers are given in "**sccompile.euler**" or use the "**Makefile**". The executable name is "**xeuler**". Run interactively with "**nohup ./xeuler &**" on any Linux machine; or use appropriate batch file at your installation.

Initialization:- Before starting a simulation, it is best to visit the "**directives**" file and decide upon the kind of algorithms that will be used. Also decide which application you want to run by visiting the "**Applications_Euler_src**" file and make sure that the desired application is called in "PROGRAM RIEMANN_EULER". Make sure that the appropriate dimensioning of the problem is specified by choosing "**ix1, iy1, iz1, ioffx, ioffy, ioffz**".

Main Variables:- The code begins with a declaration of variables and the most important multidimensional arrays that store the flow data. Notes on the usage of those variables/arrays are also provided right next to the variables.

Standard Input/Output:- Standard output is to "unit = 6", which is a file called "**riemann.out**". During each dump, the code also rewrites a file called "**riemann.in**" which will help with restart. Thus "riemann.in" and "riemann.out" should be saved when archiving a code for future restart. On each restart, "riemann.out" will be overwritten, so do save prior versions if needed.

Dump and Image File Names:- The name of the dump/checkpoint file defaults to "dump0001" and the code increments dump file names automatically. The file name has the general format "????0001", so up to 9999 dump files can be written per simulation, which is plenty. Dump files are written every "**ndumpstep**" timesteps. Imaging file names default to "rhoa0001, prsa0001, vlxa0001, vlya0001, vlza0001" and are incremented automatically as the images are written out. Species fractions can also be included, if they are present.

To restart the code from a pre-existing dump/checkpoint file, please look up the variable "RESTART" in the "directives" file and also please make sure that you have saved the "riemann.in" file from the dump file that you are restarting from.

Timestep Control:- The variable "**time**" keeps track of the simulation time. The current timestep is "**dtcur**", which auto-adjusts as the simulation progresses. The timestep is regulated by the CFL number, which is stored in "**cfl_coef**". The simulation is run till it reaches a simulated time of "**timestop**". Images are written after every interval of "**dtimage**" of simulation time. The variable "**timeimage**" is the last time when an image was written out. All these variables are set soon after the declaration of the main data arrays in the code.

Other Variables:- Floor values for density and pressure are "**rhofloor, prsfloor**". The polytropic index is "**gamma_euler**". There are other variables that can also be reset; and they are also given at the beginning of the code. However, it is recommended that they should not be changed much by the end-user. All variables that are set are also mirrored back by the code at the top of the file "riemann.out" so that the user has a good idea of the kind of run s/he has asked for. I.e., the code tells you back the algorithms and setting that you chose. Some of those settings are, of course, controlled by the "directives" file too. The min and max of the major variables will be monitored every timestep and reported in "riemann.out". Watch out for wild swings in these values because they can indicate that the simulation is in trouble. The code also reports on the time taken per timestep. The presence of unintended NANs can also cause the time taken per timestep to change dramatically, signifying a troubled simulation. Thus one can always check up on the status of a run that is ongoing by doing "tail -50 riemann.out".

Problem Specification:- Choose problem dimensions by choosing "**xmin, xmax, ymin, ymax, zmin, zmax**". Uniform zoning is the default and the code does that for you automatically. Choose zone sizes as needed. Also choose the boundary conditions consistently with the application needs. Look for the pattern "CALL INIT_" and make sure that the correct physical problem is being run by the code.

Stopping Conditions:- The code is designed to detect certain types of inconsistent set-ups. In that case, it writes a helpful message and stops graciously. It cannot detect all types of inconsistent set-ups though. It helps to pay attention to those stopping messages and accommodate to them. It does not pay to try and override the stopping messages.

Accuracy Testing:- Any code should meet its design accuracy. If the intended accuracy is not met, it can often reveal a code bug or a design flaw in the underlying algorithms. For that reason, some of the applications focus on accuracy testing. There are subroutines that will set up

smoothly varying hydrodynamical vortex test problems. Because the initial conditions are very smooth, the test problems are amenable to accuracy testing. There are supporting subroutines that will evaluate the error as a function of time when the test problem is run. This will demonstrate whether the chosen algorithm meets its intended accuracy for a particular test problem when the test is carried out on a sequence of meshes with increasing resolution.

The distribution already includes the results of such accuracy analysis for hyrdodynamical test problems. The results show that ADER-WENO schemes meet their design accuracies quite nicely. This may not be so for other well-known options. That is why we have included ADER-WENO methods and also other options (like ADER-TVD and ADER-PPM) that may be better-known to some astronomers.

Main Timestep Loop:- The main timestep loop starts with "DO istep = 1, ntstep" and it is usually best to keep that timestep loop undisturbed. Its logic has already been described in Figs. 1 and 2 and in the previous Sections. It may sometimes become necessary to force the flow in say a turbulence simulation. In that case, the best place to put in forcing is after the time update, so that all variables in conserved and primitive variables have been refreshed and can help in determining how to force the flow. If the code is run under OpenMP, it can also be made to keep track of the time spent on one CPU and the time spent by all CPUs. That can help in gauging parallel efficiency.

HAPPY COMPUTING!!

References

D.S. Balsara, *Linearized Formulation of the Riemann Problem for Adiabatic and Isothermal Magnetohydrodynamics*, Ap.J. Supp., Vol. 116, Pg. 119-131 (1998a)

D.S. Balsara, *Total Variation Diminishing Algorithm for Adiabatic and Isothermal Magnetohydrodynamics*, Ap.J. Supp., Vol. 116, Pgs. 133-153 (1998b)

D.S. Balsara & D. Spicer, *A Staggered Mesh Algorithm Using Higher Order Godunov Fluxes to Ensure Solenoidal Magnetic Fields in MHD Simulations*, J. Comput. Phys., Vol. 149, Pgs. 270-292 (1999a)

D.S. Balsara & D. Spicer, *Maintaining Pressure Positivity in MHD Flows*, J. Comput. Phys., Vol. 148, Pg. 133-148 (1999b)

D.S. Balsara & C.-W.Shu *Monotonicity Preserving Weighted Essentially Non-Oscillatory Schemes with Increasingly High Order of Accuracy*, J. Comput. Phys., Vol. 160, Pgs. 405-452 (2000)

D.S. Balsara, *Divergence-Free Adaptive Mesh Refinement for Magnetohydrodynamics*, J. Comput. Phys., Vol. 174(2), Pgs. 614-648 (2001)

D.S. Balsara, *Second Order Accurate Schemes for Magnetohydrodynamics With Divergence-Free Reconstruction*, Ap.J.Supp., Vol. 151(1), Pgs. 149-184, (2004)

D.S. Balsara, T. Rumpf, M. Dumbser & C.-D. Munz, *Efficient, High Accuracy ADER-WENO Schemes for Hydrodynamics and Divergence-Free Magnetohydrodynamics*, J. Comp. Phys., Vol. 228, Pgs. 2480-2516 (2009)

D.S. Balsara, *Divergence-free Reconstruction of Magnetic Fields and WENO Schemes for Magnetohydrodynamics*, J. Comp. Phys., Vol. 228, Pgs. 5040-5056 (2009)

D.S. Balsara, *Multidimensional Extension of the HLLE Riemann Solver; Application to Euler and Magnetohydrodynamical Flows*, J. Comp. Phys. Vol. 229, Pgs. 1970-1993 (2010)

D.S. Balsara, *A Two-Dimensional HLLC Riemann Solver with applications to Euler and MHD Flows*, J. Comp. Phys., Vol. 231 (2012a) Pgs. 7476-7503

D.S. Balsara, *Self-Adjusting, Positivity Preserving High Order Schemes for Hydrodynamics and Magnetohydrodynamics*, J. Comp. Phys., Vol. 231 (2012b) Pgs. 7504-7517

D.S. Balsara , M. Dumbser and R. Abgrall, *Multidimensional HLL and HLLC Riemann Solvers for Unstructured Meshes – With Application to Euler and MHD Flows*, Journal of Computational Physics, 261 (2014) 172-208

D.S. Balsara, *Multidimensional Riemann Problem with Self-Similar Internal Structure – Part I – Application to Hyperbolic Conservation Laws on Structured Meshes*, Journal of Computational Physics 277 (2014) 163-200

D.S. Balsara and M. Dumbser, *Multidimensional Riemann Problem with Self-Similar Internal Structure – Part II – Application to Hyperbolic Conservation Laws on Unstructured Meshes*, Journal of Computational Physics, 287 (2015) 269-292

D.S. Balsara and M. Dumbser, *Divergence-Free MHD on Unstructured Meshes using High Order Finite Volume Schemes Based on Multidimensional Riemann Solvers*, Journal of Computational Physics 299 (2015) 687-715

D.S. Balsara, *Three Dimensional HLL Riemann Solver for Structured Meshes; Application to Euler and MHD Flow*, Journal of Computational Physics 295 (2015) 1-23

D.S. Balsara, J. Vides, K. Gurski, B. Nkonga, M. Dumbser, S. Garain, E. Audit, *A Two-Dimensional Riemann Solver with Self-Similar Sub-Structure – Alternative Formulation Based on Least Squares Projection*, Journal of Computational Physics 304 (2016) 138-161

D.S. Balsara and J. Kim, *A Subluminal relativistic Magnetohydrodynamics Scheme with ADER-WENO predictor and multidimensional Riemann solver-based corrector*, Journal of Computational Physics , Vol. 312 (2016) 357-384

D.S. Balsara, T. Amano, S. Garain, J. Kim, *High Order Accuracy Divergence-Free Scheme for the Electrodynamics of Relativistic Plasmas with Multidimensional Riemann Solvers*, Journal of Computational Physics 318 (2016) 169-200

J. U. Brackbill and D. C. Barnes, *The effect of nonzero* $\nabla \cdot B$ *on the numerical solution of the magnetohydrodynamic equations*, Journal of Computational Physics 35 (1980) 426-430

S. H. Brecht, J. G. Lyon, J. A. Fedder, K. Hain, *A simulation study of east-west IMF effects on the magnetosphere*, Geophysical Research Lett. 8 (1981) 397

M. Brio & C.C. Wu , *An upwind differencing scheme for the equations of ideal magnetohydrodynamics*, Journal of Computational Physics 75 (1988) 400

Colella, P. & Woodward, P., *The Piecewise Parabolic Method (PPM) for Gas-Dynamical Simulations*, Journal of Computational Physics, 54 (1984) 174

Colella, P., A *Direct Eulerian MUSCL Scheme for Gas Dynamics*, SIAM Journal of Scientific and Statistical Computing, 6 (1985) 104

P. Colella and J. Sekora, *A limiter for PPM that preserves accuracy at smooth extrema*, Journal of Computational Physics, 227, (2008) 7069-7076

W. Dai and P.R. Woodward, *On the divergence-free condition and conservation laws in numerical simulations for supersonic magnetohydrodynamic flows*, Astrophysical Journal 494 (1998) 317-335

C. R. DeVore, *Flux-corrected transport techniques for multidimensional compressible magnetohydrodynamics*, Journal of Computational Physics 92 (1991) 142-160

M., Dumbser, M., Käser, *Arbitary high order non-oscillatory finite volume schemes on unstructured meshes for linear hyperbolic systems*, Journal of Computational Physics, 221 (2007) 693-723

Dumbser, M., Balsara, D.S., Toro, E.F., Munz, C.-D., *A unified framework for the construction of one-step finite volume and discontinuous Galerkin schemes on unstructured meshes*, Journal of Computational Physics, 227 (2008) 8209-8253

M. Dumbser and D.S. Balsara, *A New, Efficient Formulation of the HLLEM Riemann Solver for General Conservative and Non-Conservative Hyperbolic Systems*, Journal of Computational Physics 304 (2016) 275-319

B.Einfeldt, C.-D. Munz, P.L. Roe & B. Sjogreen, *On Godunov-type methods near low densities*, J. Comput. Phys., 92 (1991) 273-295

C.R.Evans and J.F.Hawley, *Simulation of Magnetohydrodynamic Flows: A Constrained Transport Method*, Astrophysical Journal 332 (1989) 659

T. Gardiner & J.M. Stone, *An unsplit Godunov method for ideal MHD via constrained transport*, Journal of Computational Physics, 205 (2005), 509

Godunov, S.K., *A difference method for the numerical calculation of discontinuous solutions of the equations of hydrodynamics*, Mat. Sb., 47 (1959) 271-306

K.F. Gurski, *An HLLC-type approximate Riemann solver for ideal magnetohydrodynamics*, SIAM J. Sci. Comput. 25 (2004) 2165

Harten, A., *High resolution schemes for conservation laws*, Journal of Computational Physics, 49 (1983) 357-393

A. Harten, B. Engquist, S.Osher and S. Chakravarthy, *Uniformly high order essentially non-oscillatory schemes III*, Journal of Computational Physics, 71 (1987) 231-303

Jiang, G.-S. and Shu, C.-W., *Efficient implementation of weighted ENO schemes*, Journal of Computational Physics, 126 (1996) 202-228

S.-T. Li, *An HLLC Riemann solver for magnetohydrodynamics*, J. Comput. Phys., 203 (2005) 344

P. Londrillo and L. DelZanna, *On the divergence-free condition in Godunov-type schemes for ideal magnetohydrodynamics: the upwind constrained transport method*, Journal of Computational Physics 195 (2004) 17-48

P. McCorquodale and P. Colella, *A high order finite volume method for conservation laws on locally refined grids*, Communications in Applied Mathematics and Computational Science, 6(1) (2011) 1

T. Miyoshi and K. Kusano, *A multi-state HLL approximate Riemann solver for ideal magnetohydrodynamics*, J. Comput. Phys., 208 (2005) 315-344

Riemann, B., *Uber die Fortpflanzung ebener Luftwellen von endlicher Schwingungsweite*, Abhandlungen der Gesselschaft der Wissenschaften zu Gottingen, Mathematisch-physicalische Klasse 8 (1860) 43

P. L. Roe and D. S. Balsara, *Notes on the eigensystem of magnetohydrodynamics*, SIAM Journal of applied Mathematics 56 (1996), 57

D. Ryu, F. Miniati, T. W. Jones, and A. Frank, *A divergence-free upwind code for multidimensional magnetohydrodynamic flows*, Astrophysical Journal 509 (1998) 244-255

Titarev, V.A. and Toro, E.F., *ADER: arbitrary high order Godunov approach*, Journal of Scientific Computing 17 (1-4) (2002) 609-618

Titarev, V.A. and Toro, E.F., *ADER schemes for three-dimensional nonlinear hyperbolic systems*, Journal of Computational Physics, 204 (2005) 715-736

Toro, E.F. and Titarev, V.A., *Solution of the generalized Riemann problem for advection reaction equations*, Proceedings of the Royal Society of London, Series A 458 (2002) 271-281

van Leer, B., *Towards the ultimate conservative difference scheme. IV. A new approach to numerical convection*, Journal of Computational Physics, 23 (1977) 276-299

van Leer, B., *Toward the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method*, Journal of Computational Physics, 32 (1979) 101

K.S. Yee, *Numerical Solution of Initial Boundary Value Problems Involving Maxwell Equation in an Isotropic Media*, IEEE Trans. Antenna Propagation 14 (1966) 302

Z. Xu, D.S. Balsara and H. Du, *Divergence-Free WENO Reconstruction-Based Finite Volume Scheme for Ideal MHD Equations on Triangular Meshes*, Communications in Computational Physics, 19(04) (2016) 841-880