Chapter 2

Exercises

2.1 Usage of the example package `scalar`

In this section, we will explain how to use the package for solving a scalar equation. The content of this package is as follows:

```
# ls scalar
Makefile    anime.pro   main.f      pldt.pro    pldtps.pro    rddt.pro
```

The program is written in Fortran that is one of the most popular programing language in the field of astronomical simulations. It is contained in the file 'main.f' in this example.

2.1.1 Compilation and execution of the program

Before executing a program, we need to 'compile' it – change a format of the program from a human readable one into a machine executable one. After moving to the directory "`scalar/`", execute the UNIX command '`make`'. Then, the program will be executed after a compilation. If succeed, you will find several new files, `main.o`, `a.out` and `out.dat` in this directory. The file `main.o` is an 'object' file corresponding to 'main.f', and the file 'a.out' is an 'executable' file. The result of the simulation is contained in the output data file 'out.dat'.

```
# cd scalar
# make
f77   -c -o main.o main.f
main.f:
 MAIN:
f77 -o a.out main.o
./a.out
  write    step=       0 time= 0.000E+00
  write    step=      50 time= 0.125E+02
  write    step=     100 time= 0.250E+02
   ### normal stop ###
# ls
Makefile    anime.pro   main.o      pldt.pro     rddt.pro
a.out*      main.f      out.dat     pldtps.pro
```

2.1.2 Output data file (out.dat)

You can read the content of the 'out.dat' file by using an editor or an appropriate UNIX command (e.g. `more`, `less`, `head` etc.) since it is written in a human readable format. The following is an example of the content. The first line indicates the spatial size (`jx`) of the data array and the number of outputs (`nx`) in the temporal sequence of the simulation. In this example, there are 3 sets of arrays with length of 100. The next line is for the 'time' information of the first data set. Here the item '0' and '0.00' correspond to the step (`ns`) and the time (`time`), respectively, of the first data set in the output. From the next to the 102nd line, the data is placed. The left and right column indicates the spatial coordinate (`x`) and the value (`u`) of the simulation result. The next data set starts from the 103rd line in the same order, and so on. This output format is defined at the 53, 55, and 59th lines in the Fortran program file '`main.f`'.

```
# head out.dat
  100,    3
    0,  0.00
  1.0, 1.0000000
  2.0, 1.0000000
  3.0, 1.0000000
  ......
```

2.1.3 Visualization of a result

We usually use a special software for the visualization of the simulation results. Here we introduce "IDL" that is one of such commercial (expensive!) softwares and is very popular in astronomical data analysis both for simulations and observations.

Startup of IDL (idl)

To startup IDL, type `idl`.

```
# idl
```

Then, it starts as follows:

```
IDL Version ....
Installation number: XXXXX.
Licensed for use by: XXXXX

IDL>
```

You can enter the IDL commands after its prompt "`IDL>`". You may also run an IDL program.

Loading the data into the IDL session (.r rddt)

To load the simulation data into the IDL session, use the IDL program `rddt.pro` as follows:

```
IDL> .r rddt
```

After this, you can refer to, process, and visualize the data in the IDL session. Here "`.r`" means

"run".

Plot of the data (.r pldt)

To plot the data, use the IDL program `pldt.pro` as follows:

```
IDL> .r pldt
```

The result will be like Fig 2.1

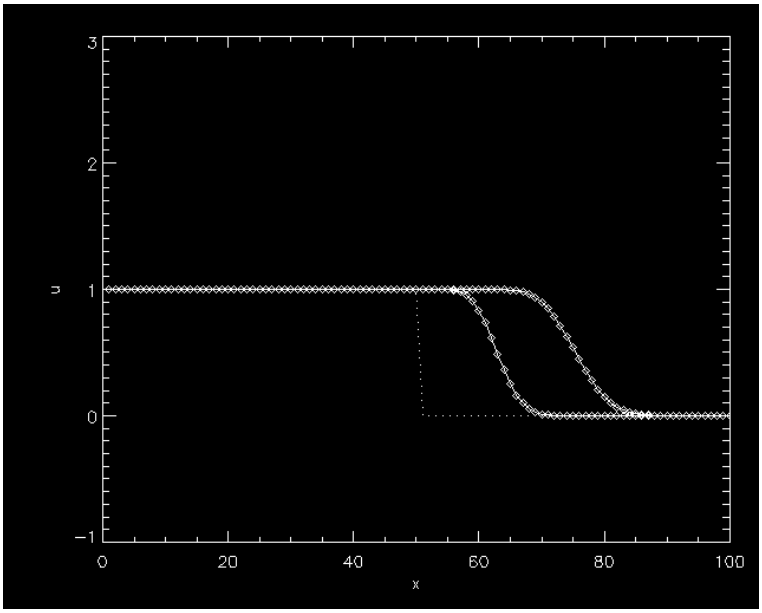

Figure 2.1: Plot of the simulation results in the 'scalar' package

Animation of the simulation results (.r anime)

To make an animation of the simulation results, use the IDL program `anime.pro` as follows:

```
IDL> .r anime
```

As a result of this IDL program, there appears a new window showing an animation. Note that an error occurs if you try to open another animation window simultaneously. Keep only one window.

Finish an IDL session (exit)

To finish an IDL session, type "`exit`" after the IDL prompt.

```
IDL> exit
```

2.1.4 Modification of the program

Change the hydrodynamic solver

The main solver in the example package is written with the upwind algorithm The Fortran statements of the algorithm is in the 72nd to 88th lines of 'main.f'. You can change it by modyfing these lines.

```
c-------------------------------------------------------------------------|
c     solve equation
c
c                                                       upwind - start
>>>
      do j=1,jx-1
         f(j)=0.5*(cs*(u(j+1)+u(j))-abs(cs)*(u(j+1)-u(j)))
      enddo

      f(jx)=f(jx-1)

      do j=2,jx-1
         u(j)=u(j)-dt/dx*(f(j)-f(j-1))
      enddo

      u(1)=u(2)
      u(jx)=u(jx-1)
c                                                       upwind - end   >>>
```

Change the number of mesh points (jx)

The number of mesh points is defined by the value of the variable `jx` at the 5th line of the program. The spatial resolution of the simulation can be controled by modyfing this part.

```
      parameter (jx=100)
```

Change the finishing step, the output settings (nstop, nskip)

The finishing step and interval step of the output are defined by the value of the variables `nstop` and `nskip`, respectively.

```
c   time control parameters

      nstop=100
      nskip = 50
```

Change the interval of the temporal stepping (safety)

The interval of the temporal stepping is determined by the CFL condition – a stability condition corresponding to each algorithm. This condition gives only an upper limit for the temporal interval. So we usually determine it by giving a 'safety number' (`safety`) less than unity. By changing this value, the stability, quality, and cost of simulations can be controlled.

```
c     obtain time spacing
      safety=0.25
```

2.1.5 Appendix

Sample Fortran program, main.f

```
c=======================================================================|
c     array definitions
c=======================================================================|
      implicit real*8 (a-h,o-z)
      parameter (jx=100)
      dimension x(1:jx),u(1:jx),f(1:jx)
c=======================================================================|
c     prologue
c=======================================================================|
c   time control parameters
      nstop=100
      nskip = 50
c-----------------------------------------------------------------------|
c  initialize counters
      time  = 0.0
      ns    = 0
      nx = nstop/nskip+1
c-----------------------------------------------------------------------|
c     Set initial condition
c-----------------------------------------------------------------------|
      pi=4.*atan(1.0)
c  grid
      dx=1.0
      x(1)=dx
      do j=1,jx-1
         x(j+1)=x(j)+dx
      enddo
c
c  variable
      do j=1,jx/2
         u(j)= 1.0
      enddo
      do j=jx/2+1,jx
         u(j)= 0.0
      enddo
c
c  velocity
      cs=1.0
c-----------------------------------------------------------------------|
c     Output initial condition
c
      write(6,103) ns,time
 103  format (1x,' write    ','step=',i8,' time=',e10.3)
      open(unit=10,file='out.dat',form='formatted')
      write(10,100) jx,nx
 100  format(i5,',',i5)
      write(10,101) ns,time
```

```
 101  format (i5,',',f6.2)
      do j=1,jx
         write(10,102) x(j),u(j)
      enddo
 102  format(f5.1,',',f10.7)
c=======================================================================|
c     time integration
c=======================================================================|
1000  continue
      ns = ns+1
c-----------------------------------------------------------------------|
c     obtain time spacing
      safety=0.25
      dt=safety*dx/cs
      time=time+dt
c-----------------------------------------------------------------------|
c     solve equation
c
c                                                     upwind - start >>>
      do j=1,jx-1
         f(j)=0.5*(cs*(u(j+1)+u(j))-abs(cs)*(u(j+1)-u(j)))
      enddo
      f(jx)=f(jx-1)

      do j=2,jx-1
         u(j)=u(j)-dt/dx*(f(j)-f(j-1))
      enddo
      u(1)=u(2)
      u(jx)=u(jx-1)
c                                                     upwind - end   >>>
c-----------------------------------------------------------------------|
c     data output
      if (mod(ns,nskip).eq.0) then
         write(6,103) ns,time
         write(10,101) ns,time
         do j=1,jx
            write(10,102) x(j),u(j)
         enddo
      endif

      if (ns .lt. nstop) goto 1000
      close(10)
*=======================================================================|
      write(6,*) '  ### normal stop ###'
      end
```

Sample IDL program, rddt.pro

```
; rddt.pro
openr,1,'out.dat'
readf,1,jx,nx

; define array
ns=intarr(nx)
t=fltarr(nx)

x=fltarr(jx)
u=fltarr(jx,nx)

; temporary variables for read data
ns_and_t=fltarr(2,1)
x_and_u=fltarr(2,jx)

for n=0,nx-1 do begin
 readf,1,ns_and_t
 readf,1,x_and_u
 ns(n)=fix(ns_and_t(0,0))
 t(n)=ns_and_t(1,0)
 u(*,n)=x_and_u(1,*)
endfor

close,1
free_lun,1

x(*)=x_and_u(0,*)

delvar,ns_and_t,x_and_u

help
end
```

Sample IDL program, pldt.pro

```
!x.style=1
!y.style=1
!p.charsize=1.4

plot,x,u(*,0),xtitle='x',ytitle='u',linest=1,yrange=[-1,3],xrange=[0,100]
for n=1,nx-1 do begin
oplot,x,u(*,n)
oplot,x,u(*,n),psym=4
endfor

end
```

Sample IDL program, anime.pro

```
!x.style=1
!y.style=1
!p.charsize=1.4

window,xsize=480,ysize=480
xinteranimate,set=[480,480,nx]

for n=0,nx-1 do begin

plot,x,u(*,n),xtitle='x',ytitle='u',yrange=[-1,3],xrange=[0,100]
oplot,x,u(*,n),psym=4

xinteranimate,frame=n,window=0

endfor

xinteranimate

end
```

2.1.6 Exercise

Linear wave equation

Run the example package of the 'scalar' by referring to Section 2.1.1 to 2.1.5 of this textbook. The package is for solving the linear wave equation by the upwind algorithm. The initial values are $u_j = 1$ for $j = 1, ...50$ and $u_j = 0$ for $j = 51, ...100$. The Courant number is $\nu = c\Delta t/\Delta x = 0.25$. Make following new programs by modifying the original one, namely,

1. a program solving by the FTCS algorithm, and

2. a program solving by the Lax-Wendroff algorithm,

3. a program solving with the minmod limiter (see Equation 1.146).

Plot and compare the results of these programs with each other.

Note: A finite difference form of the one-dimensional wave equation

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \tag{2.1}$$

can be written like

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}(f_{j+1/2}^n - f_{j-1/2}^n). \tag{2.2}$$

By using the FTCS (Forward in Time and Centered in Space) algorithm, the numerical flux is given as

$$f_{j+1/2}^n = \frac{1}{2}(f_{j+1} + f_j) = \frac{1}{2}c\ (u_{j+1} + u_j). \tag{2.3}$$

Examples of the numerical flux of other algorithms for *the linear wave equation* as follows:
Lax-Friedrich algorithm:

$$f_{j+1/2}^n = \frac{1}{2}\left[(1 - \frac{1}{\nu})\ cu_{j+1} + (1 + \frac{1}{\nu})\ cu_j\right] \tag{2.4}$$

Upwind algorithm:

$$f_{j+1/2}^n = \frac{1}{2}[\ c\ (u_{j+1} + u_j) - |c|\ (u_{j+1} - u_j)\ ] \tag{2.5}$$

Lax-Wendroff algorithm:

$$f_{j+1/2}^n = \frac{1}{2}[\ (1 - \nu)\ cu_{j+1} + (1 + \nu)\ cu_j\ ] \tag{2.6}$$

Here, $\nu \equiv c\Delta t/\Delta x$.

Burgers equation

Make and run a program for solving the Burgers equation,

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{u^2}{2}\right) = 0, \tag{2.7}$$

by the 1st-order upwind algorithm. Plot the results and compare them with those in Figures 1.9 – 1.12. The numerical flux for this program can be written as

$$f_{j+1/2}^n = \frac{1}{2}\left\{\left(\frac{{u_{j+1}}^2}{2} + \frac{{u_j}^2}{2}\right) - \frac{1}{2}|u_{j+1} + u_j|(u_{j+1} - u_j)\right\}. \tag{2.8}$$

Diffusion equation

Make and run a program for solving the diffusion equation,

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2} \tag{2.9}$$

by the FTCS algorithm. Plot the results and compare them with those in Figure 2.2. Set up an appropriate initial distribution, e.g. a Gaussian distribution, and define the diffusion coefficient κ instead of the wave speed c as follows:

```
c  variable
      do j=1,jx
         u(j)= exp(-(((x(j)-x(jx/2))/5.)**2))
      enddo
c
c  kappa
      kappa=1.0
```

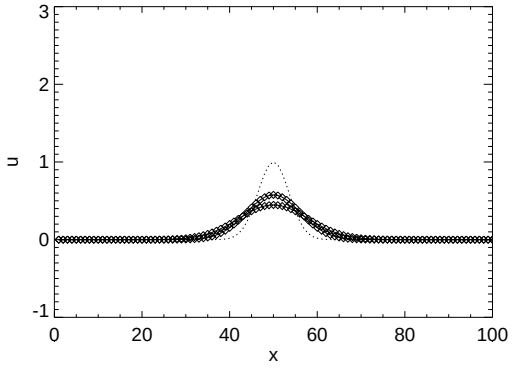

Figure 2.2: Result of a simulation for solving the diffusion equation.

2.2 Usage of the CANS package: shock tube problem

2.2.1 CANS1D

The CANS1D consists of many sets of *subroutines* and *model packages*. For example, the subroutines to solve the hydrodynamic / magnetohydrodynamic (MHD) equations are contained under the directory "`hdmlw`". The files are as follows:

```
# ls hdmlw
Makefile       mlw_ht.f       mlw_m3_g.f     mlw_m_g.f      mlwfull.f
README         mlw_ht_c.f     mlw_m3t.f      mlw_mt.f       mlwhalf.f
Readme.tex     mlw_ht_cg.f    mlw_m3t_c.f    mlw_mt_c.f     mlwsrcf.f
mlw_a.f        mlw_ht_g.f     mlw_m3t_cg.f   mlw_mt_cg.f    mlwsrch.f
mlw_h.f        mlw_m.f        mlw_m3t_g.f    mlw_mt_cgr.f
mlw_h_c.f      mlw_m3.f       mlw_m_c.f      mlw_mt_g.f
mlw_h_cg.f     mlw_m3_c.f     mlw_m_cg.f     mlw_rh.f
mlw_h_g.f      mlw_m3_cg.f    mlw_m_cgr.f    mlwartv.f
```

The model packages are collections of programs for solving the 'typical problems' that are considered to be basic for understanding the hydrodynamic / MHD simulations, e.g. the shock-tube problem, the Sedov point explosion problem and so on. Each package is contained in a separate directory whose name start with "`md_`". From here, we will explain the shock-tube problem as an example to use the CANS1D. The files in the shock-tube problem package are as follows:

```
# ls md_shktb
Makefile            bnd.f               pldt.pro
README              cipbnd.f            rddt.pro
Readme.pdf          main.f              shktb_analytic.pro
Readme.tex          main.pro
anime.pro           model.f
```

The solving program consists of several files with a file-name extension '.f' written in the Fortran language. The documents are in the files `README` and `Readme.pdf`.

2.2.2 Compilation of the subroutines in CANS1D

Before executing a program, we need to 'compile' subroutines. By this procedure, several 'library archive' files will be made with a file extension '.a' under the CANS top directory. After moving to the CANS top directory, execute the UNIX command '`make`'. (Warning! It will take much time if the CPU speed is low.) The products of this procedure are the library-archive files, `libcansnc.a`, `libcans1d.a`, `libcans2d.a`, and `libcans3d.a`. Each of these is an archive of object files of the subroutines.

```
# cd cans
# make
.......
# ls
Develop.txt   Models.tex  Readme.log  cans1d/  idl/         xmhdshktb.ps
Makefile      NonLTE/     Readme.pdf  cans2d/  libcans1d.a  xshktb.ps
Makefile.rel  README      Readme.ps   cans3d/  libcans2d.a
Models.pdf    Readme.aux  Readme.tex  cansnc/  libcans3d.a
Models.ps     Readme.dvi  avs/        htdocs/  libcansnc.a
```

2.2.3 Compilation and execution of the main program

For the compilation of the main program of the shock tube problem, move to the directory `cans1d/md_shktb`. Execute the UNIX command '`make`'. Then, the program will be executed after a compilation. If succeed, you will find several new files, `main.o`, `a.out`, `params.txt` and several files with extension of '`.dac`' in this directory. The file `main.o` is an 'object' file corresponding to 'main.f', and the file 'a.out' is an 'executable' file. The result of the simulation is contained in the output data file '*.dac'.

```
# cd cans1d/md_shktb
# make
f77   -c -o main.o main.f
f77   -c -o model.o model.f
f77   -c -o bnd.o bnd.f
f77   -c -o cipbnd.o cipbnd.f
f77 -o a.out main.o model.o bnd.o cipbnd.o \
        -L../.. -lcans1d -lcansnc
./a.out
  write    step=       0 time= 0.000E+00 nd =  1
  write    step=      51 time= 0.101E-01 nd =  2
  write    step=      93 time= 0.201E-01 nd =  3
  .....
  write    step=     585 time= 0.142E+00 nd = 16
  stop     step=     585 time= 0.142E+00
   ### normal stop ###
```

2.2.4 Visualization of a result

We usually use a special software for the visualization of the simulation results. Here we introduce "IDL" that is one of such commercial (expensive!) softwares and is very popular in astronomical data analysis both for simulations and observations.

Startup of IDL (idl)

To startup IDL, type `idl`.

```
# idl
```

Then, it starts as follows:

```
IDL Version ....
Installation number: XXXXX.
Licensed for use by: XXXXX

IDL>
```

You can enter the IDL commands after its prompt 'IDL>'. You may also run an IDL program.

Loading the data into the IDL session (.r rddt)

To load the simulation data into the IDL session, use the IDL program `rddt.pro` as follows:

```
IDL> .r rddt
```

After this, you can refer to, process, and visualize the data in the IDL session. Type '`help`' to obtain a list of available arrays and variables in the IDL session.

```
IDL> help
......
GM              FLOAT     =       1.40000
IX              LONG      =         1026
......
NX              LONG      =           16
PR              DOUBLE    = Array[1026, 16]
PR1             FLOAT     =      0.100000
RO              DOUBLE    = Array[1026, 16]
RO1             FLOAT     =      0.125000
......
T               DOUBLE    = Array[16]
TE              DOUBLE    = Array[1026, 16]
VX              DOUBLE    = Array[1026, 16]
......
```

Here `PR`, `RO`, `TE` and `VX` are arrays of the pressure, density, temperature and (x-component of) velocity, respectively. Note that in IDL sessions, the letter case of the variable names will be ignored, namely 'pr' and 'PR' correspond to the same variable.

Plot of the data (.r pldt)

To plot the data, use the IDL program `pldt.pro` as follows:

```
IDL> .r pldt
```

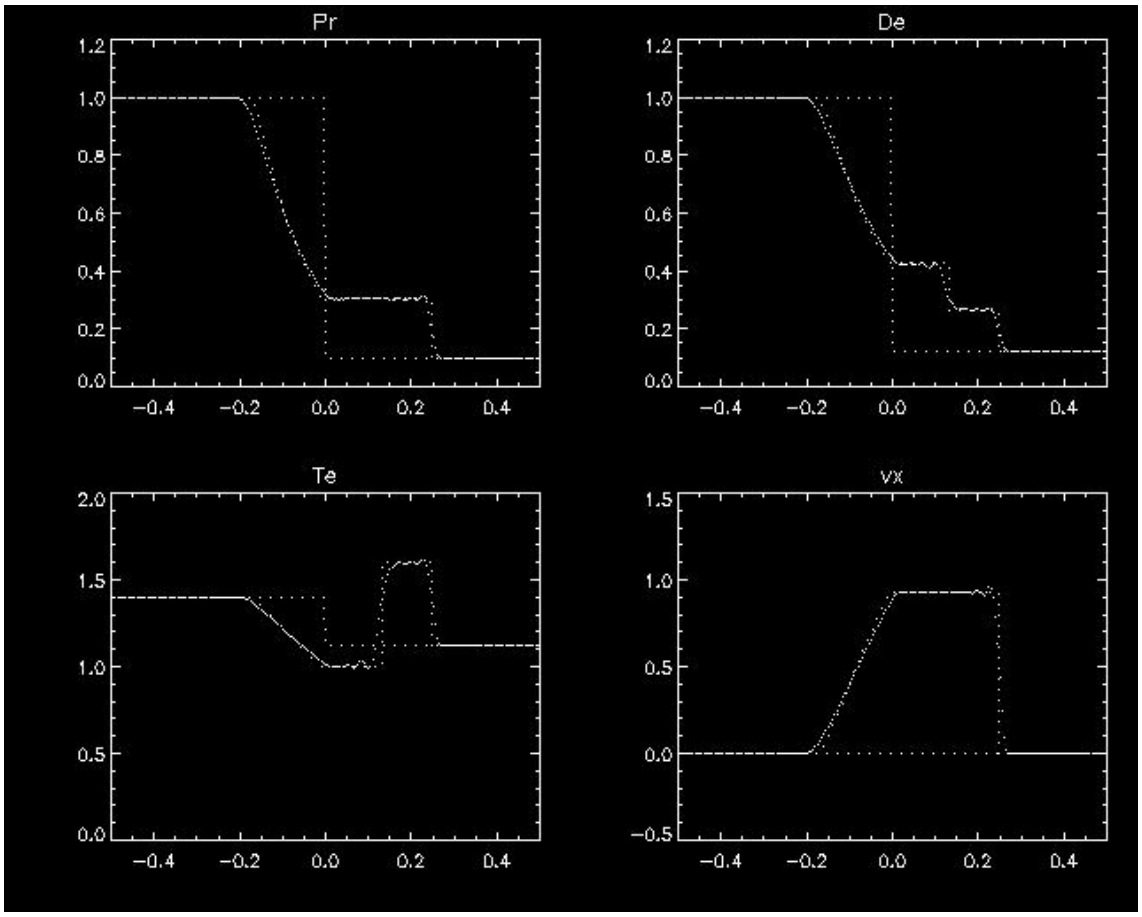

Figure 2.3: Result of the package `md_shktb`

Animation of the simulation results (.r anime)

To make an animation of the simulation results, use the IDL program `anime.pro` as follows:

```
IDL> .r anime
```

Finish an IDL session (exit)

To finish an IDL session, type `exit` after the IDL prompt.

```
IDL> exit
```

2.3 Exercise

2.3.1 Try CANS1D

1. Try the model package "Isothermal shock tube (`md_itshktb`)". Run the program and visualize the results by using IDL.

2. Try the model package "Shock tube (`md_shktb`)". Run the program and visualize the results by using IDL.
3. Try the model package "Shock formation (`md_shkform`)". Run the program and visualize the results by using IDL.
4. Try the model package "MHD shock tube (`md_mhdshktb`)". Run the program and visualize the results by using IDL (Fig. 2.4).
5. Try any of the model packages.

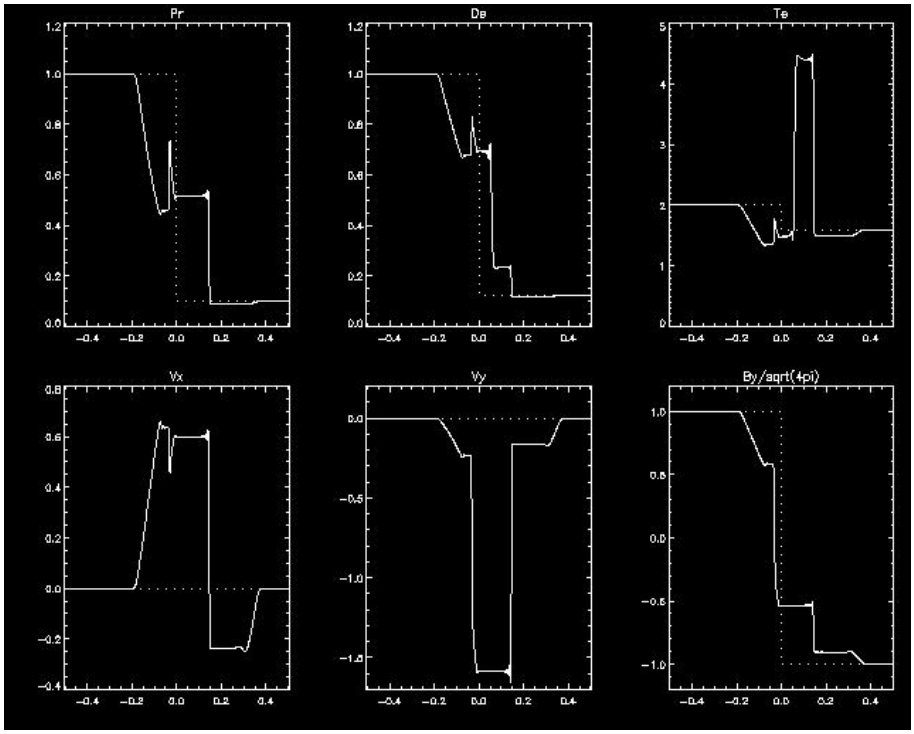

Figure 2.4: Results of `md_mhdshktb`

Note:

- When one runs a Fortran program, the output files, `params.txt` and `***.dac` are all overwritten. Rename these files or back up to any other directory before executing a program to avoid overwriting.
- To remove the object and executable files, type "`make clean`" after the UNIX prompt.

2.3.2 Try and modify the package md_shktb

Change the number of the mesh points by modyfing the appropriate file(s) in the model package "Shock tube (`md_shktb`)", run the program, and compare the results with the original one. Also change the interval of the data output and try an animation in IDL.

2.3.3 Try and modify the package md_sedov

Change the specific heat ratio γ by modyfing the appropriate file(s) in the model package "Supernova: the Sedov solution (`md_sedov`)", run the program, and compare the results with

the original one.

2.4 Advanced Exercise

Referring to Section 1.5, answer the following questions. We consider a one-dimensional hydrodynamic flow. The initial condition is given by,

$$(\rho_j, P_j, u_j) = \begin{cases} (1,\ 1,\ 0) & (j \le 0) \\ (0.81,\ 0.6,\ 0) & (j > 0) \end{cases} .$$

The gas is ideal one and specific heat ratio is $\gamma = 5/3$.

1. Compute the Roe average density from ρ_0 and ρ_1.
2. Compute H_0, H_1, and $\bar{H}$.
3. Compute the sound speed, a.
4. Compute the amplitudes, w_1, w_2 and w_3.
5. Modify the package `md_shktb` and obtain the numerical solution. Explain the numerical solution in terms of a, w_1, w_2 and w_3.
6. Try the package `md_shkin` and compare the results with Figure 1.17.