The `athena1.1` *User's Guide*

James M. Stone
Department of Astrophysical Sciences
Princeton University
Princeton, NJ 08544-1001

Thomas A. Gardiner, Peter J. Teuben
Department of Astronomy
University of Maryland
College Park MD 20742-2421

and

John F. Hawley
Department of Astronomy
University of Virginia
PO Box 3818 University Station
Charlottesville, VA 22903

1 Introduction

Athena is a grid-based code for astrophysical gas dynamics being developed with support of the NSF Information Technology Research (ITR) program. This *User's Guide* describes version 1.1 (hereafter referred to as `athena1.1`). `athena1.0` was the first publically released version of the code, version 1.1 corrects several serious errors in the isothermal eigensystems, and some minor bugs throughout the code.

The primary goal of the *Athena* project is to develop a robust and flexible code with multiple-physics options in a variety of geometries, using flexible gridding methods, and optimized for modern shared or distributed memory parallel machines. This code is necessary to enable increasingly sophisticated investigations of a wide variety of problems in astrophysical gas dynamics. As part of the project, *Athena* will be made freely available to the astrophysics community, along with complete documentation and web-based training material. Although the capabilities of `athena1.1` are quite restricted compared to the ultimate goals of the project, it is being freely distributed consistent with this open source philosophy.

The `athena1.1` code contains algorithms for the following:

- compressible hydrodynamics and ideal MHD in one spatial dimension
- ideal gas equation of state with arbitrary γ (including $\gamma = 1$, an isothermal EOS)
- second- or third-order characteristic interpolation in the primitive variables
- numerical fluxes computed using Roe's, or the HLLE, approximate Riemann solver (the HLLC solver is also available but for hydrodynamics only)

Since later versions of the code will always be backwards compatible (in particular, they will run one-dimensional problems simply and efficiently), **it is strongly advised to use the latest release of the code for research problems (even those in one-dimension) to take advantage of improvements and potential bug fixes.**

There are four basic sources of documentation for *Athena*

1. *The Method Paper:* (in preparation)
2. *The User's Guide:* (this document) gives an overview of how to install, configure, compile, and run `athena1.1` and visualize the resulting output. Necessary reading for all users.
3. *The Programmers's Guide:* (in preparation)
4. *Web-based Tutorials:* (in preparation)

Since so much of this documentation is still in preparation, the equations of motion solved by `athena1.1` are given below as a starting point.

Users of `athena1.1` should have a basic, working knowledge of the Unix operating system, access to a C compiler, and a graphics package for plotting one-dimensional tabular data. Some familiarity with code management using Makefiles is helpful but not necessary.

The equations solved by `athena1.1` can be written in conservative form as

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F_x}(\mathbf{U})}{\partial x} = 0, \tag{1}$$

where $\mathbf{U}$ is a vector of conserved quantities and $\mathbf{F_x}$ the vector of their fluxes. The number and form of the components of both $\mathbf{U}$ and $\mathbf{F_x}$ depend on the physics of the problem to be solved. For adiabatic MHD the vector of conserved quantities and their fluxes are

$$\mathbf{U} = \begin{bmatrix} \rho \\ \rho v_x \\ \rho v_y \\ \rho v_z \\ E \\ B_y \\ B_z \end{bmatrix}, \qquad \mathbf{F_x} = \begin{bmatrix} \rho v_x \\ \rho v_x^2 + P + B^2/(8\pi) - B_x^2/(4\pi) \\ \rho v_x v_y - B_x B_y/(4\pi) \\ \rho v_x v_z - B_x B_z/(4\pi) \\ (E + P + B^2/(8\pi)) - (\mathbf{B}\cdot\mathbf{v})B_x/(4\pi) \\ B_y v_x - B_x v_y \\ B_z v_x - B_x v_z \end{bmatrix}. \tag{2}$$

where

$$E = \frac{P}{\gamma - 1} + \frac{1}{2}\rho v^2 + \frac{B^2}{8\pi} \tag{3}$$

and $B^2 = \mathbf{B}\cdot\mathbf{B}$.

For isothermal MHD, the fifth components of $\mathbf{U}$ and $\mathbf{F_x}$ are dropped, and equation (3) is replaced with $P = \rho C^2$, where C is a constant (the isothermal speed of sound). For adiabatic hydrodynamics, the last two components of $\mathbf{U}$ and $\mathbf{F_x}$ are dropped, as well as all terms involving the magnetic field in equations (2) and (3). For isothermal hydrodynamics, the last three components of $\mathbf{U}$ and $\mathbf{F_x}$ are dropped, as well as all terms involving the magnetic field in equations (2) and (3).

2 Obtaining and Installing `athena1.1`

The source code for `athena1.1` can be downloaded as a gzipped tar file from the web. Only the source code can be downloaded. To install and run the code requires only a C compiler.

After downloading the tar file, uncompressing and untarring it, the following directory structure should be created:

```
/athena1.1
          /doc          documentation (this User's Manual)
          /src          source code, and include files
              /prob     problem files (see /src/problem.c for a symbolic link)
          /tst          various input files for tests
              /1D-hydro
              /1D-mhd
          /vis          visualization tools and scripts
              /dx       OpenDX scripts
              /idl      IDL (rsinc.com) scripts
              /nemo     NEMO scripts
```

The `athena1.1` directory will also contain a shell script called `configure` which has been generated by the GNU `autoconf` toolkit, as well as several other files used by this toolkit. As described in §3, this script provides command-line control of all of the physics and algorithm options in the code.

The contents of the directories in `/athena1.1` are largely self-explanatory. The `athena1.1/src` directory contains all the source code for `athena1.1`, including functions to initialize various test problems in the sub-directory `/prob`. The `athena1.1/tst` directory contains input files read by the code when running the tests (§8 contains a description and examples of how to run `athena1.1`). Finally, the directory `athena1.1/vis` contains some preliminary scripts for visualizing the data created by athena1.1 using three popular graphics packages: OpenDX, IDL, and PGPLOT within the NEMO package. The code will also output data as formatted tables which can be read by other plotting packages such as sm; see §6 for a complete description of the data output formats implemented in athena1.1. Note that the visualization scripts included in athena1.1 are under development, and later versions will have much more sophisticated capabilities.

In addition to the directories listed above (which are created when you untar the source code), two more directories will be created when athena1.1 is compiled or run. These are:

```
/athena1.1
         /bin           contains executable, created by Makefile
         /run           directory created by executable as needed
```

The /run directory is created through command line input to the athena1.1 executable, and need not have the name "run". This feature allows multiple jobs to create data in distinct directories as an aid to managing results (see §5.2 for an example of usage).

3 Configuring athena1.1

After you have installed athena1.1, the next step is to configure the code for a specific problem, a specific machine, and for a specific level of accuracy and optimization. A shell script called configure generated by the GNU autoconf toolkit[1] is used for this step.

There are several basic functions served by the configure script. The first is to enable or disable *features* in the code, and to choose between different *packages* implemented in athena1.1. Configure *features* are used when a particular option has only two choices: enabled (or "on") or disabled ("off"): examples include choosing whether the executable uses single or double precision, or enabling or disabling various data output formats. Configure *packages* are used when an option may have more than one choice. Examples of *packages* include the basic physics options (such as hydrodynamics or MHD, adiabatic or isothermal equation of state, etc.), as well as the algorithm options (order of accuracy for the reconstruction step, choice of Riemann solver, etc.). These *package* options are controlled at the source code level using C precompiler macros. The configure script provides a powerful way of setting these macros through a command line interface that does not require the user to edit any special files.

A second basic function served by configure is to set compiler and linker options and flags using environment variables. Since optimization is not an important issue for a one-dimensional code, this feature is not used in athena1.1.

Finally, a third function of configure is to query the system automatically to see that a C-compiler, linker, and any external libraries necessary for compilation are installed and accessible. If they are not, configure can automatically disable features so as to allow installation to complete. For example, if the HDF library is not installed on the target system, configure can automatically disable HDF data output (even if the user requests this feature be enabled in the command line), allowing the code to be compiled and linked without errors due to unresolved references (although the resulting executable will not generate HDF files, of course)[2].

The basic syntax of the configure command is:

configure [--enable-*feature*] [--disable-*feature*] [--with-*package*=*choice*],

[1] autoconf: see http://www.gnu.org/software/autoconf/

[2] Actually, since HDF output is not yet implemented in athena1.1 this example is moot

where *feature* and *package* are valid options in `athena1.1`, and *choice* is the value to which *package* is to be set. The valid optional features in `athena1.1` are given in Table 1, and the valid optional packages (including all possible choices and their default values) are given in Table 2.

Table 1: Optional features controlled by `configure` in `athena1.1`

FEATURE	DEFAULT	Comments
double	enabled	computations performed in double precision
dx	disabled	writes header files for OpenDX scripts
fortran	disabled	makes binary dumps readable by fortran77 programs[a]

[a]Current implementation almost certainly not portable, but works with Solaris and Linux.

Table 2: Optional packages controlled by configure in `athena1.1`

PACKAGE	CHOICE[a]	Comments
gas	hydro	create code for hydrodynamics
	mhd	create code for MHD
eos	**adiabatic**	use adiabatic equation of state
	isothermal	use isothermal equation of state
problem	*function-name*	use *function-name* in directory `/src/prob` for initial conditions
flux	**roe**	use Roe's Riemann solver
	hlle	use the HLLE Riemann solver
	hllc	use the HLLC Riemann solver (hydrodynamics only)
order	**2**	use second-order (piecewise-linear) spatial reconstruction
	3	use third-order (piecewise-parabolic) spatial reconstruction

[a] Default choices shown in bold.

Only one choice can be made for each optional package given in Table 2 (the choices are mutually exclusive). The "problem" package should be set to the function name in the directory `athena1.1/src/prob` that is to be used by the code to initialize the problem of interest. Several choices are included for the test problems that can be run by `athena1.1`, the default is `athena1.1/src/prob/shkset.c`. Note that `configure` creates a symbolic link between `problem.c` in `athena1.1/src` and the appropriate function name in `athena1.1/src/prob`.

The `configure` script should be run in the root directory `/athena1.1`. Running `configure` with the help option (`configure --help`) gives more information, including a list of all optional features and packages.

For example, to configure `athena1.1` to run an isothermal hydrodynamical shocktube problem initialized with the function `shkset.c` problem using Roe fluxes and third-order interpolation in single precision, use the following:

```
configure --enable-single --with-eos=isothermal --with-gas=hydro --with-order=3
```

To configure `athena1.1` to run the linear wave test problem in adiabatic MHD using HLLE fluxes and second-order interpolation in double precision, use the following:

```
configure --with-flux=hlle --with-problem=linear_wave
```

When `configure` runs, it creates a custom Makefile for the problem in the directory `athena1.1/src` using the file `athena1.1/src/Makefile.in` as a template. After successful execution, `configure` will echo the options that have been set (including all the default values).

4 Compiling athena1.1

After running `configure`, all that is required to compile the code is to type `make all` in the top level directory `/athena1.1` (within which the configure script is run). This automatically creates the directory `athena1.1/bin`, which will contain the executable, and runs make in the `athena1.1/src` directory to compile and link the code. The top-level Makefile in the `/athena1.1` directory also contains other targets which may be of use to users listed in Table 3.

Table 3: Makefile targets in `athena1.1`

TARGET	Comments
all	creates `athena1.1/bin` directory and compiles code
compile	compiles code
clean	removes .o files in `athena1.1/src`
help	prints help message
check	runs benchmark (see §8.1)

Normally the Makefiles should never need to be edited by hand. However, since compiler options and flags are not currently controlled through configure in `athena1.1`, it is possible to edit the Makefile in the `athena1.1/src` directory to change compiler flags, for example, to improve optimization. For example, setting `GNUFLAGS` to a blank (`''`) value will speed-up the code already.

5 Running athena1.1

After configuring and compiling `athena1.1`, there should be an executable called `athena` in the directory `athena1.1/bin`. There are two steps to running the code: (1) editing parameters in the input file, and (2) running the executable.

5.1 Editing the input file

Run time parameters are set in an input file, usually given the name `athinput.`*problem-name*, where *problem-name* is a string identifier. Often this string is the same as the name of the problem-generator, i.e. the function in `athena1.1/src/prob` which is used to initialize the data. In some cases, a name which is more specific to the problem at hand is used (since some problem-generators can be used to initialize more than one problem).

As an example of an `athena1.1` input file, the file `/athena1.1/tst/1D-mhd/athinput.brio+wu` is reproduced below. Other examples can be found in this and the `/athena1.1/tst/1D-hydro` directory.

```
<job>

problem_id      = Brio_Wu   # problem ID: basename of output filenames
restart_flag    = 0         # 0 for new job, 1 for restart
restart_file    = res000aa  # name of restart file

<time>

cour_no         = 0.8       # The Courant, Friedrichs, & Lewy (CFL) Number
dt_res          = 1.0       # time increment between restart dumps
nlim            = 1000      # cycle limit
tlim            = 1.0       # time limit
```

```
dt_hst          = 0.01      # time increment between history dumps
dt_bin          = 0.01      # time increment between binary  dumps
dt_hdf          = 0.1       # time increment between HDF-SDS dumps
dt_tab          = 0.01      # time increment between tabular dumps

<grid>

nxzones         = 800       # Number of zones in X-direction
x1min           = 0.0       # minimum value of X
x1max           = 1.0       # maximum value of X
ibc_x1          = 2         # inner-I (X) boundary flag
obc_x1          = 2         # outer-I (X) boundary flag

<problem>

gamma           = 2.0       # gamma = C_p/C_v

dl              = 1.0       # density on left half of grid
pl              = 1.0       # pressure
v1l             = 0.0       # X-velocity
v2l             = 0.0       # Y-velocity
v3l             = 0.0       # Z-velocity
b1l             = 0.75      # X-magnetic field
b2l             = 1.0       # Y-magnetic field
b3l             = 0.0       # Z-magnetic field

dr              = 0.125     # density on right half of grid
pr              = 0.1       # pressure
v1r             = 0.0       # X-velocity
v2r             = 0.0       # Y-velocity
v3r             = 0.0       # Z-velocity
b1r             = 0.75      # X-magnetic field
b2r             = -1.0      # Y-magnetic field
b3r             = 0.0       # Z-magnetic field
```

Note the syntax of the parameter specification used in this file. Parameters are grouped into named *blocks*, with the name of each block appearing on a single line within angle brackets. Block names must always appear in angle brackets on a seperate line (although the blank lines above and below the block names are not required)[3].

Below each block name is a list of parameters, with syntax

parameter-name = value [# comments]

White space after the parameter name, after the '=', and before the '#' character is ignored. Everything after (and including) the '#' character is also ignored. Only one parameter value can appear per line. Comment lines (i.e. a line beginning with '#') are allowed for documentation purposes. A maximum number of 256 characters is permitted per line in the input file. Both block names and parameter names are case sensitive.

The input file is read by a very flexible parser written for `athena1.1` (`athena1.1/src/par.c`). The entire input file is read at the very beginning of the main program, and the parameter names and their values stored in memory. Thereafter, these values can be accessed as necessary by any function at any time during execution. The parser allows the parameter names to appear in any order within a named block, extra (or misspelled) parameter names will be parsed and never used.

[3] In $ 5.2 you will see that a parameter can also be specified on the commandline by using "*block/parameter=value*"

There are no default values for any of the run-time parameters in the input file; each parameter must be supplied a value through the input file. If a value is requested from the parser but its name does not exist, the parser will print an error message and terminate the execution of the program. In this way, both missing or misspelled parameter names will be detected at run time. The parameters may be integers, floating point numbers, or strings. The parser will do automatic type conversion, for example converting floating point numbers to double precision if necessary (though the user is expected to know the basic difference between real, integer, and string data types). Parameter values can also be set at run time through the command line, which provides a very flexible way of testing the code and running parameter searches, see §5.2.

Below we describe each of the parameters in the input file and their function.

5.1.1 The `<job>` block

Parameters in this block control properties of the jobs run by `athena1.1`. They are accessed by the function `main.c`.

- **problem_id:** string added as basename of output filenames (see §6). Usually same as *problem-name* used in input file. There is no maximum length for this name[4].
- **restart_flag:** currently not used[5].
- **restart_file:** currently not used.

5.1.2 The `<time>` block

Parameters in this block control times in a job (such as ending time, or data output times). They are accessed by the function `main.c`.

- **tlim:** time to stop integration, in units defined by problem
- **nlim:** maximum number of cycles of the main loop before stopping.
- **cour_no:** CFL number, must be less than 1.0
- **dt_hst:** time interval between history dumps, in units defined by problem
- **dt_bin:** time interval between binary dumps, in units defined by problem
- **dt_hdf:** currently not used
- **dt_tab:** time interval between tabular dumps, in units defined by problem
- **dt_res:** currently not used.

[4]Although remember that the maximum length of a line in the input file is 256 characters.

[5]The ability to control the output of restart (or checkpoint) files from the executable is really only useful for multidimensional simulations, thus this capability is not implemented in `athena1.1`.

5.1.3 The `<grid>` block

Parameters in this block control the properties of the grid. They are accessed by the function `init_grid_block.c`.

- **nxzones:** number of grid points
- **x1min:** X-coordinate of left-edge of first cell
- **x1max:** X-coordinate of right-edge of last cell. The entire computational domain spans $x_{max} - x_{min}$, the grid spacing is $\triangle x = (x_{max} - x_{min})/N_{xzones}$, and the center of the first cell is located at $x = x_{min} + \triangle x/2$.
- **ibc_x1, obc_x1:** integer flags for boundary conditions applied at "inner" (left) and "outer" (right) edges of grid. Currently three values are implemented: 1 = reflecting, 2 = inflow and outflow (projection), and 4 = periodic.

5.1.4 The `<problem>` block

Parameters in this block are accessed by the problem-generator, and therefore depend on the problem being run. For example, the problem-generator `athena1.1/src/prob/shkset.c` (which is used for the Brio & Wu test problem) requires the following parameter values in the `<problem>` block:

- **gamma:** ratio of specific heats used in equation of state
- ***l:** values of variable * in left-state
- ***r:** values of variable * in right-state

The ***l** and ***r** parameter names are specific to the Brio & Wu shocktube problem. In general, the input file for other problems will have different variable names in the problem block.

5.2 Execution of `athena1.1`

Once the run parameters have been set by editing the input file, the `athena1.1` executable can be run using the `-i` option to specify the name of the input file. For example, to run the Brio & Wu shocktube use the command

```
% athena -i ../tst/1D-mhd/athinput.brio+wu
```

This will first dump out the values of all input parameters, and then when the code runs, it will print the cycle number and timestep to standard output; when it concludes it will print final diagnostic information. For example, as a simple diagnostic of code performance, the number of zones updated per cpu second is output.

A variety of command line options have been implemented in `athena1.1`. A list is given by the `-h` switch:

```
% athena -h

Usage: ../bin/athena [options] [block/par=value ...]
Options:
  -i <athinput_file>      Specify input file [default 'athinput']
  -d <run_directory>      Specify run dir [default current dir]
  -h                      This Help
  -n                      Parse input, but don't run program
```

The -d option can be used to create a new directory in which athena1.1 will run and write the output files. The -n option is useful for debugging any parsing errors, as it will dump the contents of all parsed block/parameters.

A value for any of the valid parameter names can also be input from the command line. This, in combination with the -d option, might be useful for parameter surveys. For example, suppose a problem has three parameters, a,b,c, and athena1.1 is to be used for a survey looping over 6, 5 and 7 different values of a,b,c respectively In C-shell notation, this could be accomplished quite easily as follows:

```
#! /bin/csh -f
  foreach      a  (1.0  1.5  2.0  2.5  3.0  3.5)
    foreach    b  (10   20   30   40   50)
      foreach  c  (1    3    10   30   100  300  1000)
        set dir=run_$a_$b_$c
        mkdir -p $dir
        athena -i your_athinput -d $dir problem/a=$a  problem/b=$b  problem/c=$c > $dir/athena.log
      end
    end
  end
```

Note that one winds up with 210 different run directories this way. An alternative would be to construct a hierarchy, with a on the top level, and b and c below. This could be accomplished with, for example, set dir=run210/$a/$b/$c.

6 Data output formats

The athena1.1 code can output data in a variety of formats. The time interval at which each data type is written can be selected independently using the <job>dt_XXX variables in the input file. Setting an output interval to 0.0 means no output is given for that type. Output filenames use a naming convention *basename.dumpid.type*, where the basename is inherited from the <job>problem_id parameter, the *dumpid* is a zero filled unsigned integer with <job>numdigits[6], and the *type* is either bin (for binary dumps), tab (for tabular dumps), or hst (for history dumps – note that history dump filenames do not include a *dumpid*).

1. history dumps: contains a formatted table of a variety of volume integrated values, with one line in the table created every dt_hst in time. Thus, at the end of execution, the output file contains tlim/dt_hst lines which form a time-history of these quantities. The file is created by the function dump_history.c; more (or problem specific) quantities can be added by editing this file. The data is appended to the file each time the *dump_history()* function is called.

2. binary dumps: contains an unformatted write of all dependent variables over all active zones. If the fortran option is enabled by configure, then the file will contain header (and footer) bytes which makes the file readable by fortran77 programs (at least on SGI, Solaris and Linux systems). If the dx option is enabled by configure, then an OpenDX header file with the same name as the corresponding binary file but with the extension .dx will be created. This header file allows binary dumps to be read by OpenDX networks (see §7.2). A new file is created with a time interval of dt_bin. Created by the function dump_binary.c.

3. tabular dumps: contains a formatted table of all dependent variables. A new file is created with a time interval of dt_tab. Created by the function dump_table.c.

[6]currently fixed at three

HDF dumps are currently not implemented.

It is important to note that **output files in** `athena1.1` **will always be silently overwritten!**

7 Visualizing output

Since `athena1.1` solves the equations in one-dimension, visualizing the data generated by the code is particularily simple as only 1D graphs are needed. Rudimentary scripts for several different graphics packages are supplied with the source code; future versions will incorporate much more sophisticated visualization tools. The following subsections describe useful visualization packages for `athena1.1` data files (the discussion assumes the code has already been run to produce output).

7.1 IDL procedures

The IDL (Interactive Data Language)[7] package can read the binary dump files. A simple procedure for reading `athena1.1` binary dumps is included in `athena1.1/vis/idl`; to run this procedure you will need IDL installed on your system. (Note that binary dumps must be created with the fortran option **disabled** to be compatible with IDL.) From the `athena1.1/bin` directory, use the following.

```
% idl
IDL> .run ../vis/idl/pltath.pro
IDL> readbin,'Brio_Wu.040.bin'
IDL> six_plot
```

A variety of procedures are included in the `pltath.pro` file, including a procedure to read binary files (`readbin`), and make simple plots and animations.

7.2 OpenDX networks

The OpenDX[8] package can read our binary dump files, provided the `.dx` header files exist. This requires `athena1.1` be configured with the dx option enabled (the fortran option may be either enabled or disabled), and of course OpenDX must be installed on the system.

An OpenDX session can be started from the `athena1.1/bin` directory with the following:

```
% cp ../vis/dx/PlotLine.* .          # for convenience make a local copy
% dx
  <Run Visual Programs>
  click on PlotLine.net in the ``Files''
  click on OK
```

7.3 NEMO

Although designed primarily for studying stellar dynamics, the NEMO[9] package contains graphics that can be used to to display and animate the tabular files generated by `athena1.1`. Some examples are given below (it is assumed that NEMO has been compiled with PGPLOT as the graphics driver). From the `athena1.1/bin` directory, use the following[10]:

```
% astroload nemo
% ../vis/nemo/1d-animate xcol=2 ycol=3
```

[7] see also: `http://www.rsinc.com`

[8] see also: `http://www.opendx.org`

[9] see also: `http://www.astro.umd.edu/nemo`

[10] astroload is part of `http://www.astro.umd.edu/nemo/linuxastro/astromake/`

```
        # this should show 28 time frames of density vs. position in rapid succession
% tkrun ../vis/nemo/1d-animate
        # this should show sliders for xcol and ycol (2=X-pos, 3=den, 4=M1, ... 11=B1i)
% ../vis/nemo/1d-pt-map
        # this will have produced a FITS file map-3.fits
% astroload ds9
% ds9 &
% mirds9 map-3.fits
        # the X axis (position) is 800 pixels, and the Y axis (time), so not very impressive
```

In addition, there are many other packages that can be used to plot the tabular files, (e.g. gnuplot, mongo, sm, etc.).

8 Examples of Running athena1.1

8.1 The athena1.1 Benchmark

To test the installation of `athena1.1` a benchmark can be automatically run using the Makefile. This benchmark consists of running the Brio & Wu shocktube problem on a grid of 800 zones to a time of 0.1, and then computing the L1 error norm between the resulting solution and a precomputed result (stored in the file `athena1.1/tst/1D-mhd/test.Brio_Wu.040.tab`). If the benchmark fails to run, or if the resulting error norm is large, then something has gone wrong in the installation or in the compilation of the code.

To run the benchmark, `athena1.1` must be configured with the default settings. The steps required are given below, including the output when the code is run on a 2.2GHz Athlon processor.

```
% cd athena1.1
% configure
% make all check
(cd tst/1D-mhd; ./run.brio+wu)
zone-cycles/cpu-second = 1.366816e+05
L1 norm for density: 0
```

The benchmark not only provides a test of the installation of `athena1.1`, but also is a convenient way to test how various compilers and compiler options affect the performance and accuracy of the code on a variety of machines.

8.2 Running a test problem with athena1.1: The Brio & Wu shocktube

As an example of how to configure, compile and run `athena1.1` and visualize the output, we show the steps required to run the Brio & Wu shocktube problem. Assuming the code has already been installed (see §2), the first step is to configure:

```
% cd athena1.1
% configure
```

The `autoconf` utility will print a variety of diagnostic statements during this step. Next, the code must be compiled:

```
% make all
```

The default compiler options will print diagnostic statements. Then the code can be run in the `athena1.1/bin` directory:

```
% cd bin
% athena -i ../tst/1D-mhd/athinput.brio+wu
```

The code will print information about every timestep as it executes. It should generate 40 binary (tabular) dumps named `Brio_Wu.*.bin`, as well as a history file `Brio_Wu.hst`. There are a variety of ways that the data in these files can be visulaized; one way is to use the IDL scipts included in `athena1.1/vis/idl`.

```
% idl
IDL> .run ../vis/idl/pltath.pro
IDL> readbin,'Brio_Wu.040.bin'
IDL> four_plot
```

The resulting plots which should now appear on the screen is shown in Figure 1.

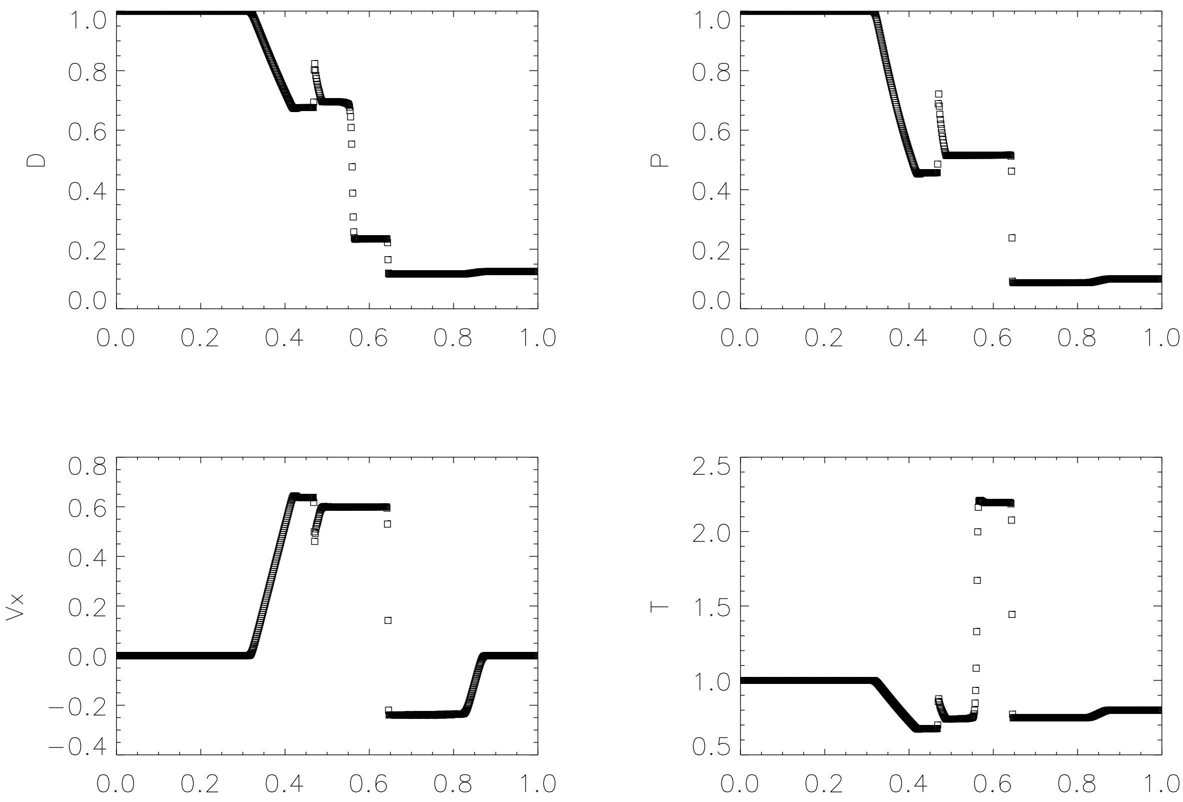

Figure 1: Results from Brio & Wu shocktube test problem plotting using our IDL routine `four_plot`