# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>v</td>
</tr>
<tr>
<td>Implicit user agreement</td>
<td>vii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>viii</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 VERSION 3.0 ........................................... 1  
1.2 VERSION 3.2 ........................................... 3  
1.3 VERSION 3.3 ........................................... 4  
1.4 VERSION 3.4 ........................................... 6  
1.5 VERSION 3.5 ........................................... 7  
1.6 VERSION 3.6 ........................................... 9  

## 2 Running ZEUS-3D

2.1 Overview ........................................... 12  
2.2 The macro file `zeus36.mac` ......................... 13  
  2.2.1 The `EDITOR` definitions ....................... 15  
  2.2.2 The `EDITOR` aliases ........................... 16  
2.3 The script file `dzeus36.s` ............................. 19  
  2.3.1 Select FORTRAN compiler ....................... 20  
  2.3.2 Get files from home directory .................. 21  
  2.3.3 If necessary, create the directory `dzeus3.6` .... 21  
  2.3.4 Create the change deck .......................... 21  
  2.3.5 Create the input deck for `EDITOR`, and execute ... 22  
  2.3.6 Create the input deck for `ZEUS` ................. 23  
  2.3.7 Make the `ZEUS` executable ....................... 24  
  2.3.8 Tidy up directory ............................... 24  
2.4 Executing ZEUS-3D .................................. 24  

## 3 Output from ZEUS-3D

3.1 Restart dumps ....................................... 25  
3.2 1-D plot files ..................................... 25  
3.3 2-D plot files ..................................... 26  
3.4 Pixel dumps/movie frames ............................ 26  
3.5 HDF4 files .......................................... 27  
3.6 Time slice dumpfiles ................................ 28  
3.7 Cork dumpfiles .................................... 28  
3.8 Display dumpfiles ................................... 28  
3.9 RADIO dumps ....................................... 29  
3.10 Message log files .................................. 30  
3.11 Userdump ......................................... 30  
3.12 Recognised plotting variables ..................... 30
4 Interacting with *ZEUS-3D* 33

5 Adding source code to *ZEUS-3D* 36
  5.1 Adding an entire subroutine .................................. 36
  5.2 Microsurgery using *EDITOR* ................................ 42
  5.3 Debugging in *ZEUS-3D* ...................................... 45

6 Quick summary 48

A The *ZEUS-3D* skeleton 49
  A.1 Legacy transport ............................................. 49
  A.2 Finely Interleaved Transport .............................. 50
  A.3 *EDITOR* aliases ......................................... 51

B The namelists 52
  B.1 IOCON—/O CONtrol (subroutine MSTART) .................. 53
  B.2 RESCON—REStart dump CONtrol (subroutine MSTART) .... 53
  B.3 GGEN1—Grid GENerator for x1 (subroutine GRIDX1) .... 55
  B.4 GGEN2—Grid GENerator for x2 (subroutine GRIDX2) .... 56
  B.5 GGEN3—Grid GENerator for x3 (subroutine GRIDX3) .... 57
  B.6 PCON—Problem CONtrol (subroutine NMLSTS) ............ 58
  B.7 HYCON—HYdro CONtrol (subroutine NMLSTS) ............... 58
  B.8 IIB—Inner I Boundary control (subroutine NMLSTS) ..... 61
  B.9 OIB—Outer I Boundary control (subroutine NMLSTS) ..... 64
  B.10 IJB—Inner J Boundary control (subroutine NMLSTS) ..... 65
  B.11 OJB—Outer J Boundary control (subroutine NMLSTS) ..... 66
  B.12 IKB—Inner K Boundary control (subroutine NMLSTS) ..... 66
  B.13 OKB—Outer K Boundary control (subroutine NMLSTS) ..... 67
  B.14 GCON—Grid motion CONtrol (subroutine NMLSTS) ....... 71
  B.15 AMBCON—AMBipolar Diffusion CONtrol, (subroutine NMLSTS) 69
  B.16 EQOS—Equation Of State control (subroutine NMLSTS) ... 70
  B.17 GCON—Grid motion CONtrol (subroutine NMLSTS) ....... 71
  B.18 EXTCON—EXTension CONtrol (subroutine NMLSTS) ....... 71
  B.19 PL1CON—PLot (1-D) CONtrol (subroutine NMLSTS) ...... 72
  B.20 PL2CON—PLot (2-D) CONtrol (subroutine NMLSTS) ...... 76
  B.21 PIXCON—PIXel/movie graphics CONtrol (subroutine NMLSTS) 80
  B.22 USRCON—USer dump CONtrol (subroutine NMLSTS) ...... 84
  B.23 HDFCON—HDF dump CONtrol (subroutine NMLSTS) ....... 84
  B.24 TSLCON—Time SLice (history) dump CONtrol (subroutine NMLSTS) 85
  B.25 CRKCON—C0RK dump CONtrol (subroutine NMLSTS) ...... 86
  B.26 DISCON—DISplay dump CONtrol (subroutine NMLSTS) ..... 86
  B.27 RADCON—RAdio dump CONtrol (subroutine NMLSTS) ..... 87
  B.28 PGEN—Problem GENerator (subroutine aliased to PROBLEM) 93
C The ZEUS-3D variables

C.1 Grid variables .................................................. 95
C.2 Field variables (3-D arrays) ................................ 96
C.3 Boundary variables (2-D arrays) .............................. 97
C.4 Scratch variables ............................................. 98
C.5 Sundry variables (an abbreviated list) .................... 98
C.6 Parameters ..................................................... 99

Index .................................................................. 100
Preface

Most, if not all of the astrophysical MHD codes used around the world bearing the name ZEUS can trace their roots to the original 2-D code developed by Michael Norman and myself in 1986. Of these, this manual describes a version of the code that I have been developing and maintaining ever since. The interested reader will find a complete history of this code from its inception to the present release in the “history deck” of the source code.

The pervasiveness of ZEUS throughout the world is in large part due to the generous spirit of Michael Norman whose vision included “astrophysical community codes” to serve theorists much like AIPS serves radio astronomers. ZEUS-3D was first developed at the National Center for Supercomputing Applications (NCSA) between 1988 and 1990, and in 1992 version 3.2 was made available to the public. A few years later, MLN released the MPI version of the code (ZEUSMP) and use of the code spread. Many versions of the code now exist and have been significantly modified by various users to perform simulations from comet-planet collisions to cosmology.

While the ZEUS-family of codes are not fully-upwinded (like a Godunov scheme, for example), they have proven to be flexible and robust. One can add almost any physical process to the code without worrying too much about its effects on the underlying MHD scheme. It has therefore found a niche amongst numerically literate, though perhaps not expert, astrophysicists who have a computational problem to investigate but neither the time nor the resources to develop their own code.

One of the roles of the Institute for Computational Astrophysics (ICA) at Saint Mary’s University is to provide and, in some cases, support code to the astrophysical community. To this end, the ICA web page (www.ica.smu.ca/zeus3d) now places into the public domain this author’s double-precision version of ZEUS-3D (version 3.6; dzeus36). This code is distinct from the NCSA/UCSD/LCA version of the code (ZEUSMP; http://lca.ucsd.edu) which has not enjoyed significant algorithm development in more than a decade. As evident in §1, version 3.6 has undergone significant code development and bears little resemblance algorithmically to version 3.2 released around 20 years ago and, for that matter, to ZEUSMP. This includes, for example, a “planar-split” MHD algorithm, fully conservative in 3-D, self-consistent boundary conditions, an “$N$–log $N$” Poisson solver, and a full suite of graphics. The code now runs under OpenMP with 98% parallelism [yielding a speed-up factor of 12.5 (20) with 16 (32) processors] and AZEuS\(^1\), an AMR version of the code and still under development at the time of this writing, will become available from a parallel site (www.ica.smu.ca/azeus) in the near future.

Conditions for use of this code are on the next page. The proper citations for referencing the algorithms used in dzeus36 are:


It is requested that any publication reporting results performed by dzeus36 or any of its

\(^1\)Adaptive Zone Eulerian Scheme
derivatives include the following acknowledgement:

Use of ZEUS-3D, developed by D. A. Clarke at the Institute for Computational Astrophysics (www.ica.smu.ca) with financial support from the Natural Sciences and Engineering Research Council of Canada (NSERC), is hereby acknowledged.

If length is an issue, the following will also suffice:

Use of ZEUS-3D, developed by D. A. Clarke at the ICA (www.ica.smu.ca) with support from NSERC, is acknowledged.

Inquiries about this software, constructive criticism, bug reports, etc., should be directed to the author at the address below.

David Clarke, March, 2015
Institute for Computational Astrophysics
Saint Mary’s University
Halifax, NS, Canada B3H 3C3
dclarke@ap.smu.ca
Implicit user agreement

In what follows, *this software* refers to “ZEUS-3D, version 3.6” (*dzeus36*), and *the author* refers to David A. Clarke, ICA, Halifax. It is assumed that anyone using this code has read, understood, and agreed to the following conditions of use:

1. Distribution of this software shall remain the purview of the author. A user is free to share this software with students and co-workers, but requests from those not working directly with the user should be directed to [www.ica.smu.ca/zeus3d](http://www.ica.smu.ca/zeus3d).

2. This software shall be used exclusively for education, research, non-profit, and non-military purposes. Specific written permission from the author must be obtained before any commercial use of this software is undertaken.

3. The banner and history decks (first two modules of the source code) shall remain with this software and any descendent developed from and still based substantially upon this software.

4. The name(s) of the institution(s) with which the author is or has been affiliated shall not be used to publicise any data and/or results generated by this software. All findings and their interpretations are the opinions of the user and do not necessarily reflect those of the author nor the institution(s) with which the author is or has been affiliated.

The author makes no representations about the suitability of this software for any purpose. Subject to the above conditions, this software and accompanying manual are provided “as is” without expressed or implied warranty.
Acknowledgements

The author wishes to express his gratitude to students, research associates, collaborators, and mentors past and present, for their valuable contributions toward the development of dzeus36, and in particular in debugging, providing and/or developing subroutines and algorithms, giving advice, and development of this user manual. In alphabetical order, these include Jack Burns, Stephen Campbell, Mike Casey, Jean Pierre DeVilliers, Kevin Douglas, Logan Francis, Phil Hardee, John Hawley, Chris Howard, Tom Jones, Byung-Il Jun, Chris Loken, Pierre-Yves Longaretti, Nick MacDonald, Chris MacMackin, Mordecai-Mark Mac Low, Alexander Men'shchikov, Michael Norman, Rachid Ouyed, Jon Ramsey, Mark Richardson, Alex Rosen, Jim Stone, Martin Sulkanen, and Joel Tanner.

Acknowledgement is made of the use and incorporation of routines from Numerical Recipes by William Press, Saul Teukolsky, William Vetterling, and Brian Flannery. This is an epic tomb of enormous benefit to the computational science community, and the ZEUS-3D project has benefited from this classic text on numerous occasions.

The author thanks the late Kevin Kohler, formerly of the Oceanographic Center at Nova Southeastern University, for allowing the source code for PSPLOT to be distributed with dzeus36. PSPLOT has improved and broadened in-line graphics—which had traditionally been accomplished with NCAR graphics—as well as simplify their installation.

The author also wishes to thank Professor Tom Jones of the Department of Astronomy at the University of Minnesota for his permission to include his Riemann solver into this release. These modules provide the “analytical” comparator for the suite of 1-D Riemann problems that comprise a significant portion of the test suite used to confirm ZEUS-3D.

Over the years, financial and technical support for the ZEUS development project(s) has been provided by many sources, including the NCSA and the University of Illinois, the American National Science Foundation and NASA, the Harvard-Smithsonian Center for Astrophysics, Saint Mary’s University, and NSERC.

Finally, and most profoundly, the author wishes to thank his former advisor and mentor, Michael Norman, for his vision of a community astrophysics code which came to be known as ZEUS. Some of the coding in dzeus36 still bears Mike's signature, and certainly the fundamental structure of the program follows the Jim Wilson and Mike Norman school of thought.
1 Introduction

1.1 VERSION 3.0

ZEUS-3D is a 3-D magnetohydrodynamics (MHD) solver, and although it was designed with astrophysical applications in mind, fluid dynamic problems in the other physical sciences can be addressed with this code too. The code is now about 35,000 lines of FORTRAN and growing, and represents many years of work by many people. During the past two years, I have been the primary developer of the code, although algorithms and structures developed by many others over the past 20 years have been freely used. These include Philip Colella, Chuck Evans, John Hawley, Michael Norman, Larry Smarr, Jim Stone, Bram van Leer, Jim Wilson, Karl-Heinz Winkler, Paul Woodward, and others.

ZEUS-3D was created as part of the ZEUS development project, begun and headed by Dr. Michael Norman at the NCSA (National Center for Supercomputing Applications). It has been Mike’s continuing efforts to support this project, both financially and intellectually, that have made the development of ZEUS-3D possible. Dr. Jim Stone, also a member of the ZEUS development project, was the principle creator of ZEUS-2D, the predecessor to ZEUS-3D. Although the two codes now differ substantially, the efforts that Jim and Mike made to develop the magnetic field algorithm and the modularity of the code are still very evident in ZEUS-3D.

In its present incarnation, ZEUS-3D is a three-dimensional ideal (non-resistive, non-viscous, adiabatic) non-relativistic magnetohydrodynamical (MHD) fluid solver which solves the following coupled partial differential equations as a function of time and space:

\[
\begin{align*}
\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) &= 0; \\
\frac{\partial \vec{s}}{\partial t} + \nabla \cdot (\vec{s} \vec{v}) &= -\nabla p - \rho \nabla \Phi + \vec{J} \times \vec{B}; \\
\frac{\partial e}{\partial t} + \nabla \cdot (e \vec{v}) &= -p \nabla \cdot \vec{v}; \\
\frac{\partial \vec{B}}{\partial t} &= \nabla \times (\vec{v} \times \vec{B}).
\end{align*}
\]

where:
- \(\rho\) = matter density
- \(\vec{v}\) = velocity flow field
- \(\vec{s}\) = \(\rho \vec{v}\) = momentum density vector field
- \(p\) = thermal pressure
\[ \Phi = \text{gravitational potential} \]
\[ \vec{B} = \text{magnetic induction} \]
\[ \vec{J} = \text{current density } (\nabla \times \vec{B}) \]
\[ e = \text{internal energy density (per unit volume)} \]

The code possesses the following numerical attributes:

1. finite differencing on an Eulerian mesh (but possibly moving in an average sense with the fluid);
2. fully explicit in time and therefore subject to the CFL limit;
3. operator and directional splitting of the MHD equations;
4. can be used efficiently for 1-D and 2-D simulations with any of the coordinates reduced to symmetry axes;
5. Cartesian geometry for 3-D simulations, Cartesian and cylindrical coordinates for 2-D simulations, Cartesian, cylindrical, and spherical coordinates for 1-D simulations;
6. written in a “covariant” fashion to minimise the effects of the different coordinate systems on the structure of the code;
7. fully staggered grid, with scalars (density and internal energy) zone-centred and vector components (velocity and magnetic field) face-centred [derived vector components (current density and \textit{emfs}) are edge-centred];
8. von-Neumann Richtmyer artificial viscosity to smear shocks;
9. upwinded, monotonic interpolation using one of donor cell (first order), van Leer (second order), or piecewise parabolic interpolation—PPI (third order) algorithms;
10. Consistent Advection used to evolve internal energy and momenta; and
11. Constrained Transport modified with the Method of Characteristics used to evolve the magnetic fields.

This code is strictly Newtonian. Relativistic astrophysics cannot be simulated in any way with this version. No explicit account for relativistic particles is incorporated either. The code assumes strict charge neutrality at all times—it is not a plasma code. It is assumed that the fluid is thermal, and is coupled to the magnetic fields via collisions with an ionised component which never undergoes charge separation. Pressure is assumed to be isotropic and gravitation is limited to the specification of a point mass. A fully three-dimensional Poisson-solver is planned for the next version (3.1) which will account for the self-gravity of the fluid.

The purpose of this manual is not to educate the potential user on numerical techniques, physical justification of the assumptions inherent to the code, or even what the potential problems to be solved are. Instead, it is assumed that the user is intimately familiar with the
fundamentals of MHD and has come up with a complex problem to solve which is completely 
described by equations 1 through 4. It is also assumed that the user has a working knowledge 
of UNIX. In this spirit, this manual is designed to instruct the user on the mechanics of using 
ZEUS-3D to solve the equations that pen and paper cannot attempt.

D. Clarke, February 1991

1.2 VERSION 3.2

The code has undergone numerous changes since the release of version 3.0 and has grown 
to nearly 45,000 lines of FORTRAN and more than 160 subroutines. Version 3.1 was never 
actually released as such, and so there is no corresponding manual. This, then, is the first 
revision of the user manual. The major differences between versions 3.0 and 3.2 include:

1. Line-of-sight integrations through the data volume for a variety of variables, including 
the Stokes parameters (see §3.9) are possible in both XYZ and ZRP coordinates. The 
EDITOR definition RADIO (§2.2.1) must be set to invoke this display option.

2. An option has been added to generate time slice plots. The EDITOR definition TIMESL 
has been added which now must be set in order to get time slice output.

3. Subroutines peculiar for generating polar pixel dumps (written by Carol Song) have 
been expunged. ZEUS-3D now converts polar slices to Cartesian slices “on the fly” 
before generating pixel dumps.

4. 2-D NCAR graphics have been enhanced with better annotation. Polar contours and 
vector plots now work properly.

5. An EDITOR alias FINISH has been added which represents a subroutine called after 
the main loop of the main program zeus3d. This gives the user a slot in which to 
perform various tasks at the end of the run.

6. The code can be micro-tasked for the Crays. Tests indicate that for typical runs, a 
real-time speed-up of 3.9 can be achieved with 4 dedicated processors.

7. The code will now run efficiently (i.e., it vectorises) as a uni-tasked process on the 
Convex. This is done by defining the EDITOR definition CONVEXOS. Multi-tasking on 
a Convex using the -O3 option can be done, but yields a real-time speed-up of only 
about 2.5 on a four processor machine. For runs on the Crays, UNICOS must now be 
defined.

8. More combinations of dimension and geometry are now known to work. The list now 
includes Cartesian (XYZ) with any two, any one, or no symmetry flag(s) set, cylindrical 
(ZRP) with either JSYM+KSYM or KSYM set, spherical polar (RTP) with either JSYM+KSYM 
or KSYM set. Other combinations will be debugged as needed.

9. One can now select an isothermal equation of state. A new EDITOR definition ISO 
has been added to take advantage of the reduction in memory and computation re-
quirements for isothermal systems.
10. Yu Zhang (NCSA) has implemented a 3-D self-gravity module using the so-called DADI (Dynamical Alternating Direction Implicit) scheme. The \textit{EDITOR} definition \texttt{GRAV} must be set if self-gravity is to be invoked.

11. One now has the choice of solving either the total energy equation or the internal energy equation (the only choice in previous versions). The toggle \texttt{itote} has been introduced to the namelist \texttt{hycon} to specify which of these formalisms is to be used (Byung-Il Jun, NCSA).

12. Pixel, Voxel and \textit{RADIO} dumps may now be made in \textit{HDF} format. This avoids the cumbersome process of “bracketing” the images, but at the cost of more than four times the disc space requirements.

13. The common blocks have been radically restructured, and the way restart dumps are generated has been overhauled entirely. It is now possible to read a restart dump, for example, that was generated by a compiled version of the code with different \textit{EDITOR} macro settings and different values for the array parameters.

14. Ragged boundaries are no longer available. This feature has been expunged from the code for lack of use and because of the increasing effort necessary to incorporate it into new features. Boundaries must now be regular.

Users of version 3.0 will be happy to note that there are no major changes in the way \textit{ZEUS-3D} is compiled or executed, and the namelist parameters have remained more or less fixed. Still, there are enough subtle changes that it might do the experienced user some good to review these notes before attempting to run a job with this new version. Also note that version 3.2 cannot read restart dumps created by version 3.0, and vice versa.

D. Clarke, August, 1992

1.3 VERSION 3.3

The NCSA, under the auspices of the Laboratory for Computational Astrophysics and the leadership of Dr. Michael Norman, has developed \texttt{zeus32} into an MHD-cosmology code and continues to make their code available to the community. Independent of the NCSA effort, I and my co-workers have developed \texttt{zeus32} into an CR-MHD (CR $\equiv$ cosmic rays) code (\texttt{zeus33}). This manual, therefore, describes the first non-NCSA version of the code and was developed at the Harvard-Smithsonian Center for Astrophysics. This version contains more than 52,000 lines of source code and is the most extensive re-write of the code since version 3.0 was first generated from the 2-D template. Most of the routines in the \texttt{PHYSICS} group—including the hydrodynamics—have been rewritten in order to implement the new Consistent Method of Characteristics (CMoC). The CMoC was developed to solve the chronic problem of magnetic field explosions in previous MHD algorithms. While substantive to the code, these changes are mostly transparent to the user. Changes of significance to the user include:

1. The \textit{EDITOR} alias \texttt{MOC} has been removed, since the MoC algorithm has been replaced with the CMoC algorithm. The option to use the original CT scheme has also been
eliminated since, unlike MoC, CMoC reduces to the original CT scheme in the weak field limit. The EDITOR alias FASTCMOC has been added to activate the more efficient version of CMoC for cases where the ratio of the flow and Alfvén velocities is not expected to exceed $10^8$ for 64-bit words, and $10^4$ for 32-bit words.

2. A two-fluid approximation for a relativistic fluid has been installed (Byung-Il Jun, NCSA). It is turned on by specifying the EDITOR macro TWOFLUID. The two-fluid approximation takes after Jones & Kaing (1991, ApJ, 363, 499) and can reproduce all of their results. The diffusion coefficient is determined by a subroutine linked with the EDITOR alias DIFFUSION. The diffusion operator is performed using a time-centred, sub-cycling algorithm which allows the CFL limit to be specified independently of the diffusion time scale.

3. A time-centred subcycling option for the artificial viscosity has been installed and is activated by setting iscyqq=1 in namelist hycon. This renders the CFL limit independent of the viscous time scale. For applications with strong shocks, this can reducing computational time by a factor of 2 or more.

4. An additional option for ARTIFICIALVISC has been introduced (gasdiff) by Byung-Il Jun. This routine uses ordinary gas diffusion to stabilise shocks. It does so without any excess heating often associated with viscosity, but tends to render the solution very smooth since it is applied everywhere.

5. The variables iordd, iorde, etc. and istpd, istpe, etc. have been expunged. In this release, iord and istp specify respectively the order of the interpolation algorithm and whether the steepener is to be applied in the third order PPI algorithm for all variables.

6. The I/O has been updated with the two-fluid variables. In addition, the conventions used in the various I/O routines have been standardised. In particular, with the exception of RADIO variables, virtually all variables available for output in any one I/O routine are available in all. By necessity, the RADIO variables remain limited.

7. A “pseudogravity” option has been added. The pseudogravity “holds onto” artificial pressure gradients (e.g., a King atmosphere) much like ordinary gravity was used in ZEUS04 (the predecessor to ZEUS-2D). The pseudogravity is activated by setting the EDITOR macro PSGRAV which is mutually exclusive with GRAV. The pseudo-gravitational potential has the same units as pressure (i.e., $pv^2$) rather than the usual units of gravitational potential (i.e., $v^2$). The pseudo-gravitational acceleration is given by $dv/dt = -\nabla \phi/\rho$ and is treated exactly as a pressure in the source term routines. Thus, to “hold onto” an artificial atmosphere in a problem initialisation routine, simply define PSGRAV and set $gp(i,j,k) = p(i,j,k)$.

8. Bremsstrahlung emission has been added to the RADIO dumps.

9. The code has been generalised to run on SUN SPARCstations. The EDITOR macro SUNOS must be specified for either SUNOS or SOLARIS operating systems.

D. Clarke, December 1993
1.4 VERSION 3.4

It has been more than a decade since version 3.3 was completed. Except for one bug found in the CMoC algorithm (to do with density, and described below), it has proven to be an extremely robust algorithm. The main problem with the code is its boundary conditions, and this version has seen numerous rewrites and experimentation with its boundary condition routines, none satisfactory.

The main problem is that as released, version 3.3 could not do something as simple as launch an Alfvén wave from an inflow boundary. As released, version 3.4 can, but at the cost of introducing monopoles at an inflow boundary in some circumstances (such as Ouyed-type jets from problem generator CORONA), and numerous patches have been installed to prevent or at least limit these. In particular, inflow/outflow boundary conditions should be used with great caution. On the plus side, with help form Pierre-Yves Longaretti, periodic boundary conditions are now exact, with both sides of a periodic grid committing identical machine round-off errors.

Other changes to the code include:

1. The code has been upgraded to double precision, and is now called dzeus34. Creating the executable xdzeus34 now requires linking the double precision libraries: dnamelist.a and dsci01.a.

2. The problem generator for launching jets from accretion discs (à la Ouyed and Pudritz) has been added (corona). A new EDITOR definition POLYTROPE has been added if the results of solving the internal energy equation are to be set aside in favour of a strict polytrope ($p = \kappa \rho^n$). This feature should be used with extreme caution as a polytrope is not physically equivalent to an adiabatic equation of state (the former forbidding irreversible processes).

3. The problem generator for Couette type flows (Longaretti) has been added.

4. PSGRAV and GRAV may now be set simultaneously, if needed. A new 3-D array, psgp, has been added to keep the pseudo-gravity separate from the physical gravitational potential, gp.

5. Yu Zhang’s DADI gravity routines, which never worked properly, have all been expunged and two new Poisson solvers have been installed by A. Men’shchikov: SOR (Successive Over-Relaxation), and FMG (Full Multi-Grid). The algorithm is chosen by setting gravalg to 1 or 2 for SOR or FMG. Only SOR is fully debugged.

6. The code is now portable to AIX (IBM) and LINUX.

7. A bug in the CMoC algorithm was fixed. The original scheme used four-point averages of the density to the location of the $\text{emf}$ when estimating the characteristic velocities. However, it was found that at steep density gradients, this proved disastrous. A degree of freedom overlooked in the original CMoC implementation was exploited to allow the density to be upwinded too, thus preventing steep gradients from over- or under-estimating $\text{emfs}$. 
8. Kinematic viscosity has been added to the code (constant viscosity only), and is triggered by specifying a non-zero value for “nu”, a global variable, in namelist HYCON. “nu” is the kinematic viscosity defined by \( \nu = V L / R \), where \( R \) is the Reynolds number of the flow and \( L \) and \( V \) are length and velocity scales of the problem.

9. A. Men’shchikov has introduced PSPLLOT\(^2\) to the plotting library grfx03.a. Three namelist parameters (norpp11, norpp12, norptsl) will allow for publication-quality graphics with colour without linking any NCAR libraries. Two additional user-creatable libraries psplot.a and noncar.a must be linked instead for this option to work.

10. The subroutines CURRENT* have been replaced with CURL*, which compute components of the curl. It is a generalisation that may be used to compute the vorticity as well.

11. Vector potentials are now available in all graphics options (e.g., §B.19, §B.20, §B.21, etc.). To allow this, “inverse curl” routines have been added (following Arfken, ed. 5, pp. 73–74) that compute a vector potential from a given magnetic field.

D. Clarke, September, 2005

1.5 VERSION 3.5

Modifications to the code from version 3.4 are numerous and invasive, and needed to correct long-standing problems with boundary conditions and to complete the installation of the total energy equation. The code now has more than 90,000 lines of FORTRAN and 350 subroutines.

A self-consistent framework for setting magnetic boundary conditions has been developed and installed in this release. In particular, inflow and outflow boundary conditions, while still not perfect, are now much cleaner than in dzeus34, and can be used with some confidence. For the user, the visible consequences are four-fold:

1. A distinction is now made between the skin values (e.g., variables on the i=is face at the inner-i boundary) and proper boundary values (e.g., variables, face-centred or zone-centred, at i=ism1 and i=ism2 at the inner-i boundary). For inflow conditions, the user must now set the skin values of the transverse (to the boundary normal) components of the emf (thus, \( \varepsilon_2 \) and \( \varepsilon_3 \) at the inner-i boundary), and the boundary values of the transverse magnetic field components (thus, \( B_2 \) and \( B_3 \) at the inner-i boundary). Note that there are no skin values for the transverse magnetic field and the boundary values of the transverse emfs are set directly by a new routine BVALEMFS called at the top of CT.

2. Arrays containing desired inflow values for the normal (to the boundary) magnetic field components (e.g., b1ib1, b2ib1, etc.) are no longer available. Instead, the normal magnetic field component is now set by the solenoidal condition and therefore “floats”.

\(^2\)with kind permission from its author, Kevin Kohler. Please see Acknowledgements for the full citation.
3. Routines BVALB1, BVALB2, and BVALB3 used to set magnetic boundary values in previous versions are no longer available. Instead, the user must initialise every element of the magnetic field arrays (b1, b2, and b3) in their problem generator including boundary zones making sure that $\nabla \cdot \vec{B} = 0$ at $t = 0$ everywhere.

4. Boundary type 8, a *selective inflow* boundary condition, can now be set and is suitable for sub-magnetosonic inflow conditions. At a given inflow boundary, as many boundary conditions may be set as there are characteristics pointing into the grid (e.g., see Bogovalov, 1997, A&A, 323, 634). For supermagnetosonic flow, this means all seven variables, namely $\rho$, $p$ (or $e$), all three components of the velocity, and the two transverse components of the magnetic field (the normal component being determined by the solenoidal condition). On the other hand, for sub-slowmagnetosonic inflow, only four characteristics point into the grid, and three of the inflow variables must be allowed to “float”. A variable floats if one sets the boundary type [e.g., niib(j,k)] to 8, and if the inflow array for that variable [e.g., diib1(j,k) for density, etc.] is set to “huge” (a global parameter; see §C.6).

Complete details are given in §B.8. Other changes made to this release include:

1. A new parameter, iords (§B.7), now lets one specify the order of interpolation for the scalars (e.g., $\rho$, $e$, etc.) separately from the vector components, which are still controlled by iord. The default value for iords is iord, whose default is still 2 (van Leer). Note that with the scalars interpolated with PPI (iord=3) and the contact steepener activated (istp=2), contacts in most 1-D Riemann problems are as steep as fast shocks—two or three zones.

2. Additional interpolation schemes have been partially added (i.e., available for scalars only), most notably a velocity-corrected second-order van Leer scheme (iord=-2). See §B.7 for details.

3. The new boundary conditions fix many problems, including a long standing one with 3-D toroidal fields in propagating jets. Previously, it was noted that such a field introduced a strong axial field from the i=zs skin, and tapering the magnetic profile to zero before the jet radius was necessary to avoid this. 3-D toroidal fields now leave the i=zs skin perfectly cleanly, with zero (to machine round off errors) $B_i$ left on the skin itself.

4. Stone’s MOC and the Hawley-Stone variation, HSMOC, have been installed to allow for comparisons among the various algorithms. New EDITOR aliases MOC and HSMOC are used to engage these algorithms (§2.2.1).

5. Parameter ijkx is now ijkx (x ⇒ maximum) and a new parameter ijkx (n ⇒ minimum) has been added. Confusion between the old and new uses of ijkx is minimised as these are now set automatically and no longer need to be set by the user in the script file, dzeus35.s (§2.3).
6. Installation of the total energy equation is complete, and is now the default ($itote=1$). This has mitigated numerous changes throughout the code, including removal of consistent advection from the energy equations, and the introduction of a new global variable, $et$, that always contains the total energy density, regardless of which energy equation is being solved. The old energy variable, $e$, and its related inflow variables, e.g., $eiib1$, etc., have been changed to $e1$ and $eiib1$, etc., and always contains the internal energy density, regardless of which energy equation is being solved. See §B.7 for further details.

7. Tom Jones’ “Reimann solver” has been included, and is compiled when the EDITOR macro $RIEMANN$ is defined. Its purpose is strictly to provide the “analytical” solution for the suite of 1-D Riemann test problems, and is controlled via namelist $pl1con$ (§3.2).

8. EDITOR macro $VECPOT$ has been added to allow the vector potential to be used as the primary magnetic variables instead of the magnetic field. In principle, this should cause differences only at the machine round-off level, and other than setting the macro, requires no changes by the user. For example, the user would still initialise the magnetic field components everywhere. With $VECPOT$ set, the code would then use the “anticurl” routines, $ACURL*$, $* = 1, 2, 3$, to compute the initial vector potential from the initial magnetic field.

9. Toru Okuda’s flux-limited diffusion algorithm for radiation HD has been included, though in an incomplete and untested form. It is activated by setting the EDITOR macro $RADIATION$.

10. The code has been tuned for OpenMP, and its directives can be inserted automatically by setting $iutask=2$ in EDITOR’s input deck $inedit$ (part of $dzeus35.s$; see §2.3).

D. Clarke, September, 2007

1.6 VERSION 3.6

The most significant change to the code is a new “finely interleaved transport” (FIT) algorithm which retains most of the algorithmic details of CMoC (planar split, fully Eulerian transport, etc.), but rearranged into finer modules, with more frequent updates to $v$ and $B$. As such, the code is now largely impervious to a “striping instability” (affecting 2-D shear Alfvén waves and 2-D advection) to which the code has been vulnerable since $zeus04$.

The code is now over 130,000 lines of FORTRAN with nearly 500 subroutines and functions.

1. $GRFX03$ has been overhauled. All 1-D plotting routines have been merged to a new $GRFX1D$ which now does single and multiple plots using lines or one of a variety of symbols. For 2-D plots, the user may now select as many as two scalars and three vector fields (along with “corks”; see change 3 below) for each plot in this “buffet-style” of plotting. (See §B.20.)
Also for 2-D plots, one may now rebin the 2-D data-slices to a lower resolution grid before plotting. *e.g.*, for high-resolution runs, one doesn’t always need contour plots of the full-resolution data which, for *PSPLLOT*, can require significant cpu, memory, and disc resources to generate.

2. Nick MacDonald’s “aging” module has been added to track the synchrotron age of a zone, thereby allowing spectral indices to be calculated. This should still be considered experimental.

3. Nick MacDonald’s and Mark Richardson’s Lagrangian particle (“corks”) routine has been added. These tracer particles can be initialised by the problem generator and/or injected during the simulation, and carry with them a whole host of measurables. To engage, *EDITOR* macro *CORKS* must be set. See namelist *crkcon*.

4. From the outset (*zeus30*), differences among compilers were inaccurately attributed to the operating system (OS) rather than the compiler. As such, a longish list of *EDITOR* macros for OSs (*e.g.*, SUNOS, CTSS, *etc.*) had accumulated which couldn’t cleanly accommodate different compilers running under the same OS. In this release (and in version 2.2 of *EDITOR*), all *EDITOR* OS macros have been replaced with compiler macros. The user now specifies in the *dzeus36.s* file (rather than the *zeus36.mac* file where the OS used to be set) a supported compiler with the compiler option (*coption*) “debug”, “optimise” or “openmp”, and *EDITOR* creates the appropriate *makezeus* file. Supported compilers include: *cf77* (Cray), *f90* (Sun), *g95* (*g95.org*), *gfortran* (Gnu), *ifort* (Intel), *pgf77*, *pgfortran* (Portland Group), and *xlf* (IBM).

The compilers *g95* and *gfortran* can give numerous inscrutable warning messages from the loader which seem to be entirely innocuous:

```
ld: warning for symbol _g01wdt_ tentative definition of size 16 from ... 
```

There is some discussion of these on various web forums, none of which reveals the origin or cause. Further, the *g95* compiler does not seem to be ANSI-standard, as some test problems (*e.g.*, Fig. 3b from Ryu & Jones, 1995) exhibits noise in $v_2$ at the $10^{-17}$ level which is entirely absent with all other compilers.

In the 1-D tests with optimisation turned on, *ifort* is 75% faster than *gfortran* which is 67% faster than *g95*.

5. Slip periodic boundary conditions have been added (see *gp boundary*).

6. The old voxel dumps have been purged out of complete disuse.

7. Stephen Campbell (B.Sc. 2014) has introduced a new animation-making option in the pixel dump routine (*morppix=1*). All images are stored as *real*4 (instead of *character*4) and assembled at the end of the run into mpeg files. The original *real*4 frames are gathered in a tarball to allow the user to remake the animation with different palette, bracketing, *etc.*, without having to rerun the simulation. See *PIXDMP* for further details.

A similar feature is available for the radio dumps (*RADDMP*).
8. Logan Francis (B.Sc. 2014) has added an FFT/FST gravity solver (grvalg=3) which can handle all types of boundary conditions (periodic by using a Fourier transform, Dirichlet conditions by using a sine transform). As with the full multi-grid algorithm (grvalg=2), active grid dimensions must be powers of 2. See GRAVFFT for further details.

9. The long-standing problem of 2-D shear Alfven wave “wings” has been resolved, resulting in a nearly complete overhaul of the transport step. ZEUS can now perform 2-D shear Alfven wave propagation, 2-D advection, and vortex ring transport totally free from what I call the “striping instability” using “Finely Interleaved Transport” (FIT; trnvrn=1). The traditional transport algorithm (“legacy transport”) is still available by setting trnvrn.le.0.

FIT represents the most significant overhaul of the transport step since the introduction of CMoC in 1993. There are no new algorithms per se, rather the granularity of existing algorithms has been reduced and the order in which quantities are updated altered. Thus, and for example, for various values of $ix1x2x3$, $v2$ is updated from 3-transport of $s2$ before 2-transport of $s1$ (using $v2$). Likewise, $b2$ and $b3$ are updated from $emf1$ before $emf2$ is computed from $b1$ and $b3$, etc. With these changes, dzeus36 is not vulnerable to the “striping instability” (which it has been since zeus04), and 2-D transport is now CFL-limited for the first time.

10. The one-fluid ambipolar diffusion algorithm described by Duffin & Pudritz (2008, MNRAS, 391, 1659) has been implemented by Chris MacMackin (B.Sc 2015). It should still be considered experimental and be used only in the limit of low ionisation. For a highly ionised fluid, one needs a two-fluid approach. Chris has also implemented the super-stepping (Runge-Kutta) method of Meyer, Balsara, & Aslam (2012, MNRAS, 422, 2102) to address the exceedingly short time step ambipolar diffusion otherwise requires. This allows an AD problem to be evolved on the CFL-limited MHD time scale.

D. Clarke, December, 2015
2 Running ZEUS-3D

2.1 Overview

At the time of this writing, ZEUS-3D may be compiled with any one of cf77 (Cray), ifort (Intel), f90 (Sun), g95 (g95.org), gfortran (Gnu), pgf77, pgfortran (Portland Group), and xlf (IBM). Other compilers can be used with only minor changes required to the code, and the user is referred to the installation instructions in the document install_dz36.txt for details.

This manual is written assuming the use of Gnu’s gfortran, the most recent and still-supported version of this most-commonly used publicly available FORTRAN compiler, though differences with other compilers are mostly minor and transparent. For those familiar with versions 3.5 of ZEUS-3D and earlier, the OS-specific EDITOR macros AIX, CONVEXOS, LINUX, SUNOS, and UNICOS have been replaced with the compiler-specific macros listed above. This long-overdue change was made to correct an early and incorrect notion that FORTRAN differences were necessary to accommodate differences in the operating system rather than the compiler. The move to compiler-specific macros affects primarily the installation instructions and introduces minor changes to the zeus36.mac and dzeus36.s files; all other aspects of using ZEUS-3D remain unchanged.

With the code installed, the user controls what simulation is performed by editing the two files zeus36.mac and dzeus36.s. These are relatively short and painless to edit, and their complete descriptions are included in the next two subsections. Once edited to the user’s satisfaction, the ZEUS-3D executable is created by running the dzeus36.s script file which is done by typing:

csh -v dzeus36.s

Running this file performs sequentially the following tasks:

1. sets the FORTRAN compiler;
2. retrieves all necessary files from user-specified home directories;
3. creates a directory called dzeus3.6 within the user’s present working directory (pwd) to store all the source and object files created during compilation;
4. creates a change deck for dzeus36 containing preprocessor macros and aliases (zeus36.mac, next subsection), and changes to the source code (if any) required for the application (the most common and often the only changes necessary to the source code are to the parameter statements which set the size of the arrays needed for the run.);
5. fires up the EDITOR preprocessor;
6. creates the input deck for the dzeus36 run; and finally

---

3Available from the ICA web site www.ica.smu.ca/zeus3d
4gcc.gnu.org/fortran
7. makes the executable xdzeus36 (using the UNIX facility MAKE).

A description of the file naming convention is required at this point. ZEUS-3D refers in a general way to the package and its capabilities while dzeus36 is more specific, and is a mnemonic for "double precision ZEUS-3D, version 3.6". zeus36 is the common denominator for the names of the principle files required to create the executable. Thus, the source code itself is dzeus36, the script file is dzeus36.s, the macro file is zeus36.mac (there is no leading “d” since no changes were needed in this file during migration to double precision), and the executable is xdzeus36. However, to confuse matters, the minor files don’t follow this convention. The input deck is inzeus and the change deck is chgzeus —no “36” suffix—and the libraries don’t even have ZEUS as part of their names. And so it goes. The bottom line, though, is that if the only changes to be made to the source code are the values of the parameters which set the array dimensions, then there are only two files to be concerned with: dzeus36.s and zeus36.mac. The rest is automatic.

2.2 The macro file zeus36.mac

Below is an example of a zeus36.mac file. A similar file can be downloaded from the ICA web site. It is suggested that this file be copied and used as a general template since all the macros used by dzeus36 are listed in this example.

```plaintext
**--+-1--------2---------3---------------3--------------2---1---1---**
**
**************  CONDITIONAL COMPILATION SWITCHES **************
**
** 1) symmetry axes: ISYM, JSYM, KSYM
**
*define JSYM, KSYM
**
** 2) geometry: XYZ, or ZRP, or RTP
**
*define XYZ
**
** 3) physics: AGING, AMBIDIFF, GRAV, ISO, MHD, POLYTROPE, PSGRAV,
** RADIATION, TWOFLUID
**
*define MHD
**
** 4) data output modes: CORKS, DISP, HDF, PIX, PLT1D, PLT2D, RADIO,
** TIMESL
**
*define PLT1D, TIMESL
**
** 5) other: DEBUG, FASTCMOC, HSMOC, MOC, NOMOC, Riemann, VECTORISE
**
*define FASTCMOC, RIEMANN
**
************** MODULE NAME ALIASES **************
**
The modules "BNDYUPDATE", "SPECIAL", "SPECIALSRC", "SPECIALTRN",
** "FINISH", "PROBLEM", "PROBLEMRESTART", "USERSOURCE", and "USERDUMP"
** are slots available for user-developed subroutines.
**
```
These are all preprocessor commands (the preprocessor used here is called EDITOR—also developed by the author—and for those familiar with the old Cray OS CTSS, it has the “look and feel” of HISTORIAN), and become part of the “change deck” chgzeus created by the script file dzeus36.s, described in the next subsection. A change deck is a file which is merged with the source code during the preprocessing step of dzeus36.s. Both the source code and the change deck can contain preprocessor commands which are interpreted, carried out, and then expunged from the code by EDITOR before the code is compiled by the FORTRAN compiler. All preprocessor commands have an asterisk (*) in column 1; two asterisks (**) in columns 1 and 2 indicate a comment. When the preprocessor has finished, the result is a pure FORTRAN source code tailored specifically for the problem to be solved. Therefore, in order to customise the code, it is necessary to set the EDITOR “definitions” and “aliases” (generically referred to as “macros”) found in zeus36.mac.

The combined effect of the macros is two-fold. First, they determine what parts of the code are activated and what parts are ignored. Thus, it is possible to eliminate the computations and the memory requirements necessary to evolve the magnetic fields, for example, by not defining the MHD macro [this can be done by “commenting out” (double asterisk) the *define MHD statement in the example above]. The preprocessor will then remove all coding
peculiar to the magnetic field including the declarations of the magnetic field arrays during
the preprocessing step. The compiler never sees the magnetic stuff, and the executable is
streamlined for the hydrodynamical problem. Of course, the original source code is not al-
tered by preprocessing it. Rather, the preprocessor creates a precompiled version of the code
and stores each subroutine into its own file (to facilitate MAKE and debuggers such as DBX)
in the directory dzeus3.6 which is created by the script file dzeus36.s. Second, the alias
macros can be used to substitute any character string in the code during the preprocessing
step.

This manual discusses only those aspects of the EDITOR preprocessor necessary for the
user to make changes to the code, compile it, and then execute it. A full account of EDITOR
is given in the manual, edit22_man.ps, found in the manuals directory of dzeus36.tar from
www.ica.smu.ca/zeus3d.

2.2.1 The EDITOR definitions

A description of the definition macros (called “Conditional Compilation Switches” at the
top of the given example of zeus36.mac above) follows:

1. The code can be streamlined (optimised) for 1-D and 2-D problems by setting the
   appropriate symmetry macros. If symmetry along any of the $i$ ($x_1$), $j$ ($x_2$), or $k$ ($x_3$)
   axes is desired, then set the ISYM, JSYM, or KSYM macros. If the macros are not set and
   a 1-D or 2-D calculation is initialised by the input deck, ZEUS-3D will still carry out
   the sub 3-D computation correctly, but will do so less efficiently.

2. The geometry is set by setting one of the XYZ (Cartesian), ZRP (cylindrical), or RTP
   (spherical polar) macros. Obviously, these macros are mutually exclusive.

3. AGING activates the subroutines and define an additional scalar array necessary to track
   the “age” of high-energy electrons required for calculations of synchrotron emissivity.
   AMBIDIFF turns on the single-fluid approximation for ambipolar diffusion described by
   alias GRAVITY to gravity will turn on the Poisson solver and one of three algorithms
   (SOR, FMG, FFT) will be used to solve the self-gravitational potential. The ISO
   macro should be set if an isothermal equation of state is desired. With ISO defined,
   an isothermal equation of state is presumed and the energy variables are not declared
   saving both computational time and memory. By setting the MHD macro, the algorithm
   for evolving the magnetic fields is activated. With MHD on, additional field arrays are
   declared and the code peculiar to updating the magnetic field is compiled. POLYTROPE
   forces a strict polytropic equation of state. PSGRAV (no longer mutually exclusive with
   GRAV) activates the pseudo-gravity feature used to hold onto artificial atmospheres.
   The macro RADIATION enables the (incomplete) flux-limited diffusion algorithm for
   RMHD. Defining TWOFLUID will activate the arrays and coding necessary to solve the
   energy equation for the second thermal fluid. Note that partial densities and momenta
   are not tracked for the second fluid; only partial internal energies (and thus partial
   pressures). The second fluid may be subjected to diffusion, if desired.
4. The graphics enabled during a run are set by the graphics macros. Set CORKS to activate Lagrangian tracer particles and their output dumps, set DISP for display dumps, set HDF for HDF dump files, set PIX to enable 2-D pixel dumps, set PLT1D for 1-D line plots, set PLT2D for 2-D contour and/or vector plots, set RADIO for RADIO dump files, and set TIMESL for time slice dumps. As many as these may be set simultaneously as necessary. See §3 for a discussion of the various ZEUS-3D dump files.

5. The DEBUG macro turns on portions of the code designed for development and debugging, and will send all sorts of messages to the terminal and may even cause the code to crash. It should be invoked only by developers of the code. The faster CMoC algorithm may be invoked by setting the macro FASTCMOC. This macro should be set only if the accuracy of the smallest of the flow and Alfvén speeds is unimportant when it falls below $10^{-8}$ ($10^{-4}$) times the largest of the speeds for 64-bit (32-bit) words. Otherwise, the general CMoC algorithm (activated by not setting the FASTCMOC macro) can handle arbitrarily small Alfvén and/or flow speeds accurately, but at the cost of 25% more computational time. The macros HSMOC, MOC, and NOMOC invoke other MHD algorithms which are available for comparison, but are not recommended for general use. The macro RIEMANN is needed if 1-D analytical solutions are to be overlaid with the results of 1-D shock-tube tests, and VECTORISE invokes special “vectorised” versions of the CMoC routines generally used only in debugging.

### 2.2.2 The EDITOR aliases

The alias macros allow phrases in the code to be substituted for other phrases during the precompiling step. Thus, “Module Name Aliases” (in the middle of the given example of zeus36.mac above) give the user control over what subroutines are called during execution. As an example, in the main program of the source code, there is a statement: call SOURCE which becomes call srcstep after preprocessing using the given example of zeus36.mac. Note that there is no subroutine called SOURCE but there is a subroutine in the source code called srcstep. Thus, the user is free, in principle, to create their own subroutine to manage the source terms instead of srcstep, and this new routine can be linked into the code by altering the alias setting for SOURCE from srcstep to the name of the new routine. Note that by setting any of the Module Name Aliases to empty (a subroutine in dzeus36 which does nothing but return to the calling routine), a Module Name Alias can be effectively “turned off”.

Aliases can also be used to set parameters in various parameter statements scattered throughout the source code. These are the “Error Criteria Aliases” and “Iteration Limits Aliases” at the bottom of the given example of zeus36.mac above. Thus the EDITOR statement:

```
alias GRAVITYERROR 1.0e-6
```

sets the maximum convergence error in the self-gravity module to $10^{-6}$. Somewhere in the code is the statement `parameter ( errmax = GRAVITYERROR )` and the preprocessor makes the substitution. However, the majority of the parameters (array dimensions, for example) are set directly in dzeus36.s which is described in the next subsection.
Running ZEUS-3D

To understand better the descriptions of the “Module Name Aliases” which follow, the reader should examine the flow charts in App. A (ZEUS-3D Skeleton) which indicate the order in which the Module Name Aliases are called. Some subroutines are charged with reading the input data from the input deck inzeus. A description of all the input namelist parameters is given in App. B.

1. **BNDYUPDATE**: This module is called at the beginning of each loop and allows inflow boundary conditions to be evolved in time should this be necessary for the simulation. Examples of evolving inflow boundary conditions include helically perturbing the inflow at a jet orifice to break the symmetry (wiggle), generating magnetic field at the boundary (bgen), or empty if no inflow boundary update is desired. The user can, of course, supply a subroutine for this alias. See §5.1 for discussion on how to add a subroutine to the code.

2. **EXTENDGRID**: This module will allow the grid to be extended as a disturbance (shock) propagates into initially quiescent zones. Currently, the only options are extend and empty. The subroutine extend will prevent quiescent zones from being updated until the disturbance comes within five zones, potentially saving significant amounts of computational time. Care should be exercised in its use, however. If the subroutine is unsuccessful in determining when the disturbance gets close to an edge of the current computational domain, the results can be disastrous.

3. **GRAVITY**: This module updates the self-gravitational potential. Currently, the only choices are empty and gravity. If gravity is selected, the user will have to choose a Poisson solver (grvalg in namelist grvcon), as well as a method to determine boundary values (glib, etc. in namelist iib, etc.).

4. **SPECIAL**: This is a simplistic solution to the potentially complex problem of the user desiring to add a whole new type of physics to the code. It assumes that changes do not need to be intertwined into existing modules, which in practice, often will be necessary. The three accompanying “plugs” SPECIALSRC (for “special” source terms to be added after the artificial viscous step), USERSOURCE (for source terms to be added before the artificial viscous step), and SPECIALTRN (for “special” transport terms) allow for some flexibility in installing new physics within the current structure, but this still may not be enough for any type of sophisticated addition. Currently, all four macros are set to empty.

5. **SOURCE**: This is the module in which source terms are incorporated. For full dynamics, this should be set to srcstep (or the user’s module if need be) while for problems of pure advection, this should be set to empty.

6. **SPECIALSRC**: See SPECIAL.

7. **TRANSPORT**: This is the module for the transport of variables across zone boundaries and should be set to trnsprt or to the user’s equivalent module. It is unlikely that empty should ever be used here.
8. SPECIALTRN: See SPECIAL.

9. NEWTIMESTEP: This module determines how the next time step is computed. Since ZEUS-3D is an explicit code, all algorithms should incorporate the CFL limit. Current choices are newdt for full (M)HD problems, and advectdt for pure advection problems.

10. NEWGRID: This module adjusts the grid should grid velocities be set to follow the flow, at least partially. Current choices are newgrid and empty. In practise, the user will have to provide their own prescription for evaluating the grid velocities, as most of the available methods are untested. This will require replacing or adding to the subroutine newvg. See §5 for discussion on how to add or modify a subroutine in the code.

11. FINISH: This is a “plug” available to the user to have any user-supplied subroutine called once at the end of execution. This could, for example, be a routine to generate diagnostics and/or graphics for data the user may have been accumulating in USERDUMP.

12. PROBLEM: This macro is used to link the user-supplied “problem generating” subroutine that initialises all flow variables and boundary values. It is called by the subroutine setup, which is called by mstart. Alternately, a number of problem generators for a variety of applications already exist in the source code. In the present example, PROBLEM is set to shkset, an existing problem generator which initialises the variables for a 1-D Riemann problem (“shock tube”); in this case, the so-called Brio and Wu problem.

13. PROBLEMRESTART: This macro allows the specifications of the problem to be altered should the job be restarted from a restart dump. Set the macro to empty if no alteration of the problem is desired (as, for example, to simply extend the evolution time).

14. USERSOURCE: See SPECIAL.

15. ARTIFICIALVISC: This macro specifies which artificial viscosity algorithm should be used. Current options are viscous, which uses the von-Neumann Richtmyer artificial viscosity algorithm, and gasdiff which invokes ordinary gas diffusion.

16. DIFFUSION: This macro specifies the subroutine to use to compute the diffusion coefficient for the two-fluid model. Currently, the only options are empty and diffco.

17. USERDUMP: This macro allows the user to link their own I/O routine if additional I/O capacity is needed other than what is currently available in dataio (1-D and 2-D plot files, pixel dumps, HDF files, time slice dumps, cork dumps, display files, RADIO dumps, and full-precision restart dumps, all created at user-set time intervals; §3).

It is unlikely that the “Error Criteria Aliases” or the “Iteration Limits Aliases” should ever have to be changed.

Finally, in addition to the aliases listed above is module name alias ATMOSPHERE called by problem generator jetinit which allows a user to specify their own routine to initialise an atmosphere through which a jet is launched. Existing atmosphere routines include cloud (to set up a jet-cloud collision) and king, which sets up a King atmosphere.
2.3 The script file dzeus36.s

Below is a reproduction of the script file dzeus36.s found in the zeus directory of the file dzeus36.tar downloaded from www.ica.smu.ca/zeus3d. It can be run by typing: csh -v dzeus36.s.

#---+----1----+----2----+----3----+--+----3----+----2----+----1----+---#
#============== SOURCE FILE TO CREATE THE ZEUS EXECUTABLE =============#
# # B R I O W U . X 1 # #
# #--------------------------> Select FORTRAN compiler.
# # Supported compilers include: 
# # cf77 (Cray), f90 (Sun), g95 (g95.org), gfortran (Gnu), ifort 
# # (Intel), pgf77, pgfortran (Portland Group), xlf (IBM) 
# setenv FCOMP gfortran
#===========================================================> Get files from home directory.
if(! -e ./xedit22) cp ../editor/xedit22 .
if(! -e ./dnamelist.a) cp ../nmlst/dnamelist.a .
if(! -e ./dsci01.a) cp ../sci/dsci01.a .
if(! -e ./grfx03.a) cp ../grfx/grfx03.a .
if(! -e ./psplot.a) cp ../grfx/psplot.a .
if(! -e ./noncar.a) cp ../grfx/noncar.a .
#=======================================> If necessary, create the directory "dzeus3.6".
if(! -e ./dzeus3.6) mkdir ./dzeus3.6
#=======================================> Create the change deck.
rm -f ./chgzeus
cat << EOF > ./chgzeus
*read zeus36.mac
*define $FCOMP
*d par.44,45
parameter ( in = 555, jn = 1, kn = 1 )
parameter ( nxpx = 1, nypx = 1, nxrd = 1, nyrd = 1 )
**read chguser
EOF
#=======================================> Create the input deck for EDITOR, and execute.
rm -f ./inedit
cat << EOF > ./inedit
\$editpar innam='dzeus36', chgdk='chwzeus', idump=1, job=3
  , ipre=1, inmlst=1, iupdate=1, iutask=0, safety=0.20
  , branch='dzeus3.6', xeq='xdzeus36', make='makezeus'
  , compiler='$FCOMP'
  
c  , cptions='debug'
  , cptions='optimise'
  
c  , specdk='corona', 'phistv', 'nmlsts', 'plotid'
  , lib='checkin.o dnamelist.a dsci01.a grfx03.a psplot.a
    noncar.a'
  , lib='checkin.o dnamelist.a dsci01.a grfx03.a psplot.a
    noncar.a -L/opt/local/hdf4/$FCOMP/lib -ldf -ljpeg -lz -lsz'
  , lib='checkin.o dnamelist.a dsci01.a grfx03.a psplot.a
    -L/opt/local/hdf4/$FCOMP/lib -ldf -ljpeg -lz -lsz
  , lib='checkin.o dnamelist.a dsci01.a grfx03.a psplot.a
    -L/opt/local/ncarg/$FCOMP/lib -lnccarg -lnccarg_gks -lnccarg_c
    -XI11 -Icairo -Ifreetype'
EOF
chmod 755 ./xedit22
./xedit22
#=======================================> Create the input deck for ZEUS.
rm -f ./inzeus
cat << EOF > ./inzeus
\$iocon iotty=6, iolog=2
\$rescon dtdmp=0.0, idtag='xd'
\$ggen1 nbl=550, x1min=0.0, x1max=550., igrid=1, x1rat=1.0, lgrid=.t.\$
\$ggen2 \$
\$ggen3 \$
\$pcon nlim = 999999, tlim=-80.0, ttotal=900.0, tsave=10.0
\$hycon qcon=1.0, qlin=0.2, courno=0.75, iord=2, iords=3, istp=2
, itote=1, iscyqq=0
\$iib niib(1,1)=9
\$oib noib(1,1)=9
\$ijb
\$ojb
\$ikb
\$okb
\$grvcon
\$ambcon
\$eqos gamma=2.0
\$gcon
\$extcon
\$pl1con dpl1=80.0, pl1dir=1, corl=1, aspect=1.0, pl1h=2, npl1v=2
, norppl1=2, pl1soln=12*1, xdiscpl1=200.0
, pl1var = 'd ', 'se', 'p ', 'et', 'v1', 'v2', 'v3', 'ma'
, 'b1', 'b2', 'b3', 'bd'
\$pl2con
\$pixcon
\$usrcon
\$hdfcon
\$tsclcon
\$crkcon
\$discon
\$radcon
\$pgen idirect=1, n0=200, d0=1.000, e10=1.0, v10=0.0, b10 =0.75
, b20=0.6, b30=0.8
\$pgen idirect=1, n0=350, d0=0.125, e10=0.1, v10=0.0, b10=0.75
, b20=-0.6, b30=-0.8
EOF
#============================================> MAKE the ZEUS executable.
make -f ./makezeus
#===================================================> Tidy up directory.
rm -f ./editlp ./inedit ./output ./xedit22 ./*.a

Note that a # in column 1 indicates a comment in a script file. In this example, two
flavours of comment lines are used. Comments led with a double dashed line (=========)
indicate portions of the script file which rarely, if ever, need to be changed by the user.
Comments with a single dashed line (--------) indicate portions of the script file that will
probably need to be changed with every simulation. Below are descriptions of the eight
segments found in the script file dzeus36.s.

2.3.1 Select FORTRAN compiler

The first segment sets the environment variable FCOMP to the desired FORTRAN compiler
(e.g., gfortran). This may be one of the compilers already supported within dzeus36
(cf77, f90, g95, gfortran, ifort, pgf77, pgfortran, or xlf), or to a compiler the user has
introduced into dzeus36, as per the installation instructions install_dz36.txt.

2.3.2 Get files from home directory

The second segment retrieves the files necessary to create the ZEUS-3D executable and are retrieved only if they do not already exist on disc [if (! -e filename)]. This example assumes that the script file is launched from the directory zeus created when dzeus36.tar from www.ica.smu.ca/zeus3d is unpacked. Files already in this directory and thus not retrieved include:

- **dzeus36** the nearly 130,000 lines of source code divided up into more than 450 subroutines
- **zeus36.mac** file containing all the EDITOR macros (§2.2)
- **checkin.o** the object file of the C-routine checkin.c (the only C-routine used by ZEUS-3D) which allows interrupt messages to be read from the terminal during interactive runs (§4)

while those retrieved from other directories include:

- **xedit22** the preprocessor executable
- **dnamelist.a** the double precision library of subroutines which emulate the namelist feature (§2.3.6)
- **dsci01.a** the double precision library of four specialised max-min subroutines
- **grfx03.a** library of subroutines calling routines in external graphics libraries (NCAR and PSPLOT)
- **psplot.a** library of routines for PSPLOT graphics
- **noncar.a** library of dummy NCAR routines used when NCAR graphics are not installed at the site

2.3.3 If necessary, create the directory dzeus3.6

The third segment creates the directory dzeus3.6 on condition that it does not already exist. The precompiled source files (one subroutine per file) and the compiled object files are put here.

2.3.4 Create the change deck

The fourth segment creates the change deck chgzeus which is merged with the source code dzeus36 during the preprocessing step. The first line in chgzeus reads the EDITOR macros in zeus36.mac using the EDITOR command *read. This command replaces the statement with the contents of the named file. Thus, the macros in zeus36.mac become part of the change deck chgzeus, and get merged with the source code.

The second line uses the environment variable set in the first segment to set the appropriate EDITOR FORTRAN macro.

The third line is the EDITOR *delete (or *d for short) command used to replace lines 44 and 45 in the common deck par in the main source code dzeus36 with the two following parameter statements which set the parameters to the desired values for the simulation.
This is where the user should indicate the size of the arrays required for the simulation to be performed. The parameters set in the given example of the script file \texttt{dzeus36.s} are all described in §C.6.

Finally, should the user have their own changes to the code, these can be most conveniently put into a file called \texttt{chguser}, for example, and the statement \texttt{**read chguser} would then be “de-commented” by deleting one of the asterisks. This will ensure that the user’s changes will be incorporated just like those in \texttt{zeus36.mac} and the two parameter statements discussed above. Changes should be specified using the language of \texttt{EDITOR} (code prepared for the old CTSS precompiler \texttt{HISTORIAN} can be processed by \texttt{EDITOR}), and would include additional subroutines such as the problem generator which need to be compiled with the rest of the source code. Full description of how to do this is found in §5.

### 2.3.5 Create the input deck for \texttt{EDITOR}, and execute

The fifth segment creates the input deck for the preprocessor \texttt{EDITOR} and then fires it up. Changes to this segment should be needed rarely. If it becomes necessary to change the name of the main source file from \texttt{dzeus36}, or to change the name of the change deck from \texttt{chgzeus}, or to change the name of the directory created for the precompiled and compiled subroutine files from \texttt{dzeus3.6}, or to change the name of the makefile from \texttt{makezeus}, or to change the name of the \texttt{ZEUS-3D} executable from \texttt{xdzeus36}, or to specify compiler options other than the default “debug” or “optimise” settings, these changes should be made in the \texttt{EDITOR} input deck \texttt{inedit}.

On the latter, by setting \texttt{coptions='optimise'}, \texttt{EDITOR} will insert the appropriate compiler options for the selected compiler (FCOMP) so long as the selected compiler is one of the eight compilers supported by \texttt{EDITOR}\textsuperscript{5}. Similarly if \texttt{coptions='debug'}, the appropriate debugging compiler options are set including array size and over-flow checking. For any other compiler, the user will have to set \texttt{coptions} explicitly. Similarly, loader options can be specified in \texttt{loptions}.

On occasion and depending on the compiler, a few subroutines can cause a run to generate significantly different results when compiled with full optimisation than with little or no optimisation [often traceable to exponentiation (**)]. For example, Sun’s old \texttt{f77} compiler generated errant object code when compiling the routines \texttt{corona}, \texttt{phistv}, and \texttt{couette} with full optimisation. Other routines, such as \texttt{plot1d}, \texttt{plot2d}, and \texttt{mlists} compiled correctly, but took forever to optimise. For such occasions, \texttt{EDITOR} allows one to specify troublesome routines in \texttt{specdk} (a 1-D character*8 array) and the special compiler options to be used for these routines in \texttt{speccopt}. Thus, “de-commenting” the line:

\begin{verbatim}
c , speccopt='-O0', specdk='corona','phistv','nmlsts','plot1d'
\end{verbatim}

in the sample file \texttt{dzeus36.s} above would apply no optimisation (-00) to the four routines listed, and full optimisation (-02) to the rest. For additional details, the reader is referred to the \texttt{EDITOR} user manual: \texttt{edit22_man.ps}.

\textsuperscript{5}At the time of the writing, this list is the same as those supported by \texttt{dzeus36}, namely \texttt{cf77}, \texttt{f90}, \texttt{g95}, \texttt{gfortran}, \texttt{ifort}, \texttt{pgf77}, \texttt{pgfortran}, and \texttt{xlf}. 
For parallel processing at the do-loop level, set iutask (second line of the namelist editpar) to 1 for Cray microtasking, or 2 for OpenMP. This will cause EDITOR to insert the appropriate scoping commands at the beginning of the major do-loops in the code. One then has to set the appropriate compiler options for your compiler to compile the code for multiple processors. If the user is using OpenMP and one of the eight supported compilers, one can set coption='openmp' to let EDITOR set the appropriate compiler options.

Libraries are specified by setting libs, of which three examples are given in the sample file dzeus36.s above. The first and uncommented setting of libs requires only libraries included in dzeus36.tar (from www.ica.smu.ca/zeus3d); no systems or third-party libraries such as NCAR (for graphics, e.g., §3.2, §3.3) or HDF (§3.5) are required. By virtue of the PSPLOT library, publication-quality and full-colour graphics are possible without NCAR (originally, the only graphics capability ZEUS possessed). The second libs command (commented out) is the variation used at the ICA for HDF libraries, while the third (also commented out) is when both HDF and NCAR are linked. Additional libraries may be linked by appending them to whatever libs list is used. Note that lines “commented out” in a namelist are echoed on the CRT as the input deck is read, which is a feature of the EDITOR namelist. (See §2.3.6 and App. B for a discussion of the EDITOR namelist feature.)

With this input deck, the preprocessor will merge the change deck chgzeus with dzeus36, carry out the precompiler commands according to the aliases and definitions in the macro file zeus36.mac, split up the precompiled source code (now containing nothing but FORTRAN syntax) into separate files for each subroutine, search the directory dzeus3.6 and write to disc only those files which do not already exist or have been changed, and finally create the makefile makezeus, described in §2.3.7.

2.3.6 Create the input deck for ZEUS

The sixth segment is where the input deck for the ZEUS-3D executable is created (inzeus) and so the user should set all input parameters here (described fully in App. B). In this example, inzeus is set up for the 1-D MHD Brio and Wu shock tube problem. ZEUS-3D uses namelists to specify input parameters but does not use the standard namelist utility. Historically, the first versions of namelist available under UNICOS were horrid (character variables could not be set, vectors could only be set one element at a time, error messages were unreadable), and so a more useful namelist utility was incorporated into the preprocessor EDITOR. Thus, as one of its duties, EDITOR can be instructed (inmlst=1) to replace all references to namelists with calls to subroutines found in the library dnamelist.a which is linked to the executable during the MAKE process. This step is entirely transparent to the user. Namelists can be used as always, with the usual (more or less) syntax, bearing in mind that once defined, a namelist must be read before the next namelist is defined. Since this time, namelist has become a standard feature of Fortran90 and has been significantly improved. Should the user prefer to use the namelist utility of the local OS, then the input parameter inmlst in the EDITOR input deck inedit should be set to 0 (§2.3.5). Be warned that doing this may make some of the namelists in the dzeus36.s (inzeus) file unreadable and generate run-time error messages. Syntactic errors may even arise during compilation.

One major difference between the Fortran90 namelist and the EDITOR namelist is the latter allows for rank 2 arrays to be specified in an extremely intuitive fashion. For
example, to set \((\text{diib1}(i,j), i=20,30), j=70,80)\) to 1.0, while setting the rest of the 100 by 100 array to 0.1, one merely needs to type:

\[
\text{diib1}(1:100,1:100) = 0.1, \text{diib1}(20:30,70:80) = 1.0
\]

where the order is important. This capacity is not supported by Fortran90, and so some of the \text{namelist} syntax will have to be changed in the input decks \text{inzeus} and \text{inedit} should the user wish to use the standard \text{namelist}. If using the \text{EDITOR namelist} feature, remember not to allow any of the \text{namelist} lines to extend beyond the 72nd column. The first column in each line can be a blank or a ‘c’ (to comment out the line) and nothing else. The second column may contain a blank or a ‘$’ and nothing else. (Note that because \text{dzeus36.s} is a script file, the $ must be “protected” by a \). Otherwise, the script file will try to interpret the $ as a control character rather than treating it as a character to be written to a disc file. The user will note that a \ does not precede the $ in the input deck \text{inzeus} once it is written to disc by \text{dzeus36.s}.) Text specifying the input parameters may start in column 3. If a character string is too long to fit in the 72 column format, one simply types as much as one can in the first line (i.e., up to and including the 72nd column), then resumes typing the character string on the next line, beginning in column 3. A single quote must appear before the first character in the first line of the character string and after the last character in the last line of the character string only.

A detailed description of all the \text{namelist} parameters is contained in App. B.

2.3.7 Make the ZEUS executable

The seventh segment fires up the makefile \text{makezeus} created by the preprocessor \text{EDITOR}. The makefile will compile only those \text{FORTRAN} files in the directory \text{dzeus3.6} which have been written since the last time they were compiled, then link all the object files together with the specified libraries to create the executable \text{xdzeus36}.

2.3.8 Tidy up directory

The eighth and final segment deletes the temporary files no longer needed.

2.4 Executing ZEUS-3D

Once the script file has completed successfully, simply type \text{xdzeus36} followed by a carriage return, and ZEUS-3D will begin running. In general, one can move the two files \text{xdzeus36} and \text{inzeus} to any other directory and the executable can be launched from that directory simply by typing \text{xdzeus36}, followed by a carriage return (enter).

Alternatively, one can run ZEUS-3D in batch mode, and for this the user should consult their systems administrator as batch facilities are highly system and installation dependent.
3 Output from ZEUS-3D

A variety of methods for dumping data to disc during execution are available in ZEUS-3D. Each of these methods has their specific use, and at times all types are used simultaneously. In this section, a brief description of each method is given, along with a list of the most vital statistics. These include: the EDITOR definition (if any) which enables the data dump, the logical unit to which the dumps are attached during execution, the namelist which controls the data dump (App. B), the convention used for naming the disc file for this type of data dump, and the format of the data in the disc file created.

3.1 Restart dumps

These are full precision dumps of all flow variables at specified time intervals suitable for resuming a simulation. Execution can be instructed to overwrite the previous even (odd) numbered dump with the new even (odd) numbered dump should disc space be limited. Thus, only two restart dumps would exist at any one time. For MHD runs, the size of a restart dump is about $10 \times in \times jn \times kn$ words, while for HD runs, $6.5 \times in \times jn \times kn$ words.

The first data written to a restart dump are the array dimensions and parameters that indicate which EDITOR macros are defined. Values of EDITOR aliases are not stored. These, then, are the first data read from a restart dump and are used to allow a restart dump to be read regardless of the differences between the array dimensions and EDITOR definition settings in the new executable (that which is reading the restart dump) and the old executable (that which created the restart dump). Thus, it is possible to resume an MHD run without the MHD definition set (and thus resume the calculation hydrodynamically), to read the inner eighth of a $64^3$ data volume into any part of a new $128^3$ grid, etc.

   EDITOR definition: none
   logical unit: iodmp
   namelist: rescon
   filename: $zrnnn$id, where $zr$ is the common prefix to all restart dumps, $nnn$ is a three digit integer distinguishing the multiple dumps created during a run, and $id$ is a two character, user-specified problem tag.
   format: binary, one word (8 bytes) per datum

3.2 1-D plot files

These are metacode (NCAR) or postscript (PSPLLOT) files containing publication-quality 1-D plots for every variable selected. A separate file is created for each slice specified.

   EDITOR definition: PLT1D
   logical unit: iopl1
   namelist: plicon
   filename: $zpnnn$id.mm, where $zp$ is the common prefix to all 1-D plot files, $nnn$ and $id$ are as defined for restart dumps, and $mm$ is an extension indicating the slice number. For PSPLLOT, the suffix .ps is added to the filename.
format: metacode—use idt to read NCAR-generated metafiles
postscript—use mgv/gv to read PSPLLOT-generated postscript files

If, after the run is complete, the user needs 1-D plots of variables that were not selected before execution, these can be created after the run using the stand-alone ancillary program PLOTZ, provided the user has created a total HDF dump at or near the desired problem time. See the User Manual for ZEUS Ancillaries found in the manuals directory of dzeus36.tar from www.ica.smu.ca/zeus3d for details.

3.3 2-D plot files

These are metacode (NCAR) or postscript (PSPLLOT) files containing publication-quality 2-D plots of a user-specified combination of variables (colour contours, line contours, and/or vectors) using the so-called buffet-style plotting introduced in this version (§B.20). A separate file is created for each slice specified.

EDITOR definition: PLT2D
logical unit: iopl2
namelist: pl2con
filename: zqnnnid.mm, where zq is the common prefix to all 2-D plot files, nnn and id are as defined for restart dumps, and mm is an extension indicating the slice number. For PSPLLOT, the suffix .ps is added to the filename.
format: metacode—use idt to read NCAR-generated metafiles
postscript—use mgv/gv to read PSPLLOT-generated postscript files

See the note at the end of §3.2 if, after the run is complete, the user needs 2-D plots of variables that were not selected before execution.

3.4 Pixel dumps/movie frames

These are “binned” 2-D slices through the data volume of a single variable designed for animation. Files are written in one, two, or all three of three formats: real*4 “movie” (new to version 3.6); Mike Norman’s char*1 “pixel dumps”; and HDF4.

Movie frames (pixfmt=1, default) are 4-byte unformatted files written with a header that includes grid dimensions, grid, and data extrema. If this format is selected then, at the end of the run, they are automatically converted to PPM format then assembled into an mpeg file assuming FFmpeg (ffmpeg.org) is available on the user’s platform. The PPM files may then be deleted, and the original real*4 movie frames are assembled into a compressed tarball which can be used by the stand-alone ancillary program ANIM8Z to re-render the animation if the bracketing was not chosen to the user’s satisfaction.

Pixel dumps (pixfmt=2) are unformatted files just one byte deep (one ascii character per datum) and include no header whatever, not even array dimensions. These were designed for use with Carol Song’s IMAGETOOL, long since abandoned by the NCSA. To render these into an animation, one has to convert each to a PPM file (e.g., Dave Lane’s RAW2PPM) and then assemble the PPM files into an mpeg using FFmpeg.
In either movie or pixel format, polar plots are rebinned to a Cartesian plane, then dumped as a regular movie frame or pixel plot. Both formats are designed to create animations, though with the passage of `IMAGETOOL`, the movie frames are more straightforward to use. Multiple slices per variable may be requested so that the number of animations per run can be substantial (e.g., four slices of six variable means 24 animations).

Finally, `HDF4` files are written for `pixfmt=3`.

**EDITOR** definition: `PIX`
logical unit: `iopix`
namelist: `pixcon`
filename: `zi**nnnid.mm{.f}`, where `zi` is the common prefix to all image frames, `**` is a two-character representation of the variable (see Table 3.1 in §3.12), `nnn` and `id` are as defined for restart dumps, `mm` is an extension indicating the slice number, and `f` is one of `m` for movie format (`pixfmt=1`), blank for pixel format (`pixfmt=2`), or `h` for `HDF4` (`pixfmt=3`). On exit, movie files (`pixfmt=1`) are converted to PPM, used to create an mpeg file `zi**id.mm.mpg`, and assembled in a tarball `zi**id.mm.tar.gz`.

format: `pixfmt=1`: unformatted, four bytes per datum plus header
`pixfmt=2`: unformatted, one byte per datum, no header
`pixfmt=3`: `HDF4`

### 3.5 HDF4 files

`HDF4` (Hierarchical Data Format; version 4) files contain 3-D data of one or more variables in the `HDF` format developed at the NCSA. The data are written in four byte words which is adequate for quantitative graphical study but not for checkpointing (see §3.1). All graphical software packages from the NCSA use this format for data dumps, as do many other commercially available products (e.g., IDL). `HDF` files contain header information which include array dimensions, data extrema, and grid coordinates. The size of an `HDF` file containing a single variable is a little more than the number of active zones times 4 bytes. For a “total” dump (all primary variables to the same `HDF` file) with none of `GRAV`, `PSGRAV`, or `TWOFLUID` defined, the size is the number of active zones times 32 bytes for MHD runs, or times 20 bytes for HD runs.

**EDITOR** definition: `HDF`
logical unit: `none`
namelist: `hdfcon`
filename: `zh**nnnid`, where `zh` is the common prefix to all `HDF` files, `**`, `nnn`, and `id` are as defined for pixel dumps.

format: `HDF4`, four bytes per datum

If, after the run is complete, the user needs 1-D, 2-D, or `RADIO` dumps that were not created during the simulation, these can be created after the run using the stand-alone ancillary programs `PLOTZ` and `RADIO`, which use `HDF` “total dumps” as input (see §B.23). For this reason, it is recommended that HDF total dumps be created at least as often as 1-D
and 2-D plots and, if disc space is not a problem, as often as RADIO dumps. See the User Manual for ZEUS Ancillaries found in the manuals directory of dzeus36.tar from www.ica.smu.ca/zeus3d for details on how to install and use PLOTZ and RADIO.

3.6 Time slice dumpfiles

There are two types of time slice dumps, and either, both, or neither may be selected. The first is a single ascii file which contains values of various scalars at specified time intervals such as the total mass, angular momenta, magnetic monopoles, energy, etc., as well as extrema of quantities such as density, pressure, divergence of magnetic field, etc. The second is a plot file (metacode or postscript) containing 1-D plots of these scalars as a function of time. The user selects the time interval for the ascii and plot dumps independently.

EDITOR definition: TIMESL
logical units: iotsl1 and iotspl
namelist: tslcon
filenames: ztnnnid (ascii file), where zt is the common prefix to all time slice ascii files, nnn and id are as defined for restart dumps.
ztpnnnid (plot file), where ztp is the common prefix to all time slice plots.
formats: ascii and metacode/postscript

3.7 Cork dumpfiles

New to version 3.6, one can specify a set of Lagrangian (tracer) particles scattered throughout the grid at \( t = 0 \) as well request others to be injected onto the grid at specified time intervals. These particles (a.k.a. “corks”) are completely passive in nature, and gather user-specified information about their local conditions as the simulation progresses. At each time step where cork information is dumped, a single line of data for each cork is appended to an accumulating ascii file which can be used by the user for post-processing to display the data as desired. In addition, the location of the corks (but not their accumulated data) may be added to the 2-D NCAR/PSPLLOT files.

EDITOR definition: CORKS
logical unit: iocrk
namelist: crkcon
filename: zcnnnid, where zc is the common prefix to all cork files, nnn and id are as defined for restart dumps.
format: ascii

3.8 Display dumpfiles

Display dumps are single ascii files (maximum of 80 characters per line) which contains a quantitative display (matrix format) of a specified portion of various 2-D slices through any of many variables at evenly spaced time slices during a simulation. The data are scaled and converted to integers before being written to the ascii file. The dynamic range of the scaled
data depends on the specified “width” of the field of view (no more than 38), and ranges from $10^2$ to $10^6$. For very small widths ($\leq 8$), the data are not scaled and written as real numbers, with three or four significant figures. This utility is much like PRTIM in AIPS, for those familiar with the Astronomical Image Processing System. Its primary use is in debugging, or when one needs to view a small portion of data quantitatively and simultaneously.

**EDITOR** definition: DISP
logical unit: iodis
namelist: discon
filename: zdnnnid, where zd is the common prefix to all display files, nnn and id are as defined for restart dumps.

format: ascii

### 3.9 RADIO dumps

RADIO\textsuperscript{6} dumps are similar to the pixel dumps/movie frames (§3.4), but contain line-of-sight integrations of various quantities rather than 2-D slices through the data volume. In this release, RADIO dumps are possible in both Cartesian (XYZ) and cylindrical (ZRP) coordinates, though the latter are not fully tested. The integrands include all three Stokes parameters and numerous other scalars (e.g., density, Mach numbers, bremsstrahlung, etc.) and are integrated using a very fast binning algorithm that is as much as 50 times faster than traditional direct ray-tracing algorithms.

As with pixel/movie files, RADIO files may be written in one, two, or all three of three formats: real*4 “movie frame” (radfmt=1, new to version 3.6); Mike Norman’s char*1 “pixel dumps” (radfmt=2); and HDF4 (radfmt=3), with the latter suitable for graphical analysis if desired.

**EDITOR** definition: RADIO
logical unit: iorad
namelist: radcon
filename: zR**nnnid{.f}, where zR is the common prefix to all RADIO dumps, and where **, nnn, id, and f are as defined for pixel dumps (§3.4). On exit, movie files (radfmt=1) are converted to PPM, used to create a movie file zR**id.mm.mpg, and assembled in a tarball zR**id.mm.tar.gz.

format:
- radfmt=1: unformatted, four bytes per datum plus header
- radfmt=2: unformatted, one byte per datum, no header
- radfmt=3: HDF4

If, after the run is complete, the user needs RADIO dumps at different vantage points than selected before execution and/or animations of certain stills rotating about an axis, these can be created after the run using the stand-alone ancillary program RADIO, provided

\textsuperscript{6}The original post-processing program, RADIO, was designed to take line-of-sight integrations through an MHD datacube to compute the Stokes parameters, and thus mimic radio observations from telescopes such as the VLA, whence the name.
the user has created a total HDF dump at or near the desired problem time. See the 
User Manual for ZEUS Ancillaries found in the manuals directory of dzeus36.tar from 
www.ica.smu.ca/zeus3d for details.

3.10 Message log files

The message log file contains all the messages that are written to the terminal by the code 
during execution. In addition, the grid and all the values of the namelist parameters specified 
in the file inzeus are dumped here. It serves as the log for the execution.

EDITOR definition: none
logical unit: iolog
namelist: none
filename: zlnnnid, where zl is the common prefix to all log files, nnn and 
id are as defined for restart dumps.
format: ascii

3.11 Userdump

USERDUMP is an EDITOR alias available for the user to include their own special type of I/O 
which may be desired in addition to those currently available. See §5 for details on how to 
add subroutines to the code.

EDITOR definition: none
logical unit: iousr
namelist: usrcn
filename: zuinnid, where zu is the common prefix to all user dump files, 
nnn and id are as defined for restart dumps.
format:  

3.12 Recognised plotting variables

Table 3.1 below and continued on the following page lists the two-character variable repre-
sentations [corresponding to the double asterisks (**) used in §3.4, §3.5, and §3.9 above] used 
for generating the filenames for pixel (P), HDF (H), and RADIO (R) dumps. These two-
character representations are identical to those used to specify the variables to be dumped 
(see pixvar in namelist pixcon, hdfvar in namelist hdfcon, and radvar in namelist radcon, 
App. B) with the exception that variables specified by a single character (e.g., d) appear 
with a trailing underscore (e.g., d_1) in the dump file name. The third column indicates the 
I/O types in which the variable may be dumped.
<table>
<thead>
<tr>
<th>**</th>
<th>Variable</th>
<th>Dumps</th>
<th>**</th>
<th>Variable</th>
<th>Dumps</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_</td>
<td>vector potential norm</td>
<td>PH</td>
<td>p2</td>
<td>magnetic pressure</td>
<td>PH</td>
</tr>
<tr>
<td>a1</td>
<td>1-vector potential</td>
<td>PH</td>
<td>p3</td>
<td>1st thermal + mag. pres.</td>
<td>PH</td>
</tr>
<tr>
<td>a2</td>
<td>2-vector potential</td>
<td>PH</td>
<td>p4</td>
<td>2nd thermal pressure</td>
<td>PH</td>
</tr>
<tr>
<td>a3</td>
<td>3-vector potential</td>
<td>PH</td>
<td>p5</td>
<td>1st + 2nd thermal pres.</td>
<td>PH</td>
</tr>
<tr>
<td>ag</td>
<td>synchrotron age</td>
<td>PH</td>
<td>p6</td>
<td>2nd thermal + mag. pres.</td>
<td>PH</td>
</tr>
<tr>
<td>an</td>
<td>normal vector pot.</td>
<td>P</td>
<td>p7</td>
<td>1st + 2nd + mag. pres.</td>
<td>PH</td>
</tr>
<tr>
<td>ap</td>
<td>poloidal vector pot.</td>
<td>P</td>
<td>pa</td>
<td>pitch angle; tan^{-1}(B_1/B_\phi)</td>
<td>P</td>
</tr>
<tr>
<td>b_</td>
<td>magnetic field norm</td>
<td>PH</td>
<td>pg</td>
<td>pseudo-grav. potential</td>
<td>PH</td>
</tr>
<tr>
<td>b1</td>
<td>1-magnetic field</td>
<td>PH</td>
<td>s1</td>
<td>1-momentum</td>
<td>PH</td>
</tr>
<tr>
<td>b2</td>
<td>2-magnetic field</td>
<td>PH</td>
<td>s2</td>
<td>2-momentum</td>
<td>PH</td>
</tr>
<tr>
<td>b3</td>
<td>3-magnetic field</td>
<td>PH</td>
<td>s3</td>
<td>3-momentum</td>
<td>PH</td>
</tr>
<tr>
<td>bP</td>
<td>φ-magnetic field</td>
<td>P</td>
<td>sd</td>
<td>skew-density</td>
<td>P</td>
</tr>
<tr>
<td>bR</td>
<td>radial magnetic field</td>
<td>P</td>
<td>sn</td>
<td>normal momentum</td>
<td>P</td>
</tr>
<tr>
<td>bn</td>
<td>normal magnetic field</td>
<td>P</td>
<td>sp</td>
<td>poloidal momentum</td>
<td>P</td>
</tr>
<tr>
<td>bp</td>
<td>poloidal magnetic field</td>
<td>P</td>
<td>sy</td>
<td>synchrotron emissivity</td>
<td>P</td>
</tr>
<tr>
<td>bt</td>
<td>plasma beta = 2p/B^2</td>
<td>PH</td>
<td>to</td>
<td>all field arrays</td>
<td>H</td>
</tr>
<tr>
<td>cs</td>
<td>sound speed</td>
<td>P</td>
<td>u1</td>
<td>1st specific int. energy</td>
<td>PH</td>
</tr>
<tr>
<td>d_</td>
<td>density</td>
<td>PH</td>
<td>u2</td>
<td>2nd specific int. energy</td>
<td>PH</td>
</tr>
<tr>
<td>da</td>
<td>synchrotron density-age</td>
<td>PH</td>
<td>v_</td>
<td>velocity norm (speed)</td>
<td>PH</td>
</tr>
<tr>
<td>e1</td>
<td>first internal energy</td>
<td>PH</td>
<td>v1</td>
<td>1-velocity</td>
<td>PH</td>
</tr>
<tr>
<td>e2</td>
<td>second internal energy</td>
<td>PH</td>
<td>v2</td>
<td>2-velocity</td>
<td>PH</td>
</tr>
<tr>
<td>er</td>
<td>radiation energy density</td>
<td>PH</td>
<td>v3</td>
<td>3-velocity</td>
<td>PH</td>
</tr>
<tr>
<td>et</td>
<td>total energy density</td>
<td>PH</td>
<td>vA</td>
<td>Alfvén speed</td>
<td>P</td>
</tr>
<tr>
<td>fn</td>
<td>normal flux function</td>
<td>P</td>
<td>vf</td>
<td>fast magnetosonic speed</td>
<td>P</td>
</tr>
<tr>
<td>gp</td>
<td>gravitational potential</td>
<td>PH</td>
<td>vn</td>
<td>normal velocity</td>
<td>P</td>
</tr>
<tr>
<td>j_</td>
<td>current density norm</td>
<td>PH</td>
<td>vp</td>
<td>poloidal velocity</td>
<td>P</td>
</tr>
<tr>
<td>j1</td>
<td>1-current density</td>
<td>PH</td>
<td>vs</td>
<td>slow magnetosonic speed</td>
<td>P</td>
</tr>
<tr>
<td>j2</td>
<td>2-current density</td>
<td>PH</td>
<td>vv</td>
<td>( \nabla \cdot \vec{v} )</td>
<td>PH</td>
</tr>
<tr>
<td>j3</td>
<td>3-current density</td>
<td>PH</td>
<td>w_</td>
<td>vorticity norm</td>
<td>PH</td>
</tr>
<tr>
<td>jn</td>
<td>normal current density</td>
<td>P</td>
<td>w1</td>
<td>1-vorticity</td>
<td>PH</td>
</tr>
<tr>
<td>jp</td>
<td>poloidal current density</td>
<td>P</td>
<td>w2</td>
<td>2-vorticity</td>
<td>PH</td>
</tr>
<tr>
<td>k1</td>
<td>first specific entropy</td>
<td>PH</td>
<td>w3</td>
<td>3-vorticity</td>
<td>PH</td>
</tr>
<tr>
<td>k2</td>
<td>second specific entropy</td>
<td>PH</td>
<td>wn</td>
<td>normal vorticity</td>
<td>P</td>
</tr>
<tr>
<td>ka</td>
<td>averaged specific entropy</td>
<td>PH</td>
<td>wp</td>
<td>poloidal vorticity</td>
<td>P</td>
</tr>
<tr>
<td>lu</td>
<td>radiative luminosity</td>
<td>PH</td>
<td>A_</td>
<td>pol’n position angle</td>
<td>R</td>
</tr>
<tr>
<td>m_</td>
<td>Mach number</td>
<td>PH</td>
<td>AV</td>
<td>A with pol’n vectors</td>
<td>R</td>
</tr>
<tr>
<td>ma</td>
<td>Alfvénic Mach number</td>
<td>PH</td>
<td>B_</td>
<td>magnetic field strength</td>
<td>R</td>
</tr>
<tr>
<td>mf</td>
<td>fast magnetosonic number</td>
<td>PH</td>
<td>BR</td>
<td>bremsstrahlung</td>
<td>R</td>
</tr>
<tr>
<td>ms</td>
<td>slow magnetosonic number</td>
<td>PH</td>
<td>D_</td>
<td>density</td>
<td>R</td>
</tr>
<tr>
<td>nb</td>
<td>break frequency</td>
<td>P</td>
<td>E1</td>
<td>1st internal energy</td>
<td>R</td>
</tr>
<tr>
<td>p1</td>
<td>1st thermal pressure</td>
<td>PH</td>
<td>F_</td>
<td>fractional pol’n</td>
<td>R</td>
</tr>
</tbody>
</table>
Table 3.1, continued. Two Character Variable Representations

<table>
<thead>
<tr>
<th>**</th>
<th>Variable</th>
<th>Dumps</th>
<th>**</th>
<th>Variable</th>
<th>Dumps</th>
</tr>
</thead>
<tbody>
<tr>
<td>FV</td>
<td>F with pol’n vectors</td>
<td>R</td>
<td>P_</td>
<td>polarised intensity</td>
<td>R</td>
</tr>
<tr>
<td>I_</td>
<td>total intensity</td>
<td>R</td>
<td>PV</td>
<td>P with pol’n vectors</td>
<td>R</td>
</tr>
<tr>
<td>IV</td>
<td>I with pol’n vectors</td>
<td>R</td>
<td>SH</td>
<td>scalar velocity shear</td>
<td>R</td>
</tr>
<tr>
<td>M_</td>
<td>Mach number</td>
<td>R</td>
<td>U1</td>
<td>1st specific int. energy</td>
<td>R</td>
</tr>
<tr>
<td>MA</td>
<td>Alfvénic Mach number</td>
<td>R</td>
<td>V_</td>
<td>pol’n vectors (black)</td>
<td>R</td>
</tr>
<tr>
<td>MF</td>
<td>fast magnetosonic num.</td>
<td>R</td>
<td>VR</td>
<td>pol’n vectors (white)</td>
<td>R</td>
</tr>
<tr>
<td>MS</td>
<td>slow magnetosonic num.</td>
<td>R</td>
<td>VV</td>
<td>\nabla \cdot \vec{v}</td>
<td>R</td>
</tr>
</tbody>
</table>
4 Interacting with ZEUS-3D

During an interactive execution (as opposed to batch), the user may probe ZEUS-3D for its status, change select input parameters, and submit instructions to create a dump, stop, pause, resume, etc. This is done by typing a recognised three-character “interrupt message” followed by a carriage return. Once every “time step”, ZEUS-3D “glances” at the terminal buffer (by virtue of the lone C routine checkin.c introduced in §2.3.2). If an interrupt message has been entered, ZEUS-3D will carry out the instruction. If no interrupt message is found, execution proceeds without pause. Below is a list of the interrupt messages recognised by ZEUS-3D, along with a brief description of their function. Only the first three characters of each command (those in typewriter font) need be entered. Note that there are several synonyms for a number of the commands, which are separated by commas.

On some batch systems, having the code checking the terminal buffer can interfere with execution. In this case, one can turn this feature off by setting intractv=0 in namelist iocon (App. B).

Controlling execution:

- **time**, **cycle**, **status**, **t**, **n**, **?**  
  prints a time and cycle report, then resumes execution

- **quit**, **abort**, **crash**, **break**  
  immediate emergency termination, no final dumps are made

- **stop**, **end**, **exit**, **finish**, **terminate**  
  clean stop—all final dumps are made

- **halt**, **pause**, **wait**, **interrupt**  
  halt execution and wait for a message from the crt or controller.

- **restart**, **go**  
  restarts execution after a halt

- **tlimit**, **tfinish** (followed by a real number)  
  resets the physical (problem) time limit (when computation will stop)

- **nlimit**, **nfinish** (followed by an integer)  
  resets the cycle limit

- **ttotal**, **tcpu** (followed by an integer number of seconds)  
  resets maximum cpu time to consume.

- **tsave**, **treset** (followed by an integer number of seconds)  
  resets the save time reserved for cleanup and termination

Controlling data output:

- **dump**  
  creates a restart dump at current time
• dtdmp (followed by a real time interval) resets the problem time interval between restart dumps
• pl1 creates a 1-D plot at current time
• dt1 (followed by a real time interval) resets the problem time interval between 1-D plots
• pl2 creates a 2-D plot at current time
• dt2 (followed by a real time interval) resets the problem time interval between 2-D plots
• pixel creates a pixel dump at current time
• dtpix (followed by a real time interval) resets the problem time between pixel dumps
• usr creates a user dump (calls USERDUMP) at current time
• dtusr (followed by a real time interval) resets the problem time between user dumps
• hdf creates an HDF dump at current time
• dth (followed by a real time interval) resets the problem time between HDF dumps
• tslice adds a time slice dump at current time to time slice file
• dtttslice (followed by a real time interval) $> 0 \Rightarrow$ resets the problem time between time slice ascii dumps $< 0 \Rightarrow$ resets the problem time between time slice plot dump
• display adds a display dump at current time to display dump file
• dty (followed by a real time interval) resets the problem time between display dumps
• radio creates a radio dump at current time
• dtradio (followed by a real time interval) resets the problem time between radio dumps
• corks creates a cork dump at current time
• `dtcorks` (followed by a real time interval)
  resets the problem time between cork dumps

• `1ddiagnostics`
  creates a 1-D diagnostic dump at current time (for programmers only)

• `1dt` (followed by a real time interval)
  resets the problem time between 1-D diagnostic dumps

• `2ddiagnostics`
  creates a 2-D diagnostic dump at current time (for programmers only)

• `2dt` (followed by a real time interval)
  resets the problem time between 2-D diagnostic dumps
5 Adding source code to ZEUS-3D

5.1 Adding an entire subroutine

Adding source code to the ZEUS-3D package is not as difficult as one might think, especially if all one wants to do is add new subroutines or replace existing ones. Below is the subroutine myprob which can be used as a template to create a problem generator. A soft copy of myprob may be found in the zeus directory of dzeus36.tar from www.ica.smu.ca/zeus3d. The style is that which is used for all subroutines currently in dzeus36.

*insert zeus3d.9999
*deck myprob

```fortran
subroutine myprob
abcde:zeus3d.myprob <------------------------ initialises my problem
 september, 1990

written by: A Busy Code Developer
 modified 1: July, 2006 by ABCD; updated for dzeus35
 modified 2: October, 2010 by ABCD; updated for dzeus36
 modified 3: November, 2014 by ABCD; added div(B) check.

PURPOSE: Initialises all the flow variables for my problem. More
description of my problem can go here. Boundary values are set in
the calling routine immediately after MYPROB is called.

INPUT VARIABLES:

OUTPUT VARIABLES:

LOCAL VARIABLES:

EXTERNALS:
 DIVERG

*call comvar
```

The 1-D and 2-D arrays "array1d" and "array2d" are never used and
are included only to show how arrays can be declared and equivalenced
to "global worker arrays". The 3-D array "divb" is used at the end
c of the template to check on whether the magnetic field as set up
c satisfies the solenoidal condition.
c
real*8    array1d (ijkx)
real*8    array2d (idim,jdim)
real*8    divb    ( in, jn, kn)
c
equivalence ( array1d , wa1d  )
equivalence ( array2d , wa2d  )
equivalence ( divb , wa3d  )
c
c external    diverg
c
-----------------------------------------------------------------------
c
Input parameters:

da , db    array and boundary values for density
e1a, e1b    array and boundary values for first internal energy
e2a, e2b    array and boundary values for second internal energy
v1a, v1b    array and boundary values for 1-velocity
v2a, v2b    array and boundary values for 2-velocity
v3a, v3b    array and boundary values for 3-velocity
b1a, b1b    array and boundary values for 1-magnetic field
b2a, b2b    array and boundary values for 2-magnetic field
b3a, b3b    array and boundary values for 3-magnetic field

c      namelist / pgen /
1     da , db , e1a , e1b ,  e2a
2             , e2b , v1a , v1b , v2a , v2b
3             , v3a , b1a , b1b , b2a , b2b
4             , b3a , b3b
c
Default values
c
da = 1.0d0
db = 0.1d0
e1a = 0.9d0
e1b = 0.9d0
e2a = 0.0d0
e2b = 0.0d0
v1a = 0.0d0
v1b = 0.0d0
v2a = 0.0d0
v2b = 0.0d0
v3a = 0.0d0
v3b = 0.0d0
b1a = 0.0d0
b1b = 0.0d0
b2a = 0.0d0
b2b = 0.0d0
b3a = 0.0d0
b3b = 0.0d0
c
Read namelist pgen, and store contents to log file.
c
read (ioin , pgen)
write (iolog, pgen)
Adding source code to ZEUS-3D

Set field arrays. Metric factors in the magnetic field settings are necessary to preserve the solenoidal condition.

do 30 k=ksm2,kep3
   do 20 j=jsm2,jep3
      do 10 i=ism2,iep3
         d(i,j,k) = da
         v1(i,j,k) = v1a
         v2(i,j,k) = v2a
         v3(i,j,k) = v3a
         *if -def,ISO
            e1(i,j,k) = e1a
         *endif -ISO
      *if def,TWOFLUID
         e2(i,j,k) = e2a
      *endif TWOFLUID
      *if def,MHD
         b1(i,j,k) = b1a
         b2(i,j,k) = b2a
         b3(i,j,k) = b3a
      *endif MHD
      10 continue
   20 continue
30 continue
*if -def,ISYM
   Set inflow boundary arrays. Setting niib=10 imposes inflow boundary conditions, for which quantities such as diib1, etc., set the desired inflow variable values.

*if def,MHD
   q11 = ( v2b * b3b - v3b * b2b ) * dx1a(ism1)
   q12 = ( v2b * b3b - v3b * b2b ) * dx1a(ism2)
   q2 = ( v3b * b1b - v1b * b3b ) * h2a(is)
   q3 = ( v1b * b2b - v2b * b1b ) * h31a(is)
*endif MHD
do 50 k=ksm2,kep3
   do 40 j=jsm2,jep3
      niib(j,k) = 10
      diib1(j,k) = db
      diib2(j,k) = db
      v1iib1(j,k) = v1b
      v1iib2(j,k) = v1b
      v1iib3(j,k) = v1b
      v2iib1(j,k) = v2b
      v2iib2(j,k) = v2b
      v3iib1(j,k) = v2b
      v3iib2(j,k) = v2b
      *if -def,ISO
         e1iib1(j,k) = e1b
         e1iib2(j,k) = e1b
      *endif -ISO
      *if def,TWOFLUID
         e2iib1(j,k) = e2b
         e2iib2(j,k) = e2b
      *endif TWOFLUID
      *if def,MHD
         b2iib1(j,k) = b2b
         b2iib2(j,k) = b2b
b3iib1 (j,k) = b3b
b3iib2 (j,k) = b3b
emf1iib1(j,k) = q11
emf1iib2(j,k) = q12
emf2iib1(j,k) = q2 * dx2a(j)
emf3iib1(j,k) = q3 * dx3a(k) * h32a(j)
*endif MHD
40 continue
50 continue
*endif -ISYM
*if def,MHD
  c
  c As a check, compute div(B), and integrate it over the grid
  c ("sumdivb"). The fourth parameter in the calling list for DIVERG is
  c set to 1 and thus the returned values "divb" and "sumdivb" will be
  c normalised by the specified magnetic field configuration and the
  c grid. Therefore, "sumdivb" should be of order 10**-12 or less (e.g.,
  c machine round off) for the field to be divergence-free.
  c
  call diverg ( b1, b2, b3, 1, 1, divb, sumdivb )
*else MHD
  sumdivb = 0.0d0
*endif MHD
  c
  if (iotty .gt. 0) write (iotty, 2010) sumdivb
  if (iolog .gt. 0) write (iolog, 2010) sumdivb
  c
  c-----------------------------------------------------------------------
  c----------------------- Write format statements -----------------------
  c-----------------------------------------------------------------------
  c
  2010 format('MYPROB : Initialisation complete; normalised div(B) = ',
        1 ,1pg12.5,'.')
  c
  return
end
 c
 c=======================================================================
 c\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\n
There are many ingredients to this template which warrant discussion. In order of appearance, these are:

1. Ignoring for the moment the *EDITOR statement *insert zeus3d.9999, the first line of each subroutine must be an *EDITOR *deck (*dk for short) statement. Without this statement, the precompiler won’t put the subroutine into a separate file, inhibiting the debugger should it be necessary. It is easiest, although not necessary, to give the deck the same name as the subroutine.

2. Note that there is no parameter list in the subroutine statement. A parameter list is unnecessary since all variables that need to be used and/or set are accessible via
the common blocks. In fact, using a parameter list would inhibit the inclusion of a user-supplied subroutine using the present structure of the code.

3. All of the important variables declared in `dzeus36` are in common blocks, and can be included into a subroutine simply by inserting the `EDITOR` statement `*call comvar` just before the local declarations are made. The `EDITOR` `*call` (*ca for short) statement is much like `INCLUDE` whereby a section of code known as a “common deck” (called `comvar` in this case) is inserted at the location of the `*call` statement. Every variable of any possible interest is declared in `comvar`, including many that the user would never need. (A description of the most widely used variables is given in App. C.) At the beginning of `comvar` is an “implicit none” statement, which requires that the attributes (`real*8`, `integer`, etc.) of all variables used in the subroutine be declared. Note that should the user inadvertently try to use a variable name already declared in `comvar`, the compiler will flag the repetition and abort compilation. While the “implicit none” does not require that all externals called by the program unit be declared in an `external` statement, it is still good practice to do so. In fact, if undeclared externals appear inside a nested do-loop construct, this may inhibit `EDITOR`’s auto-tasking feature (parameter `iutask`; see §2.3).

4. Should local 1-D arrays be needed to store data at grid points along an axis, it is best to declare them with dimension `(ijkx)`, such as `array1d` in the template. The parameter `ijkx` is declared and defined in `comvar` as the largest of `in`, `jn`, and `kn` (the dimensions of the 3-D arrays), also declared in `comvar`. So that no additional memory is occupied by this local array, it can be equivalenced to one of the 26 1-D scratch arrays declared in `comvar`, as done in the template. Similarly, local 2-D arrays (e.g., `array2d`) should be dimensioned `(idim,jdim)` (where `idim` and `jdim` are set by `comvar` to one of `in`, `jn`, and/or `kn`, depending on the defined symmetries) and equivalenced to one of the 2-D worker arrays. Finally, local 3-D arrays can be declared and equivalenced as exemplified by `divb` in the template. The names of all the scratch arrays (1-D, 2-D, and 3-D) are given in §C.4 and their dimensions are defined in §C.6.

5. The namelist `pgen` is reserved for the namelist in the Problem GENerator. Of course, any name other than `pgen` could be used, so long as it is not already used in the input deck `inzeus` and the new name for the namelist is substituted for `pgen` in `inzeus`. Note how default values for the input parameters can be assigned before the namelist is read.

6. Loop 30 is a typical way the 3-D field variables (d = density, e1 = first internal energy per unit volume, etc.) are assigned values. In this very simple case, the variables are assigned to the scalars read from the namelist `pgen`. Note that all energy variables (e.g., `e1`, `e1iib1`, etc.) should be considered “pressure-like” (energy per unit volume) and not “temperature-like” (energy per unit mass). Appendix C has a list of all the variable names and their dimensions. The do-loop indices declared in `comvar` are all assigned values in the subroutine `nmlsts` which is called immediately before the user’s problem generator (`PROBLEM`) is called (see App. A) and so they can be used explicitly in any user-supplied subroutine called thereafter. Thus, the index for loop 30 (`k`)
ranges from \texttt{ksmm2} (k-start minimum minus 2) to \texttt{kemxp3} (k-end maximum plus 3), which includes all boundary zones. This is particularly important for the magnetic field variables. Similarly for the indices of loops 20 (j) and 10 (i). Note the use of the \texttt{EDITOR *if define, *endif} (*if def, *ei for short) structure which conditionally includes or excludes a segment of coding depending on whether, in this case, MHD was defined during precompilation. Similar conditionals can be based on the “truth” of any \texttt{EDITOR} definition, and on how aliases are set. For example, one could place an \texttt{EDITOR *if alias PROBLEM.eq.myprob} just after the \texttt{subroutine} statement, and the matching \texttt{*endif} just before the \texttt{return} statement. In this way, the subroutine would be empty (nothing between the \texttt{subroutine} and \texttt{return} statements) unless the \texttt{EDITOR} alias \texttt{PROBLEM} were set to \texttt{myprob}. This would prevent it from being compiled when it is not needed.

7. Loop 50 illustrates how inflow boundary values (to be applied only to those boundary zones where matter is flowing into the grid in a known fashion) can be set for supermagnetosonic flow. (See \S\S 1.5 and B.8 for variations required for submagnetosonic inflow conditions.) In this case, the “inner-i-boundary” (iib) values of the flow variables are being initialised. Alternatively, one could set the in-flow boundary values as input parameters using the namelists \texttt{iib, oib, etc.} (\S B.8, \S B.9, etc.). Note the use of the \texttt{EDITOR *if define, *endif} construct to prevent this loop from being compiled in the event that ISYM is defined. If ISYM has been defined, the variables \texttt{niib, etc.} are not declared in \texttt{comvar}. Variables that are conditionally declared (depending on which \texttt{EDITOR} definitions are set) are noted in App. C.

8. Note the use of the \texttt{ZEUS}-subroutine \texttt{DIVERG} which, in this case, computes the normalised average magnetic field divergence as initialised by the user. This quantity (\texttt{sumdivb}) should be of the order $10^{-12}$ if the initialised magnetic field is to be declared solenoidal. Note carefully that the MHD algorithm in \texttt{ZEUS-3D} is designed to conserve $\nabla \cdot \vec{B}$ to machine round-off error, not to ensure that $\nabla \cdot \vec{B}$ is specifically zero. Thus, if the user initialises $\vec{B}$ with a non-zero divergence, the MHD algorithm will diligently conserve that divergence to machine round-off error throughout the simulation. Therefore, if the user wants a divergence-free field, it must be initialised that way.

9. Finally, if desired, the user can write various messages to the terminal (logical unit \texttt{iotty}) or to the message log file (logical unit \texttt{iolog}). Both \texttt{iotty} and \texttt{iolog} are declared in \texttt{comvar} and set by the subroutine \texttt{mstart}, and thus available in \texttt{PROBLEM} so long as this subroutine starts off with \texttt{*ca comvar} as exemplified in \texttt{myprob}.

Once the subroutine is written, it should be placed in its entirety into a change deck called, for example, \texttt{chguser} and the line **\texttt{read chguser} in the script file \texttt{dzeus36.s} should be “de-commented” by deleting one of the asterisks (\S 2.3). Upon its first pass (the merge step), the preprocessor will, in this case, insert the user’s subroutine into \texttt{dzeus36} immediately after line 9,999 of the main program \texttt{zeus3d} (by virtue of the \texttt{EDITOR} statement \texttt{*insert zeus3d.9999} appearing at the top of the subroutine template). Since \texttt{zeus3d} doesn’t have 9,999 lines, \texttt{EDITOR} will simply stick the subroutine after the last line of the main program. It doesn’t matter where in \texttt{dzeus36} the subroutine gets inserted so long as it
isn’t in the middle of an existing subroutine (deck). Immediately after the main program is as good a place as any. Upon the second pass, the precompiler will find the user’s subroutines and treat them as it would any other it encounters. Thus, if there are any EDITOR commands in the user’s routines (such as *call comvar, *if define,MHD), they will be carried out and then expunged from the working copy of the source code. The user’s subroutine will then be placed in its own file in the directory dzeus3.6, and the name of the subroutine will be included in the makefile makezeus which will then compile the subroutine and link it with the rest of the object files and libraries. Provided the EDITOR alias PROBLEM has been set to myprob (or whatever it’s called) in the macro file zeus36.mac, the user’s problem generator will be called at the appropriate time during execution. Similarly, if the subroutine should be called at the location of any of the other available “plugs” in the code, set the appropriate alias (i.e. SPECIAL, SPECIALSRC, USERSOURCE, SPECIALTRN, USERDUMP, PROBLEM, PROBLEMRESTART, or FINISH; see §2.2.2 and the ZEUS-3D skeleton in App. A) in zeus36.mac to the subroutine name.

5.2 Microsurgery using EDITOR

For the truly adventurous, it is possible to alter individual lines of code in dzeus36 without actually changing the original source code. In this way, the changes made can be kept separate from the code, and thus not lost in the abyss of dzeus36. In addition, the user’s changes could, in principle, be incorporated into the master code at a later date and become part of the next release. To do this, there are two things required: an EDITOR listing of the code and a short tutorial on how to use EDITOR. For those who have worked with HISTORIAN, all this should seem very familiar. For those who haven’t, take heart—the structure is very intuitive. The real problem will be ensuring that the changes made don’t break something else in the code. This is where the headaches will lie, and those who really want to change the code do so at their own peril!

To get an EDITOR listing of the code, copy the script file number.s (a soft-copy of which may be found in the editor directory of dzeus36.tar from www.ica.smu.ca/zeus3d):

```bash
#============= SOURCE FILE TO CREATE A NUMBERED LISTING ===============#
#
#=======================================> Get files from home directory.
if(! -e ./xedit22) cp ../editor/xedit22 .
#-----------------------> Create the input deck for EDITOR, and execute.
rm -f ./inedit
cat << EOF > ./inedit
$editpar inname='dzeus36'
  , ibanner=1, job=1, inumber=3, itable=1, ixclude=1
EOF
chmod 755 ./xedit22
./xedit22
#=================================================> Tidy up directory.
rm -f ./editlp ./inedit ./output ./xedit22

to the directory in which dzeus36 lives, and run it by typing:
csh -v number.s
```
Adding source code to ZEUS-3D

This script file will fire up EDITOR in its numbering mode (job=1), and produce a listing with a table of contents, and various labels on each line. The numbered file will be called dzeus36.n, and can be viewed in a wide (132 character) window. Printed copy is not recommended; at 60 lines per page, there will be more than 1,800 pages of output! The third column to the right of the source listing is the number of lines since the most recent EDITOR *deck or *cdeck statement. This is the column needed to perform microsurgery on the master file.

During preprocessing, EDITOR makes two major passes over the code. The first pass does the merging of the change deck chgzeus (which contains zeus36.mac and possibly chguser) into the main code. EDITOR commands performed during this pass include:

1. *insert deckname.n—ininserts text immediately following the *insert command into the source code directly after line \(n\) in deck (or cdeck: common deck) deckname. The value of \(n\) is determined from the third column to the right of the source code in the numbered listing, dzeus36.n.

2. *delete deckname.n,m—deletes lines \(n\) through \(m\) in deck (or cdeck) deckname, and replaces it with the text immediately following the *delete command, if any. Note that \(m\) must be greater than \(n\). If \(m\) is missing altogether, then \(m = n\) will be assumed.

That’s it. An example:

\[
*\text{delete zeus3d.10,20} \\
\quad a = b \\
\quad b = c \\
*\text{insert mstart.100} \\
\quad d(i,j,k) = 1.0 \\
*i zeus3d.100 \\
\quad c = d \\
*d zeus3d.120
\]

Note that *d and *i are short forms for *delete and *insert respectively. In addition, *replace (*rp for short) is a synonym for *delete. In the example, lines 10 through 20 in the main program zeus3d are replaced with the two lines which set \(a\) and \(b\), a single line setting \(d(i,j,k)\) is inserted after line 100 in subroutine mstart, a single line setting \(c\) is inserted after line 100 in zeus3d, and line 120 in zeus3d is simply deleted.

To aid the user in deciding what changes to make and where to make them, a flow chart showing the sequence of the major subroutine calls in ZEUS-3D is given in App. A. This will be particularly useful once faced with the task of comprehending the source code listing, dzeus36.n.

If EDITOR detects any merge syntax errors or conflicts during the merge, it will write the merged file [as best as could be done given the error(s) detected] into a file named dzeus36.m and insert an error message immediately after each offending line. A merge error will prevent the second pass of preprocessing (i.e., precompilation) from being executed and the user will be told what character pattern to search for in the file dzeus36.m in order to find the generated error messages.

Should the merge step be successful, EDITOR goes through a second pass and performs all the precompilation commands. These include:
1. *if define, macro—the following source code is kept provided the macro is defined by a *define statement somewhere in the file.

2. *if -define, macro—the following source code is kept provided the macro is not defined by a *define statement somewhere in the file.

3. *if def,.not. macro—same as 2. Note that def is an acceptable short form for define.

4. *if def, macro1 .and. macro2—the following source code is kept provided both macros are defined by a *def statement somewhere in the file.

5. *if def, macro1 .or. macro2—the following source code is kept provided either macro is defined by a *def statement somewhere in the file.

6. *if alias macro.eq. phrase—the following source code is kept provided the alias macro has been set to the character string phrase by an *alias statement somewhere in the file.

7. *if alias macro.ne. phrase—the following source code is kept provided the alias macro has not been set to the character string phrase by an *alias statement somewhere in the file.

8. *else—the following source code is kept if the truth value of the previous *if is false.

9. *endif—closes the previous *if, *else structure. All source code following the *endif statement is not affected by the previous *if or *else statements. For every *if statement, there must be an *endif statement which follows.

10. *call deckname—including the contents of the common deck deckname at the location of the *call statement.

These precompiler commands can be used as part of the changes to be inserted into dzeus36 using the EDITOR *delete and *insert commands. All changes should be placed in the user’s change deck which, in our example, has been called chguser. These changes are then incorporated into the code by “de-commenting” the line **read chguser in the script file dzeus36.s by deleting one of the asterisks (§2.3).

Note that during both passes, the *deck and *cdeck statements are used as reference points, and are then expunged from the source code during the second pass. If any precompilation syntax errors are detected, EDITOR will write the precompiled file [as best as could be done given the error(s) detected] into a file named dzeus36.f and insert an error message immediately after each offending line. EDITOR will abort further processing and the user will be told what character pattern to search for in the file dzeus36.f in order to find the generated error messages. On the other hand, if the precompilation pass is successful, EDITOR will make yet another pass through the code to substitute namelist statements with subroutine calls, perform auto-tasking, update the files in the directory dzeus3.6, and create the makefile, makezeus. This makefile compiles only those subroutines affected by the
Adding source code to ZEUS-3D

changes made, links all the subroutines and libraries together, and creates the new executable
xdzeus36.

A complete discussion of EDITOR’s merge and precompilation features can be found in
the EDITOR user manual edit22_man.ps found in the directory manuals of dzeus36.tar
from www.ica.smu.ca/zeus3d.

5.3 Debugging in ZEUS-3D

It is the author’s experience that virtually no change of significance can be introduced into
the code without tripping up some problems requiring a debugger. And while the vast
majority of these bugs will eventually be traced back to the user’s initialisation routine or
other change made by the user, it will often necessitate probing other parts of the code to
find these problems. To someone not knowing their way around ZEUS-3D, this will come as
a daunting task indeed. Therefore, this section attempts to offer—in a generic way—some
guidance in starting a debugging session.

The debugger DBX, originally a Cray then Sun Microsystems product, can be used with
code compiled with f90 -g (which may be available on non Cray and non-Sun platforms;
check with your sysadmin). The debugger GDB may be used with code compiled with g95 -g
or gfortran -g. Other debuggers are required for other compilers. In any case, to prepare
xdzeus36 for a debugger, it should be compiled with options='debug'; see §2.3. One then
enters the debugging environment by typing:

dbx xdzeus36

(or whatever debugger is being used) at the UNIX prompt. From there the user can set
“breakpoints”, reassign variable values, and navigate pretty well anywhere within the code
probing variable values as one moves along. For the uninitiated, a very short three-page
primer on using dbx, dbxprimer.ps, may be found in the directory manuals of dzeus36.tar
from www.ica.smu.ca/zeus3d. For GDB and other debuggers, the user is directed to on-line
resources.

While not specific to this package, the following discussion assumes dbx to be the default
debugging environment.

1. Stop in your initialisation routine (PROBLEM)

The first task is to make sure all variables are set as you think they should be. Stop at the
return statement of your initialisation routine, and probe all variable values, particularly
those around the periphery of the grid where users often forget to initialise the flow variables.
90% of all bugs a user introduces into the code can be traced to flow variables (density,
velocity, etc.) not being assigned properly or fully. Make sure, for example, that each array
is set from 1:in, 1:jn, and 1:kn and not, for example, simply from is:ie, js:je, and
ks:ke (see App. C for a review of the variable and parameter names and definitions). In
addition, if any boundaries are set to “inflow” (nflo=10), the user will need to set the inflow
boundary arrays such as diib1, diib2, viib1, etc. (see the template routine myprob.f in
§5.1).

2. Stop in srcstep
Stopping at the top of routine \texttt{srcstep} (where the source terms are updated; see the \texttt{dzeus36} “skeleton” in App. A for guidance on how to navigate through the code) will allow you to make certain that once set in the initialisation routine, all variables have been passed to the beginning of the first MHD cycle correctly. If execution dies before reaching \texttt{srcstep}, it is possible there has been a problem in one of the graphics routines, and you should probe the variable values there.

If execution makes it correctly to the top of \texttt{srcstep}, one can advance through \texttt{srcstep} one call after the other, making sure that each of \texttt{stv1}, \texttt{stv2}, \texttt{stv3}, \texttt{viscous}, etc., is executing correctly.

3. \textit{Stop in trnsprt}

If \texttt{srcstep} reveals no anomalies, one next ventures into \texttt{trnsprt}, which takes care of the transport terms (fluxes), the induction equation, and the transverse Lorentz forces. Navigating through this routine is complicated by the fact that the order in which the constituent routines (\texttt{ctran1}, \texttt{ctran1}, \texttt{ctran3}, \texttt{ttran1}, \texttt{ttran2}, and \texttt{ttran2} for FIT; \texttt{tranx1}, \texttt{tranx2}, \texttt{tranx3}, \texttt{cmoc1}, \texttt{cmoc2}, and \texttt{cmoc3} for legacy transport) are executed depends on how many MHD cycles have been run through. This is an attempt to reduce any favouritism among the directions in the directional- and planar-split algorithms, but it can make debugging more challenging.

The variable controlling the order in which the constituent routines are called is the integer \texttt{ix1x2x3} which can take on any integral value between and including 1 and 6. Knowing the value of \texttt{ix1x2x3} will tell you to which segment of \texttt{trnsprt} you have to go.

With any luck, you will not have to venture into the \texttt{cmoc*} routines, as these are long with many local variables, most of which are scalars which can make debugging all the more painful.

4. \textit{Double-debug sessions}

For the really stubborn bugs, often I have to do a “double debug session” in which I have \texttt{dzeus36} without the changes compiled and open in a debug session in one window on the left side of my screen, and the code with the changes opened up in a debug session in another window on the right side of my screen. From there, I undertake the tedious task of advancing through the code routine by routine, line by line until I find where the two versions diverge, and go from there.

5. “Gotchas”

Depending on your installation, \texttt{dbx} is capable of a number of very annoying “gotchas” which can slow progress markedly. I mention a few below.

While any variable declared either globally or local to the subroutine should, in principle, be accessible (\textit{i.e.}, their values probed) from within the subroutine, this is not always the case. In some installations, only variables actually set or modified by the subroutine may be probed and, if you really need to see these variable values, one either has to go to a routine where these values are modified or, if that is insufficient, put a “dummy” assignment statement in the routine (\textit{e.g.}, \texttt{var = var + 0.0d0}), recompile, and restart the debug session.

In my own installation of \texttt{dbx}, local variables that are equivalenced in the subroutine
to a globally declared variable are often inaccessible by the local variable name—one has to use the global variable name to probe values. This, like the first “gotcha”, is a completely stupid design “feature”, but you may be stuck with it. Sometimes using the global variable name is no big deal. However, in some situations such as in the cmoc* routines where the local variable is not dimensioned the same as the global array [e.g., in cmoc1, local variable v2t(kn,jn) is equivalenced to wi2d(ijkx,ijkx) in 3-D], it isn’t always so simple to determine the indices needed for the global array to retrieve the desired element of the local array. In these situations, it may be necessary to comment out the equivalence statements and recompile, hoping that this doesn’t somehow affect the nature of the problems you are trying to uncover.

Finally, dbx will only report the first 15 decimal places of the variable (for double precision), and sometimes problems start to occur in the 16th decimal place or beyond. Even though values beyond the 15th or 16th significant figure are normally considered “noise”, unlike real noise computer noise is always repeatable and thus can serve as a useful indicator of when deviation from the correct answer begins. In dbx, for example, one can probe these additional digits by subtracting the reported result from the variable itself. Thus, if v1(i,j,k) is reported as 3.14159265358979, you could type:

\[
\text{print } v1(i,j,k) - 3.14159265358979
\]

which may, for example, then report:

\[
v1(i,j,k) - 3.14159265358979 = 2.5430056384790e-16
\]

which can be compared to another session to make sure differences aren’t creeping in at this extremely low but sometimes significant level. It is my experience that if dbx reports two number to be equal to 15 decimal places, they aren’t always equal. However, if the noises are also equal then the variable values can safely be taken as identical.

These are only a few general guidelines to debugging within dzeus36, and may cover 95% of the situations a typical user may encounter. If bugs or “undesired features” are found or strongly suspected in the code itself that are unrelated to changes introduced by the user, the user is encouraged to report these to the author at dclarke@ap.smu.ca or at the address given in the Preface.
6 Quick summary

This final section is intended to serve as a quick reference sheet for those who are already familiar with running ZEUS-3D.

1. Set the macros in the file zeus36.mac (§2.2 and App. A).

2. Make the necessary changes to the dzeus36.s script file, including the parameters in the change deck chgzeus (§2.3.4) and the input parameters in the input deck inzeus (§2.3.6 and App. B).

3. Put the desired source code changes, if any, into the file chguser (§5), and “de-comment” the line **read chguser in the script file dzeus36.s by deleting one of the asterisks (§2.3).

4. Run the script file to create the ZEUS-3D executable by typing csh -v dzeus36.s

5. Fire up the executable by either typing xdzeus36, or by submitting the job to the appropriate batch queue.
A The ZEUS-3D skeleton

A.1 Legacy transport

The following flowchart depicts "legacy transport" (hycon namelist parameter \texttt{trnrsn}=0; MHD, GRAV on, RADIATION off) as of November, 2013 with no further modifications anticipated. Modules in upper case are \textit{EDITOR} aliases set in \texttt{zeus36.mac}, with exemplary choices (appropriate for the 1-D Brio & Wu problem; \S 2.2, 2.3) given parenthetically. Modules in lower case are actual subroutine names in the source code.
A.2 Finely Interleaved Transport

New to version 3.6: The following flowchart depicts the FIT algorithm (hycon namelist parameter trnrse=1; MHD, GRAV on, RADIATION off; trnrse=2 engages the fully conservative variant of FIT) as of November, 2013, representing the most significant change to ZEUS transport since the introduction of CMoC in 1993, and designed to address the “striping instability”. Modules in upper case are EDITOR aliases set in zeus36.mac, with exemplary choices (appropriate for the 1-D Brio & Wu problem; §2.2, 2.3) given parenthetically. Modules in lower case are actual subroutine names in the source code.
A.3 EDITOR aliases

The “empty” routine empty (returns immediately to calling routine) is an option for just about any EDITOR alias macros (other than TRANSPORT, NEWTIMESTEP, and PROBLEM) that effectively “turns them off”. This subsection lists most of the other more substantive subroutines within ZEUS-3D that can be aliased to each macro, along with their intended purpose.

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNDYUPDATE</td>
<td>breset resets flow-in boundary values for some test problems</td>
</tr>
<tr>
<td></td>
<td>jetbndy generates magnetic field and wiggles at jet boundary</td>
</tr>
<tr>
<td></td>
<td>jetcork injects corks at jet inlet</td>
</tr>
<tr>
<td></td>
<td>krasdisc maintains “Krasnopolsky boundary conditions” (corona)</td>
</tr>
<tr>
<td></td>
<td>... user-defined module to maintain boundaries</td>
</tr>
<tr>
<td>EXTENDGRID</td>
<td>extend to extend computational domain</td>
</tr>
<tr>
<td>GRAVITY</td>
<td>gravity one of three Poisson solver algorithms may be chosen</td>
</tr>
<tr>
<td>SPECIAL</td>
<td>... user-defined module for additional physics</td>
</tr>
<tr>
<td>SOURCE</td>
<td>empty for advection tests</td>
</tr>
<tr>
<td></td>
<td>srcstep standard source term module</td>
</tr>
<tr>
<td>SPECIALSRC</td>
<td>twinjet sustains two outflowing jets from “fertile zones”</td>
</tr>
<tr>
<td></td>
<td>resetalf for launching Alfvén waves from the boundary (ALFVTST)</td>
</tr>
<tr>
<td></td>
<td>... user-defined module for additional source terms</td>
</tr>
<tr>
<td>TRANSPORT</td>
<td>trnsprt standard transport module</td>
</tr>
<tr>
<td>SPECIALTRN</td>
<td>minden resets density to floor value</td>
</tr>
<tr>
<td></td>
<td>... user-defined module for additional transport terms</td>
</tr>
<tr>
<td>NEWTIMESTEP</td>
<td>newdt full dynamics</td>
</tr>
<tr>
<td></td>
<td>advectdt for advection tests</td>
</tr>
<tr>
<td>FINISH</td>
<td>spectra generates Fourier spectra from gathered data</td>
</tr>
<tr>
<td></td>
<td>pertest reports deviations from periodicity</td>
</tr>
<tr>
<td></td>
<td>... user-defined module called once at the end of execution</td>
</tr>
<tr>
<td>USERSOURCE</td>
<td>phistv non-conservative point mass gravity (see corona)</td>
</tr>
<tr>
<td></td>
<td>... user-defined module for additional source terms</td>
</tr>
<tr>
<td>ARTIFICIALVISC</td>
<td>viscous von Neumann-Richtmyer artificial viscosity</td>
</tr>
<tr>
<td></td>
<td>gasdiff heat and mass diffusion</td>
</tr>
<tr>
<td>DIFFUSION</td>
<td>diffuse second fluid diffusion</td>
</tr>
<tr>
<td>USERDUMP</td>
<td>... user-defined I/O module</td>
</tr>
<tr>
<td>PROBLEM</td>
<td>shkset for shock tube tests</td>
</tr>
<tr>
<td></td>
<td>corona sets up jet-disc problem</td>
</tr>
<tr>
<td></td>
<td>jetinit sets up propagating jet problem</td>
</tr>
<tr>
<td></td>
<td>... user-defined module to initialise flow variables</td>
</tr>
<tr>
<td>PROBLEMRESTART</td>
<td>... user-defined module to alter variables for restarted job</td>
</tr>
</tbody>
</table>
B The namelists

There are well over 500 namelist parameters to specify a unique initialisation. Take heart—most defaults can be used for most applications. As a start, use the input deck given in the dzeus36.s template (§2.3), and then alter as needed.

On the next page begins a complete catalogue of all the input parameters in dzeus36. The parameters are grouped together in “namelists” and discussion for each namelist is contained within a segment headed by the name of the namelist and the subroutine in which the namelist is called. For example, the first namelist is iocon (input/output control) and is called by the subroutine mstart. After each heading is a discussion of what the namelist controls, a list of all the parameters which are elements of the namelist, and finally the syntax used in dzeus36 to declare the namelist.

For the uninitiated, namelist is a non-standard feature of most FORTRAN77 compilers and a standard feature of Fortran90 which provides a convenient way to specify input data. Before Fortran90 was released in 1994, each platform had its own namelist with its own syntax, and this made it difficult to port ZEUS-3D even among different flavours of UNIX. Thus, a namelist emulator was built into EDITOR which, during one of its many passes through the code, replaces all namelist references (including reads and writes) with calls to subroutines in the dnamelist.a library. The following discussion, therefore, reflects the syntax for the EDITOR namelist, which differs somewhat from the Fortran90 version. If desired, EDITOR can be instructed not to replace the namelist syntax (inmlst=0), in which case your compiler’s namelist would be invoked. This may cause syntax errors to be issued since standard FORTRAN namelists don’t allow variables passed via a subroutine to be used as a namelist parameter, whereas the EDITOR namelist does.

In order to specify an input parameter, one merely needs to set it to the desired value as done in the input deck inzeus found in the sample script file dzeus36.s (§2.3). The order in which the variables appear in the namelist declaration need not be adhered to in the input deck nor must all the variables be set. So long as the variable specified in the input deck is a member of the namelist, then namelist will set the variable as specified.

There are a few rules to bear in mind. The namelists (but not necessarily the variables within a namelist) in the input deck must be in the same order as they are encountered during execution. If no parameters are to be set, an empty namelist (one with the namelist name between two $ sentinels) must appear in the correct sequence. There is no problem with namelists appearing that are never read, but a read to a non-existent namelist will generate a namelist error message. In this catalogue, the order of the namelists is the same as the order in which they appear in inzeus and in which they are encountered in dzeus36.

The syntactic rules of setting the variables can be gleaned from the input deck inzeus (§2.3). Column 1 is reserved for a ‘c’ to “comment out” a namelist line which is then echoed on the CRT when encountered in the input deck. Column 2 is reserved for the leading $ sentinel. The specification of the namelist may start in column 3 and must terminate with a second $ sentinel. Until the second $ sentinel is found, all lines will be interpreted as part of the same namelist. All characters appearing after the 72nd column will be ignored, including the closing $ sentinel, should it inadvertently be placed there.
B.1 IOCON—I/O CONTROL (subroutine MSTART)

This namelist sets the logical units to be used during execution. Typically, these parameters will not need to be set to anything other than their default values. These parameters are not written to the restart dump. Therefore, all non-default values for any of the parameters in this namelist must be set each time the job is resumed.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>iotty</td>
<td>logical unit for terminal (standard output)</td>
<td>6</td>
</tr>
<tr>
<td>iopl1</td>
<td>logical unit for 1-D plots using NCAR/PSPLOT graphics</td>
<td>99</td>
</tr>
<tr>
<td>iopl2</td>
<td>logical unit for 2-D plots using NCAR/PSPLOT graphics</td>
<td>99</td>
</tr>
<tr>
<td>iolog</td>
<td>logical unit for message log dump</td>
<td>30</td>
</tr>
<tr>
<td>iodmp</td>
<td>logical unit for restart dumps</td>
<td>31</td>
</tr>
<tr>
<td>iopix</td>
<td>logical unit for pixel dumps</td>
<td>32</td>
</tr>
<tr>
<td>iousr</td>
<td>logical unit for user dumps</td>
<td>33</td>
</tr>
<tr>
<td>iotsl</td>
<td>logical unit for time slice (history) ascii dumps</td>
<td>34</td>
</tr>
<tr>
<td>iotsp</td>
<td>logical unit for time slice (history) plot dumps</td>
<td>99</td>
</tr>
<tr>
<td>iodis</td>
<td>logical unit for display dumps</td>
<td>36</td>
</tr>
<tr>
<td>iorad</td>
<td>logical unit for RADIO dumps</td>
<td>37</td>
</tr>
<tr>
<td>iocrk</td>
<td>logical unit for cork dumps</td>
<td>38</td>
</tr>
<tr>
<td>iotmp</td>
<td>logical unit for temporary files; unit should always be closed after using so that other routines can expect them to be available when needed.</td>
<td>39</td>
</tr>
<tr>
<td>intractv</td>
<td>= 1 =&gt; interactive mode on (see INTCHK)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>= 0 =&gt; interactive mode off; to prevent batch jobs from calling CHECKIN at every time step.</td>
<td></td>
</tr>
</tbody>
</table>

WARNING: AVOID LOGICAL UNIT 3. APPARENT CONFLICT WITH NCAR.

NOTE: IOTTY MAY BE SET TO 6 (TO GET CRT OUTPUT) OR 0 (NO OUTPUT).

```
   namelist / iocon /
   1 iotty , iopl1 , iopl2 , iolog , iodmp
   2 , iopix , iousr , iotsl , iotsp , iodis
   3 , iorad , iocrk , iotmp , intractv
```

B.2 RESCON—RESTART dump CONTROL (subroutine MSTART)

This namelist determines if the job is to be started from initial conditions, or if it is to be restarted from a previous run. These parameters are not written to the restart dump. Therefore, all non-default values for any of the parameters in this namelist must be set each time the job is resumed.

The default values are set for starting from initial conditions, which occurs when the third to fifth characters in resfile are 000. To restart a job, one can usually use the same input deck as was used for the original run with resfile set to the desired restart dump name. In addition, parameters in the namelist pcon may have to be changed.

The parameters *getm?: *=i,j,k, ?=n,x are designed so that only a portion of the restart dump may be read, and/or so that the data may be read into a larger grid. That is, it is not necessary for the field arrays in a restarted job to be dimensioned the same as those in the run which generated the restart dump.

Example 1: For a straight restart without altering the grid or the EDITOR macros, leave
Appendix B: The namelists

the values of igetmn, etc. to their defaults and make sure that the parameters in, etc. are set to the same values as in the run which generated the restart dump.

Example 2: If the first run was on a 64^3 grid and the user wishes to read the inner eighth of the data and position them at the centre of a 100^3 grid, and if the new portion of the grid is to be determined from the existing grid, then the following settings are necessary:

\[
\begin{align*}
\text{igetmn} &= 17, \text{jgetmn} = 17, \text{kgetmn} = 17, \text{iaddz} = 1 \\
\text{igetmx} &= 48, \text{jgetmx} = 48, \text{kgetmx} = 48, \text{jaddz} = 1 \\
\text{iputmn} &= 35, \text{jputmn} = 35, \text{kputmn} = 35, \text{kaddz} = 1
\end{align*}
\]

The desired portion of the restart dump will be read and loaded into the 100^3 grid between \(i=35,66, j=35,66, k=35,66\). In addition, the 1-grid \(x1a(35:66)\) (see §C.1 for a discussion of the naming convention for the grid variables) will be filled by the values of \(x1a(17:48)\) in the restart dump. The code will detect that the grids \(x1a, x2a, x3a\) are now incomplete, and will call the appropriate modules to add zones to the \(x1-, x2-,\) and \(x3\)-grids. If the user wishes, (*addz=1, *=i,j,k), the new portion of the grid may be determined automatically from the existing grid. In this example, \(x1a(1:34)\) would be determined (i.e., \(dx1min, x1rat, etc.,\) see namelist \(ggen1)\) from \(x1a(35:37)\). Similarly, \(x1a(67:100)\) would be determined from \(x1a(64:66)\). Alternatively, the user may opt to set the new portion of the grid manually. In this case, one should set *addz=0 and proceed with setting the namelists \(ggen1, ggen2, ggen3\). (See discussion in \(ggen1).\) Note that if the user selects the manual option, it is imperative that the portion of the new grid that overlaps the old grid be, in fact, identical to the old grid. Next, all arrays will be padded with values at the edges of the portion read. Thus \(d(1:34,j,k)=d(35,j,k), d(67:100,j,k)=d(66,j,k)\) (where \(d\) is the density array—see §C.2), etc. Of course, the user is free to set the values of the padded portion of the arrays to whatever values they want by linking a user-supplied subroutine to the \(EDITOR\) macro PROBLEMRESTART (§2.2.2).

Finally, a job may be resumed from a restart dump with different \(EDITOR\) macros defined or not. Thus, if a job that began with magnetic fields is to be resumed without them, the user may recompile \(dzeus36\) without magnetic fields (MHD not defined) and then blindly read the restart dump which contains magnetic field arrays. There is enough information in the restart dump that the code can selectively read the non-magnetic part of the dump and resume the calculation as though there were never any magnetic fields. Of course, whether suddenly disappearing the magnetic fields is physically realistic is another matter!

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtdmp</td>
<td>problem time interval between restart dumps</td>
<td>0.0</td>
</tr>
<tr>
<td>nresdmp</td>
<td>the sequential number for the next restart dump</td>
<td>-1</td>
</tr>
<tr>
<td>nlogdmp</td>
<td>the sequential number for the next log file</td>
<td>-1</td>
</tr>
</tbody>
</table>
idtag character*2 problem tag appended to filenames 'aa'
resfile restart dump filename 'zr000aa'
igetmn minimum x1-index (i) to be read from restart dump 1
igetmx maximum x1-index (i) to be read from restart dump in
iputmn i-index at which xla(igetmn) is stored 1
iaddz < 0 => no new zones are generated 0
  = 0 => call GRIDX1 to redo entire grid
  > 0 => new zone spacing determined from existing grid

The variables (jgetmn, jgetmx, jputmn, jaddz) and (kgetmn, kgetmx, kputmn, kaddz) are analogous to (igetmn, igetmx, iputmn, iaddz) for the 2- and 3-directions respectively.

```
namelist / rescon /
1    dtdmp , nresdmp , nlogdmp , idtag , resfile
2    , igetmn , igetmx , jgetmn , jgetmx , kgetmn
3    , kgetmx , iputmn , jputmn , kputmn , kaddz
4    , jaddz , kaddz
```

### B.3 GGEN1—Grid GENerator for x1 (subroutine GRIDX1)

This namelist controls how the grid is determined in the 1-direction. All the parameters in this namelist, as well as those in namelists ggen2, ggen3, and those read by subroutine nmlsts are written to the restart dump. The stored values, therefore, will become the “default” values of the parameters for any run resumed from the restart dump.

The grid can be created all at once or in several blocks. Each block requires a separate read of this namelist specifying how that portion of the grid is to be computed. The parameter lgrid should be set to .true. (or equivalently .t. for the EDITOR namelist) only for the last block. (Note that the EDITOR namelist also allows .f. as a short form for .false.)

There are two types of gridding. The first is “ratioed gridding” where the distance across a zone is a fixed multiple of the distance across the previous zone. If this multiple is 1, then the zones are uniform. If the multiple is 1.1, then each zone is 10% larger than the previous one. If the multiple is 0.9, then each zone is 10% smaller than the previous one. To determine a block of ratioed zones uniquely, one must specify the number of zones in the block (nbl), the minimum and maximum extent of the block in coordinate units (x1min, x1max), and either the smallest zone size in the block (dx1min) or the ratio to use between zones (x1rat). Specifying either dx1min or x1rat will allow the other to be computed.

The second type of gridding is “scaled gridding” where the coordinate value is some fixed multiple of the previous coordinate value. For ratioed grids, dx(n)=mult*dx(n-1). For scaled grids, x(n)=mult*x(n-1). For example, scaled gridding would be appropriate for the r-direction in spherical polar coordinates if the zones were all to have the same shape.

To determine a block of scaled zones uniquely, one must specify the number of zones in the block (nbl) and the minimum and maximum extent of the block in coordinate units (x1min, x1max). Neither dx1min nor x1rat are needed.

The grid can be scaled to physical units most conveniently by setting the multiplicative factor x1scale to the desired scaling value.

For restarted jobs, there is a third gridding option. Setting igrid to zero will cause the grid generator to skip over the nbl zones specified for this block. Thus, in the second
example in the discussion for namelist *rescon*, one could set the new zones for the \( x_1 \)-direction manually with three *ggen1* namelist “cards”. The first card would set zones \((1:34)\) in whatever manner desired with the condition that the last zone of the new grid ends where the first zone of the old grid begins. The second card would set \( igrid=0 \) and \( nbl=32 \). This would leave zones \((35:66)\) alone since they were set when the restart dump was read. Finally, the third card would set zones \((67:100)\) in whatever manner desired with the condition that the first zone of the new grid begins where the last zone of the old grid ends.

Other than remaining within the memory limits of the machine, there are two practical considerations when choosing the number of zones for each of the three dimensions. First, if at all possible, the greatest number of zones should be along the \( 1 \)-direction so that the vector length of the vectorised loop is as long as possible\(^7\). Second, if the code is to be multi-tasked, the number of zones (including the five boundary zones) in each direction should be an integral multiple of the number of parallel processors available on the machine. This will yield the best overall degree of parallelism.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nbl</td>
<td>number of active zones in block being generated</td>
<td>1</td>
</tr>
<tr>
<td>x1min</td>
<td>( x1a(imin); ) bottom position of block</td>
<td>0.0</td>
</tr>
<tr>
<td>x1max</td>
<td>( x1a(imax); ) top position of block</td>
<td>0.0</td>
</tr>
<tr>
<td>x1scale</td>
<td>arbitrary scaling factor for ( &quot;x1\text{min}&quot; ) and ( &quot;x1\text{max}&quot; )</td>
<td>1.0</td>
</tr>
<tr>
<td>igrid</td>
<td>method of computing zones.</td>
<td>1</td>
</tr>
<tr>
<td>( +0 )</td>
<td>( \text{block has already been set (restarted runs only)} )</td>
<td></td>
</tr>
<tr>
<td>( +1 )</td>
<td>( \text{(ratioed) use input } &quot;x1rat&quot; \text{ to compute } &quot;dx1min&quot;. } )</td>
<td></td>
</tr>
<tr>
<td>( -1 )</td>
<td>( \text{(ratioed) use input } &quot;x1rat&quot; \text{ to compute } &quot;dx1min&quot;. } )</td>
<td></td>
</tr>
<tr>
<td>( +2 )</td>
<td>( \text{(ratioed) use input } &quot;dx1min&quot; \text{ to compute } &quot;x1rat&quot;. } )</td>
<td></td>
</tr>
<tr>
<td>( -2 )</td>
<td>( \text{(ratioed) use input } &quot;dx1min&quot; \text{ to compute } &quot;x1rat&quot;. } )</td>
<td></td>
</tr>
<tr>
<td>( +3 )</td>
<td>( \text{(scaled) compute } &quot;x1rat&quot; \text{ and } &quot;dx1min&quot; \text{ from } &quot;nbl&quot;. } )</td>
<td></td>
</tr>
<tr>
<td>x1rat</td>
<td>desired ratio ( dx1a(i+1) / dx1a(i) )</td>
<td>1.0</td>
</tr>
<tr>
<td>dx1min</td>
<td>size of first ((igrid&gt;0)) or last ((igrid&lt;0)) zone in block</td>
<td>0.0</td>
</tr>
<tr>
<td>lgrid</td>
<td>( = .false. ) ( \Rightarrow ) read another block (namelist card). ( = .true. ) ( \Rightarrow ) all blocks are read in. Do not look for another &quot;ggen1&quot; namelist card.</td>
<td>.false.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>namelist</th>
<th>ggen1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>nbl</td>
<td>x1min , x1max , x1scale , igrid</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>, x1rat , dx1min , lgrid</td>
</tr>
</tbody>
</table>

**B.4 GGEN2—Grid GENerator for \( x_2 \) (subroutine GRIDX2)**

See comments for GGEN1.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nbl</td>
<td>number of active zones in block being generated</td>
<td>1</td>
</tr>
<tr>
<td>x2min</td>
<td>( x2a(jmin); ) bottom position of block</td>
<td>0.0</td>
</tr>
<tr>
<td>x2max</td>
<td>( x2a(jmax); ) top position of block</td>
<td>0.0</td>
</tr>
</tbody>
</table>

\(^7\text{This is an issue only if one is using a vector machine.}\)
Appendix B: The namelists

x2scale  arbitrary scaling factor for "x2min" and "x2max"  1.0
igrid  method of computing zones.  1
  = 0 => block has already been set (restarted runs only)
  =+1 => (ratioed) use input "x2rat" to compute "dx2min",
     "dx2min" = size of first zone in block
  =-1 => (ratioed) use input "x2rat" to compute "dx2min",
     "dx2min" = size of last zone in block
  =+2 => (ratioed) use input "dx2min" to compute "x2rat",
     "dx2min" = size of first zone in block
  =-2 => (ratioed) use input "dx2min" to compute "x2rat",
     "dx2min" = size of last zone in block
  = 3 => (scaled) compute "x2rat" and "dx2min" from "nbl".

x2rat  desired ratio dx2a(j+1) / dx2a(j)  1.0
dx2min  size of first (igrid>0) or last (igrid<0) zone in block  0.0
units  sets the angular units (character*2, RTP only) 'rd'
   'rd' => radians, 'pi' => pi radians, 'dg' => degrees
lgrid  = .false. => read another block (namelist card). .false.
   = .true.  => all blocks are read in. Do not look for
     another "ggen2" namelist card.

namelist / ggen2 /
  1  nbl , x2min , x2max , x2scale , igrid
  2  , x2rat , dx2min , units , lgrid

B.5  GGEN3—Grid GENERator for x3 (subroutine GRIDX3)

See comments for GGEN1.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nbl</td>
<td>number of active zones in block being generated</td>
<td>1</td>
</tr>
<tr>
<td>x3min</td>
<td>x3a(kmin); bottom position of block</td>
<td>0.0</td>
</tr>
<tr>
<td>x3max</td>
<td>x3a(kmax); top position of block</td>
<td>0.0</td>
</tr>
<tr>
<td>x3scale</td>
<td>arbitrary scaling factor for &quot;x3min&quot; and &quot;x3max&quot;</td>
<td>1.0</td>
</tr>
<tr>
<td>igrid</td>
<td>method of computing zones.</td>
<td>1</td>
</tr>
<tr>
<td>x3rat</td>
<td>desired ratio dx3a(k+1) / dx3a(k)</td>
<td>1.0</td>
</tr>
<tr>
<td>dx3min</td>
<td>size of first (igrid&gt;0) or last (igrid&lt;0) zone in block</td>
<td>0.0</td>
</tr>
</tbody>
</table>
| units     | sets the angular units (character*2, ZRP and RTP only) | 'rd'
   'rd' => radians, 'pi' => pi radians, 'dg' => degrees
| lgrid     | = .false. => read another block (namelist card). | .false.
   = .true.  => all blocks are read in. Do not look for
     another "ggen3" namelist card.

namelist / ggen3 /
  1  nbl , x3min , x3max , x3scale , igrid
  2  , x3rat , dx3min , units , lgrid
B.6 PCON—Problem Control (subroutine NMLSTS)

Determines the criteria for terminating the job.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlim</td>
<td>cycles to run</td>
<td>0</td>
</tr>
<tr>
<td>tlim</td>
<td>physical (problem) time to stop calculation</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>if tlim &lt; 0, problem is stopped at exactly abs(tlim)</td>
<td></td>
</tr>
<tr>
<td>ttotal</td>
<td>total seconds of execution time permitted for job</td>
<td>0.0</td>
</tr>
<tr>
<td>tsave</td>
<td>seconds of execution (cpu) time reserved for cleanup</td>
<td>0.0</td>
</tr>
</tbody>
</table>

WARNING: should a job be restarted from a job with tlim<0, the sum of the first and restarted job will not be identical to machine round-off to a job that did the simulation in one run. This is because the single-run job would not have the abbreviated time step in the middle as needed by the first partial job to finish exactly on "prtime".

```plaintext
   namelist / pcon /
   1 nlim , tlim , ttotal , tsave
```

B.7 HYCON—Hydro Control (subroutine NMLSTS)

Sets the parameters which control the hydrodynamics. One of the most important selectors in this namelist, itote, chooses between the internal (itote=0) and total (itote=1; default) energy equations, with the latter often but not always giving superior results. Pros and cons for choosing between the two energy equations include: Execution time is generally faster for the internal energy equation (by about 20%), and pressures are guaranteed positive definite for courno<0.5. However, the algorithm is not strictly conservative which, among other things, causes it to converge on incorrect values (by as much as 20%) in some 1-D shock-tube tests and lead to qualitatively different results in some multi-dimensional problems. The total energy equation is conservative and somewhat more stable allowing a slightly larger Courant number (e.g., courno=0.75) than the internal energy equation in some applications. However, internal energies are not positive definite and where they become negative, are reset to e1floor. In the opinion of this author, the requirement that a code be strictly conservative has been somewhat overblown in the literature. It is true that only conservative codes will converge correctly on 1-D shock tube problems but, in the messy universe where the sum of mechanical and thermal energies is known not to be conserved, a strictly conservative code may less important than the assurance of positive-definite pressures. This version of the code gives the user both options.

All energy variables should be interpreted as energy per unit volume. In setting up a problem, the user should always initialise the internal energy density (variable e1 and boundary values e1iib1, etc.; see §C.2 and §C.3) and not the total energy density, (et), regardless of (itote). Being a primitive variable, boundary conditions are always applied to the internal energy density. Note that if ISO is defined, itote is set to 0.

The steepest discontinuities this code can sustain are obtained with iord=2, iords=3, and istp=2, where the latter assures only contact discontinuities are steepened. istp=1 will cause any discontinuity to be steepened and is intended for advection tests only. These settings maintain contacts in 2 or 3 zones and most shocks in qcon+1 zones (although some
slow shocks may be smeared out over as many as ten zones), but can also cause the bases of discontinuities or rarefactions to undershoot slightly and even “ring” in some 1-D shock tube tests. More conservative settings are the defaults, for which the code runs 20% faster and maintains contacts in 5 to 7 zones, and most shocks to qcon+2 zones.

A note on Consistent Advection: CA is a modification of how fluxes are computed to keep them consistent with mass flux. Thus, “numerical drift” between angular momentum and mass, such as that which caused early simulations of mass accretion to form rings rather than discs, is significantly reduced (see Norman, Barton, and Wilson, 1980, ApJ).

Since its earliest inception, ZEUS has used CA on all flow variables, including all momentum components and energy, although its original purpose was for angular momentum transport. In 2006, I discovered that CA applied to energy transport causes significant ringing in Figures 1b and 2b of Ryu & Jones (1995, ApJ) and subsequently introduced a parameter into ZEUS-3D (cadvs) to turn off CA for all scalar transport, including energy density. At the same time, I introduced cadvc and cadvt to turn off CA for compressive and transverse transport of momenta. I have found no test problem nor application in which disengaging CA from energy transport has any deleterious consequence and, accordingly, the default setting for cadvs is 0 (off). Conversely and not surprisingly, numerous tests show that CA should remain engaged with momentum transport. Accordingly, the default settings for cadvc and cadvt is 1 (on). Indeed, in a private communication, Mike Norman recalled there to be no compelling reason at the time to include energy in the CA algorithm, other than “what’s good for the goose is good for the gander”, and the failure of CA in energy transport in the two R&J tests was the first evidence he saw of any problems introduced by the algorithm.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>qcon</td>
<td>quadratic artificial viscosity (q) constant</td>
<td>2.0</td>
</tr>
<tr>
<td>qlin</td>
<td>linear artificial viscosity (q) constant</td>
<td>0.0</td>
</tr>
<tr>
<td>courno</td>
<td>The CFL-determined time step is multiplied by the &quot;Courant number&quot;, a &quot;safety factor&quot;.</td>
<td>0.5</td>
</tr>
<tr>
<td>dtrat</td>
<td>ratio of &quot;dtmin&quot; to initial value of &quot;dt&quot; Set to 0.2 for no ramp-up of time step.</td>
<td>0.001</td>
</tr>
<tr>
<td>dtmax</td>
<td>maximum time step to use</td>
<td>huge</td>
</tr>
<tr>
<td>dtset</td>
<td>constant value of &quot;dt&quot; to be set throughout simulation, used only for testing; 0 =&gt; use computed &quot;dt&quot;.</td>
<td>0.0</td>
</tr>
<tr>
<td>iord</td>
<td>order of interpolation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 1 =&gt; first order (donor cell) interpolation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>= -1 =&gt; first order implicit interpolation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 2 =&gt; second order (van Leer) interpolation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>= -2 =&gt; second order implicit interpolation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 3 =&gt; piecewise parabolic interpolation (PPI)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 4 =&gt; second order velocity-corrected interpolation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 5 =&gt; second order non-harmonic interpolation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 6 =&gt; second order interpolation for non-uniform grid</td>
<td></td>
</tr>
</tbody>
</table>

Implicit interpolations (iord=-1,-2) affect compressive s* transport only; interpolations for all other transport steps use abs(iord).

| iords | order of interpolation for scalars to override "iord" | iord |
|       | iord |
| istp  | contact discontinuity steepener (third order only)   | 0    |
|       | = 0 => always off, 1 => always on, 2 => on only at    |
Appendix B: The namelists

**floor**
smallest value desired for variable **

scalars tiny
vectors 0.0

icool = 0 => use PDV in SRDSTEP
= 1 => use PDVCool in SRDSTEP for pdv work with
arbitrary cooling function

itote = 0 => solve the internal energy equation (positive 1
definitions but energy not conserved).
= 1 => solve the total energy equation (energy
conserved but pressure not positive definite)

iscydf = 0 => no subcycling on diffusion
= 1 => subcycle on diffusion

iscyqq = 0 => no subcycling on artificial viscosity
= 1 => subcycle on artificial viscosity (itote=0 only)

ix1x2x3 vector seed for directional splitting sequence

mind minimum value subroutine MINDEN will allow for density dfloor

nu kinematic viscosity (in units of LV) 0.0

isetemf affects flow-in skin values for emf(perp) and flow-in 0
boundary values for emf(par)
= 0 => SVALEMF* and BVALEMF don't overwrite (C)MOC*-computed flow-in emfs with pre-set flow-in arrays.
= 1 => SVALEMF* overwrite (C)MOC*-computed flow-in emfs with preset flow-in arrays, but BVALEMF doesn’t.
= 2 => Both SVALEMF* and BVALEMF overwrite (C)MOC*-computed flow-in emfs with preset flow-in arrays.

tspinup time to add all the desired angular velocity (SPINUP). 1.0
1.0 is characteristic time scale of the Bondi problem.

delta amplitude of imparted angular velocity (SPINUP) sqrt(2.0)
Default puts centrifugal barrier at critical point (r=1) for perturbed Bondi flow.

orbchk = 0 => BNDYCHK will abort at a non-physical boundary. 0
= 1 => BNDYCHK issues a warning at a non-physical boundary, but execution continues (dangerous!).

pjump minimum pressure jump that defines a "strong shock";
sets the limit for diffusive shock acceleration 2.0

cadvs = 1 => consistent advection (CA) used with scalar (energy) transport
= 0 => CA not used with scalar transport

cadvc = 1 => CA used with compressive momentum transport 1
= 0 => CA not used with compressive momentum transport

cadvt = 1 => CA used with transverse momentum transport 1
= 0 => CA not used with transverse momentum transport

trnvrn = 0 => use legacy CMoC algorithm (MoC, HSMoC, CT) 1
= 1 => use the FIT algorithm
= 2 => same as 1, but with transverse Lorentz terms rendered in conservative form.
= -2 => same as 2 but for legacy transport
= 3 => same as 2, but with new source terms in CTRAN* expressed as a true flux triggering "magnetic centrifugal" terms in STV*
= -3 => same as 3 but for legacy transport

Toggles for highly experimental "downwinding" algorithm:

uordq = 1 => upwinded interpolations of qq (art. visc.) 1
= 2 => downwinded interpolations of qq
Appendix B: The namelists

avordw = 1 => use 2-point averages for \( v^* \) for \( e_t \leftrightarrow e_1 \)
= 2 => use downwinded \( v^* \) for \( e_t \leftrightarrow e_1 \)

voutbc = 1 => outflow BC, \( v_1(\text{is}) \) set to \( \min(0,v_1(\text{isp}1)) \),
original Wilson scheme (same @ other boundaries)
= 2 => outflow BC, \( v_1(\text{is}) \) set to \( \min(0,v_1(\text{is})) \) to
reduce negative pressures at boundaries

The routine SPINUP and the associated namelist variables "tspinup" and "delta"
were designed to perturb Bondi flow to form discs, but can be used in other
applications in which a gradual spin-up of the grid is desired.

```
  namelist / hycon /
  1      qcon , qlin , courno , dtrat , dtmax
  2 , dtset , iord , iords , istp , dfloor
  3 , efloor , e2floor , v1floor , v2floor , v3floor
  4 , b1floor , b2floor , b3floor , dafloor , icool
  5 , itote , isycdf , iscyqq , ix1x2x3 , mind
  6 , nu , isetemf , tspinup , delta , orbcck
  7 , pjump , cadvs , cadvc , cadvt , trnvrsn
  8 , uordq , avordw , voutbc
```

B.8 IIB—IInner I Boundary control (subroutine NMLSTS)

This namelist specifies both the boundary type and the inflow values for the variables that
can be set at the inner-\( i \) boundary. These variables are not declared if the EDITOR macro
ISYM is set. Any one of ten MHD boundary conditions may be specified independently at
every boundary zone by setting \( \text{niib}(j,k) \) to the desired value of \( \text{btype} \), as follows:

\[
\begin{align*}
\text{btype} = 1 (-1) & \Rightarrow \text{reflecting; grid singularity or symmetry axis} \\
= 2 (1) & \Rightarrow \text{reflecting; non-conducting boundary (b_par=0 on bdy)} \\
= 3 (5) & \Rightarrow \text{reflecting; conducting boundary (b_perp=0 on bdy)} \\
= 4 (6) & \Rightarrow \text{reflecting; no B components change sign across boundary} \\
= 5 (4) & \Rightarrow \text{periodic} \\
= 6 & \Rightarrow \text{self-computing (for AMR)} \\
= 7 & \Rightarrow \text{outflow (not yet functional)} \\
= 8 & \Rightarrow \text{selective inflow} \\
= 9 (2) & \Rightarrow \text{non-characteristic outflow} \\
= 10 (3) & \Rightarrow \text{non-characteristic inflow}
\end{align*}
\]

where the values of \( \text{btype} \) used in versions of ZEUS-3D previous to 3.5 are given parenthetical-
ically.

The boundary values for the variables are used only in the event that a zone along
the boundary is inflow (\( \text{btype}=8,10 \)). Otherwise, the boundary value is determined from
the flow variables on the active portion of the computational grid. The flow variables are \( d \)
(density), \( e_1 \) (first internal energy density), \( e_2 \) (second internal energy density), \( e_r \) (radiation
energy density), \( v_1 \) (1-velocity), \( v_2 \) (2-velocity), and \( v_3 \) (3-velocity). In addition, skin values
for the transverse \( \text{emf} \) components and boundary values for the transverse magnetic field
components can be set (see extensive discussion below). Note that boundary values for the
\textit{total} energy (\( e_t \)) are not set directly. These are computed (in BVALET) from the boundary
values of the primitive variables.

The boundary type for the gravitational potential (\( \text{gtype} \)) is treated independently of
the MHD boundaries, since the nature of the Poisson equation (elliptical) is different from
that of the MHD equations (hyperbolic). Gravitational boundary type is specified by setting 
\texttt{giib} to the desired value of \texttt{gtype}, as follows:

\begin{verbatim}
gtype =  5 (4) => periodic
  =  9 (2) => six-term multipole expansion
  = 10 (3) => analytical (or preset) boundary values stored in gpiib. Time-
              varying boundaries can be updated by a routine aliased to
              \texttt{BNDYUPDATE}
\end{verbatim}

where the values of \texttt{gtype} used in versions of \texttt{ZEUS-3D} previous to 3.5 are given parenthetically. Any other value for \texttt{gtype} means the boundaries of $\phi$ are never updated, which would be appropriate for constant boundary values set as part of the initial conditions.

\textbf{New to version 3.5:} Magnetic boundary conditions have been completely revamped, and a new, stable algorithm has been implemented. To start, a distinction is now made between the \textit{skin} (which can receive characteristic information directly from the boundary region and/or the active grid) and the \textit{boundary}, which can only receive characteristic information from the boundary region within a given CFL-limited timestep. For example, at the inner-$i$ boundary, the $i=\text{is}$ face constitutes the “skin”, while all zones—face- or zone-centred—at $i=\text{ism1}$, $i=\text{ism2}$ are in the boundary. Because of this distinction, magnetic “skin” values and “boundary” values are now treated differently in \texttt{ZEUS-3D}. Skin values are set in routines \texttt{SVALEMF*} which are called by the (C)\texttt{MOC*} routines where the distinction between the two “terms” in the \textit{emfs} is needed to set some skin conditions. Boundary values are set by \texttt{BVALEMFS} called at the top of \texttt{CT} where all components of the \textit{emfs} are needed to set the boundary conditions for each.

The main problem with the algorithm used in version 3.4 was that skin values of the magnetic field were set directly when setting boundary conditions. This is folly, since resetting the magnetic flux through the face of a zone lying along the skin changes the net magnetic flux into the adjacent grid zone, introducing a magnetic monopole. This problem was particularly acute when setting inflow boundaries, and thus all inflow arrays such as $b1iib1$ which allowed the normal (to the boundary) magnetic field to be set directly both on the skin and inside the boundary have been purged from the code.

In this version, the user must now completely initialise all magnetic field arrays ($b1$, $b2$, and $b3$) in their problem generator, including all boundaries. For inflow boundaries, \textit{skin} values of the parallel field (e.g., $B_1$ at the $i$-skins) are maintained by user-set arrays $\texttt{emf2iib1}$ and $\texttt{emf3iib1}$ which lie along the inner $i$-skin. Typically, these components of the \textit{emf} are set by physical boundary conditions on the skin (e.g., $v_2 = v_3 = 0$; $B_2 = B_3 = 0 \Rightarrow \varepsilon_2 \propto v_3 B_1 - v_1 B_3 = 0$; $\varepsilon_3 \propto v_1 B_2 - v_2 B_1 = 0$). Meanwhile, \textit{boundary} values for the parallel field (e.g., $b1(\text{ism2:ism1})$) are determined by the solenoidal condition and thus are allowed to “float”, regardless of boundary conditions. Accordingly, there are no arrays $\texttt{emf2iib2}$, $\texttt{emf3iib2}$, etc.

As in previous versions of the code, inflow conditions on the \textit{transverse} magnetic field components (e.g., $B_2$ and $B_3$ at the $i$-boundaries) are controlled by the user-set arrays $b2iib1$, $b2iib2$, $b3iib1$, and $b3iib2$ at the inner-$i$ boundary. Note that there are no \textit{skin} values for the transverse components, only boundary values. Should constant boundary values be desired, these are most conveniently set to the corresponding initial values of $b2$ and $b3$. Should these components need to vary in time, the user must supply updated values...
Appendix B: The namelists

of b2iib1, b2iib2, b3iib1, and b3iib2 at the current time and proper location at the beginning of each MHD cycle. For this, the EDITOR alias BNDYUPDATE can be aliased to a user-supplied routine that sets the boundary arrays as needed (§2.2.2).

Further, the concept of selectively setting inflow conditions depending on whether various characteristics arrive at the skin from the boundary or grid has been introduced. For the emfs, selective inflow conditions is controlled by the parameter isetemf, set in namelist hycon. In general, if one expects the skin to receive information only from the boundary region (as in super-fast inflow), isetemf should be set to 1 and the user should supply values for the transverse emfs based on boundary conditions alone. If flow across the boundary is sub-fast, sub-Alfvén, or even sub-slow, isetemf probably should be set to 0 (in which case, the skin emfs will remain as computed by the (C)MOC* routines), though there are circumstances where it still may be best to set isetemf differently (e.g., CORONA). Regardless, care must be taken not to over-specify the boundary.

Selective inflow conditions are enabled using the new boundary type 8. This behaves just like boundary type 10 (inflow), except it allows designated variables to “float” at the boundary (take on the nearest grid value) rather than being set to the pre-determined boundary arrays (e.g., diib1 for density). To specify that a variable float, its boundary array should be set to the global parameter huge (§C.6), a nonsensical value that triggers logic in the boundary routines to set the variable according to its value in the nearest active grid zone (very similar to reflecting boundary type 2, except v1 is not set to zero at i=ls). An example of the use of selective inflow conditions may be found in the problem generator CORONA for the case where local parameter oork=2. This establishes the so-called Krasnopolsky conditions, first introduced into an NCSA version of ZEUS-3D by Krasnopolsky, et al. (1999, ApJ, 526, 631) to launch sub-Alfvénic (initially) astrophysical jets from an accretion disc maintained as a boundary condition.

Additional discussion may be found in §1.5.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>niib (j,k)</td>
<td>&quot;btype&quot; of inner i boundary on sweep j,k</td>
<td>9</td>
</tr>
<tr>
<td>giib</td>
<td>&quot;gtype&quot; of entire inner i boundary</td>
<td>2</td>
</tr>
<tr>
<td>**iib1(j,k)</td>
<td>first inner i boundary value of variable **</td>
<td>**floor</td>
</tr>
<tr>
<td></td>
<td>for sweep j,k (flow in only)</td>
<td></td>
</tr>
<tr>
<td>**iib2(j,k)</td>
<td>second inner i boundary value of variable **</td>
<td>**floor</td>
</tr>
<tr>
<td></td>
<td>for sweep j,k (flow in only)</td>
<td></td>
</tr>
<tr>
<td>**iib3(j,k)</td>
<td>third inner i boundary value of variable **</td>
<td>**floor</td>
</tr>
<tr>
<td></td>
<td>for sweep j,k (flow in only)</td>
<td></td>
</tr>
<tr>
<td>gpiib*(j,k)</td>
<td>analytical or preset values for gp on iib</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>for sweep j,k (giib=3 only)</td>
<td></td>
</tr>
<tr>
<td>jsli</td>
<td>number of zones in j-direction to slip periodic i-boundaries &gt;0 =&gt; oib is ahead of the iib.</td>
<td>0</td>
</tr>
<tr>
<td>ksli</td>
<td>number of zones in k-direction to slip periodic i-boundaries &gt;0 =&gt; oib is ahead of the iib.</td>
<td>0</td>
</tr>
</tbody>
</table>

```
  namelist / iib /
  1     niib , giib , diib1 , diib2 , v1iib1
  2     , v1iib2 , v1iib3 , v2iib1 , v2iib2 , v3iib1
  3     , v3iib2 , jsli , ksli
*if -def,ISO
```
Only $v_1$ has three boundary values that can be set (at $i=i_{is},i_{ism1},i_{ism2}$). Since just the skin values of $emf_2$ and $emf_3$ can be set, there is no second or third boundary variable available. All remaining variables are zone-centred in the $i$-direction, and thus have two boundary values to set (at $i=i_{ism1},i_{ism2}$).

### B.9 OIB—Outer I Boundary control (subroutine NMLSTS)

This namelist specifies both the boundary type and the in-flow values of all the flow variables for the outer-$i$ boundary. These variables are not declared if the EDITOR macro ISYM is set. See comments for IIB.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>noib (j,k)</td>
<td>&quot;btype&quot; of outer i boundary on sweep j,k</td>
<td>2</td>
</tr>
<tr>
<td>goib</td>
<td>&quot;gtype&quot; of entire outer i boundary</td>
<td>2</td>
</tr>
<tr>
<td>**oib1(j,k)</td>
<td>first outer i boundary value of variable ** for sweep j,k (flow in only)</td>
<td>**floor</td>
</tr>
<tr>
<td>**oib2(j,k)</td>
<td>second outer i boundary value of variable ** for sweep j,k (flow in only)</td>
<td>**floor</td>
</tr>
<tr>
<td>**oib3(j,k)</td>
<td>third outer i boundary value of variable ** for sweep j,k (flow in only)</td>
<td>**floor</td>
</tr>
<tr>
<td>gpoib*(j,k)</td>
<td>analytical or preset values for gp on oib for sweep j,k (goib=3 only)</td>
<td>0.0</td>
</tr>
</tbody>
</table>

```plaintext
namelist / oib /
1     noib , goib , doib1 , doib2 , v1oib1
2     , v1oib2 , v1oib3 , v2oib1 , v2oib2 , v3oib1
3     , v3oib2
*if -def,ISO
4     , e1oib1 , e1oib2
*endif -ISO
*if def,TWOFLUID
5     , e2oib1 , e2oib2
*endif TWOFLUID
*if def,RADIATION
6     , eroib1 , eroib2
*endif RADIATION
```
### Appendix B: The namelists

*if def, GRAV
  7 , gpoib1 , gpoib2
*endif GRAV
*if def, MHD
  8 , emf1oib1, emf1oib2, emf2oib1, emf3oib1, b3oib1
  9 , b3oib2 , b2oib1 , b2oib2
*endif MHD
*if def, AGING
  1 , daoib1 , daoib2
*endif AGING

### B.10 IJB—Inner J Boundary control (subroutine NMLSTS)

This namelist specifies both the boundary type and the in-flow values of all the flow variables for the inner-\(j\) boundary. These variables are not declared if the \textit{EDITOR} macro \texttt{JSYM} is set. See comments for IIB.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>(nijb) ((k,i))</td>
<td>&quot;btype&quot; of inner (j) boundary on sweep (k,i)</td>
<td>2</td>
</tr>
<tr>
<td>(gijb)</td>
<td>&quot;gtype&quot; of entire inner (j) boundary</td>
<td>2</td>
</tr>
<tr>
<td>(**ijb1(k,i))</td>
<td>first inner (j) boundary value of variable (**) (\text{floor}) for sweep (k,i) (flow in only)</td>
<td></td>
</tr>
<tr>
<td>(**ijb2(k,i))</td>
<td>second inner (j) boundary value of variable (**) (\text{floor}) for sweep (k,i) (flow in only)</td>
<td></td>
</tr>
<tr>
<td>(**ijb3(k,i))</td>
<td>third inner (j) boundary value of variable (**) (\text{floor}) for sweep (k,i) (flow in only)</td>
<td></td>
</tr>
<tr>
<td>(gpijb*(k,i))</td>
<td>analytical or preset values for (gp) on (ijb)</td>
<td>0.0</td>
</tr>
<tr>
<td>(kslj)</td>
<td>number of zones in (k)-direction to slip periodic (j)-boundaries &gt;0 =&gt; (ojb) is ahead of the (ijb).</td>
<td>0</td>
</tr>
<tr>
<td>(islj)</td>
<td>number of zones in (i)-direction to slip periodic (j)-boundaries &gt;0 =&gt; (ojb) is ahead of the (ijb).</td>
<td>0</td>
</tr>
</tbody>
</table>

```
namelist / ijb /
  1 nijb , gijb , dijb1 , dijb2 , vi1jb1
  2 , vi1jb2 , v2ijb1 , v2ijb2 , v2ijb3 , v3ijb1
  3 , v3ijb2 , kslj , islj
*if -def, ISO
  4 , e1ijb1 , e1ijb2
*endif -ISO
*if def, TWOFLUID
  5 , e2ijb1 , e2ijb2
*endif TWOFLUID
*if def, RADIATION
  6 , erijb1 , erijb2
*endif RADIATION
*if def, GRAV
  7 , gpijb1 , gpijb2
*endif GRAV
*if def, MHD
  8 , emf2ijb1, emf2ijb2, emf3ijb1, emf1ijb1, b1ijb1
  9 , b1ijb2 , b3ijb1 , b3ijb2
*endif MHD
*if def, AGING
  1 , daijb1 , daijb2
*endif AGING
```
B.11 OJB—Outer J Boundary control (subroutine NMLSTS)

This namelist specifies both the boundary type and the in-flow values of all the flow variables for the outer-\( j \) boundary. These variables are not declared if the EDITOR macro JSYM is set. See comments for IIB.

Parameter description default

nojb (k,i) "btype" of outer \( j \) boundary on sweep k,i 2

gojb "gtype" of entire outer \( j \) boundary 2

**ojb1(k,i) first outer \( j \) boundary value of variable ** **floor for sweep k,i (flow in only)

**ojb2(k,i) second outer \( j \) boundary value of variable ** **floor for sweep k,i (flow in only)

**ojb3(k,i) third outer \( j \) boundary value of variable ** **floor for sweep k,i (flow in only)

gpojb*(k,i) analytical or preset values for gp on ojb 0.0

namelist / ojb /

1   nojb , gojb , dojb1 , dojb2 , v1ojb1
2   , v1ojb2 , v2ojb1 , v2ojb2 , v2ojb3 , v3ojb1
3   , v3ojb2

*if -def,ISO
4   , e1ojb1 , e1ojb2
*endif -ISO

*if def,TWOFLUID
5   , e2ojb1 , e2ojb2
*endif TWOFLUID

*if def,RADIATION
6   , erojb1 , erojb2
*endif RADIATION

*if def,GRAV
7   , gpojb1 , gpojb2
*endif GRAV

*if def,MHD
8   , emf2ojb1 , emf2ojb2 , emf3ojb1 , emf1ojb1 , b1ojb1
9   , b1ojb2 , b3ojb1 , b3ojb2
*endif MHD

*if def,AGING
1   , daojb1 , daojb2
*endif AGING

B.12 IKB—Inner K Boundary control (subroutine NMLSTS)

This namelist specifies both the boundary type and the in-flow values of all the flow variables for the inner-\( k \) boundary. These variables are not declared if the EDITOR macro KSYM is set. See comments for IIB.

Parameter description default

nikb (i,j) "btype" of inner \( k \) boundary on sweep i,j 2

gikb "gtype" of entire inner \( k \) boundary 2

**ikb1(i,j) first inner \( k \) boundary value of variable ** **floor for sweep i,j (flow in only)

**ikb2(i,j) second inner \( k \) boundary value of variable ** **floor for sweep i,j (flow in only)
**ikb3(i,j) third inner k boundary value of variable ** **floor
for sweep i,j (flow in only)**
gpikb*(i,j) analytical or preset values for gp on ikb 0.0
islk number of zones in i-direction to slip periodic 0
k-boundaries >0 => okb is ahead of the ikb.
jslk number of zones in j-direction to slip periodic 0
k-boundaries >0 => okb is ahead of the ikb.

```
namelist / ikb /
  1 nikhb , gikb , dikb1 , dikb2 , v1ikb1
  2 , v1ikb2 , v2ikb1 , v2ikb2 , v3ikb1 , v3ikb2
  3 , v3ikb3 , islk , jslk
*if -def,ISO
  4 , e1ikb1 , e1ikb2
*endif -ISO
*if def,TWOFLUID
  5 , e2ikb1 , e2ikb2
*endif TWOFLUID
*if def,RADIATION
  6 , erikb1 , erikb2
*endif RADIATION
*if def,GRAV
  7 , gpikb1 , gpikb2
*endif GRAV
*if def,MHD
  8 , emf3ikb1 , emf3ikb2 , emf1ikb1 , emf2ikb1 , b2ikb1
  9 , b2ikb2 , b1ikb1 , b1ikb2
*endif MHD
*if def,AGING
  1 , daikb1 , daikb2
*endif AGING
```

B.13 OKB—Outer K Boundary control (subroutine NMLSTS)

This namelist specifies both the boundary type and the in-flow values of all the flow variables for the outer-

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nokb (i,j)</td>
<td>&quot;btype&quot; of outer k boundary on sweep i,j</td>
<td>2</td>
</tr>
<tr>
<td>gokb</td>
<td>&quot;gtype&quot; of entire outer k boundary</td>
<td>2</td>
</tr>
<tr>
<td>**okb1(i,j)</td>
<td>first outer k boundary value of variable ** <strong>floor for sweep i,j (flow in only)</strong></td>
<td>2</td>
</tr>
<tr>
<td>**okb2(i,j)</td>
<td>second outer k boundary value of variable ** <strong>floor for sweep i,j (flow in only)</strong></td>
<td>2</td>
</tr>
<tr>
<td>**okb3(i,j)</td>
<td>third outer k boundary value of variable ** <strong>floor for sweep i,j (flow in only)</strong></td>
<td>2</td>
</tr>
<tr>
<td>gpokb*(i,j)</td>
<td>analytical or preset values for gp on okb</td>
<td>0.0</td>
</tr>
</tbody>
</table>

```
namelist / okb /
  1 nokb , gokb , dokb1 , dokb2 , v1okb1
  2 , v1okb2 , v2okb1 , v2okb2 , v3okb1 , v3okb2
  3 , v3okb3
*if -def,ISO
  4 , e1okb1 , e1okb2
*endif -ISO
```
*if def, TWOFLUID
  5 , e2okb1 , e2okb2
*endif TWOFLUID
*if def, RADIATION
  6 , erokb1 , erokb2
*endif RADIATION
*if def, GRAV
  7 , gpokb1 , gpikb2
*endif GRAV
*if def, MHD
  8 , emf3okb1, emf3okb2, emf1okb1, emf2okb1, b2okb1
  9 , b2okb2 , b1okb1 , b1okb2
*endif MHD
*if def, AGING
  1 , daokb1 , daokb2
*endif AGING

B.14 GRVCON—GRavity CONtrol, (subroutine NMLSTS)

Gravitational self-potential is switched on by defining GRAV and aliasing GRAVITY to the desired gravity routine. If GRAVITY is aliased to gravity, the user must select the desired Poisson-solver by specifying a value for grvalg.

A point mass potential can be included by specifying a positive value for pmass. A point mass potential does not require defining GRAV, does not call the GRAVITY module, and is not included in the array gp which is strictly used for self-gravity. Its effect is explicitly added as velocity source terms in the routines stv1, stv2, and stv3. Thus, a point-mass potential may be used in conjunction with self-gravity or with self-gravity turned off. Note that pmass should scale with d*x1**3.

Finally, a uniform background acceleration (as on the surface of the Earth) may be imposed along one of the grid directions by specifying a non-zero value for gacc (negative value ⇒ toward inner boundary) and gdir. As with pmass, gacc is applied directly in STV* without requiring GRAV to be defined and independent of array gp. Note that gacc should scale with cs**2/dx1. Thus, on the surface of the Earth and with cs = 331.5 m/s = 1 (scaled) and dx1 = 1 m = 1 (scaled), g = 9.81 m/s^2 = 8.93 × 10^{-5} (scaled).

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcnst</td>
<td>gravitational constant</td>
<td>0.25/pi</td>
</tr>
<tr>
<td></td>
<td>= 0.25/pi for unitless calculations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 6.67428d-11 for mks (2006 CODATA value)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 6.67428d-08 for cgs</td>
<td></td>
</tr>
<tr>
<td>pmass</td>
<td>fixed central point mass object. If using scaled units, &quot;pmass&quot; (scaled point mass M) will depend on &quot;gcnst&quot;, the scaling for density, and grid size. For gcnst = 1/(4 pi), pmass = (4 pi G M) / (ds rs<strong>3), where &quot;ds&quot; is the density scale and &quot;rs&quot; the length scale. For M = 1 solar mass, ds = 3.0e5 H atoms per m</strong>3, and rs = 1.0e3 AU, pmass ~ 1.</td>
<td>0.0</td>
</tr>
<tr>
<td>iptmass</td>
<td>i index of point mass</td>
<td>ismn</td>
</tr>
<tr>
<td>jptmass</td>
<td>j index of point mass</td>
<td>jsmn</td>
</tr>
<tr>
<td>kptmass</td>
<td>k index of point mass</td>
<td>ksmn</td>
</tr>
<tr>
<td>gacc</td>
<td>uniform background acceleration field</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Appendix B: The namelists

Appendix B: The namelists

< 0 => toward inner boundary
> 0 => toward outer boundary

gdir
direction of gacc

grvalg
self-gravitational algorithm to be used.
1 = 1 => Successive Overrelaxation (SOR)
2 => Full Multi-grid (FMG)
3 => FFT/FST solvers

gcycle
maximum number of iterations for SOR
number of V-cycles for FMG

epsgrv
maximum tolerance for convergence of Poisson solvers

name_list / grvcon /
1
gcnst , ptmass , iptmass , jptmass , kptmass
2 , gacc , gdir , grvalg , gcyle , epsgrv

B.15 AMBCON—AMBipolar Diffusion CONTROL, (subroutine NMLSTS)

This namelist includes the parameters to control ambipolar diffusion (AD), and is included in the precompiled code if the AMBIDIFF macro is defined. Note that if gammaad<0, AD will not be applied.

Ambipolar diffusion is the physical process by which magnetic field is not fully coupled to the fluid. If only a portion of the fluid is ionised, the neutral particles are coupled to the magnetic field only via collisions with the ionised particles which are coupled to the field. Failure of such collisions to keep the ionised and neutral components together can cause some mass to “slip” relative to the magnetic field, a process known as ambipolar diffusion.

Done properly, one would track both the neutral and ionised components as two separate but co-spatial fluids. Given the current capability of ZEUS-3D to track a second fluid, this may not be terribly difficult to implement though, at the time of this writing, has not been tried. Instead, the one-fluid approach of Duffin & Pudritz (2008, MNRAS, 391, 1659) has been implemented which is computationally relatively inexpensive, but does limit its use to low-ionisation fluids.

As a parabolic term in an otherwise hyperbolic system of equations, the CFL-limited AD time step is typically much less (by a factor of ~100) than the MHD time step. Running the simulation on the AD time step (iscyad = 0), while being the most accurate, can be prohibitively expensive computationally. Alternately, one can run on the MHD time step and sub-cycle on the AD time step (iscyad = 1), much like is currently possibly for a diffusive second fluid (iscydf) and the artificial viscosity (iscyqq). This speeds up the 1-D C-shock test by a factor of ~5–6. Better still, one can “super-step” on the AD time step (Meyer et al., 2012, MNRAS, 422, 2102) giving a further factor of ~6 speed-up with little if any appreciable cost in accuracy over sub-cycling.

The default settings assume that the gas is a mix of 10% He and 90% H, by number of atoms. The ions are taken to be HCO+ and Na+, which both have masses of approximately 29.0 a.m.u., and hydrogen is assumed to be molecular. gammaad is given by:

\[ \gamma_{AD} = \frac{\langle \sigma v \rangle_{ni}}{m_i + m_n}, \]

where \( \langle \sigma v \rangle_{ni} \) is the collision rate between neutral and ionized particles \( 1.7 \times 10^{-9} \text{ cm}^3 \text{ s}^{-1} \) for
H$_2$), $m_i$ is the mass of an ion, and $m_n$ is the mass of a neutral particle. For the gas described here, $\gamma_{AD} = 3.28 \times 10^{13} \text{g}^{-1} \text{cm}^3 \text{s}^{-1}$.


### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| iscyad    | =0 => no subcycling for AD  
=1 => subcycling for AD  
=2 => super time-stepping for AD | 1       |
| gammaad   | drag coefficient representing coupling between ions and neutrals. <0 => huge, turning off AD. Default masses = 3.28d13 g^-1 cm^-3 s^-1. | -1.0    |
| betacoef  | numerator in expression for AD beta. For 10% He, 90% H, use 1.4. For 100% H, use 1.0. | 1.4     |
| ionconst  | =0 => calculate ion density using parameterisation  
=1 => constant ion density (for C-shock tests) | 0       |
| dscale    | scaling to convert ion density from ZEUS to cgs units. | 2.0d-18 |
| mion      | mass of ions in cgs units | 4.81d-23 |
| mneutral  | mass of neutrals in cgs units | 3.82d-24 |

**namelist / ambcon /**

1 iscyad, gammaad, betacoef, ionconst, dscale  
2 , mion, mneutral

B.16 EQOS—Equation Of State control (subroutine NMLSTS)

This namelist specifies the parameters which control the equation of state. Using all the defaults is recommended, unless a different adiabatic constant (gamma) is required. Note that if an isothermal equation of state is desired, setting the EDITOR definition ISO in addition to setting niso = 1 will allow execution to take advantage of the reduced computations necessary for isothermal systems. Parameters dimensioned with nmat allow for values to be set for both fluids if TWOFLUID is set, with the first element reserved for the first fluid (that which exists when TWOFLUID is not set), and the second element for the second (possibly diffusive) fluid enabled when TWOFLUID is set.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>gamma</td>
<td>(nmat) ratio of specific heats</td>
<td>5/3</td>
</tr>
<tr>
<td>rgas</td>
<td>(nmat) gas constant</td>
<td>1.0</td>
</tr>
</tbody>
</table>
| niso      | (nmat) = 0 => adiabatic eos  
= 1 => isothermal eos | 0       |
| ciso      | (nmat) isothermal sound speed                                              | 1.0     |
| rmetal    | (nmat) metallicity => cooling strength M-MML                               | 0.0     |
| diffc1, diffc2 | diffusion coefficient (for the second fluid) is  
set to diffc1 / B**diffc2 | 0.0     |

**namelist / eqos /**

1 gamma, rgas, niso, ciso, rmetal  
2 , diffc1, diffc2
B.17  GCON—Grid motion CONtrol (subroutine NMLSTS)

This namelist sets the parameters for grid motion, should a partial tracking of the flow be required. This feature has been dormant for years, and should this feature be desired, some code development may be required.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1fac</td>
<td>x1 motion factor</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>&lt; 0 gives &quot;Lagrangian&quot; tracking in x1 lines</td>
<td></td>
</tr>
<tr>
<td>x2fac</td>
<td>x2 motion factor</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>&lt; 0 gives &quot;Lagrangian&quot; tracking in x2 lines</td>
<td></td>
</tr>
<tr>
<td>x3fac</td>
<td>x3 motion factor</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>&lt; 0 gives &quot;Lagrangian&quot; tracking in x3 lines</td>
<td></td>
</tr>
<tr>
<td>ia</td>
<td>i&lt;ia =&gt; zone ratio is preserved in x1 lines</td>
<td>is=3</td>
</tr>
<tr>
<td>ja</td>
<td>j&lt;ja =&gt; zone ratio is preserved in x2 lines</td>
<td>js=3</td>
</tr>
<tr>
<td>ka</td>
<td>k&lt;ka =&gt; zone ratio is preserved in x3 lines</td>
<td>ks=3</td>
</tr>
<tr>
<td>igcon</td>
<td>selects grid treatment:</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>=0 =&gt; separate motion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>=1 =&gt; averaged motion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>=2 =&gt; tracking x1, x2, and x3 boundaries</td>
<td></td>
</tr>
<tr>
<td></td>
<td>=3 =&gt; averaged boundary tracking</td>
<td></td>
</tr>
<tr>
<td></td>
<td>=4 =&gt; input grid boundary speeds</td>
<td></td>
</tr>
<tr>
<td></td>
<td>vg1(ie) = x1fac * central sound speed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>vg2(je) = x2fac * central sound speed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>vg3(ke) = x3fac * central sound speed</td>
<td></td>
</tr>
</tbody>
</table>

```
namelist / gcon /
1     x1fac , x2fac , x3fac , ia , ja
2     , ka , igcon
```

B.18  EXTCON—grid EXTension CONtrol (subroutine NMLSTS)

This namelist controls the grid extension feature of the code. This is useful only for problems in which a shock separates quiescent material (which does not require updating) from material requiring computations. As the shock propagates across the grid, more zones are added to the computational domain until the entire domain has been included. Because quiescent zones are not being updated, a substantial savings in computation time could be realised. Use this feature with caution; improper use can be disastrous.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>instretch(1)</td>
<td>.le. 0 =&gt; perform computations over entire i-domain</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>.gt. 0 =&gt; i-index of first zone in initial i-domain</td>
<td>0</td>
</tr>
<tr>
<td>instretch(2)</td>
<td>.le. (1) =&gt; instretch(2)=istretch(1)+istretch(3)-1</td>
<td>0</td>
</tr>
<tr>
<td>instretch(3)</td>
<td>.le. 0 =&gt; 10</td>
<td>0</td>
</tr>
<tr>
<td>instretch(4)</td>
<td>.le. 0 =&gt; instretch(3)</td>
<td>0</td>
</tr>
<tr>
<td>jstretch(1,2,3,4)</td>
<td>same as &quot;istretch&quot;, but for 2-direction.</td>
<td></td>
</tr>
<tr>
<td>kstretch(1,2,3,4)</td>
<td>same as &quot;istretch&quot;, but for 3-direction.</td>
<td></td>
</tr>
<tr>
<td>extvar</td>
<td>specifies variable used to detect dis-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>turbance in the quiescent ambient medium</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(character*2). Legal values are: 'd ',</td>
<td></td>
</tr>
<tr>
<td></td>
<td>'e ' (pressure), 'se' (temperature).</td>
<td></td>
</tr>
</tbody>
</table>
Appendix B: The namelists

Note that \texttt{ismn} and \texttt{iemx} are the user-imposed limits of the grid in the \textit{i}-direction, while \texttt{is} and \texttt{ie} are the \textit{i}-limits of the do-loops. With grid extension off, \texttt{is = ismn} and \texttt{ie = iemx}. With grid extension on, \texttt{is .ge. ismn} and \texttt{ie .le. iemx} (§C.1). \texttt{is} is decremented by \texttt{istretch(3)} and/or \texttt{ie} is incremented by \texttt{istretch(4)} whenever the quiescent value of the specified variable (\texttt{extvar}) changes by 3\% within 5 zones of the current domain boundary. Note that \texttt{is} will not be permitted to fall below \texttt{ismn} and \texttt{ie} will not be permitted to rise above \texttt{iemx}. Grid extension in the \textit{i}-direction is turned off by keeping \texttt{istretch(1) = 0} (its default value).

An entirely analogous discussion holds for the \textit{j}- and \textit{k}-directions.

\begin{verbatim}
  namelist / extcon /
    istretch, jstretch, kstretch, extvar
\end{verbatim}

\section*{B.19 PL1CON—PLot (1-D) CONtrol (subroutine NMLSTS)}

This namelist controls the 1-D graphics. During a run, as many as \texttt{nios} 1-D slices may be specified for each variable plotted, where \texttt{nios} is a parameter set before compilation. For every slice chosen, a file (in either metacode or postscript) is created with a plot generated for each variable specified. These plots may be arranged in the same frame or separate frames, and can have any rectangular shape desired. All plots are of publication quality. Each 1-D slice (bounded by \texttt{x1ipl1mn}, \texttt{x1ipl1mx}, \texttt{etc.}) runs parallel to one of the axes of the computational grid. To specify the slice uniquely, two of \texttt{ipl1}, \texttt{jpl1}, and \texttt{kpl1} must be set. If \texttt{ipl1}, \texttt{etc.} is 0, \texttt{x1ipl1mn}, \texttt{etc.} is used instead.

For 1-D runs such as shock-tube tests, the analytical solution may be overlaid by setting \texttt{EDITOR} macro \texttt{RIEMANN} and setting the namelist parameters \texttt{pl1soln} and \texttt{xdiscpl1}. The Riemann solver (courtesy of Tom Jones) takes the end points of the 1-D run as the left and right states, and thus the run should stop before the boundaries are reached.

\textit{N.B.} For restarted runs in which the computation is resumed on a larger or smaller grid and where the default values for \texttt{x1ipl1mn}, \texttt{x1ipl1mx}, \texttt{etc.} were used in the initial run, it will be necessary to set \texttt{x1ipl1mn}, \texttt{x1ipl1mx}, \texttt{etc.} in the input deck for the restarted run to the extrema of the new grid if the plots are to extend to the bounds of the new grid. Otherwise, the plots will be bound by the old grid.

\begin{verbatim}
  parameter  description                          default

  General

  dtp1l      physical (problem) time interval between 1-D 0.0
             plot dumps. 0.0 => no plots.
  npl1dmp    the sequential number for the next 1-D plot file -1
             < 0 => npl1dmp = npl1dmp (default)
  units      sets the units of angular dimensions (character*2) 'rd'
             'dg' => degrees, 'rd' => radians
             'pi' => units of pi radians
  aspect     < 1.0 => plots are wide and short 1.0
             = 1.0 => square plots
             > 1.0 => plots are tall and narrow
  pwtpl1     pen weight for 1-D plots 1.0
  norppl1    = 1 => use NCAR graphics library for 1-D plots 2
             = 2 => use PSPLOT graphics library for 1-D plots
\end{verbatim}
Appendix B: The namelists

corl  selector for lines and symbols for plots  1
    =-6 => open diamonds; one per plot element
    =-5 => open pentagrams
    =-4 => asterisk (*)
    =-3 => open squares
    =-2 => open triangles
    =-1 => plus signs (+)
    = 0 => crosses (x)
    = 1 => open circles (o)
    = 2 => smooth curve drawn through points
    any other value => no plot

npl1h  number of frames horizontally per page  1
npl1v  number of frames vertically per page  1
allxl  = 2 => all x labels/annotations are drawn  2
        = 1 => only the x-label/annotations on the bottom
        row of plots on the page are drawn.
        = 0 => labels aren’t even drawn on the bottom row.
        allxl < 0 same as abs(allxl) except last annotation
        is not drawn if it falls at right edge of frame.
        If plots are assembled a posteriori, this is useful
        to prevent right-most annotation of left plot from
        overwriting left-most annotation of right plot.
allyl  = 2 => all y labels/annotations are drawn  2
        = 1 => only the y-label/annotations on the left
        column of plots on the page are drawn.
        = 0 => labels aren’t even drawn on the left column.
pl1hdr  = 0 => suppresses header  1
        = 1 => header is printed on top of plot
pl1ftr  = 0 => suppresses footer  1
        = 1 => footer is printed on bottom of plot

Slice control

pl1dir (nios) direction of 1-D plot: 1, 2, or 3 => 1-, 2-, or 3-direction. 0 => no plots.
    (is+ie)/2
ipl1 (nios) i index of 1-D plot in 2- or 3-direction
    (js+je)/2
jpl1 (nios) j index of 1-D plot in 3- or 1-direction
    (ks+ke)/2
kpl1 (nios) k index of 1-D plot in 1- or 2-direction

x1pl1mn (nios) minimum x1 of slice; or used by "pl1mean" x1a(is)

x1pl1mx (nios) maximum x1 of slice; or used by "pl1mean" x1a(ie+1)

x2pl1mn (nios) minimum x2 of slice; or used by "pl1mean" x2a(js)

x2pl1mx (nios) maximum x2 of slice; or used by "pl1mean" x2a(je+1)

x3pl1mn (nios) minimum x3 of slice; or used by "pl1mean" x3a(ks)

x3pl1mx (nios) maximum x3 of slice; or used by "pl1mean" x3a(ke+1)

ipl1mn (nios) i-index of minimum x1 of slice  0
ipl1mx (nios) i-index of maximum x1 of slice  0
jpl1mn (nios) j-index of minimum x2 of slice  0
jpl1mx (nios) j-index of maximum x2 of slice  0
kpl1mn (nios) k-index of minimum x3 of slice  0
kpl1mx (nios) k-index of maximum x3 of slice  0
pl1xlab (nios) character string containing NCAR mark-up language
(see below) for the desired x-label. Default is
the generic coordinate label (x1, x2, x3).

NCAR markup syntax:

    Fonts: |F0| = times-roman (default)
    |F10| = times-roman italic
    |F14| = times-roman italic bold
Appendix B: The namelists

\[ |F5| = \text{greek italic}: \; |F5|r \Rightarrow \rho; \; |F5|w \Rightarrow \omega \]
\[ |F18| = \text{math characters}: \; |F18|A \Rightarrow \perp; \; |F18|R \Rightarrow \parallel \]
\[ |F34| = \text{math characters}: \; |F34|W \Rightarrow \cdot; \; |F34|Q \Rightarrow \nabla \]

Position: \(|B| \Rightarrow \text{subscript}; \; |S| \Rightarrow \text{superscript}; \; |N| \Rightarrow \text{normal}\)

One may combine one position designator with one font change in each "field" (bounded by \(||\)), and then string as many fields together as one needs:

\[ "|F10|e|BF0|t|NF10|+e|BF0|r" = e_{\text{t}} + e_{\text{r}} \]

Variable control

pl1var (niov) names of variables to be plotted (character*2).

Valid names are: 'a' (vector potential norm), 'a1', 'a2', 'a3' (vector potential components), 'ag' (synchrotron age), 'B3' (b3*b3), 'b' (magnetic field norm), 'b1', 'b2', 'b3', 'ba', 'bb', 'bc' (magnetic field components), 'bd' (magnetic field norm/density), 'BP', 'BR' (phi- and radial magnetic field components), 'br' (bremsstrahlung emissivity), 'bt' (plasma beta), 'cs' (sound speed), 'd' (density), 'da' (density-age), 'e1' (1st int. energy), 'e2' (2nd int. energy), 'er' (radiation energy), 'et' (total energy density), 'g1' [(d(r 1t) / dr), 'gk' [(d(r kt) / dr], 'gp' (gravitational potential), 'hy' [((1/rho) dp/dr - 3/(2r**2)], 'j' (current density norm), 'j1', 'j2', 'j3' (current density components), 'k1' (entropy of 1st fluid), 'k2' (2nd entropy), 'ka' (avg. 1st + 2nd ent.), 'kt' (kinetic viscous radial trans.), 'lu' (radiative luminosity), 'l' (angular momentum), 'lt' (angular momentum radial transport), 'm' (Mach number), 'mA' (Alfven Mach number), 'mf' (fast MS Mach number), 'ms' (slow MS Mach number), 'om' (angular velocity), 'p1' (1st thermal pressure), 'p2' (magnetic pressure), 'p3' (1st ther. + mag. pres.), 'p4' (2nd thermal pressure), 'p5' (1st ther. + 2nd ther. pres.), 'p6' (mag. + 2nd ther. pres.), 'p7' (1st ther. + mag. + 2nd ther. pres.), 'pa' (pitch angle; atan(b1/bP]), 'pg' (pseudo-gravitational potential), 'ps' [(atan(b3/b2)], 's1', 's2', 's3' (momentum components), 'sy' (synchrotron emissivity), 'u1' (1st specific int. energy), 'u2' (2nd specific int. energy), 'v' (speed), 'v1', 'v2', 'v3', 'va', 'vb', 'vc' (velocity components), 'vA' (Alfven speed), 'vf' (fast speed), 'vs' (slow speed), 'vv' [div(v)], 'w' (vorticity norm), 'w1', 'w2', 'w3' (vorticity components).

nlpl1 (niov) = 1 => linear-linear plot
= 2 => log(y)-linear(x) plot
= 3 => linear(y)-log(x) plot
= 4 => log-log plot

pl1min (niov) minimum value to be plotted. 0.0
pl1max (niov) maximum value to be plotted. 0.0
plimm = 1 => "pl1min" and "pl1max" are set to extrema 1
of each variable.

= 0 => Use input values of "pl1min" & "pl1max" to
set range of plot. If pl1max .le. pl1min,
they are set to extrema (as though pl1mm=1)

= -1 => same as 0, but extend plot range by one
minor tick mark in each direction

pl1mean (niov) = 0 => ordinary 1-D slices; no means taken.
= 1 => 1-D slices are filled with means of
variable across plane (limited by x*pl1mn
and x*pl1mx) orthogonal to the slice.

thetapl1 meridional rotational angle between 1-2-3 & a-b-c 0.0
coordinates (to compute va vb vc from v1 v2 v3)

phipl1 azimuthal rotational angle between 1-2-3 & a-b-c 0.0
coordinates (to compute va vb vc from v1 v2 v3)

pl1ylab (niov) character string containing NCAR mark-up la
nguage for the desired y-label. Most defaults are fine,
unless, for e.g., v_\phi is desired over v_3.

Overlays

pl1grid every (pl1grid)th a-grid line is drawn as a fine
vertical line (for both regular and diagn. plots)
= 0 => no grid lines are drawn

pl1ref (niov) dashed reference line drawn on plot
(< -1.0d30 => no line)

plisoln (niov) = 0 => no overlay
= 1 => Analytical solution to Riemann problem
using end points of slice as left- and
right-states overlaid.
= 2 => analytical solution plotted without
numerical solution

xdiscpl1(nios) location of discontinuity (default: middle of
slice)

Diagnostic dumps

dtgrid physical (problem) time interval between diag-
nostic 1-D graph dumps (0.0 => no plots)

If you are using NCAR graphics (norppl1=1), you will need to link all NCAR graphics
libraries to your executable (see your system administrator if you do not know what or where
these libraries are) as well as two user-created libraries, grfx03.a and psplot.a. If you are
using PSPLIT (norppl1=2) exclusively, you can dispense with all the NCAR libraries, and
just link grfx03.a, psplot.a, and noncar.a, the last of which serves only to satisfy external
calls to non-existing NCAR subroutines.

```
name list / pl1con /
 1 dtp1 , npl1dmp , units , aspect , pwtpl1
 2 norppl1 , corl , npl1h , npl1v , allxl
 3 allyl , pl1hdr , pl1ftr , pl1dir , ip1l
 4 jpl1 , kppl1 , xpl1mn , xpl1mx , x2pl1mn
 5 x2pl1mx , x3pl1mn , x3pl1mx , ipl1mn , ipl1mx
 6 jpl1mn , jpl1mx , kpl1mn , kpl1mx , pl1xlab
 7 plivar , nlpl1 , pl1min , pl1max , pl1mm
 8 pl1mean , thetapl1 , phipl1 , pl1ylab , pl1grid
 9 , pl1ref , plisoln , xdiscpl1 , dtgrid
```
Appendix B: The namelists

B.20 PL2CON—PLOT (2-D) CONtrol (subroutine NMLSTS)

This namelist controls the 2-D graphics. During a run, as many as nios 2-D slices may be specified for each variable plotted. For every slice chosen, a file (metacode or postscript) is created with a plot generated for each individual or combination of variables specified. The normal to each slice is parallel to one of the axes of the computational grid and is specified by pl2dir. The extent of the slice is limited by x1pl2mn, x1pl2mx, etc., while the index at the base of the normal to the slice is given by 1p12. If ipl2mn, etc. is 0, x1pl2mn, etc. is used instead.

New to version 3.6 is buffet-style plotting in which the user may select for each plot as many as two scalars, three vector fields, and the positions of some or all of the “corks” (Lagrangian particles). This is accomplished by the addition of several new namelist parameters to allow more than one variable to be plotted per plot, and users used to previous versions of ZEUS-3D should pay special attention to these new namelist parameters described below. In previous versions of ZEUS-3D, only a select few combination plots were available (e.g., density plus velocity vectors). With the introduction of buffet-style plotting, the combination of variables that may be plotted together is now virtually limitless.

For PSPL0T graphics, the first scalar may be plotted using colour, grey scale, or line contours while the second scalar may be plotted using line contours only. If colour or grey scale contours is selected for the first scalar (pl2ccol=1,2), a colour bar is included immediately to the right of the plot (to indicate what values of the scalar correspond to each colour); this may be suppressed by negating pl2ccol. Each vector field may be drawn with different colours and a legend indicating which variables are represented by which contours and vectors is drawn in the upper right corner. The introduction of a legend (controlled by pl2leg) has obviated the need for a title, and thus pl2hdr is no longer a valid namelist parameter.

For NCAR graphics, both scalars must be line contours, and only one vector field may be specified. A legend is also produced instead of a title.

PSPL0T 2-D graphics can generate enormous post-script files; a single file with several variables plotted in full colour from a high-resolution data-slice (e.g., 1,000 zones on a side) can approach 1 Gbyte. Not only does this needlessly occupy disc space and cause commonly-available graphics-viewers such as ghostview to take forever opening each page, it can also soak up all the available memory on the machine while the plots are being generated, limiting the number of zones in the simulation. Such plots tend to be hyper-resolved often with multiple graphics instructions assigned to each screen pixel, and are rarely worth the computational effort.

New to version 3.6, one can now mitigate against this by specifying parameters np12i and np12j to be smaller than the actual grid dimensions (e.g., of the order 250–500). These parameters set the dimensions of the re-binning grids into which the 2-D data-slices are rebinned (much like pixel plots) before plotting. This can decimate the size of the post-script file while still generating perfectly acceptable plots, qualitatively identical to the hyper-resolved ones. (N.B., If the “hyper-resolved” plots are still wanted—e.g., for a poster—one can always dump HDF files as often as or instead of 2-D plots, then use PLOTZ after the simulation is completed to generate the desired plots.)

All plots are of publication quality and come fully labelled, including a time stamp.
for easy identification. Unlike the 1-D plots, only one plot may be written to each frame. However, the plot may be scaled down (pl2scale) if desired.

N.B. For restarted runs in which the computation is resumed on a larger or smaller grid, and where the default values for x1pl2mn, x1pl2mx, etc. were used in the initial run, it will be necessary to set x1pl2mn, x1pl2mx, etc. in the input deck for the restarted run to the extrema of the new grid if the plots are to extend to the bounds of the new grid. Otherwise, the plots will be bound by the old grid.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dtp12</td>
<td>physical (problem) time interval between 2-D plot dumps. 0.0 =&gt; no plots.</td>
<td>0.0</td>
</tr>
<tr>
<td>npl2dmp</td>
<td>the sequential number for the next 2-D plot file</td>
<td>-1</td>
</tr>
<tr>
<td>units</td>
<td>sets units of angular dimensions (character*2). 'dg' =&gt; degrees, 'rd' =&gt; radians, 'pi' =&gt; units of pi radians</td>
<td>'zz'</td>
</tr>
<tr>
<td>pl2scale</td>
<td>.lt. 1.0d0 =&gt; scaling factor to reduce plot size .ge. 1.0d0 =&gt; full size</td>
<td>1.0</td>
</tr>
<tr>
<td>pwtpl2</td>
<td>pen weight for 2-D plots</td>
<td>1.0</td>
</tr>
<tr>
<td>norppl2</td>
<td>= 1 =&gt; use NCAR graphics library for 2-D plots = 2 =&gt; use PSPLDT graphics library for 2-D plots</td>
<td>2</td>
</tr>
<tr>
<td>pl2grid</td>
<td>= 0 =&gt; do not overlay grid lines onto plots &gt; 0 =&gt; overlay a-grid &lt; 0 =&gt; overlay b grid every mod(</td>
<td>pl2grid</td>
</tr>
<tr>
<td>slice control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pl2dir (nios)</td>
<td>direction of normal to image plane: 1, 2, or 3 =&gt; 1-, 2-, or 3-direction. 0 =&gt; no plots.</td>
<td>0</td>
</tr>
<tr>
<td>lpl2 (nios)</td>
<td>level of 2-D plot (value of 1-, 2-, or 3-index) (is+ie)/2</td>
<td>(is+ie)/2</td>
</tr>
<tr>
<td>pl2avg (nios)</td>
<td>= 1 =&gt; averages slice with lpl2-1 = 0 =&gt; no average taken (default)</td>
<td>0</td>
</tr>
<tr>
<td>x1pl2mn (nios)</td>
<td>minimum x1 of plot window</td>
<td>x1a(ismn )</td>
</tr>
<tr>
<td>x1pl2mx (nios)</td>
<td>maximum x1 of plot window</td>
<td>x1a(iemxp1)</td>
</tr>
<tr>
<td>x2pl2mn (nios)</td>
<td>minimum x2 of plot window</td>
<td>x2a(jsmn )</td>
</tr>
<tr>
<td>x2pl2mx (nios)</td>
<td>maximum x2 of plot window</td>
<td>x2a(jemxp1)</td>
</tr>
<tr>
<td>x3pl2mn (nios)</td>
<td>minimum x3 of plot window</td>
<td>x3a(ksmn )</td>
</tr>
<tr>
<td>x3pl2mx (nios)</td>
<td>maximum x3 of plot window</td>
<td>x3a(kemxp1)</td>
</tr>
<tr>
<td>ipl2mn (nios)</td>
<td>i-index of minimum x1 of plot window</td>
<td>0</td>
</tr>
<tr>
<td>ipl2mx (nios)</td>
<td>i-index of maximum x1 of plot window</td>
<td>0</td>
</tr>
<tr>
<td>jpl2mn (nios)</td>
<td>j-index of minimum x2 of plot window</td>
<td>0</td>
</tr>
<tr>
<td>jpl2mx (nios)</td>
<td>j-index of maximum x2 of plot window</td>
<td>0</td>
</tr>
<tr>
<td>kpl2mn (nios)</td>
<td>k-index of minimum x3 of plot window</td>
<td>0</td>
</tr>
<tr>
<td>kpl2mx (nios)</td>
<td>k-index of maximum x3 of plot window</td>
<td>0</td>
</tr>
<tr>
<td>iflpl2 (nios)</td>
<td>= 0 =&gt; no flipping of plots = 1 =&gt; plot flipped about x-axis before writing</td>
<td>0</td>
</tr>
<tr>
<td>jflpl2 (nios)</td>
<td>= 0 =&gt; plot not flipped about x-axis = 1 =&gt; plot flipped about y-axis before writing</td>
<td>0</td>
</tr>
<tr>
<td>irefpl2 (nios)</td>
<td>= 0 =&gt; plot not reflected about x-axis = 1 =&gt; plot reflected about x-axis before writing rendering plot twice as tall</td>
<td>0</td>
</tr>
<tr>
<td>jrefpl2 (nios)</td>
<td>= 0 =&gt; plot not reflected about y-axis = 1 =&gt; plot reflected about y-axis before writing</td>
<td>0</td>
</tr>
</tbody>
</table>
Appendix B: The namelists

rendering plot twice as long

\begin{verbatim}
np12i (nios)  no. of horizontal zones in rebinned grid (<= idim)  0
  = 0 => no rebinning in x- or y-direction
np12j (nios)  no. of vertical zones in rebinned grid (<= jdim)  0
  = 0 => no rebinning in x- or y-direction
pl2xlab (nios)  character string containing NCAR mark-up language
  for the desired x- and y-labels. Default is the
  generic coordinate label (x1, x2, x3).
pl2y1 (nios)  = 0 => suppresses xlabel (but not x-annotations)  1
  = 1 => xlabel is put below horizontal axis
pl2y1 (nios)  = 0 => suppresses ylabel (but not y-annotations)  1
  = 1 => ylabel is put beside vertical axis
pl2ftr (nios)  = 0 => suppresses footer 1
  = 1 => footer is printed on bottom of plot
pl2leg (nios)  = 0 => suppresses legend 2
  = 1 => legend is printed above plot (as a header)
  = 2 => legend is printed on inside top right
  corner of plot

Contours; for contour variables, *= 1,2

pl2sca* (niov)  names of scalars to plot (character*2)
  Valid names are: 'a ', 'a1', 'a2', 'a3', 'ag',
  'an' (normal vector potential), 'b ', 'b1', 'b2',
  'b3', 'bn' (normal magnetic field), 'bP', 'bR',
  'bt', 'cs', 'd ', 'da', 'e1', 'e2', 'er', 'et',
  'fn' (normal magnetic flux function), 'gg', 'j',
  'j1', 'j2', 'j3', 'jn' (normal current density),
  'k1', 'k2', 'ka', 'lu', 'm ', 'mA', 'mf', 'ms',
  'nb' (break frequency), 'p1', 'p2', 'p3', 'p4',
  'p5', 'p6', 'p7', 'pa', 'pg', 's1', 's2', 's3',
  's' (momentum norm), 'sn' (normal momentum),
  'sy', 'u1', 'u2', 'v ', 'v1', 'v2', 'v3', 'vA',
  'vf', 'vn' (normal velocity), 'vs', 'vv', 'w',
  'w1', 'w2', 'w3', 'wn' (normal vorticity).
  See "pl1ivar" in pl1con for other definitions.
pl2ng* (niov)  = 1 => contours are spaced evenly
  > 1 => contours are spaced geometrically such
  that first two contours are spaced "pl2ng*"1
  times closer than evenly spaced contours.
  <-1 => contours are spaced geometrically such
  that last two contours are spaced "-pl2ng*"2
  times closer than evenly spaced contours.
pl2min* (niov)  minimum contour level to be plotted. 0.0
pl2max* (niov)  maximum contour level to be plotted. 0.0
pl2mm*  = 0 => use input "pl2min*" & "pl2max*" for plots
  = 1 => compute "pl2min*" & "pl2max*" for plots
  If both "pl2min*" and "pl2max*" are 0.0, compute
  them as though "pl2mm*" were 1 for that variable.
theta_pl2  meridional rotation angle between 1-2-3 and a-b-c
  coordinates (compute va, vb, vc from v1, v2, v3)
phi_pl2  azimuthal rotation angle between 1-2-3 and a-b-c
  coordinates (compute va, vb, vc from v1, v2, v3)
numcl* (niov)  = 0 => min(20,ncls) evenly spaced contours are
drawn between "pl2min*" and "pl2max*"
  > 0 => number of evenly spaced contours to draw
  between "pl2min*" and "pl2max*" (max ncls).
  < 0 => abs(numcl*) contours drawn and are
    specified by "pl2min*" and "levs*".
\end{verbatim}
Appendix B: The namelists

if numcl*(2:niov) are not set by user, they are set to numcl*(1).

levs* (ncls) real array specifying multiples of "pl2min*" to use for contour levels (for numcl* < 0 only). If levs2(1) not set by user, levs2 = levs1.

negcon = 0 => negative contours are indistinguishable from positive contours.
= 1 => negative contours drawn with dotted lines
hilo = 0 => highs and lows are not labelled
= 1 => highs and lows are labelled H/L
= 2 => highs and lows are labelled H/L with values as subscripts (NCAR only).

pl2ccol (niov) = 0 => do not use colour or grey-scale
= 1 => use grey-scale to fill between contour
= 2 => use colour to fill between contour
if pl2ccol(2:niov) are not set by user, they are set to pl2ccol(1). Used for PSPLLOT only.
pl2ccol < 0 same as abs(pl2ccol) except colour bar not drawn (contour levels listed in footer instead)

pl2clin1(niov) controls type of contour lines for first scalar
= 0 => no contours (default if pl2ccol .ne. 0)
= 1 => solid contours (default if pl2ccol=0 or NCAR)
=-1 => dashed contours
pl2clin2(niov) controls type of contour lines for second scalar
= 0 => no contours
= 1 => solid contours (default if pl2clin1 .ne. 1)
=-1 => dashed contours (default if pl2clin1=1)
abs(pl2clin*) = 2 => contour pen weight halved (recommended).
If pl2clin*(2:niov) not set by user, they’re set to pl2clin*(1).

Vectors; for vector variables, θ = 1,2,3

pl2vec@ (niov) names of poloidal vectors to plot (character*2) ‘zz’
Valid names are: ‘a ’ (vector potential), ‘b ’ (magnetic field), ‘cf’ (centrifugal force), ‘gp’ [grad(p)], ‘gr’ (gravitational force), ‘j ’ (current density), ‘lf’ (Lorentz force), ’s ’ (momentum), ’v ’ (velocity), ’w ’ (vorticity).

vnorm@ (niov) >=1.0 => normalise vectors, draw all vectors with original length > 1.0e-24 * vmax
> 0.0 => do not normalise vectors, draw vectors
< 1.0 .ge. vnorm@ * vmax only
= 0.0 => do not normalise vectors, draw vectors .ge. 0.04 * vmax only
< 0.0 => normalise vectors, draw all vectors with original length > 10**vnorm@ * vmax

vscale scaling factor for vector 0.8

nxvec (nios) |nxvec| = # of vectors in x-direction 20
> 0 => different vector fields are staggered in x
< 0 => different vector fields have same x-coord
= 0 => same spacing (not #) as y-direction

nyvec (nios) |nyvec| = # of vectors in y-direction 0
> 0 => different vector fields are staggered in y
< 0 => different vector fields have same y-coord
= 0 => same spacing (not #) as x-direction

pl2vcol@ (niov) colours to use for vector fields (PSPLLOT only)
= 0 => white; = 1 => purple; = 2 => blue;
Appendix B: The namelists

= 3 => green; = 4 => orange; = 5 => red; = 6 => black
(Defaults: 6 for 1st field; 5 for 2nd, 2 for 3rd)
if pl2vcol@2:niov are not set by user, they are set to pl2vcol@1.

Corks

pl2crk (niov) = 0 => no corks overlaid on plots
> 0 => overlay every (pl2crk)th cork as a small dot
< 0 => same as >0 plus cork number to upper left
In 3-D, corks within a half a zone of specified plane are plotted.

Diagnostic dumps

dtgr2d physical (problem) time interval between diagnostic 2-D dumps; 0.0 => no dumps

See note at the end of §B.19 describing the namelist pl1con concerning libraries required with and without NCAR graphics.

```
/ pl2con /
1  dtp12  np12dmp  units  pl2scale  pwtpl2
2 , norpp12  pl2grid  pl2dir  lp12  p12avg
3 , xpl2mn  xpl2mx  xpl2mn  xpl2mx  xpl2mn
4 , x3pl2mn  pl2mn  xpl2mx  jpl2mn  jpl2mx
5 , kpl2mn  kpl2mx  ifl1ppl2  jfl1ppl2  irfl1ppl2
6  jref1pl2  np12i  np12j  pl2xlab  pl2ylab
7  pl2x1  pl2y1  pl2ftr  pl2leg  pl2sca1
8  pl2sca2  pl2sg1  pl2ng2  pl2min1  pl2min2
9  pl2max1  pl2max2  pl2mm1  pl2mm2  thetapl2
1  phipl2  numcl1  numcl2  plevs1  plevs2
2  negcon  hilo  pl2ccol  pl2cl1n1  pl2cl1n2
3  pl2vec1  pl2vec2  pl2vec3  vnorm1  vnorm2
4  vnorm3  vscale  nxvec  nyvec  pl2vcol1
5  pl2vcol2  pl2vcol3  pl2crk  dtgr2d
```

B.21 PIXCON—PIXel/movie graphics CONtrOol (subroutine NMLSTS)

This namelist controls the pixel dumps, which are 2-D images of slices through the data volume rebinned to a uniform, square Cartesian grid. **New to version 3.6,** depending on the new parameter pixfmt, images may be dumped as movie frames (four bytes deep with grid and dimensions stored in a header) and/or traditional pixel dumps (one byte per datum, no header) and/or HDF files. As always, pixel dumps remain as separate files, to be dealt with manually by the user after the run. HDF files are a fossil from very early versions of the code, and are suitable for graphical analysis if desired. For movie frames, at the end of the run `makempg` gathers all files in a tarball (to be used at a later time by ANIM8Z to recreate the animation with a different palette or bracketing if desired), converts them to PPM format, and assembles them into an mpeg movie file according to user-set namelist parameters. Thus, the end product is an mpeg animation for each quantity selected along with an accompanying tarball of four-byte deep frames. For a smooth temporal animation, aim for about 400–600 dumps per animation.
Appendix B: The namelists

During a run, as many as $nios$ slices may be specified for each variable plotted, and a single movie frame/pixel dump is created for every variable and every slice specified. The number of files generated per run can therefore be enormous. The extent of the 2-D slice can be limited by setting $x1pixmn$, $x1pixmx$, etc. The normal to the slice is parallel to one of the axes of the computational grid and is specified by $pixdir$. The index at the base of the normal is given by $lpix$.

Pixel images ($pixfmt=2$) were introduced by Michael Norman in the late 1980s as a way to assemble animations with high temporal resolution without overrunning the limited disc space of the day. Each image is only 1 byte deep, and thus has a dynamic range of 256; sufficient for qualitative rendering only if the data are properly bracketed and scaled beforehand. To do this, two parameters were introduced.

First, by setting $pixmm=1$ (automatic bracketing), each image is bracketed separately to its own extrema and differences from one image to the next are caused by both the evolution of the flow and the fluctuations of the extrema. By setting $pixmm=0$ (manual bracketing), the same user-set values for $pixmin$ and $pixmax$ are applied to each image, and fluctuations from one image to the next are entirely because of evolution of the flow. A practical way of finding optimal values for $pixmin$ and $pixmax$ is to run a low resolution run with $pixmm$ set to 1, then use the reported extrema (in the log file, $z1nnn$) of the pixel dumps to set $pixmin$ and $pixmax$ for the high resolution run (with $pixmm$ set to 0).

Second, the parameter $nlpix$ sets the scaling (“transfer function”). Setting $nlpix=1$ gives a linear scaling whereas $nlpix>1$ gives an essentially logarithmic scaling, where the colour contours at the low end are $nlpix$ times closer together than at the high end. For positive-definite scalars, $nlpix=100$ usually works well; for vector components that can be either positive or negative, a linear transfer function is usually best. Note, however, that quantities need not be positive definite in order to have “logarithmic scaling” applied.

Pixel images were originally designed for use with Carol Song’s IMAGETOOL which was great in its time but sadly discontinued by the NCSA in the late 90s. I know of no software now which can read separate ZEUS-style pixel dumps and render them as animations the way IMAGETOOL did. Thus, pixel dumps are now largely obsolete. In order to create an animation from them, one has to convert each to PPM format (tripling their size), then assemble the PPM files in an mpeg using software such as FFmpeg.

Instead, movie frames ($pixfmt=1$, default) are the recommended way to create animations. These files are 4-bytes deep with some header information, and are automatically converted to PPM files at the end of a run which are then assembled into an mpeg by the subroutine $makempg$ (which relies on FFmpeg being available on the user’s platform; see ffmpeg.org if it isn’t). The movie files are compressed into a g-zipped tarball, and the user is left with two files per desired animation: the mpeg, and the tarball which can be used later to recreate the animation without rerunning the simulation should the bracketing or scaling of the animation need to be changed (using the standalone program ANIM8Z). Thus, while movie files still need to be bracketed and scaled as pixel dumps, choosing the right values for $pixmin$, $pixmax$, and $nlpix$ before the simulation begins is not critical. Note also for movie files, $pixmm=2$ sets $pixmin$ ($pixmax$) to the minimum (maximum) of the minima (maxima) of all the frames in a single animation. This obviates the need to run a low-resolution simulation to determine the optimal values of $pixmin$ and $pixmax$. Even if the absolute extrema of the extrema are not optimal, one can redo the animation from the tarball after
the simulation using ANIM8Z.

Polar slices are binned to a Cartesian grid before they are written to disc. If a polar grid includes very small zones near the origin, it may be best to request two animations for each slice visualised. One animation could include the entire grid and reflect the resolution near the mid-radial regions (i.e., oversample the outer grid but under-sample the inner), while the second could include only the inner radial regions to reflect the resolution of the inner grid.

The parameters which set the dimensions of the arrays for the slices (nxpix, nypix) are independent of the parameters which set the dimensions of the flow variables (in, jn, kn). Thus, in the case of a non-uniform grid, files may be written with enough pixels to preserve the highest resolution on the grid.

_N.B._ For restarted runs in which the computation is resumed on a larger or smaller grid, and where the default values for x1pixmn, x1pixmx, etc. were used in the initial run, it will be necessary to set x1pixmn, x1pixmx, etc. in the input deck for the restarted run to the extrema of the new grid if the dumps are to extend to the bounds of the new grid. Otherwise, the dumps will be bound by the old grid.

Also for a restarted run, the g-zipped movie tarballs from the previous run should be in the same directory as the executable so that the new movie frames can be added to the tarballs, and the mpegs can be made from all movie frames since the beginning of the first run, rather than just from those created from the beginning of the present run. To enable this feature, neither the problem tag nor the names of the movie tarballs should be changed from one run to the next.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dtpix</td>
<td>problem time interval between pixel dumps/movie frames. If dtpix .le. 0, no files are generated.</td>
<td>0.0</td>
</tr>
<tr>
<td>npixdmp</td>
<td>sequential number for next pixel/movie file</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>&lt; 0 =&gt; npixdmp = ipixdmp</td>
<td></td>
</tr>
<tr>
<td>units</td>
<td>sets units of angular dimensions (character*2). 'rd' =&gt; degrees, 'rd' =&gt; radians, 'pi' =&gt; units of pi radians</td>
<td></td>
</tr>
<tr>
<td>pixfmt</td>
<td>= 1 =&gt; PIXDMP creates 4-byte movie frames.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>= 2 =&gt; PIXDMP creates 1-byte raw pixel dumps.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 3 =&gt; PIXDMP creates movie and raw files (1+2).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 4 =&gt; PIXDMP creates 4-byte HDF files.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 5 =&gt; PIXDMP creates movie and HDF files (1+4).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 6 =&gt; PIXDMP creates raw and HDF files (2+4).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>= 7 =&gt; PIXDMP creates files of each format (1+2+4).</td>
<td></td>
</tr>
<tr>
<td>Slice control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pixdir</td>
<td>direction of normal to image plane. 1, 2, or 3 =&gt; 0 1-, 2-, or 3-direction. If &quot;pixdir&quot; is not set, no pixel dumps will be generated.</td>
<td>0</td>
</tr>
<tr>
<td>lpix</td>
<td>level of 2-D pixel dump (value of 1-, 2-, or 3-index)</td>
<td>(is+ie)/2</td>
</tr>
<tr>
<td>npixi</td>
<td>number of x-pixels before any j-reflection</td>
<td>nxpx</td>
</tr>
<tr>
<td>npixj</td>
<td>number of y-pixels before any i-reflection</td>
<td>nypx</td>
</tr>
<tr>
<td>x1pixmn</td>
<td>minimum x1 for pixel image</td>
<td>x1a(is)</td>
</tr>
</tbody>
</table>
### Appendix B: The namelists

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1pixmx (nios)</td>
<td>maximum x1 for pixel image</td>
</tr>
<tr>
<td>x1a(ie+1)</td>
<td></td>
</tr>
<tr>
<td>x2pixmn (nios)</td>
<td>minimum x2 for pixel image</td>
</tr>
<tr>
<td>x2a(js)</td>
<td></td>
</tr>
<tr>
<td>x2pixmx (nios)</td>
<td>maximum x2 for pixel image</td>
</tr>
<tr>
<td>x2a(je+1)</td>
<td></td>
</tr>
<tr>
<td>x3pixmn (nios)</td>
<td>minimum x3 for pixel image</td>
</tr>
<tr>
<td>x3a(ks)</td>
<td></td>
</tr>
<tr>
<td>x3pixmx (nios)</td>
<td>maximum x3 for pixel image</td>
</tr>
<tr>
<td>x3a(ke+1)</td>
<td></td>
</tr>
<tr>
<td>iflippix(nios)</td>
<td>= 0 =&gt; no flipping of images</td>
</tr>
<tr>
<td>= 1 =&gt; image flipped about x-axis before writing (size of image not changed, just flipped)</td>
<td></td>
</tr>
<tr>
<td>jflippix(nios)</td>
<td>= 0 =&gt; no flipping of images</td>
</tr>
<tr>
<td>= 1 =&gt; image flipped about y-axis before writing</td>
<td></td>
</tr>
<tr>
<td>ireflpix(nios)</td>
<td>= 0 =&gt; no reflection about x-axis</td>
</tr>
<tr>
<td>= 1 =&gt; image reflected about x-axis on output =&gt; twice the number of y-pixels requested</td>
<td></td>
</tr>
<tr>
<td>jreflpix(nios)</td>
<td>= 0 =&gt; no reflection about y-axis</td>
</tr>
<tr>
<td>= 1 =&gt; image reflected about y-axis on output =&gt; twice the number of x-pixels requested</td>
<td></td>
</tr>
</tbody>
</table>

#### Variable control

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixvar (niov)</td>
<td>names of variables to be plotted (character*2). 'zz'</td>
</tr>
<tr>
<td>Valid names are: 'a', 'a1', 'a2', 'a3', 'ag', 'ap', 'an', 'b', 'b1', 'b2', 'b3', 'bp', 'bR', 'bn', 'bp', 'bt', 'cs', 'd', 'da', 'e1', 'e2', 'er', 'et', 'fn', 'gp', 'j', 'j1', 'j2', 'j3', 'jn', 'jp', 'k1', 'k2', 'ka', 'lu', 'm', 'MA', 'mf', 'ms', 'nb', 'p1', 'p2', 'p3', 'p4', 'p5', 'p6', 'p7', 'pa', 'pg', 's', 's1', 's2', 's3', 'sd' (skew-density), 'sp', 'sn', 'sy', 'u1', 'u2', 'v', 'v1', 'v2', 'v3', 'vA', 'vf', 'vn', 'vp', 'vs', 'w', 'w1', 'w2', 'w3', 'wn', 'wp'. See &quot;pl1var&quot; in pl1con and &quot;pl2sca*&quot; in pl2con for other definitions.</td>
<td></td>
</tr>
<tr>
<td>nlpix (niov)</td>
<td>= 0 =&gt; store data</td>
</tr>
<tr>
<td>&gt; 0 =&gt; store log10(data), concentrating colours at low end. Dynamic range = nlpix, 1 =&gt; 100.</td>
<td></td>
</tr>
<tr>
<td>&lt; 0 =&gt; store log10(data), concentrating colours at high end. Dynamic range = -nlpix, -1 =&gt; -100.</td>
<td></td>
</tr>
<tr>
<td>pixmin (niov)</td>
<td>value to be assigned the minimum colour.</td>
</tr>
<tr>
<td>pixmax (niov)</td>
<td>value to be assigned the maximum colour.</td>
</tr>
<tr>
<td>pixmm (niov)</td>
<td>= 0 =&gt; use input &quot;pixmin&quot;, &quot;pixmax&quot; for each image</td>
</tr>
<tr>
<td>= 1 =&gt; use &quot;pixmin&quot;, &quot;pixmax&quot; for each image (default)</td>
<td></td>
</tr>
<tr>
<td>= 2 =&gt; extreme of computed &quot;pixmin&quot;, &quot;pixmax&quot; used for all frames (movie format only; raw and HDF formats, pixmm=1 and 2 are the same)</td>
<td></td>
</tr>
<tr>
<td>ncpix</td>
<td>number of colour contours in image</td>
</tr>
<tr>
<td>Animation control; note that a separate palette can be chosen for each variable</td>
<td></td>
</tr>
<tr>
<td>pixpal (niov)</td>
<td>= 0 =&gt; use user palette 'pfilepix'</td>
</tr>
<tr>
<td>= 1 =&gt; greyscale palette.</td>
<td></td>
</tr>
<tr>
<td>= 2 =&gt; ZEUS' spectral colour palette if pixpal(2:niov) not set by user, they're set to pixpal(1).</td>
<td></td>
</tr>
<tr>
<td>pfilepix(niov)</td>
<td>character*128 user's palette file (pixpal = 0). If file not found, pixpal reverts to default (2).</td>
</tr>
</tbody>
</table>
Appendix B: The namelists

fratepix FFmpeg's integer parameter. Animation assembled to run at "fratepix" frames per second
mpgpix = 1 => animations written in mpg format
= 4 => animations written in mp4 format
else=> neither mpegs nor PPM files created, movie frames are bundled into a tarball for post-processing (e.g., ANIM8Z)
ppmpix = 0 => PPM files are discarded, staging directory 0 makempg.dir deleted after use
= 1 => PPM files are retained in makempg.dir

B.22 USRCON—USEr dump CONtrol (subroutine NMLSTS)

This namelist is reserved for a user-supplied I/O subroutine aliased to USERDUMP (see App. A to see where USERDUMP is called). Additional namelist parameters may be added as needed.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtusr</td>
<td>physical (problem) time interval between user dumps.</td>
<td>0.0</td>
</tr>
<tr>
<td>nusrdmp</td>
<td>the sequential number for the next user dump file</td>
<td>-1</td>
</tr>
</tbody>
</table>

B.23 HDFCON—HDF dump CONtrol (subroutine NMLSTS)

This namelist controls the HDF (Hierarchical Data Format, version 4) dumps. HDF is a format for data files developed at the NCSA. This format is fairly widely used, appearing in various commercial applications such as IDL. HDF dumps are 4 bytes deep, and contain the grid coordinates along with other useful information about the data. They are suitable for graphical and quantitative analysis, but not for resuming a simulation.

In order to use HDF, it is necessary to link all the HDF4 libraries to your executable. If you don’t know what or where these libraries are on your system, ask your system administrator who may have to download the (free) HDF4 libraries from the NCSA website (www.ncsa.uiuc.edu).

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dthdf</td>
<td>physical (problem) time interval between hdf dumps.</td>
<td>0.0</td>
</tr>
<tr>
<td>nhdfdmp</td>
<td>the sequential number for the next HDF file</td>
<td>-1</td>
</tr>
<tr>
<td>hdfvar(niov)</td>
<td>names of variables to be dumped (character*2).</td>
<td>‘zz’</td>
</tr>
</tbody>
</table>
Valid names are: 'to' ("total" dump => v1, v2, v3, b1, b2, b3, d, et, e1, e2, gp, pg, er, da in the same file), 'a1', 'a2', 'a3', 'a', 'ag', 'b1', 'b2', 'b3', 'b', 'bt', 'd', 'da', 'e1', 'e2', 'er', 'et', 'gp', 'j1', 'j2', 'j3', 'j', 'k1', 'k2', 'ka', 'lu', 'm', 'mA', 'ms', 'mf', 'p1', 'p2', 'p3', 'p4', 'p5', 'p6', 'p7', 'pg', 's1', 's2', 's3', 's', 'u1', 'u2', 'v1', 'v2', 'v3', 'v', 'vv', 'w1', 'w2', 'w3', 'w'. See "pl1var" in pl1con and "pl2sca*" in pl2con for definitions.

namelist / hdfcon /
  1   dthdf , nhfdmp , hdfvar

B.24 TSLCON—Time SLice (history) dump CONTROL (subroutine NMLSTS)

This namelist controls the time slice data dumps. Various scalars, such as total mass, angular momenta, energy, extrema of variables, etc. are periodically written to an ascii file and/or a plot (NCAR or PSPLLOT graphics). See §B.19 for what libraries are needed for NCAR and PSPLLOT graphics respectively.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dttsl</td>
<td>problem time interval between ascii dumps</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.0 =&gt; no ascii timeslice dumps</td>
<td></td>
</tr>
<tr>
<td>ntsldmp</td>
<td>sequential number for next timeslice file</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>&lt; 0 =&gt; ntsldmp = itsldmp</td>
<td></td>
</tr>
<tr>
<td>itslmn</td>
<td>minimum i-index of integration domain</td>
<td>ismn</td>
</tr>
<tr>
<td>itslmx</td>
<td>maximum i-index of integration domain</td>
<td>iemx</td>
</tr>
<tr>
<td>jtslmn</td>
<td>minimum j-index of integration domain</td>
<td>jsmn</td>
</tr>
<tr>
<td>jtslmx</td>
<td>maximum j-index of integration domain</td>
<td>jemx</td>
</tr>
<tr>
<td>ktslmn</td>
<td>minimum k-index of integration domain</td>
<td>ksmn</td>
</tr>
<tr>
<td>ktslmx</td>
<td>maximum k-index of integration domain</td>
<td>kemx</td>
</tr>
<tr>
<td>Plot control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dttsp</td>
<td>problem time interval between plot dumps</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.0 =&gt; no timeslice plot dumps</td>
<td></td>
</tr>
<tr>
<td>ntspdmp</td>
<td>sequential number for next timeslice plot file</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>&lt; 0 =&gt; ntspdmp = itsldmp</td>
<td></td>
</tr>
<tr>
<td>pwttsp</td>
<td>pen weight for timeslice plots</td>
<td>1.0</td>
</tr>
<tr>
<td>norptsp</td>
<td>= 1 -&gt; use NCAR graphics for timeslice plots</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>= 2 -&gt; use PSPLLOT graphics for timeslice plots</td>
<td></td>
</tr>
<tr>
<td>ntspv</td>
<td>number of frames horizontally per page</td>
<td>1</td>
</tr>
<tr>
<td>tsphdr</td>
<td>number of frames vertically per page</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>= 1 =&gt; write headers to timeslice plot file</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>= 0 =&gt; suppresses headers</td>
<td></td>
</tr>
<tr>
<td>tspfr</td>
<td>= 1 =&gt; write footers to timeslice plot file</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>= 0 =&gt; suppresses footers</td>
<td></td>
</tr>
<tr>
<td>ttspmn</td>
<td>problem time for beginning of plot</td>
<td>0.0</td>
</tr>
<tr>
<td>ttspmx</td>
<td>problem time for end of plot (0.0 =&gt; prtime)</td>
<td>0.0</td>
</tr>
</tbody>
</table>

namelist / tslcon /
  1   dttsl , ntsldmp , itslmn , itslmx , jtslmn
Appendix B: The namelists

B.25 CRKCON—Cork dump CONtrol (subroutine NMLSTS)

New to version 3.6: This namelist controls the cork dump, which consist of data (cork positions, values of flow variables, and functions of flow variables such as $\nabla \cdot \vec{v}$, Mach number, etc.) collected at user-specified time intervals at the locations of all Lagrangian particles (corks) populating the grid at $t = 0$ (those set by GRIDCORK), as well as those injected onto the grid by the user during the simulation (e.g., JETCORK). These data are continually appended to an opened ascii file (zcnnid) which tabulates the locations of each cork at each time interval, along with each of the specified tracked variables. User-provided post-processing routines can be used to display these data after the simulation is complete and/or the user could provide a routine aliased to FINISH to generate a plot file on termination (see routine SPECTRA for a template).

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtcrk</td>
<td>problem time interval between cork dumps</td>
<td>0.0</td>
</tr>
<tr>
<td>ncrkdmp</td>
<td>the sequential number for the next cork file</td>
<td>-1</td>
</tr>
<tr>
<td>gcrk*</td>
<td>GRIDCORK sets a lattice of gcrk1 x gcrk2 x gcrk3 corks (default = 1 for *SYM).</td>
<td>0</td>
</tr>
<tr>
<td>iordcrk</td>
<td>= 1 =&gt; first order integration of cork positions = 2 =&gt; second order integration</td>
<td>1</td>
</tr>
<tr>
<td>crkvar</td>
<td>names of variables to be tracked (character<em>2). 'zz' Valid names are: 'a1', 'a2', 'a3', 'a', 'ag', 'B3', 'b1', 'b2', 'b3', 'bP', 'bR', 'b', 'bd', 'br', 'bt', 'd', 'da', 'e1', 'e2', 'er', 'et', 'gl', 'gk', 'gp', 'hy', 'ji', 'j2', 'j3', 'j', 'k1', 'k2', 'ka', 'kt', 'l1', 'lt', 'lu', 'm', 'mA', 'mf', 'ms', 'om', 'p1', 'p2', 'p3', 'p4', 'p5', 'p6', 'p7', 'pg', 'si', 's2', 's3', 's', 'sy', 'u1', 'u2', 'v1', 'v2', 'v3', 'v', 'vv', 'w1', 'w2', 'w3', 'w', 'ze' (shock ratio). See &quot;pl1var&quot; in pl1con and &quot;pl2sca</em>&quot; in pl2con for definitions.</td>
<td>'zz'</td>
</tr>
</tbody>
</table>

namelist / crkcon /
1       dtcrk , ncrkdmp , gcrk1 , gcrk2 , gcrk3
2       , iordcrk , crkvar

B.26 DISCON—DISplay dump CONtrol (subroutine NMLSTS)

This namelist controls the display dumps of 2-D slices. During a run, as many as nios slices may be specified for each variable displayed. All display dumps generated during a run are dumped to the same ascii data file. The extent of the display slice can be limited by setting idismn, idismx, etc. The normal to the display slice is parallel to one of the axes of the computational grid and is specified by disdir. The index at the base of the normal is given by ldis.
The display format allows the user to view a small portion of the data quantitatively in a matrix format. The maximum amount of data that can be visualised at once from each specified variable and slice is 38 by 38. The data are scaled and converted to integers with a dynamic range anywhere from 100 to $10^6$, depending on the amount of data being displayed. The data are arranged in a 2-D matrix and labelled with the grid indices and the scaling factor used to scale the data. (The functionality is similar to that of the task PRTIM in AIPS.)

*N.B.* For restarted runs in which the computation is resumed on a larger or smaller grid, and where the default values for `idismn`, `idismx`, *etc.* were used in the initial run, it will be necessary to set `idismn`, `idismx`, *etc.* in the input deck for the restarted run to the extrema of the new grid if the dumps are to extend to the bounds of the new grid. Otherwise, the dumps will be bound by the old grid.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dtdis</code></td>
<td>physical (problem) time interval between display dumps. 0.0 =&gt; no display dumps.</td>
<td>0.0</td>
</tr>
<tr>
<td><code>ndisdmp</code></td>
<td>the sequential number for the next display file</td>
<td>-1</td>
</tr>
<tr>
<td><code>disdir</code> (nios)</td>
<td>direction of normal to display plane: 1, 2, or 3</td>
<td>0</td>
</tr>
<tr>
<td><code>ldis</code> (nios)</td>
<td>level of 2-D display (value of 1-, 2-, or (is+ie)/3-index)</td>
<td></td>
</tr>
<tr>
<td><code>disvar</code> (niov)</td>
<td>names of variables to be displayed (character*2). 'zz'</td>
<td></td>
</tr>
<tr>
<td><code>idismn</code> (nios)</td>
<td>bottom i-index of display window</td>
<td></td>
</tr>
<tr>
<td><code>idismx</code> (nios)</td>
<td>top i-index of display window</td>
<td></td>
</tr>
<tr>
<td><code>jdismn</code> (nios)</td>
<td>bottom j-index of display window</td>
<td></td>
</tr>
<tr>
<td><code>jdismx</code> (nios)</td>
<td>top j-index of display window</td>
<td></td>
</tr>
<tr>
<td><code>kdismn</code> (nios)</td>
<td>bottom k-index of display window</td>
<td></td>
</tr>
<tr>
<td><code>kdismx</code> (nios)</td>
<td>top k-index of display window</td>
<td></td>
</tr>
</tbody>
</table>

**Appendix B: The namelists**

This namelist controls the RADIO dumps, which are 2-D images of lines-of-sight integrations through the data volume at arbitrary viewing angles ($\theta$ and $\phi$) binned on a uniform, square Cartesian grid. **New to version 3.6**, depending on the new parameter `radfmt`, images may be dumped as movie frames (four bytes deep with grid, dimensions, and viewing angles stored in a header) and/or traditional pixel dumps (one byte per datum, no header) and/or HDF files. As always, pixel dumps remain as separate files, to be dealt with manually.
Appendix B: The namelists

by the user after the run. HDF files are a fossil from very early versions of the code, and are suitable for graphical analysis if desired. For movie frames, at the end of the run `makempg` gathers all files in a tarball (to be used at a later time by `ANIM8Z` to recreate the animation with a different palette or bracketing if desired), converts them to PPM format, and assembles them into an mpeg movie file according to user-set namelist parameters. Thus, the end product is an mpeg animation for each quantity selected along with an accompanying tarball of four-byte deep frames. For a smooth temporal animation, aim for about 400–600 dumps per animation.

The integration volume can be limited by setting `x1radmn`, `x1radmx`, etc. `RADIO` dumps are currently available for Cartesian (XYZ) and cylindrical (ZRP) geometries, with the latter not fully tested. See discussion in §B.21 regarding strategy of bracketing images, dumping images logarithmically, and making animations.

There are two types of integrated quantities: flow variables and emissivities. Many of the parameters listed below are for controlling the latter. For example, the Stokes parameters once integrated can be convolved with a beam, polarisation vectors may be plotted directly (rather than raster images), polarisation vectors may be superposed on total intensity raster images, and so on.

The “masks” (`*lower`, `*upper`, `dmask*`, and `bmask`) are useful in limiting which portion of the grid is included in the integration of the non-emissivity scalars. For example, if there is a contact discontinuity (CD) enclosing the region of interest, then there will be a jump in the density (d) along this interface. Thus, if d jumps, say, from about 0.1 to about 1.0 across the CD, setting `dmask*=1.0` and `dupper=0.5` would allow only the low density region (be it interior or exterior to the CD) to contribute to the line-of-sight integration of variable *. Alternatively, if the magnetic field is found only in the material of interest, setting `bmask*=1.0` would allow only material with magnetic field to be included in the integration of variable *. Finally, the variables `*lower` and `*upper` allow each variable to be masked by its own distribution. These can be set in addition to the density and/or magnetic field masks (`dmask*`, `bmask*`). For example, if only the compressive portions of the flow are to be integrated, then setting `xupper=0.0` will mean that only negative values of \( \nabla \cdot \vec{v} \) will be included in the integration. All values excluded by the various masks will be given zero weight. In all cases, the default is no mask.

Reversing the palette (`nirad<0`) is useful for images in which `radmin<0` and `radmax<0` (e.g., negative velocity divergences). In these cases, it may be desirable to have the “maximum” colour correspond to the minimum pixel value (which has the greatest absolute value).

Note that the parameters which set the dimensions of the arrays for the `RADIO` dumps (`nxrd,nyrd`) are independent of the parameters which set the dimensions of the flow variables (`in,jn,kn`) and of the regular pixel slices (`nxpx,nypx`).

N.B. For restarted runs in which the computation is resumed on a larger or smaller grid, and where the default values for `x1radmn`, `x1radmx`, etc. were used in the initial run, it will be necessary to set `x1radmn`, `x1radmx`, etc. in the input deck for the restarted run to the extrema of the new grid if the dumps are to extend to the bounds of the new grid. Otherwise, the dumps will be bound by the old grid.

Also for a restarted run, the g-zipped movie tarballs from the previous run should be in the same directory as the executable so that the new movie frames can be added to the tarballs, and the mpegs can be made from all movie frames since the beginning of the first
run, rather than just from those created from the beginning of the present run. To enable this feature, neither the problem tag nor the names of the movie tarballs should be changed from one run to the next.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dtrad</td>
<td>problem time interval between RADIO dumps</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0 =&gt; no RADIO dumps.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nraddmp</td>
<td>the sequential number for the next radio file</td>
<td>-1</td>
</tr>
<tr>
<td>&lt;0 =&gt; nraddmp = iraddmp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>units</td>
<td>sets the angular units (character*2)</td>
<td>'rd'</td>
</tr>
<tr>
<td>'rd' =&gt; radians, 'pi' =&gt; units of pi radians</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'dg' =&gt; degrees</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thetamin</td>
<td>minimum angle between x1-axis and plane of sky</td>
<td>0.0</td>
</tr>
<tr>
<td>dtheta</td>
<td>desired increment in &quot;theta&quot; between successive dumps</td>
<td>0.0</td>
</tr>
<tr>
<td>phimin</td>
<td>minimum azimuthal angle for lines of sight.</td>
<td>0.0</td>
</tr>
<tr>
<td>dphi</td>
<td>desired increment in &quot;phi&quot; between successive dumps</td>
<td>0.0</td>
</tr>
<tr>
<td>x1radmn</td>
<td>minimum x1 for RADIO integration</td>
<td>x1a(is)</td>
</tr>
<tr>
<td>x1radmx</td>
<td>maximum x1 for RADIO integration</td>
<td>x1a(ie+1)</td>
</tr>
<tr>
<td>x2radmn</td>
<td>minimum x2 for RADIO integration</td>
<td>x2a(js)</td>
</tr>
<tr>
<td>x2radmx</td>
<td>maximum x2 for RADIO integration</td>
<td>x2a(je+1)</td>
</tr>
<tr>
<td>x3radmn</td>
<td>minimum x3 for RADIO integration</td>
<td>x3a(ks)</td>
</tr>
<tr>
<td>x3radmx</td>
<td>maximum x3 for RADIO integration</td>
<td>x3a(ke+1)</td>
</tr>
<tr>
<td>radfill</td>
<td>= 0 =&gt; diameter of image plane set to sqrt(x1radln<strong>2 + x2radln</strong>2 + x3radln**2) so that no matter the viewing angle, image fits entirely on image plane.</td>
<td>0</td>
</tr>
<tr>
<td>= 1 =&gt; diameter of image plane set to x1radln so that viewed face-on, image fills entire plane, leaving no black space around image.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>radbox</td>
<td>= 0 =&gt; no box drawn around integration region</td>
<td>2</td>
</tr>
<tr>
<td>= 1 =&gt; all box lines drawn with highest colour</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=-1 =&gt; all box lines drawn with lowest colour</td>
<td></td>
<td></td>
</tr>
<tr>
<td>= 2 =&gt; all lines except those to &quot;hidden corner&quot; drawn with highest colour (default)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=-2 =&gt; all lines except those to &quot;hidden corner&quot; drawn with lowest colour</td>
<td></td>
<td></td>
</tr>
<tr>
<td>radfmt</td>
<td>= 1 =&gt; RADDMP creates 4-byte movie frames.</td>
<td>1</td>
</tr>
<tr>
<td>= 2 =&gt; RADDMP creates 1-byte raw pixel dumps.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>= 3 =&gt; RADDMP creates movie and raw files (1+2).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>= 4 =&gt; RADDMP creates 4-byte HDF files.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>= 5 =&gt; RADDMP creates movie and HDF files (1+4).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>= 6 =&gt; RADDMP creates raw and HDF files (2+4).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>= 7 =&gt; RADDMP creates files of all 3 formats (1+2+4).</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Emission control</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>radtype</td>
<td>= 0 =&gt; emissivities are not computed</td>
<td>2</td>
</tr>
<tr>
<td>= 1 =&gt; Smith et al emissivity (p**2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>= 2 =&gt; CNB emissivity (function of d, p, B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>= 3 =&gt; synchrotron emissivity with break freq.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pindex</td>
<td>power law index of relativistic electrons</td>
<td>2.0</td>
</tr>
<tr>
<td>N(E) = kappa * E**(-pindex) (radtype = 2, 3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>freq</td>
<td>frequency of RADIO observation (Hz)</td>
<td>5.0d+9</td>
</tr>
</tbody>
</table>
Appendix B: The namelists

nubrs  nubr(Hz)/nubr(zeus); scaling factor for break frequency (radtype=3 only) 0.5
       0.0 => value is computed in JETINIT
brn0   fiducial number density for bremsstrahlung lrge
       emission (m**-3)
brt0   fiducial temp. for bremsstrahlung emission (K) lrge
brnu0, brnu1 limits of frequency band for brem. (Hz) 1.000000e17
       1.000001e17
betarel relativistic beta = v/c for fastest zone (for 0.0
       Mike Casey's Doppler boosting)

Convolution control ("beam")

Only the Stokes parameters and the Q-SRA variables are convolved with the
specified elliptical "beam".

cnvlv   = 0 => do not apply convolution 0
       = 1 => apply convolution to Stokes parameters
bmajor  major axis of convolving beam (distance units) 1.0
bminor  minor axis of convolving beam (distance units) 1.0
bpa     beam position angle (radians) measured counter-
       clockwise between major axis and +x axis 0.0
cpb     "cells" per beam 5.0

"cpb" is an accuracy parameter. The default (5.0) should be fine in most cases. Setting it lower will speed up the computation, but will yield a more ragged image.

Masking control

In what follows,

* = d, pi, t, b, sh, vv, br, w, m, ma, ms, mf, db, eb, de
q = d, p, p/d, B, shear, divv, brem, curlv, M, MA, MS, MF, d*b, e1**2b, d/e1**2

*lower  q is integrated along los provided q > *lower  *huge
*upper  q is integrated along los provided q < *upper huge
dmask* density mask toggle for variable * (except "d") -1.0
        = 1.0 => "dlower" and "dupper" set int. limits
        = 0.0 => not
        =-1.0 => use "dmask"
       
dmask  density mask toggle for all variables 0.0
       If "dmask*" .ne. -1.0, value of "dmask*" overrides "dmask" for variable * only
bmask* B-field mask toggle for variable * -1.0
        = 1.0 => B-field extent sets int. limits
        = 0.0 => not
        =-1.0 => use "bmask"

bmask  B-field mask toggle for all variables 0.0
       If "bmask*" .ne. -1.0, value of "bmask*" overrides "bmask" for variable * only

Variable control

Controls which variables are to be "carried" with the line-of-sight integrations, and how the pixel dumps are to be bracketed. The bulk of CPU is used up in setting up the lines-of-sight; the additional cpu required for each variable is nominal. The only real "cost" of selecting all variables is the additional disc space required to store all the files.
Appendix B: The namelists

radvar (niov) character*2 mnemonics for variables to be plotted 'zz'
using pixel-plotting routines. Legal mnemonics are:
'A' => pol. position angle [ 0.5 * atan (Q/U) ]
'AV' => A with polarisation vectors superposed
'F' => fractional pol'n (fpol) rastor plot
'PV' => fpol with polarisation vectors
'I' => total intensity (toti) rastor plot
'IV' => toti with polarisation vectors
'P' => polarised intensity (poli) rastor plot
'PV' => poli with polarisation vectors
'V' => pol'n vector rastor plot (black on white)
'VR' => pol'n vector rastor plot (white on black)
'D' => column density
'P1' => column pressure (internal energy density)
'B' => column magnetic field strength
'T' => column temperature (specific internal energy)
'SH' => column velocity shear
'VV' => column divergence of velocity
'BR' => bremsstrahlung emission
'W' => vorticity
'M' => Sonic Mach number
'MA' => Alfven Mach number
'MS' => Fast magnetosonic Mach number
'MF' => Fast magnetosonic Mach number
'DB' => d * B (K-S R A variable)
'EB' => p1**2 * B (K-S R A variable)
'DE' => d / p1**2 (K-S R A variable)

nlrad (niov) = 0 => store data
> 0 => store log10(data) concentrating colours at low end. Dyn. range = nlrad, 1 => 100.
< 0 => store log10(data) concentrating colours at high end. Dyn. range =-nlrad, -1 => -100.

radmin (niov) value of data to be assigned the minimum colour. 0.0
radmax (niov) value of data to be assigned the maximum colour. 0.0
radmm (niov) = 0 => use input "radmin", "radmax" for each image
If radmin=radmax=0, "radmm" reset to 1
= 1 => compute "radmin", "radmax" for each image
= 2 => extreme of computed "radmin", "radmax" used for all frames (movie format only; raw and HDF formats, radmm=1 and 2 are the same)

ncrad number of colour contours in image 255

Reversing the palette (nlrad<0) is useful for images in which radmin<0 and radmax<0 (e.g., negative velocity divergences). In these cases, it may be desirable to have the "maximum" colour correspond to the minimum pixel value (which has the greatest absolute value).

Vector control

eorb = 1 => E-vectors
  = 2 => B-vectors

porf = 1 => vector length proportional to poli
  = 2 => vector length proportional to fpol

bworb = 1 => black and white pixel vectors
  = 2 => black pixel vectors only

vlmin vectors with length < vlmin*(max vector) not plotted.

icut vectors are not plotted if total intensity 0.001
Appendix B: The namelists

```plaintext

toti < icut*max(toti)  
vector are not plotted if polarised intensity 0.001

pcut < pcut*max(poli)

pscale  
scaling factor for polarisation vectors 0.8
  = 1 => vectors lining up along image axis touch
  < 0 => vectors scaled by |pscale| and have uniform length (useful to map polarisation angles)

incpx  
plot every incpx(th) vector in x-direction 1

incpy  
plot every incpy(th) vector in y-direction

Animation control; note that a separate palette can be chosen for each variable

radpal (niov) = 0 => use user palette 'pfilerad'
  = 1 => greyscale palette.
  = 2 => spectral colour palette (default).
  if radpal(2:niov) not set by user, they're set to radpal(1).

pfilerad(niov)  
character*128 user's palette file (radpal = 0).
  If file not found, radpal reverts to default.

fraterad  
FFmpeg's integer parameter. Animation assembled to run at "fraterad" frames per second

mpgrad  
= 1 => animations written in mpg format
  = 4 => animations written in mp4 format
  else=> neither mpegs nor PPM files created, movie frames are bundled into a tarball for post-processing (e.g., ANIM8Z)

ppmrad = 0 => PPM files are discarded, staging directory 0
  = 1 => PPM files are retained in makempg.dir

Animation control; note that a separate palette can be chosen for each variable

radpal (niov) = 0 => use user palette 'pfilerad'
  = 1 => greyscale palette.
  = 2 => spectral colour palette (default).
  if radpal(2:niov) not set by user, they're set to radpal(1).

pfilerad(niov)  
character*128 user's palette file (radpal = 0).
  If file not found, radpal reverts to default.

fraterad  
FFmpeg's integer parameter. Animation assembled to run at "fraterad" frames per second

mpgrad  
= 1 => animations written in mpg format
  = 4 => animations written in mp4 format
  else=> neither mpegs nor PPM files created, movie frames are bundled into a tarball for post-processing (e.g., ANIM8Z)

ppmrad = 0 => PPM files are discarded, staging directory 0
  = 1 => PPM files are retained in makempg.dir
```

```

radvar (niov) = 0 => use user palette 'pfilerad'
  = 1 => greyscale palette.
  = 2 => spectral colour palette (default).
  if radpal(2:niov) not set by user, they're set to radpal(1).

pfilerad(niov)  
character*128 user's palette file (radpal = 0).
  If file not found, radpal reverts to default.

fraterad  
FFmpeg's integer parameter. Animation assembled to run at "fraterad" frames per second

mpgrad  
= 1 => animations written in mpg format
  = 4 => animations written in mp4 format
  else=> neither mpegs nor PPM files created, movie frames are bundled into a tarball for post-processing (e.g., ANIM8Z)

ppmrad = 0 => PPM files are discarded, staging directory 0
  = 1 => PPM files are retained in makempg.dir
```
B.28  PGEN—Problem GENerator (subroutine aliased to PROBLEM)

This namelist is reserved for the problem generator, which sets the flow variables to the desired initial conditions. Thus the parameters which appear in this namelist depend on which problem is being studied. The desired problem is specified by setting the EDITOR alias PROBLEM in the file zeus36.mac to the name of the problem generating subroutine. This subroutine should initialise the active zones of all field variables as well as any inflow boundary arrays. The routines bndyflgs and bndyall are called after PROBLEM, and thus need not be called in the user’s problem generator (§5.1).

Below is a description of the problem generator to shkset, which is used for the 1-D Brio and Wu problem and consistent with the sample of dzeus36.s given in §2.3. In general, the user will be writing their own problem generator and may, if they wish, call their namelist pgen as well. Note that it does not matter that more than one subroutine uses pgen as the name of its namelist, so long as only one problem generating subroutine is called in a run (as is typical). If the user wishes to use one of the problem generators already in dzeus36, each of their namelists are described in the comments of the problem generating routine in exactly the same format as that for shkset which follows.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>idirect</td>
<td>1 =&gt; 1-direction</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2 =&gt; 2-direction</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3 =&gt; 3-direction</td>
<td>3</td>
</tr>
<tr>
<td>isetib</td>
<td>= 1 =&gt; set inner boundary conditions based on ( v(\text{par}) )</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>= 0 =&gt; inflow BC are already set by iib, etc.</td>
<td>1</td>
</tr>
<tr>
<td>isetob</td>
<td>same as isetib for outer boundary</td>
<td>1</td>
</tr>
<tr>
<td>n0</td>
<td>Number of zones to be initialised. Namelist is ( \max(nx*z) ) read from ioin until a total of ( ie-is+1 [je-js+1, ke-ks+1] ) zones are initialised.</td>
<td>1</td>
</tr>
<tr>
<td>d0</td>
<td>input density</td>
<td>tiny</td>
</tr>
<tr>
<td>e10</td>
<td>input first internal energy density</td>
<td>tiny</td>
</tr>
<tr>
<td>p10</td>
<td>input first pressure ( ( = \text{gamm1}(1) * e10 ) )</td>
<td>-1.0</td>
</tr>
<tr>
<td>e20</td>
<td>input second internal energy density</td>
<td>tiny</td>
</tr>
<tr>
<td>p20</td>
<td>input second pressure ( ( = \text{gamm1}(2) * e20 ) )</td>
<td>-1.0</td>
</tr>
<tr>
<td>v10</td>
<td>input velocity in 1 direction</td>
<td>0.0</td>
</tr>
<tr>
<td>v20</td>
<td>input velocity in 2 direction</td>
<td>0.0</td>
</tr>
<tr>
<td>v30</td>
<td>input velocity in 3 direction</td>
<td>0.0</td>
</tr>
<tr>
<td>b10</td>
<td>input magnetic field in 1 direction</td>
<td>0.0</td>
</tr>
<tr>
<td>b20</td>
<td>input magnetic field in 2 direction</td>
<td>0.0</td>
</tr>
<tr>
<td>b30</td>
<td>input magnetic field in 3 direction</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Namelist variables idirect, isetib, and isetob must be set in the first namelist card and may, though need not be, set in all.

Parameters n0–b30 must be set in each namelist card to override defaults.

\[ p10 \text{ (p20) used if e10 (e20) not set.} \]
C The ZEUS-3D variables

This appendix contains a glossary of the variables used in dzeus36, and is meant to aid the user in writing subroutines and making changes to the source code itself. It is by no means complete, but should contain the variables needed for most purposes. All these variables are declared in the common deck comvar. Thus, adding the EDITOR command *call comvar before the local declarations makes all these variables accessible from within the subroutine.

The user should be aware of the index convention used. A 3-D array, such as the density, is denoted d(i,j,k), where i is the index for the x1 coordinate, j is the index for the x2 coordinate, and k is the index for the x3 coordinate. The coordinates x1, x2, and x3 are intentionally generic, since an attempt has been made to write the code in a covariant fashion. In Cartesian, cylindrical, and spherical polar coordinates, (x1,x2,x3) corresponds to (x,y,z), (z,r,φ) [not (r,φ,z)], and (ρ,θ,φ) respectively. In FORTRAN, the index which changes the fastest is the first one. Thus, in triple do-loops which manipulate the 3-D arrays, it is best to have the outer loop run on k, the middle loop run on j, and the inner loop run on i. If one of the directions is divided into more zones than the other two, then it is best that this direction be the 1-direction (with index i) since it is the inner loop which vectorises on vector machines. In Cartesian coordinates, this can always be arranged. The indices strictly follow a right-hand rule. Thus, the array nijb(k,i) is a 2-D array which has k as its first index and i as its second (and not i as the first index and k as the second which would follow a left-hand rule). In the tables in this appendix, arrays are given with their indexing to remind the user of the ZEUS-3D convention.

The user should also be aware of the gridding. The computational domain is divided into in by jn by kn zones. In each direction, five of these zones are “ghost” or “boundary” zones, while the remaining zones are “active” zones in which the equations of MHD are solved. In Cartesian geometry, these zones are rectangular boxes. In general, the gridding need not be uniform, so the ratio of the dimensions of each zone need not be constant across the grid. There are eight locations one can associate uniquely with each zone. Each of these locations can be tagged with the indices (i,j,k). These locations are: the centre of each box, the centre of three of the six faces, the centre of three of the twelve edges, and one of the eight corners.

In ZEUS-3D, there are two grids which are referred to as the half-grid (or the a-grid) and the full grid (or the b-grid). By convention, the (i,j,k)th point on the a-grid is half a grid spacing closer in each dimension to the (left, bottom, back) corner of the grid than the (i,j,k)th point on the b-grid. As seen in the table below, zone centres have pure b-grid coordinates, zone corners have pure a-grid coordinates, while faces and edges of each zone have mixed coordinates.

<table>
<thead>
<tr>
<th>location</th>
<th>coordinates</th>
<th>location</th>
<th>coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>zone centre</td>
<td>(x1b(i), x2b(j), x3b(k))</td>
<td>zone corner</td>
<td>(x1a(i), x2a(j), x3a(k))</td>
</tr>
<tr>
<td>1-face</td>
<td>(x1a(i), x2b(j), x3b(k))</td>
<td>1-edge</td>
<td>(x1b(i), x2a(j), x3a(k))</td>
</tr>
<tr>
<td>2-face</td>
<td>(x1b(i), x2a(j), x3b(k))</td>
<td>2-edge</td>
<td>(x1a(i), x2b(j), x3a(k))</td>
</tr>
<tr>
<td>3-face</td>
<td>(x1b(i), x2b(j), x3a(k))</td>
<td>3-edge</td>
<td>(x1a(i), x2a(j), x3b(k))</td>
</tr>
</tbody>
</table>

Part of the strength of ZEUS-3D is its use of a “staggered” grid. On such a grid, not all variables are located at the same place. Scalars (density and internal energy) are zone-
centred quantities while the components of the flow vectors (velocity and magnetic field) are face-centred quantities penetrating the face upon which they are centred. Vectors derived from vector quantities such as the current density ($\nabla \times \vec{B}$) and the induced electric field ($\vec{v} \times \vec{B}$) have edge-centred components parallel to the edges while scalars derived from vector quantities such as $\nabla \cdot \vec{v}$ are zone-centred. Thus, the two grids play equally important roles, and the user needs to be careful about which grid should be used and where the variables are located while making any changes to the code.

C.1 Grid variables

Limits for do-loops are tabulated below.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>is, ie</td>
<td>beginning and ending i-index for active zone-centres</td>
</tr>
<tr>
<td>js, je</td>
<td>beginning and ending j-index for active zone-centres</td>
</tr>
<tr>
<td>ks, ke</td>
<td>beginning and ending k-index for active zone-centres</td>
</tr>
</tbody>
</table>

Corresponding to each variable (is, ie, etc.) are the limiting variables (ismn, iemx, etc.) which indicate the extreme values possible for the do-loop indices should the grid extending option be used (§B.18). In addition, the variables ism2, ism1, isp1, isp2, and isp3 exist which are set to is-2, is-1, is+1, is+2, and is+3 respectively. If the computation is symmetric in the i-direction, ism2, ism1, isp1, isp2, and isp3 are simply set to is. Similar variables exist for je, js, ie, ks, and ke.

In order to make the grid covariant, metric factors have been introduced which carry all the dependence of the geometry. In general, the metric appears in the expression for a differential in volume, $dV = h_1 dx_1 h_2 dx_2 h_3 dx_3$. In Cartesian coordinates, $h_1 = h_2 = h_3 = 1$. In cylindrical coordinates, $h_1 = h_2 = 1$, $h_3 = x_2$. In spherical polar coordinates, $h_1 = 1$, $h_2 = x_1$, $h_3 = x_1 \sin x_2$. Note that if one is limited to XYZ, ZRP, and RTP coordinates, there is no need for $h_1$ and $h_3$ can be split into two variables, one dependent just on $x_1$, the other just on $x_2$. In this way, $h_3$ can be represented by two 1-D arrays ($h_{31}$ and $h_{32}$) rather than one 2-D array. Thus, three 1-D metric factors are used in ZEUS-3D.

The most commonly used a-grid variables are tabulated below.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1a(i)</td>
<td>zone-corner</td>
<td>x1-coordinate in grid units</td>
</tr>
<tr>
<td>x2a(j)</td>
<td>zone-corner</td>
<td>x2-coordinate in grid units</td>
</tr>
<tr>
<td>x3a(k)</td>
<td>zone-corner</td>
<td>x3-coordinate in grid units</td>
</tr>
<tr>
<td>dx1a(i)</td>
<td>1-edge</td>
<td>x1a(i+1) - x1a(i)</td>
</tr>
<tr>
<td>dx2a(j)</td>
<td>2-edge</td>
<td>x2a(j+1) - x2a(j)</td>
</tr>
<tr>
<td>dx3a(k)</td>
<td>3-edge</td>
<td>x3a(k+1) - x3a(k)</td>
</tr>
<tr>
<td>h2a(i)</td>
<td>zone-corner</td>
<td>= 1 for Cartesian and cylindrical coordinates,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>= x1a(i) for spherical polar coordinates</td>
</tr>
<tr>
<td>h31a(i)</td>
<td>zone-corner</td>
<td>= h2a(i)</td>
</tr>
<tr>
<td>h32a(j)</td>
<td>zone-corner</td>
<td>= 1 for Cartesian coordinates,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>= x2a(j) for cylindrical coordinates,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>= sin(x2a(j)) for spherical polar coordinates</td>
</tr>
</tbody>
</table>
The most commonly used b-grid variables are tabulated below.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{1b}(i)$</td>
<td>zone-centre</td>
<td>$x_1$-coordinate in grid units</td>
</tr>
<tr>
<td>$x_{2b}(j)$</td>
<td>zone-centre</td>
<td>$x_2$-coordinate in grid units (radians in spherical polar coordinates)</td>
</tr>
<tr>
<td>$x_{3b}(k)$</td>
<td>zone-centre</td>
<td>$x_3$-coordinate in grid units (radians in both cylindrical and spherical polar coordinates)</td>
</tr>
<tr>
<td>$dx_{1b}(i)$</td>
<td>1-face</td>
<td>$x_{1b}(i) - x_{1b}(i-1)$</td>
</tr>
<tr>
<td>$dx_{2b}(j)$</td>
<td>2-face</td>
<td>$x_{2b}(j) - x_{2b}(j-1)$</td>
</tr>
<tr>
<td>$dx_{3b}(k)$</td>
<td>3-face</td>
<td>$x_{3b}(k) - x_{3b}(k-1)$</td>
</tr>
<tr>
<td>$h_{2b}(i)$</td>
<td>zone-centre</td>
<td>$= 1$ for Cartesian and cylindrical coordinates, $= x_{1b}(i)$ for spherical polar coordinates</td>
</tr>
<tr>
<td>$h_{31b}(i)$</td>
<td>zone-centre</td>
<td>$= h_{2b}(i)$</td>
</tr>
<tr>
<td>$h_{32b}(j)$</td>
<td>zone-centre</td>
<td>$= 1$ for Cartesian coordinates, $= x_{2b}(j)$ for cylindrical coordinates, $= \sin(x_{2b}(j))$ for spherical polar coordinates</td>
</tr>
</tbody>
</table>

Every grid variable has a corresponding inverse variable, denoted by appending an “i” to the variable name. Thus, $dx_{1ai}/dx_{1a}$, $x_{2bi}/x_{2b}$, etc. Evidently, there are numerous grid variables. However, only the a-grid variables $x_{1a}$, $x_{2a}$, and $x_{3a}$ are written to the restart dump. All others are re-computed when a job is resumed.

Note that $x_{1a}(i) < x_{1b}(i)$. The exact relationship between the two grids is:

$$x_{1b}(i) = x_{1a}(i) + 0.5 \times dx_{1a}(i)$$

with similar expressions applying for the 2- and 3-directions.

### C.2 Field variables (3-D arrays)

The main field variables and their locations are as follows:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d(1,j,k)$</td>
<td>zone centre</td>
<td>density</td>
</tr>
<tr>
<td>$v_{1}(i,j,k)$</td>
<td>1-face</td>
<td>velocity in the 1-direction (grid units)</td>
</tr>
<tr>
<td>$v_{2}(i,j,k)$</td>
<td>2-face</td>
<td>velocity in the 2-direction (grid units)</td>
</tr>
<tr>
<td>$v_{3}(i,j,k)$</td>
<td>3-face</td>
<td>velocity in the 3-direction (grid units)</td>
</tr>
<tr>
<td>$e_{t}(i,j,k)$</td>
<td>zone centre</td>
<td>total energy density</td>
</tr>
<tr>
<td>$e_{1}(i,j,k)$</td>
<td>zone centre</td>
<td>first internal energy density ($\propto$ pressure)</td>
</tr>
<tr>
<td>$e_{2}(i,j,k)$</td>
<td>zone centre</td>
<td>second internal energy density</td>
</tr>
<tr>
<td>$g_{p}(i,j,k)$</td>
<td>zone-centre</td>
<td>gravitational potential</td>
</tr>
<tr>
<td>$b_{1}(i,j,k)$</td>
<td>1-face</td>
<td>magnetic field in the 1-direction ($\mu_0 = 1$)</td>
</tr>
<tr>
<td>$b_{2}(i,j,k)$</td>
<td>2-face</td>
<td>magnetic field in the 2-direction ($\mu_0 = 1$)</td>
</tr>
<tr>
<td>$b_{3}(i,j,k)$</td>
<td>3-face</td>
<td>magnetic field in the 3-direction ($\mu_0 = 1$)</td>
</tr>
<tr>
<td>$d_{a}(i,j,k)$</td>
<td>zone centre</td>
<td>product of density and synchrotron age</td>
</tr>
</tbody>
</table>
Appendix C: The ZEUS-3D variables

There is very little internal scaling of variables in ZEUS-3D that the user must consider. Density, energy, and velocity all may be scaled according to the user’s needs simply by setting the initial conditions as appropriate. For example, the user may wish to set the density and the sound speed at infinity to 1. This, along with some canonical length scale will set the time scale for the calculation. The only scaling implicit to ZEUS-3D is the permeability of free space ($4\pi \times 10^{-7}$ in mks, $4\pi$ in cgs) is set to 1. Thus, the total pressure (thermal plus magnetic) is given by $p_{\text{tot}} = p_{\text{th}} + B^2/2$. Having set the scale of the hydrodynamical variables, the user should set the magnetic fields with this additional scaling in mind.

If the EDITOR macro ISO is defined, the total and first internal energy densities, $e_1$ and $e_2$, are not declared. The second internal energy ($e_2$), the gravitational potential ($g_{\text{p}}$), the magnetic field components ($b_1$, $b_2$, $b_3$), and the synchrotron “density-age” ($d_{\text{a}}$) are declared only if the EDITOR macros TWOFLUID, GRAV, MHD, and AGING are defined respectively. If PSGRAV is defined, the “pseudo-gravitational potential” array ($p_{\text{sgp}}$) ($\neq g_{\text{p}}$) is declared.

C.3 Boundary variables (2-D arrays)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{\text{iib}}$</td>
<td>(j,k)</td>
<td>zone-centre at $i=\text{is}-1$</td>
</tr>
<tr>
<td>$g_{\text{iib}}$</td>
<td>boundary type for all variables except $g_{\text{p}}$</td>
<td></td>
</tr>
<tr>
<td>$d_{\text{iib1}}$</td>
<td>(j,k)</td>
<td>boundary type for gravitational potential</td>
</tr>
<tr>
<td>$v_{\text{1iib1}}$</td>
<td>(j,k)</td>
<td>1-face at $i=\text{is}$</td>
</tr>
<tr>
<td>$v_{\text{2iib1}}$</td>
<td>(j,k)</td>
<td>2-face at $i=\text{is}-1$</td>
</tr>
<tr>
<td>$v_{\text{3iib1}}$</td>
<td>(j,k)</td>
<td>3-face at $i=\text{is}-1$</td>
</tr>
<tr>
<td>$e_{\text{1iib1}}$</td>
<td>(j,k)</td>
<td>zone-centre at $i=\text{is}-1$</td>
</tr>
<tr>
<td>$e_{\text{2iib1}}$</td>
<td>(j,k)</td>
<td>zone-centre at $i=\text{is}-1$</td>
</tr>
<tr>
<td>$g_{\text{piib1}}$</td>
<td>(j,k)</td>
<td>zone-centre at $i=\text{is}-1$</td>
</tr>
<tr>
<td>$b_{\text{2iib1}}$</td>
<td>(j,k)</td>
<td>2-face at $i=\text{is}-1$</td>
</tr>
<tr>
<td>$b_{\text{3iib1}}$</td>
<td>(j,k)</td>
<td>3-face at $i=\text{is}-1$</td>
</tr>
<tr>
<td>$e_{\text{mf1ib1}}$</td>
<td>(j,k)</td>
<td>1-edge at $i=\text{is}-1$</td>
</tr>
<tr>
<td>$e_{\text{mf2iib1}}$</td>
<td>(j,k)</td>
<td>2-edge at $i=\text{is}$</td>
</tr>
<tr>
<td>$e_{\text{mf3iib1}}$</td>
<td>(j,k)</td>
<td>3-edge at $i=\text{is}$</td>
</tr>
<tr>
<td>$d_{\text{aiib1}}$</td>
<td>(j,k)</td>
<td>zone-centre at $i=\text{is}-1$</td>
</tr>
<tr>
<td>$d_{\text{aiib2}}$</td>
<td>(j,k)</td>
<td>zone-centre at $i=\text{is}-1$</td>
</tr>
<tr>
<td>$d_{\text{aiib3}}$</td>
<td>(j,k)</td>
<td>1-face at $i=\text{is}-2$</td>
</tr>
<tr>
<td>$v_{\text{1iib2}}$</td>
<td>(j,k)</td>
<td>1-face at $i=\text{is}-1$</td>
</tr>
<tr>
<td>$v_{\text{2iib2}}$</td>
<td>(j,k)</td>
<td>2-face at $i=\text{is}-2$</td>
</tr>
<tr>
<td>$v_{\text{3iib2}}$</td>
<td>(j,k)</td>
<td>3-face at $i=\text{is}-2$</td>
</tr>
<tr>
<td>$e_{\text{1iib2}}$</td>
<td>(j,k)</td>
<td>zone-centre at $i=\text{is}-2$</td>
</tr>
<tr>
<td>$e_{\text{2iib2}}$</td>
<td>(j,k)</td>
<td>zone-centre at $i=\text{is}-2$</td>
</tr>
<tr>
<td>$g_{\text{piib2}}$</td>
<td>(j,k)</td>
<td>zone-centre at $i=\text{is}-2$</td>
</tr>
<tr>
<td>$b_{\text{2iib2}}$</td>
<td>(j,k)</td>
<td>2-face at $i=\text{is}-2$</td>
</tr>
<tr>
<td>$b_{\text{3iib2}}$</td>
<td>(j,k)</td>
<td>3-face at $i=\text{is}-2$</td>
</tr>
<tr>
<td>$e_{\text{mf1ib2}}$</td>
<td>(j,k)</td>
<td>1-edge at $i=\text{is}-2$</td>
</tr>
<tr>
<td>$d_{\text{aiib2}}$</td>
<td>(j,k)</td>
<td>zone-centre at $i=\text{is}-2$</td>
</tr>
<tr>
<td>$v_{\text{1iib3}}$</td>
<td>(j,k)</td>
<td>1-face at $i=\text{is}-2$</td>
</tr>
</tbody>
</table>
The previous table lists the arrays for the first, second, and third inner-\(i\) boundaries. Note there are no boundary arrays for the total energy density; these values are computed directly from boundary values of the primitive variables where needed. Analogous boundary variables exist at the outer-\(i\) boundary (oib), inner-j boundary (ijb), outer-j boundary (ojb), inner-k boundary (ikb), and outer-k boundary (okb).

Note that the i-boundary variables use indices (\(j,k\)) and are declared so long as the \texttt{EDITOR} macro \texttt{ISYM} is \textit{not} defined. Similarly, the j-boundary variables use indices (\(k,i\)) and are declared so long as \texttt{JSYM} is \textit{not} defined while the k-boundary variables use indices (\(i,j\)) and are declared so long as \texttt{KSYM} is \textit{not} defined. All internal energy boundary variables (\(e_{1iib1}, \text{ etc.}\)) are \textit{not} declared if \texttt{ISO} is defined. The boundary variables for the second internal energy (\(e_{2iib1}, \text{ etc.}\)), gravity (\(g_{piib1}, \text{ etc.}\)), the magnetic field components (\(b_{2iib1}, \text{ etc.}\)) and the synchrotron density-age (\(d_{aiib1}, \text{ etc.}\)) are declared only if \texttt{TWOFLUID}, \texttt{GRAV}, \texttt{MHD}, and \texttt{AGING} are defined respectively.

Note that boundary variables are used only for regions of the boundary specified as inflow \([niib(j,k)=8,10 \text{ and/or } giib=10]\). For boundary type 8 (selective inflow), grid values are used where boundary variables are set to \texttt{huge}. For the gravitational potential, the boundary variable, \(g_{piib1}\), is set to known analytical or asymptotic values. For all other boundary types, the boundary values of the flow variables are determined from the values in the neighbouring active zones (§B.8).

### C.4 Scratch variables

There are a multitude of scratch arrays available which can be used to minimise the additional memory required by the user’s subroutines. These should be used wherever possible, especially for 3-D arrays. There are 26 1-D arrays dimensioned (\(ijkx\)) and named \texttt{wa1d} through \texttt{wz1d}, 18 2-D arrays dimensioned (\(idim,jdim\)) and named \texttt{wa2d} through \texttt{wr2d} (see §C.6 for definition of parameters \texttt{idim} and \texttt{jdim}), and finally nine 3-D arrays dimensioned (\(in,jn,kn\)) and named \texttt{wa3d} through \texttt{wi3d}.

### C.5 Sundry variables (an abbreviated list)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ioin</td>
<td>logical unit attached to input deck</td>
</tr>
<tr>
<td>iolog</td>
<td>logical unit attached to message log file</td>
</tr>
<tr>
<td>iotty</td>
<td>logical unit attached to terminal (TTY or CRT)</td>
</tr>
<tr>
<td>iodmp</td>
<td>logical unit attached to restart dumps</td>
</tr>
<tr>
<td>iopl1</td>
<td>logical unit attached to 1-D plot files</td>
</tr>
<tr>
<td>iopl2</td>
<td>logical unit attached to 2-D plot files</td>
</tr>
<tr>
<td>iopix</td>
<td>logical unit attached to 2-D pixel dumps</td>
</tr>
<tr>
<td>iousr</td>
<td>logical unit attached to user dumps</td>
</tr>
<tr>
<td>iotsl</td>
<td>logical unit attached to time slice ascii dump</td>
</tr>
<tr>
<td>iotsp</td>
<td>logical unit attached to time slice plot dump</td>
</tr>
<tr>
<td>iodis</td>
<td>logical unit attached to display dump</td>
</tr>
<tr>
<td>iorad</td>
<td>logical unit attached to \textit{RADIO} dump</td>
</tr>
</tbody>
</table>
### Appendix C: The ZEUS-3D Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iocrk</td>
<td>logical unit attached to cork dump</td>
</tr>
<tr>
<td>nhy</td>
<td>number of cycles (time steps) completed in simulation</td>
</tr>
<tr>
<td>nwarn</td>
<td>running total of warnings issued</td>
</tr>
<tr>
<td>prtime</td>
<td>problem time elapsed in simulation</td>
</tr>
<tr>
<td>dt</td>
<td>increment of problem time that solution is being advanced</td>
</tr>
</tbody>
</table>

In addition, all of the namelist variables (except for namelist pgen) are declared in comvar.

## C.6 Parameters

All global parameters are declared and set in common deck par. Primary parameters (those which the user can set) include:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>number of zones in 1-direction plus 5 ghost zones</td>
</tr>
<tr>
<td>jn</td>
<td>number of zones in 2-direction plus 5 ghost zones</td>
</tr>
<tr>
<td>kn</td>
<td>number of zones in 3-direction plus 5 ghost zones</td>
</tr>
<tr>
<td>nxpx</td>
<td>maximum number of pixels in the x-direction for pixel dumps</td>
</tr>
<tr>
<td>npx</td>
<td>maximum number of pixels in the y-direction for pixel dumps</td>
</tr>
<tr>
<td>nxrd</td>
<td>maximum number of pixels in the x-direction for RADIO dumps</td>
</tr>
<tr>
<td>nyrd</td>
<td>maximum number of pixels in the y-direction for RADIO dumps</td>
</tr>
<tr>
<td>niov</td>
<td>maximum number of variables plotted/dumped</td>
</tr>
<tr>
<td>nios</td>
<td>maximum number of slices for each variable plotted/dumped</td>
</tr>
<tr>
<td>ncsl</td>
<td>maximum number of slices for each variable plotted/dumped</td>
</tr>
<tr>
<td>ncls</td>
<td>maximum number of contour levels in 2-D NCAR/PSPLOT plots</td>
</tr>
<tr>
<td>ntsl</td>
<td>maximum number of time slices to be collected for plots</td>
</tr>
<tr>
<td>ncrk</td>
<td>maximum number of Lagrangian particles (&quot;corks&quot;)</td>
</tr>
<tr>
<td>nmat</td>
<td>maximum number of materials. With TWOFLUID set, this should be 2</td>
</tr>
<tr>
<td>isig</td>
<td>number of significant figures to which some real*8 numbers are rounded.</td>
</tr>
<tr>
<td>pi</td>
<td>3.14159...</td>
</tr>
<tr>
<td>tiny</td>
<td>$1.0 \times 10^{-99}$: smallest greater-than-zero number available on machine</td>
</tr>
<tr>
<td>huge</td>
<td>$1.0 \times 10^{99}$: largest number available on machine</td>
</tr>
<tr>
<td>smll</td>
<td>$1.0 \times 10^{-6}$: a convenient &quot;small&quot; number.</td>
</tr>
<tr>
<td>lrg</td>
<td>$1.0 \times 10^{6}$: a convenient &quot;large&quot; number.</td>
</tr>
</tbody>
</table>

The parameter nios is used by 1-D and 2-D NCAR/PSPLOT plots, pixel dumps, and display dumps. The parameter niov is used by these plus HDF dumps and RADIO dumps.

Secondary parameters (those which are computed from the primary parameters and the user does not set but should still be aware of) include:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ijkx</td>
<td>the maximum of in, jn, and kn</td>
</tr>
<tr>
<td>ijkn</td>
<td>the minimum of in, jn, and kn</td>
</tr>
<tr>
<td>idim</td>
<td>= jn (kn, in) if ISYM (JSYM, KSYM) is set [1- (2-, 3-) symmetry flag]</td>
</tr>
<tr>
<td></td>
<td>= ijkx if no symmetry is set</td>
</tr>
<tr>
<td>jdim</td>
<td>= kn (in, jn) if ISYM (JSYM, KSYM) is set [1- (2-, 3-) symmetry flag]</td>
</tr>
<tr>
<td></td>
<td>= ijkx if no symmetry is set</td>
</tr>
</tbody>
</table>
Index

This is an incomplete index with approximate (±1) page numbers. Those in bold face indicate primary references.

A

Adaptive Mesh Refinement (AMR; see “AZEuS”)  
ambipolar diffusion (AD; see also “AMBIGIFF”) 11, 15, 69-70  
AMBIGIFF 15, 69  
animations (see also pixel and RADIO dumps) 10, 26–27, 29, 79–81, 85, 89–90  
AZEuS v

B

batch mode 24  
boundaries 4, 7–8, 10–11, 17, 45, 61–67, 98
  bgen 17  
  wiggle 17  
bremssstrahlung 5, 29, 32, 74, 88, 89

C

CFL limit 2, 5, 11, 18  
change decks 12–14, 19, 21–23, 41, 42, 44, 48
  chgzeus 13, 14, 19, 21–23, 42, 48  
changing the code 12, 21–22, 36–44
  adding whole subroutines 36–41  
  changes to existing code 22, 41–44  
  debugging 16, 44–47  
checkin.c, checkin.o (see also “interrupt messages”) 19, 21, 33

common block (see “comvar”)

compilers 10, 12, 14, 19, 20, 22
  compiler options 22

comvar (ZEUS-3D declarations) 36, 39–41, 94, 99

Consistent Advection (CA) 2, 59, 60

Consistent Method of Characteristics (CMoC; see also “FASTCMOC”) 4–6, 9, 11, 16

contributors viii, 1

cork dumps (CORKS) 10, 13, 15, 18, 28, 34, 51, 75, 85, 99

cosmic rays 4

D

data dumps 25-30, 84
  naming conventions 25–30

DEBUG (EDITOR macro) 16

debugging (see “changing the code”)
declarations (see “comvar”)
DIFFUSION (EDITOR macro) 5, 14, 18, 49–50, 51
display dumps (DISP) 13, 15, 18, 28, 34, 53, 85–86, 98
dnamelist.a 6, 19, 21, 23, 52
dsci01.a 6, 19, 21
dzeus3.6 directory 12, 15, 19, 21–24, 41, 44
dzeus36 (source code) 6, 7, 12–13, 14, 16, 18, 20, 21, 39, 41, 46, 52, 54
dzeus36.f (see “EDITOR, error messages”)
dzeus36.m (see “EDITOR, error messages”)
dzeus36.mac (see ”zeus36.mac”)
dzeus36.n (see “EDITOR, listing files”)
dzeus36.s (script file) 10, 12–13, 14, 19–24, 41, 44, 48, 91

$E$

EDITOR 8–10, 12, 14–15, 19, 21–25, 39, 40–44, 49, 51–52, 94
   *alias, *al 13–14, 15, 43
   *call, *ca 36, 39, 41, 43, 94
   *cdeck, *cd 42, 44
   *deck, *dk 36, 39, 42, 44
   *define, *def 13, 14, 19, 43
   *delete, *d 21, 42, 44
   *else, *el 43
   *endif, *ei 36–38, 40, 43, 64–67
   *if 36–38, 40, 41, 43, 64–67
   *insert, *i 36, 39, 41, 42, 44
   *read, *r 19, 21–22, 41, 44, 48
   *replace, *rp 42
aliases (see also “skeleton”) 8, 12, 13–14, 16–18, 23, 25, 40, 51
comments 20
definitions 14–16, 23, 25, 40
error messages 23, 43, 44
inedit (EDITOR input deck) 9, 19, 20, 22, 23, 24
listing files 41–43
macros (see “zeus36.mac”)
manual 15, 22, 44
merging files 14, 21, 23, 41–44
precompiling files 15, 16, 21–23, 39–41, 43, 44
setting definitions and aliases 15–18
equation of state (EOS) 15, 70
ISO (EDITOR macro) 3, 9, 13, 15, 58, 67, 97–98
itote (total energy equation) 4, 9, 20, 49, 50, 58, 59, 69
polytropic (see also “POLYTROPE”) 15
executable (see “xdzeus36”)
\( \mathcal{F} \)

**FASTCMOC** *(EDITOR macro; see also “CMoC”)* 5, 13, 16

features
- version 3.0 1–3
- version 3.2 3–4
- version 3.3 4–5
- version 3.4 6–7
- version 3.5 7–9
- version 3.6 9–11

Finely Interleaved Transport (FIT) 9, 11, 46, 50, 60

\( \mathcal{G} \)

gas diffusion *(ARTIFICIALVISC option)* 5, 14, 18, 51

gsometry 2, 3, 13, 15, 27, 94–95

gravity (see also “self-gravity, GRAV, GRAVITY”) 15, 17, 51, 68

**GRAV** *(EDITOR macro)* 5, 6, 15, 67–68, 98

**GRAVITY** *(EDITOR macro)* 14, 15, 17, 51, 68

gdx03.a 7, 19, 21, 73, 78

grid 53, 54, 55–57, 70–71, 94–96
- changing on a restart 53–55
- extension 17, 51, 71–72
- generation 55–57
- moving 18, 69
- ratioed 55–56
- scaled 55–56
- staggered 2, 94–95
- variables 95–96

\( \mathcal{H} \)

**HDF4** dumps *(EDITOR macro)* 13, 15, 27, 34, 83, 99

**HISTORIAN** 14, 22, 41

\( \mathcal{I} \)

**implicit none** statement 39

initialising variables (see “problem generator”)

interrupt messages (see also “checkin.c”) 21, 33–35

**inzeus** *(ZEUS-3D input deck)* 12, 13, 17, 20, 23–24, 30, 40, 48, 52

isothermal equation of state (see “equation of state, ISO”)

**ISYM** *(EDITOR macro)* 13, 15, 38, 40, 60, 63, 98, 99
- example of use 38

\( \mathcal{J} \)

**JSYM** *(EDITOR macro)* 13, 15, 64, 65, 98, 99
\( \mathcal{K} \)

KSYM (EDITOR macro) 13, 15, 65, 66, 98, 99

\( \mathcal{L} \)

Lagrangian particles (see “cork dumps”)
Legacy transport 11, 46, 49, 60
limitations (of the code) 2
line of sight integrations (see “RADIO dumps”)
LINUX 6, 12
listing dzeus36 (see “EDITOR, listing files”)

\( \mathcal{M} \)

macros (see “zeus36.mac”)
MAKE (UNIX utility) 12, 15, 20, 23
makezeus (makefile) 10, 19–20, 22, 23, 24, 41, 44
merging dzeus36 (see “EDITOR, merging files”)
message log file 30, 41, 98
MHD (EDITOR macro) 13, 15, 25, 54, 62, 97, 98
MHD equations 1
movies (see animations)
multi-tasking (micro-tasking) 3, 23, 39

\( \mathcal{N} \)

namelists 23–30, 52–93
  column reserved for key characters 24
  comments 24
  comparison between EDITOR and system versions 23, 54
  error messages 23, 52
  namelist.a (see “dnamelist.a”)
  setting rank 2 arrays 23
NCAR graphics dumps 3, 19, 21, 23, 25, 26, 53, 73–74, 75, 79, 83, 98
noncar.a 7, 19, 21, 75, 79
number.s (see “EDITOR, listing files”)
numerical attributes (see “features”)

\( O \)

OpenMP (see “parallelisation”)
optimisation, warnings 22

\( P \)

parallelisation (see also “multi-tasking”) 22–23, 56
parameters 16, 21, 23, 24, 25, 37, 39, 40, 48, 52–93, 99
pixel dumps (PIX) 4, 10, 13, 15, 18, 26–27, 31, 34, 53, 79–83, 97, 98
plots 15, 17
1-D plots (PLT1D) 9, 13, 15, 18, 25–26, 27, 34, 53, 72–75
2-D contour and vector plots (PLT2D) 9, 13, 16, 18, 26, 53, 76–80

PLOTZ stand-alone code 26, 28
Poisson solver (see “self-gravity”)
polarisation 85, 89–90

POLYTROPE (EDITOR macro; see also “equation of state, polytropic”) 6, 13, 15
Postscript graphics (see “PSPLOT graphics”)
precompiling dzeus36 (see “EDITOR, precompiling files”)
problem generator 8, 18, 22, 36, 40, 61, 94–95
pseudogravity (see PSGRAV)
PSGRAV (EDITOR macro) 5, 6, 13, 15, 27, 97
PSPLOT graphics 7, 21, 23, 25–26, 53, 72, 75, 76, 79, 84, 99
rebinning 2-D data-slices 10, 76, 77
psplot.a 7, 19, 21, 75, 79

Q
qcon and qlin (see “viscosity, artificial”)

R

RADIO dumps (EDITOR macro) 4, 5, 10, 13, 16, 18, 29–30, 31, 34, 53, 86–92, 98, 99
RADIO stand-alone code 28, 29–30
ratio of specific heats (gamma) 70
restart dumps 4, 18, 25, 26, 27, 33, 53–55, 97, 99
restarting a run 25, 33, 53–55
RTP (EDITOR macro; see “geometry”)

S
scaling 56, 68, 76, 79, 85, 90, 97
scratch arrays (see “worker arrays”)
self-gravity 2, 4, 16, 67
size of ZEUS-3D 12, 21
skeleton 16, 41, 45, 49, 50
source code (see dzeus36)
Stokes parameters 3, 30, 86, 88
sub-cycling 5, 59
symmetry (see “geometry”)

\( T \)
time slice dump (TIMESL) 13, 16, 18, 28, 34, 84
transport algorithms (see “Finely Interleaved Transport” and “Legacy transport”)
TWOFLOUID (EDITOR macro) 5, 13, 15, 28, 38, 64–68, 70, 97, 98, 99
105

U

user agreement vii

USERDUMP (EDITOR macro) 13, 14, 30, 34, 41, 51, 83

\n
V

variables 17, 31–32, 39–40, 94–99
  boundary variables 60–67, 97–98
  field variables 96–97
  grid variables 95–96
  scratch variables (see “worker arrays”)
  sundry variables 98–99
viscosity, artificial 2, 5, 18, 51, 59
  setting qcon and qlin 20, 59
viscosity, kinematic 7, 59
voxel dumps (VOX) 10

W

worker arrays 40, 98

X

xdzeus36 (ZEUS-3D executable) 12, 13, 19, 22, 24, 44, 48
xedit22 (EDITOR executable) 19–21
XYZ (EDITOR macro; see “geometry”)

Z

ZEUS development project 1
ZEUS, history v
ZEUS-2D 1
zeus3.6 directory (see “dzeus3.6”)
zeus36 (see “dzeus36”)
zeus36.f (see “dzeus36.f”)
zeus36.m (see “dzeus36.m”)
zeus36.mac (EDITOR macro file) 10, 12–15, 19, 21–23, 41, 42, 48, 92
zeus36.n (see “dzeus36.n”)
zeus36.s (see “dzeus36.s”)
zdnnnid (see “display dumps”)
zh**nnnid (see “HDF dumps”)
zi**nnnid.it m (see “pixel dumps”)
zlnnnid (see “message log file”)
zpnnnid.mm (see “plots, 1-D plots”)
qanmnid.mm (see “plots, 2-D plots”)
ZR**nnnid (see “RADIO dumps”)
ZRP (EDITOR macro; see “geometry”)
zn


id (see “restart dumps”)

zt

nnid (see “time slice dumps”)

ztpnnid (see “time slice dumps”)

zu

nnid (see “USERDUMP”)