Ay 190 Assignment 14

Grid-Based Hydrodynamics

March 26, 2013

1 1D Planar Finite-Volume Hydrodynamics Code

The solution to the classic 1D shocktube problem was calculated in a number of ways and the results compared. A domain over [0,1] was initially set up with a fluid of density $\rho_L = 1.0$ and pressure $P_L = 1.0$ to the left of $x = 0.5$, and a second fluid of density $\rho_R = 0.1$ and pressure $P_R = 0.125$ to the right of $x = 0.5$. The system was evolved until $t = 0.2$.

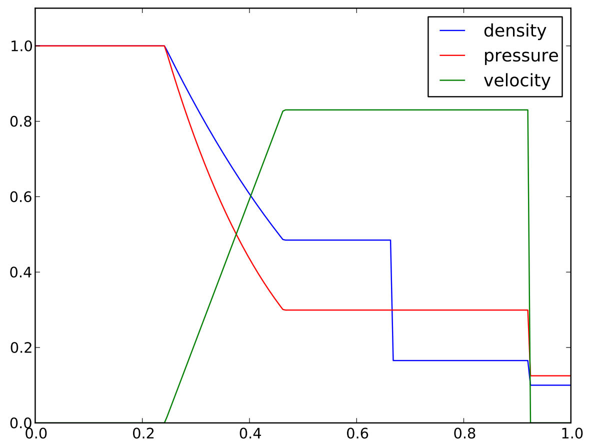

Figure 1: A solution to the shocktube problem initialied as described above, computed using Frank Timmes' exact Riemann solver.

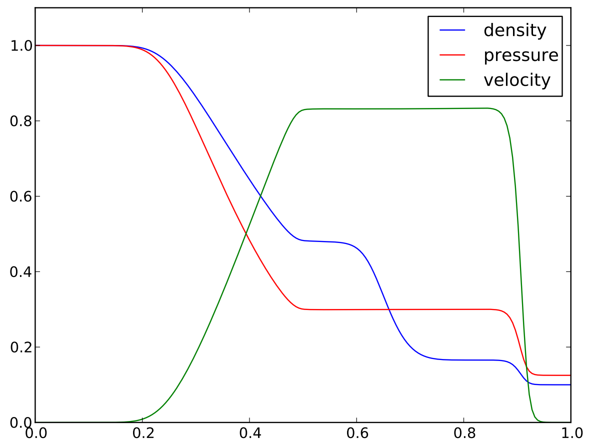

Figure 2: A solution for the same shocktube problem computed using a HLLE solver. The sharp transitions and discontinuities are smoothed out, likely due to the HLLE solver's approximation to the solution of the Riemann problem.

1.1 Comparisons of algorithm performance

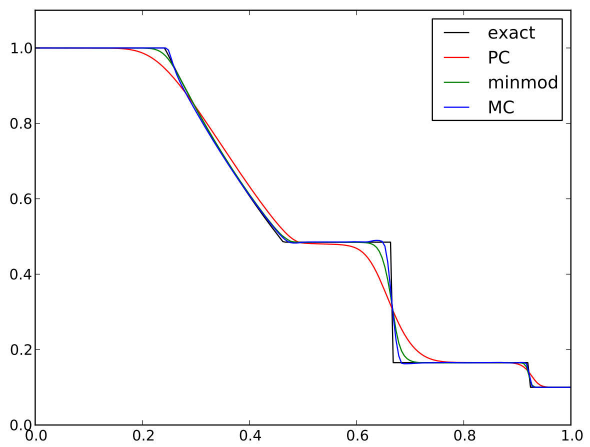

Figure 3: Comparison of density profiles produced using the HLLE approximate Riemann solver with piecewise constant, minmod, and monotonized central limiters for reconstruction compared with the exact Riemann solver. MC reproduces the transitions and discontinuities the best, though at the cost of introducing ripples in the solution, particularly at early times. At later times, the ripples exist near discontinuities. Minmod also develops ripples, but are suppressed earlier. PC reproduces the transitions and discontinuities the worst, but never develops ripples.

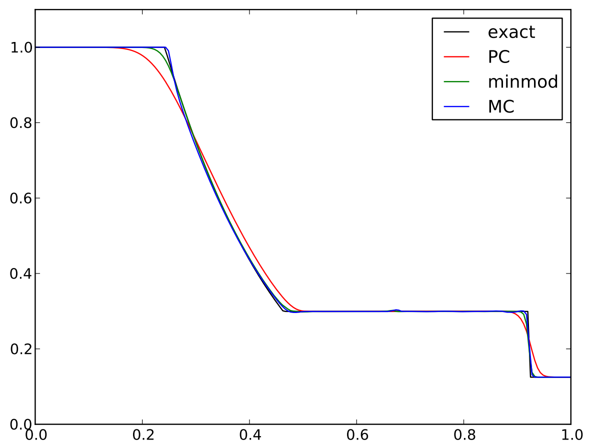

Figure 4: The same as Figure 3, but of pressures.

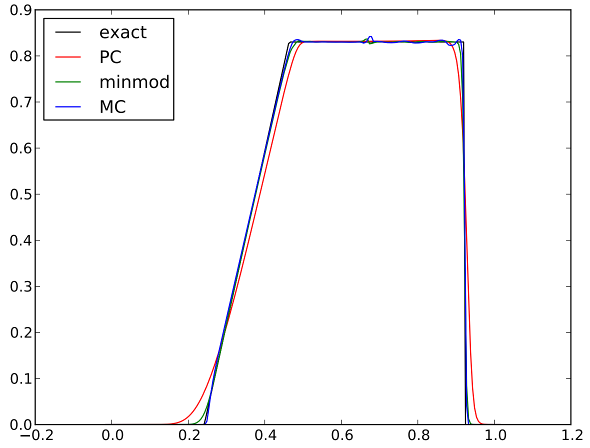

Figure 5: The same as Figure 3, but of velocities.

Appendices

The Python modules and scripts used in this assignment include

- A 1D Finite Volume Planar Hydro Code: `hydro_planar.py` (see page 3-)
- B 1D Finite Volume Spherical Hydro Code: `hydro_spherical.py` (see page 3-)

Frank Timmes' exact Riemann solver can be found at `cococubed.asu.edu/code_pages/exact_riemann.shtml`.

A 1D Planar Finite Volume Hydro Code

```
#!/usr/bin/env python
import sys,math
from pylab import *

# basic parameters
gamma = 5.0/3.0
cfl = 0.5
dt = 1.0e-5
dtp = dt
reconstruction_type = 'minmod' # minmod, mc, pc
# use nzones
nzones = 200
# run until time 0.2
tend = 0.2

##################################### class definition
class mydata:
    def __init__(self,nzones):
        self.x     = zeros(nzones) # cell centers
        self.xi    = zeros(nzones) # cell LEFT interfaces
        self.rho   = zeros(nzones)
        self.rhop   = zeros(nzones)
        self.rhom   = zeros(nzones)
        self.vel    = zeros(nzones)
        self.velp   = zeros(nzones)
        self.velm   = zeros(nzones)
        self.eps    = zeros(nzones)
        self.epsp   = zeros(nzones)
        self.epsm   = zeros(nzones)
        self.press  = zeros(nzones)
        self.pressp = zeros(nzones)
        self.pressm = zeros(nzones)
        self.q     = zeros((3,nzones)) # conserved quantities
        self.qp    = zeros((3,nzones))
        self.qm    = zeros((3,nzones))
        self.n     = nzones
        self.g     = 3 # ghost cells at each of inner/outer edges of domain
    def setup_grid(self,xmin,xmax):
        dx = (xmax - xmin) / (self.n - self.g*2 - 1)
        xmin = xmin - self.g*dx
        xmax = xmax + self.g*dx
        for i in range(self.n):
            self.x[i] = xmin + (i)*dx        # cell centers
            self.xi[i] = self.x[i] - 0.5*dx  # cell LEFT interfaces


    def shocktube_setup(self):
        # Shocktube initial data
        rchange = (self.x[self.n-self.g-1] - self.x[self.g]) / 2.0
        rho1 = 1.0
        rho2 = 0.1
        press1 = 1.0
        press2 = 0.125
        for i in range(self.n):
            if self.x[i] < rchange:
                self.rho[i] = rho1
                self.press[i] = press1
                self.eps[i] = press1 / (rho1) / (gamma - 1.0)
                self.vel[i] = 0.0
            else:
                self.rho[i] = rho2
                self.press[i] = press2
                self.eps[i] = press2 / (rho2) / (gamma - 1.0)
                self.vel[i] = 0.0


##################################### some basic functions
def prim2con(rho,vel,eps):
    q = zeros((3,len(rho)))
    q[0,:] = rho[:]
    q[1,:] = rho[:] * vel[:]
    q[2,:] = rho[:] * eps[:] + 0.5 * rho[:] * vel[:]**2

    return q

def con2prim(q):
    rho = q[0,:]
    vel = q[1,:] / rho
    eps = q[2,:] / rho - 0.5*vel**2
    press = eos_press(rho,eps,gamma)

    return (rho,eps,press,vel)

############# boundary conditions
def apply_bcs(hyd):
    hyd.rho[0:hyd.g-1] = hyd.rho[hyd.g]
    hyd.vel[0:hyd.g-1] = hyd.vel[hyd.g]
    hyd.eps[0:hyd.g-1] = hyd.eps[hyd.g]
    hyd.press[0:hyd.g-1] = hyd.press[hyd.g]

    hyd.rho[hyd.n-hyd.g:hyd.n-1] = hyd.rho[hyd.n-hyd.g-1]
    hyd.vel[hyd.n-hyd.g:hyd.n-1] = hyd.vel[hyd.n-hyd.g-1]
    hyd.eps[hyd.n-hyd.g:hyd.n-1] = hyd.eps[hyd.n-hyd.g-1]
    hyd.press[hyd.n-hyd.g:hyd.n-1] = hyd.press[hyd.n-hyd.g-1]

    return hyd

############# minmod function
def minmod(a,b):
    if(a*b < 0):
        mm = 0.0
    elif(abs(a)<abs(b)):
        mm=a
    else:
        mm=b
    return mm

############# minmod function
def tvd_minmod_reconstruct(n,g,f,x,xi):
    fp = zeros(n)
    fm = zeros(n)
    for i in range(g-1,n-g+1):
        dx_up = x[i] - x[i-1]
        dx_down = x[i+1] - x[i]
        dx_m = x[i] - xi[i]
        dx_p = xi[i+1] - x[i]
        df_up = (f[i]-f[i-1]) / dx_up
        df_down = (f[i+1]-f[i]) / dx_down
        delta = minmod(df_up,df_down)
        fp[i] = f[i] + delta*dx_p
        fm[i] = f[i] - delta*dx_m

    return (fp,fm)


############# signum functions
def signum(x,y):
    if(y >= 0):
        return abs(x)
    else:
        return -abs(x)


############# mc reconstruction
def tvd_mc_reconstruct(n,g,f,x,xi):
    fp = zeros(n)
    fm = zeros(n)
    for i in range(g-1,n-g+1):
        dx_up = x[i] - x[i-1]
        dx_down = x[i+1] - x[i]
        dx_m = x[i] - xi[i]
        dx_p = xi[i+1] - x[i]
        df_up = (f[i]-f[i-1]) / dx_up
        df_down = (f[i+1]-f[i]) / dx_down
        if(df_up*df_down < 0):
            delta = 0.0
        else:
            delta = signum(min(2.0*abs(df_up),2.0*abs(df_down),\
                               0.5*(abs(df_up)+abs(df_down))),\
                               df_up + df_down)

        fp[i] = f[i] + delta*dx_p
        fm[i] = f[i] - delta*dx_m

    return (fp,fm)


############# reconstruction top level function
def reconstruct(hyd,type):
    if(type=='pc'):
        # piecewise constant reconstruction
        for i in range(hyd.g-1,hyd.n-hyd.g+1):
            hyd.rhop[i] = hyd.rho[i]
            hyd.rhom[i] = hyd.rho[i]
            hyd.epsp[i] = hyd.eps[i]
            hyd.epsm[i] = hyd.eps[i]
            hyd.velp[i] = hyd.vel[i]
            hyd.velm[i] = hyd.vel[i]


    elif(type=='minmod'):
        (hyd.rhop,hyd.rhom) = tvd_minmod_reconstruct(hyd.n,hyd.g,hyd.rho,hyd.x,hyd.xi)
        (hyd.epsp,hyd.epsm) = tvd_minmod_reconstruct(hyd.n,hyd.g,hyd.eps,hyd.x,hyd.xi)
        (hyd.velp,hyd.velm) = tvd_minmod_reconstruct(hyd.n,hyd.g,hyd.vel,hyd.x,hyd.xi)

    elif(type=='mc'):
        (hyd.rhop,hyd.rhom) = tvd_mc_reconstruct(hyd.n,hyd.g,hyd.rho,hyd.x,hyd.xi)
        (hyd.epsp,hyd.epsm) = tvd_mc_reconstruct(hyd.n,hyd.g,hyd.eps,hyd.x,hyd.xi)
        (hyd.velp,hyd.velm) = tvd_mc_reconstruct(hyd.n,hyd.g,hyd.vel,hyd.x,hyd.xi)

    else:
        print "reconstruction type not known; abort!"
        sys.exit()


    hyd.pressp = eos_press(hyd.rhop,hyd.epsp,gamma)
    hyd.pressm = eos_press(hyd.rhom,hyd.epsm,gamma)

    hyd.qp = prim2con(hyd.rhop,hyd.velp,hyd.epsp)
    hyd.qm = prim2con(hyd.rhom,hyd.velm,hyd.epsm)

    return hyd


############# equation of state
def eos_press(rho,eps,gamma):
    press = (gamma - 1.0) * rho * eps
    return press

def eos_cs2(rho,eps,gamma):
    prs = (gamma - 1.0) * rho *eps
    dpde = (gamma - 1.0) * rho
    dpdrho = (gamma - 1.0) * eps
    cs2 = dpdrho + dpde * prs/(rho+1.0e-30)**2
    if (cs2 < 0).any(): print 'rho =',rho,'\neps =',eps
    return cs2


############# time step calculation
def calc_dt(hyd,dtp):
    global i
    cs = sqrt(eos_cs2(hyd.rho,hyd.eps,gamma))
    dtnew = [(hyd.x[j+1]-hyd.x[j]) / max(abs(hyd.vel[j]+cs[j]), abs(hyd.vel[j]-cs[j]))
             for j in range(hyd.g,hyd.n-hyd.g)]
    dtnew = min(dtnew)
    dtnew = min(cfl*dtnew,1.05*dtp)
    return dtnew

############# HLLE solver
def hlle(hyd):
    # compute eigenvalues
    evl  = zeros((3,hyd.n))
    evr  = zeros((3,hyd.n))
    smin = zeros(hyd.n)
    smax = zeros(hyd.n)
    csp  = sqrt(eos_cs2(hyd.rhop,hyd.epsp,gamma))
    csm  = sqrt(eos_cs2(hyd.rhom,hyd.epsm,gamma))
    for i in range(1,hyd.n-2):
        evl[0,i] = hyd.velp[i] - csp[i]
        evl[1,i] = hyd.velp[i]
        evl[2,i] = hyd.velp[i] + csp[i]

        evr[0,i] = hyd.velm[i+1] - csm[i+1]
        evr[1,i] = hyd.velm[i+1]
        evr[2,i] = hyd.velm[i+1] + csm[i+1]

        smin[i] = min(concatenate((evl[:,i],evr[:,i])))
        smax[i] = max(concatenate((evl[:,i],evr[:,i])))

    # set up flux left L and right R of the interface
    # at i+1/2
    fluxl = zeros((3,hyd.n))
    fluxr = zeros((3,hyd.n))

    for i in range(1,hyd.n-2): # skip only 1 boundary cell
        # switched l and r here
        fluxl[0,i]=hyd.rhop[i] * hyd.velp[i]
        fluxl[1,i]=hyd.rhop[i] * hyd.velp[i]**2 + hyd.pressp[i]
        fluxl[2,i]=(hyd.rhop[i] * hyd.epsp[i] +
                    0.5*hyd.rhop[i] * hyd.velp[i]**2 +
                    hyd.pressp[i]) * hyd.velp[i]

        fluxr[0,i]=hyd.rhom[i+1] * hyd.velm[i+1]
        fluxr[1,i]=hyd.rhom[i+1] * hyd.velm[i+1]**2 + hyd.pressm[i+1]
        fluxr[2,i]=(hyd.rhom[i+1] * hyd.epsm[i+1] +
                    0.5*hyd.rhom[i+1] * hyd.velm[i+1]**2 +
                    hyd.pressm[i+1]) * hyd.velm[i+1]

    # solve the Riemann problem for the i+1/2 interface
    ds = smax - smin
    flux = zeros((3,hyd.n))
    for i in range(hyd.g-1,hyd.n-hyd.g+1):
        for j in range(3):
            flux[j,i] = (smax[i]*fluxl[j,i] - smin[i]*fluxr[j,i] +
                         smin[i]*smax[i] * (hyd.qm[j,i+1]-hyd.qp[j,i])) / (smax[i]-smin[i])

    # flux differences
    fluxdiff = zeros((3,hyd.n))
    for i in range(hyd.g,hyd.n-hyd.g):
        for j in range(3):
            fluxdiff[j,i] = (flux[j,i]-flux[j,i-1])/(hyd.xi[i]-hyd.xi[i-1])

    return fluxdiff

############# RHS calculation
def calc_rhs(hyd):
    # reconstruction and prim2con
    hyd = reconstruct(hyd,reconstruction_type)
    # compute flux differences
    fluxdiff = hlle(hyd)
    # return RHS = - fluxdiff
    return -fluxdiff

#########################################################################
# Main program
#########################################################################


hyd = mydata(nzones)     # initialize
hyd.setup_grid(0.0,1.0)  # set up grid
hyd.shocktube_setup()           # set up initial data
```

```
dt = calc_dt(hyd,dt)     # get initial timestep

# initial prim2con
hyd.q = prim2con(hyd.rho,hyd.vel,hyd.eps)

t = 0.0
i = 0

# display stuff
ion()
figure()
plot(hyd.x,hyd.rho,"r-")
show()

# main integration loop
while t < tend:

    if i % 10 == 0:
        # output
        print "%5d %15.6E %15.6E" % (i,t,dt)
        clf()
        plot(hyd.x,hyd.rho,"b")
        plot(hyd.x,hyd.press,'r')
        plot(hyd.x,hyd.vel,'g')
        ylim([0,1.1])
        xlim([0,1])
        draw()

    # calculate new timestep
    dt = calc_dt(hyd,dt)

    # save old state
    hydold = hyd
    qold = hyd.q

    # calc rhs
    k1 = calc_rhs(hyd)
    # calculate intermediate step
    hyd.q = qold + 1.0/2.0 * dt * k1
    # con2prim
    (hyd.rho,hyd.eps,hyd.press,hyd.vel) = con2prim(hyd.q)
    # boundaries
    hyd = apply_bcs(hyd)

    #calc rhs
    k2 = calc_rhs(hyd)
    #apply update
    hyd.q = qold + dt * 0.5 * (k1 + k2)
    # con2prim
    (hyd.rho,hyd.eps,hyd.press,hyd.vel) = con2prim(hyd.q)
    # apply bcs
    hyd = apply_bcs(hyd)

    # update time
    t = t + dt
    i = i + 1

# display final result
ioff()
legend(['density','pressure','velocity'])
savefig('hlle_'+reconstruction_type+'.pdf')
show()


# print results to file
f=open('hlle_'+reconstruction_type+'.dat','w')
f.write('  i          x            density        pressure        velocity\n\n')
for i in range(hyd.n):
    f.write('{4:4d} {0:15E} {1:15E} {2:15E} {3:15E}\n'.format(hyd.x[i],hyd.rho[i],hyd.press[i],hyd.vel[
f.close()
```

B 1D Spherical Finite Volume Hydro Code

```
#!/usr/bin/env python
import sys,math
from pylab import *
from bounds import *
from eos import *

# basic parameters
K1 = 1.2435e15 * (0.5e0**(4.0/3.0))
gamma1 = 1.28
gamma2 = 2.5
gammath = 1.5
rhonuc = 2.0e14

E1 = K1/(gamma1-1.e0)
E2 = (gamma1 - 1.e0)/(gamma2-1.e0)*E1*rhonuc**(gamma1-gamma2)
K2 = (gamma2 - 1.e0)*E2
E3 = (gamma2 - gamma1)/(gamma2-1.e0)*E1*rhonuc**(gamma1-1.e0)

G = 6.672e-8 # gravitational constant
rhomin=1e6

cfl = 0.5
dt = 1.0e-5
dtp = dt
reconstruction_type = 'pc' # minmod, mc, pc
# use nzones
nzones = 2000
```

```
# run until time 0.2
tend = 0.2

####################################### class definition
class mydata:
    def __init__(self,nzones):
        self.x      = zeros(nzones) # cell centers
        self.xi     = zeros(nzones) # cell LEFT interfaces
        self.rho    = zeros(nzones)
        self.rhop   = zeros(nzones)
        self.rhom   = zeros(nzones)
        self.vel    = zeros(nzones)
        self.velp   = zeros(nzones)
        self.velm   = zeros(nzones)
        self.eps    = zeros(nzones)
        self.epsp   = zeros(nzones)
        self.epsm   = zeros(nzones)
        self.press  = zeros(nzones)
        self.pressp = zeros(nzones)
        self.pressm = zeros(nzones)
        self.q      = zeros((3,nzones)) # conserved quantities
        self.qp     = zeros((3,nzones))
        self.qm     = zeros((3,nzones))
        self.n      = nzones
        self.g      = 3 # ghost cells at each of inner/outer edges of domain


    def setup_grid(self,xmin,xmax):
        dx = (xmax - xmin) / (self.n - self.g*2 - 1)
        xmin = xmin - self.g*dx
        xmax = xmax + self.g*dx
        for i in range(self.n):
            self.x[i] = xmin + (i)*dx        # cell centers
            self.xi[i] = self.x[i] - 0.5*dx  # cell LEFT interfaces


####################################### initial conditions
def setup_star(hyd):
    poly = loadtxt('poly.dat')
    prad = poly[:,0]
    prho = poly[:,1]
    nn = len(prho)

    for i in range(hyd.n):
        hyd.rho[i] = max(rhomin,linterp(hyd.x[i],nn,prho,prad))
        hyd.press[i] = K1 * hyd.rho[i]**gamma1
        hyd.eps[i] = hyd.press[i] / (gamma1-1.0) / hyd.rho[i]
        if(hyd.rho[i] <= rhomin):
            hyd.rho[i] = hyd.rho[i] / 5.0

    return hyd
```

```
def linterp(xx,n,f,x):
    i=0
    while(i<n and x[i] < xx):  i=i+1

    if(i==n):    ff = rhomin
    elif(i==0):  ff = (f[1]-f[0])/(x[1]-x[0]) * (xx - x[0]) + f[0]
    else:        ff = (f[i]-f[i-1])/(x[i]-x[i-1]) * (xx-x[i-1]) + f[i-1]

    return ff


##################################### some basic functions
def prim2con(rho,vel,eps):
    q = zeros((3,len(rho)))
    q[0,:] = rho[:]
    q[1,:] = rho[:] * vel[:]
    q[2,:] = rho[:] * eps[:] + 0.5 * rho[:] * vel[:]**2

    return q

def con2prim(q):
    rho = q[0,:]
    vel = q[1,:] / rho
    eps = q[2,:] / rho - 0.5*vel**2
    press,cs2 = hybrid_eos(rho,eps)

    return (rho,eps,press,vel)


############# minmod function
def minmod(a,b):
    if(a*b < 0):
        mm = 0.0
    elif(abs(a)<abs(b)):
        mm=a
    else:
        mm=b
    return mm

############# minmod function
def tvd_minmod_reconstruct(n,g,f,x,xi):
    fp = zeros(n)
    fm = zeros(n)
    for i in range(g-1,n-g+1):
        dx_up = x[i] - x[i-1]
        dx_down = x[i+1] - x[i]
        dx_m = x[i] - xi[i]
        dx_p = xi[i+1] - x[i]
        df_up = (f[i]-f[i-1]) / dx_up
        df_down = (f[i+1]-f[i]) / dx_down
        delta = minmod(df_up,df_down)
        fp[i] = f[i] + delta*dx_p
        fm[i] = f[i] - delta*dx_m

    return (fp,fm)


############# signum functions
def signum(x,y):
    if(y >= 0):
        return abs(x)
    else:
        return -abs(x)


############# mc reconstruction
def tvd_mc_reconstruct(n,g,f,x,xi):
    fp = zeros(n)
    fm = zeros(n)
    for i in range(g-1,n-g+1):
        dx_up = x[i] - x[i-1]
        dx_down = x[i+1] - x[i]
        dx_m = x[i] - xi[i]
        dx_p = xi[i+1] - x[i]
        df_up = (f[i]-f[i-1]) / dx_up
        df_down = (f[i+1]-f[i]) / dx_down
        if(df_up*df_down < 0):
            delta = 0.0
        else:
            delta = signum(min(2.0*abs(df_up),2.0*abs(df_down),\
                               0.5*(abs(df_up)+abs(df_down))),\
                               df_up + df_down)

        fp[i] = f[i] + delta*dx_p
        fm[i] = f[i] - delta*dx_m

    return (fp,fm)


############# reconstruction top level function
def reconstruct(hyd,type):
    if(type=='pc'):
        # piecewise constant reconstruction
        for i in range(hyd.g-1,hyd.n-hyd.g+1):
            hyd.rhop[i] = hyd.rho[i]
            hyd.rhom[i] = hyd.rho[i]
            hyd.epsp[i] = hyd.eps[i]
            hyd.epsm[i] = hyd.eps[i]
            hyd.velp[i] = hyd.vel[i]
            hyd.velm[i] = hyd.vel[i]
    elif(type=='minmod'):
        (hyd.rhop,hyd.rhom) = tvd_minmod_reconstruct(hyd.n,hyd.g,hyd.rho,hyd.x,hyd.xi)
        (hyd.epsp,hyd.epsm) = tvd_minmod_reconstruct(hyd.n,hyd.g,hyd.eps,hyd.x,hyd.xi)
        (hyd.velp,hyd.velm) = tvd_minmod_reconstruct(hyd.n,hyd.g,hyd.vel,hyd.x,hyd.xi)

    elif(type=='mc'):
        (hyd.rhop,hyd.rhom) = tvd_mc_reconstruct(hyd.n,hyd.g,hyd.rho,hyd.x,hyd.xi)
        (hyd.epsp,hyd.epsm) = tvd_mc_reconstruct(hyd.n,hyd.g,hyd.eps,hyd.x,hyd.xi)
        (hyd.velp,hyd.velm) = tvd_mc_reconstruct(hyd.n,hyd.g,hyd.vel,hyd.x,hyd.xi)

    else:
        print "reconstruction type not known; abort!"
        sys.exit()


    hyd.pressp,cs2p = hybrid_eos(hyd.rhop,hyd.epsp)
    hyd.pressm,cs2m = hybrid_eos(hyd.rhom,hyd.epsm)

    hyd.qp = prim2con(hyd.rhop,hyd.velp,hyd.epsp)
    hyd.qm = prim2con(hyd.rhom,hyd.velm,hyd.epsm)

    return hyd


############# time step calculation
def calc_dt(hyd,dtp):
    global i
    press,cs2 = hybrid_eos(hyd.rho,hyd.eps)
    cs = sqrt(cs2)
    dtnew = [(hyd.x[j+1]-hyd.x[j]) / max(abs(hyd.vel[j]+cs[j]), abs(hyd.vel[j]-cs[j]))
             for j in range(hyd.g,hyd.n-hyd.g)]
    dtnew = min(dtnew)
    dtnew = min(cfl*dtnew,1.05*dtp)
    return dtnew

############# HLLE solver
def hlle(hyd):
    # compute eigenvalues
    evl  = zeros((3,hyd.n))
    evr  = zeros((3,hyd.n))
    smin = zeros(hyd.n)
    smax = zeros(hyd.n)

    pressp,cs2p = hybrid_eos(hyd.rhop,hyd.epsm)
    pressm,cs2m = hybrid_eos(hyd.rhop,hyd.epsm)
    csp  = sqrt(cs2p)
    csm  = sqrt(cs2m)

    for i in range(1,hyd.n-2):
        evl[0,i] = hyd.velp[i] - csp[i]
        evl[1,i] = hyd.velp[i]
        evl[2,i] = hyd.velp[i] + csp[i]

        evr[0,i] = hyd.velm[i+1] - csm[i+1]
        evr[1,i] = hyd.velm[i+1]
        evr[2,i] = hyd.velm[i+1] + csm[i+1]

        smin[i] = min(concatenate((evl[:,i],evr[:,i])))
        smax[i] = max(concatenate((evl[:,i],evr[:,i])))

    # set up flux left L and right R of the interface
    # at i+1/2
    fluxl = zeros((3,hyd.n))
    fluxr = zeros((3,hyd.n))

    for i in range(1,hyd.n-2): # skip only 1 boundary cell
        # switched l and r here
        fluxl[0,i]=hyd.rhop[i] * hyd.velp[i]
        fluxl[1,i]=hyd.rhop[i] * hyd.velp[i]**2 + hyd.pressp[i]
        fluxl[2,i]=(hyd.rhop[i] * hyd.epsp[i] +
                    0.5*hyd.rhop[i] * hyd.velp[i]**2 +
                    hyd.pressp[i]) * hyd.velp[i]

        fluxr[0,i]=hyd.rhom[i+1] * hyd.velm[i+1]
        fluxr[1,i]=hyd.rhom[i+1] * hyd.velm[i+1]**2 + hyd.pressm[i+1]
        fluxr[2,i]=(hyd.rhom[i+1] * hyd.epsm[i+1] +
                    0.5*hyd.rhom[i+1] * hyd.velm[i+1]**2 +
                    hyd.pressm[i+1]) * hyd.velm[i+1]

    # solve the Riemann problem for the i+1/2 interface
    ds = smax - smin
    flux = zeros((3,hyd.n))
    for i in range(hyd.g-1,hyd.n-hyd.g+1):
        for j in range(3):
            flux[j,i] = (smax[i]*fluxl[j,i] - smin[i]*fluxr[j,i] +
                         smin[i]*smax[i] * (hyd.qm[j,i+1]-hyd.qp[j,i])) / (smax[i]-smin[i])

    # flux differences
    fluxdiff = zeros((3,hyd.n))
    for i in range(hyd.g,hyd.n-hyd.g):
        dr = hyd.xi[i+1]-hyd.xi[i]
        for j in range(3):
            fluxdiff[j,i] = (hyd.xi[i+1]**2*flux[j,i] - hyd.xi[i]**2*flux[j,i-1]) \
                / (dr*hyd.xi[i+1]**2)

    return fluxdiff


############# Mass calculation (required for RHS calcluation)
def calc_mass(hyd):
    mass = zeros(hyd.n)
    m1 = zeros(hyd.n)
    m1[hyd.g] = 4.0/3.0*pi * hyd.rho[0] * (hyd.xi[hyd.g+1]**3)
    mass[hyd.g] = 4.0/3.0*pi * hyd.rho[0] * (hyd.x[hyd.g])**3
    for i in range(hyd.g,hyd.n-1):
        m1[i] = 4.0/3.0*pi * hyd.rho[i] * (hyd.xi[i+1]**3 - hyd.xi[i]**3)
        dmi = 4.0/3.0*pi * (hyd.rho[i-1]*(hyd.xi[i]**3 - hyd.x[i-1]**3) +
                            hyd.rho[i]*(hyd.x[i]**3 - hyd.xi[i]**3))
        mass[i] = mass[i-1] + dmi

    return (mass,m1)


############# RHS calculation
def calc_rhs(hyd):
    # reconstruction and prim2con
    hyd = reconstruct(hyd,reconstruction_type)
    # compute flux differences
    fluxdiff = hlle(hyd)

    # compute non-conserved terms
    rhs = zeros((3,hyd.n))
    mass,m1 = calc_mass(hyd)
    for i in range(hyd.g,hyd.n-hyd.g):
        dr = hyd.xi[i+1]-hyd.xi[i]
        for j in range(3):
            rhs[j,i] = (-fluxdiff[j,i]
                        - hyd.rho[i]*G*(m1[i+1]-m1[i])/(hyd.xi[i+1]**2 * dr)
                        + 2*hyd.press[i]/hyd.xi[i+1])

    return rhs

########################################################################
# Main program
########################################################################


hyd = mydata(nzones)     # initialize
hyd.setup_grid(0.0,2e9)  # set up grid
setup_star(hyd)          # set up initial data
dt = calc_dt(hyd,dt)     # get initial timestep

# initial prim2con
hyd.q = prim2con(hyd.rho,hyd.vel,hyd.eps)

t = 0.0
i = 0

# display stuff
figure()
max_rho0 = max(hyd.rho)
```

```
max_press0 = max(hyd.press)
print 'initial maxima:  rho =',max_rho0,'pressure =',max_press0
plot(hyd.x,hyd.rho/max_rho0,'b')
plot(hyd.x,hyd.press/max_press0,'r')
xlim([0,2e9])
ylim([0,1.1])
show()
ion()
# main integration loop
while t < tend:

    if i % 10 == 0:
        # output
        print "%5d %15.6E %15.6E" % (i,t,dt)
        clf()
        plot(hyd.x,hyd.rho/max_rho0,'b')
        plot(hyd.x,hyd.press/max_press0,'r')
        plot(hyd.x,hyd.vel,'g')
        xlim([0,2e9])
        ylim([0,1.1])
        draw()

    # calculate new timestep
    dt = calc_dt(hyd,dt)

    # save old state
    hydold = hyd
    qold = hyd.q

    # calc rhs
    k1 = calc_rhs(hyd)
    # calculate intermediate step
    hyd.q = qold + 1.0/2.0 * dt * k1
    # con2prim
    (hyd.rho,hyd.eps,hyd.press,hyd.vel) = con2prim(hyd.q)
    # boundaries
    hyd = apply_bcs_spherical(hyd)

    #calc rhs
    k2 = calc_rhs(hyd)
    #apply update
    hyd.q = qold + dt * 0.5 * (k1 + k2)
    # con2prim
    (hyd.rho,hyd.eps,hyd.press,hyd.vel) = con2prim(hyd.q)
    # apply bcs
    hyd = apply_bcs_spherical(hyd)

    # update time
    t = t + dt
    i = i + 1
```

```
# display final result
ioff()
legend(['density','pressure','velocity'],loc='best')
savefig('hlle_'+reconstruction_type+'.pdf')
show()


# print results to file
f=open('hlle_'+reconstruction_type+'.dat','w')
f.write('  i           x            density        pressure        velocity\n\n')
for i in range(hyd.n):
    f.write('{4:4d} {0:15E} {1:15E} {2:15E} {3:15E}\n'.format(hyd.x[i],hyd.rho[i],hyd.press[i],hyd.vel[
f.close()
```