Isothermal gas dynamics in two space dimensions
Solving multidimensional problems with splitting methods

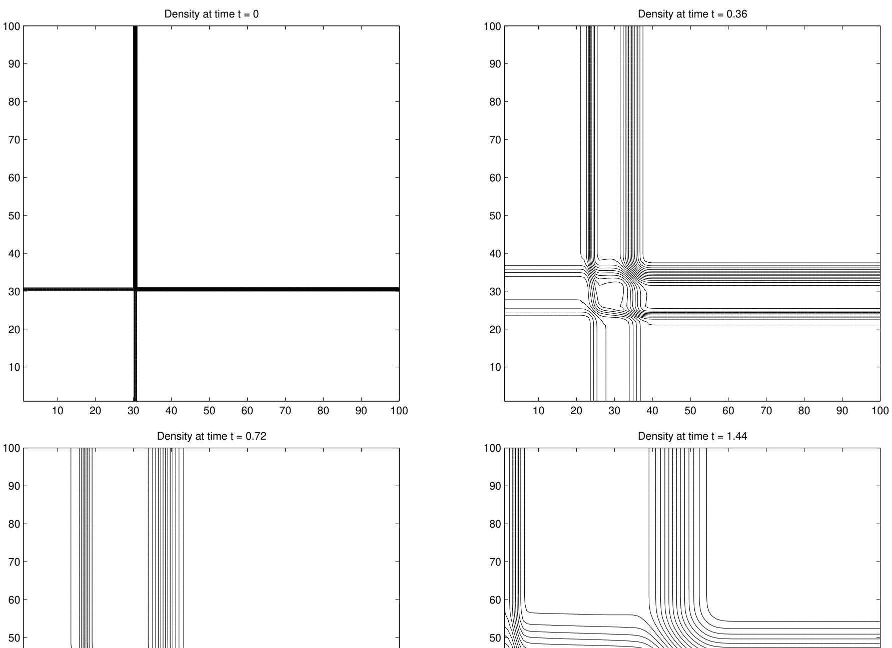
Matthias Köppe

10 July 1998

This is not a script but a collection of pictures and code; therefore you probably should rather keep attention to the talk.

1 Two-dimensional Riemann problem

Initial data is piecewise constant with four values in four quadrants. The velocities are all zero; density is least in the lower-left quadrant and greatest in the upper-right quadrant.

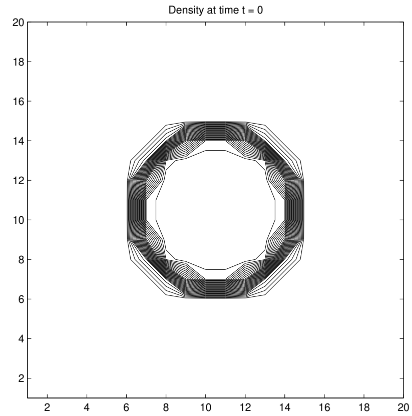


Only the density iso-curves are shown; the velocities do not give additional information.

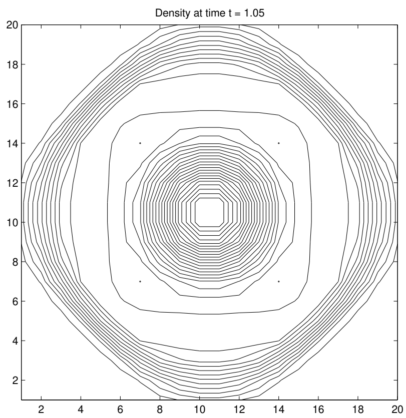
2 Transverse propagation and radial-symmetric data

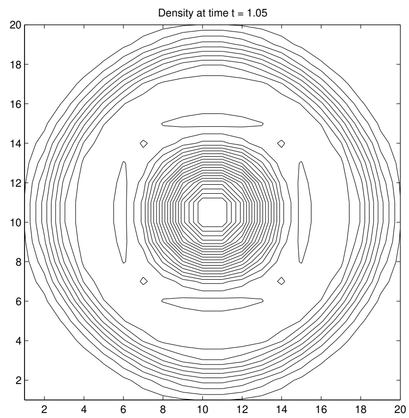
Given is a Riemann problem across a circular interface. The density inside the circle is greater than the density outside. The solution of this initial-value problem is an outwards-moving circular shock wave.

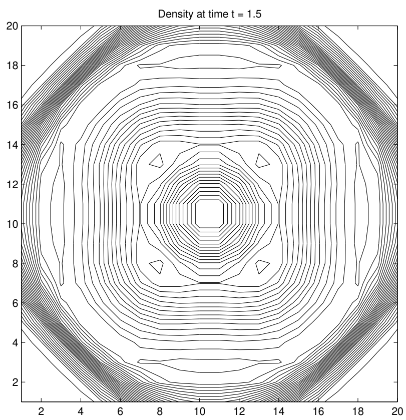
Initial conditions

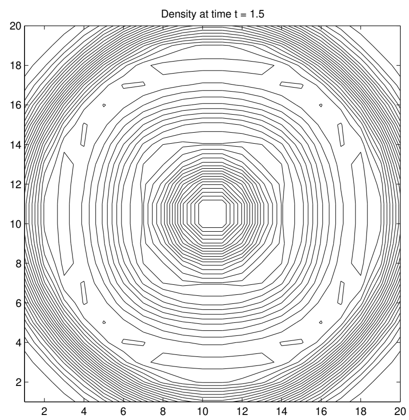


Without transverse propagation | With transverse propagation









Without transverse propagation, waves will be propagated only along the main directions of the grid. This will destroy symmetric properties of the numerical solution. – The solution calculated with transverse propagation still does not look perfectly symmetric, but this is just due to the very rough grid of 20×20 cells.

3 Driver routine

```
      program driver
c
c  Sample driver routine for claw2
c  This program solves the 2D isothermal gas equations
c
c  Derived from:
c
c  Euler solver
c  Author: Randall J. LeVeque
c  Version of November, 1995  --  CLAWPACK Version 2.1
c
c
c
      implicit double precision (a-h,o-z)
      external bc2ex0,rpn2iso,rpt2iso
      parameter (maxmx = 200)
      parameter (maxmy = 200)
c
c  Equation layout:
c  (1) density \rho, (2,3) velocity u, v
c
      parameter (meqn = 3)
c
c  We have three waves
c
      parameter (mwaves = 3)
      parameter (mbc = 2)
      parameter (mwork = 178092)
c FIXME: Check dimensions
      dimension     q(1-mbc:maxmx+mbc, 1-mbc:maxmy+mbc, meqn)
      dimension mthlim(mwaves),method(7),dtv(5),cflv(4),nv(2)
      dimension x(1-mbc:maxmx+mbc),y(1-mbc:maxmy+mbc)
      dimension work(mwork)

      common /param/ c

      open(10,file='fort.info',status='unknown',form='formatted')
      open(11,file='fort.nplot',status='unknown',form='formatted')
      open(99,file='fort.dump',status='unknown',form='formatted')
c
c     # check to see if we are restarting from a previous run:
c
      write(6,*) 'restarting from previous run?  0=no, 1=yes '
      read(5,*) irstrt
      if (irstrt .eq. 1) then
c         # restart from a previous run
c         # Note: this assumes data is in the file fort.dump
          call rstrt2(maxmx,maxmy,meqn,mwaves,mbc,mx,my,
     &       q,t,dxold,dyold,dtv,cflv,nv,method,mthlim,info)
c         # dt and method parameters may be reset by new data below.
    t0 = t
```

```
else
  t0 = 0.d0
endif
c
c     # Input parameters for this run:
c
      write(6,*) 'input dx and dy.  (grid spacing)'
      read(5,*) dx,dy
      if (irstrt.eq.1) then
 if (dx.ne.dxold .or. dy.ne.dyold) then
    write(6,*) 'dx or dy set wrong for restart'
    stop
    endif
 endif
c
      write(6,*) 'input dt and tend.  (time step and final time)'
      read(5,*) dt,tend
      write(6,*) 'input nout.  (number of solution outputs)'
      read(5,*) nout
      write(6,*) 'input method(1:3)'
      read(5,*) method(1), method(2), method(3)
      write(6,*) 'input mthlim(mw), mw=1,mwaves.'
      read(5,*) (mthlim(mw), mw=1,mwaves)
c
c     Sound speed
c
c     c = 1.3d-2
      write(6,*) 'input constant sound speed'
      read(5,*) c
c
c     Talk to user
c
      write(6,*) 'running...'
      write(6,*) ' '
c
c
c     # time increments between outputing solution:
      dtout = (tend - t0)/dfloat(nout)
c
      cflv(1) = 1.0d0  !# maximum Courant number allowed
      cflv(2) = 0.9d0  !# desired Courant number
      dtv(1) = dt   !# initial time step
      dtv(2) = 1d6   !# maximum allowed time step
      nv(1) = 500   !# maximum number of time steps to allow
      method(4) = 1 !# print dt, cfl in each step
      method(5) = 0 !# no source term
      method(6) = 0 !# no capacity function
      method(7) = 0 !# maux
c
c     # set up grid
c
c     # domain is unit square:
      ax = 0.d0
```

```
      bx = 1.d0
      ay = 0.d0
      by = 1.d0
      mx = (bx-ax + 1d-10)/dx
      my = (by-ay + 1d-10)/dy
c
      do 10 i=1,mx
 x(i) = ax + (i-0.5d0) * dx
   10    continue
      do 15 j=1,my
 y(j) = ay + (j-0.5d0) * dy
   15    continue
c
c     # set initial conditions, unless we are restarting:
c
      if (irstrt .eq. 0)  then
c        # 2D Riemann problem
c        # initial conditions are piecewise constant in 4 quadrants
c        # (or other conditions; chosen by linked-to module)
 call ic2rp(maxmx,maxmy,meqn,mbc,mx,my,x,y,dx,dy,q)
 endif
c
c     # output initial data to unit 100
      call out2iso(maxmx,maxmy,meqn,mbc,mx,my,x,y,q,tstart,100)
c
c     ----------
c     Main loop:
c     ----------
c
      do 100 n=1,nout
 tstart = t0 + (n-1) * dtout
 tend = t0 + n*dtout
c
         call claw2(maxmx,maxmy,meqn,mwaves,mbc,mx,my,
     &  q,dummy,dx,dy,tstart,tend,dtv,
     &           cflv,nv,method,mthlim,
     &           work,mwork,info,bc2ex0,rpn2iso,rpt2iso,src2xx)
c
         dtv(1) = dtv(5)  !# use final dt as starting value on next call
c
c        # output solution at this time
         call out2iso(maxmx,maxmy,meqn,mbc,mx,my,x,y,q,tend,100+n)
c
c        # write out information about this call to claw:
c
 write(6,601) nv(2),tend,info
  601    format('CLAW2 returning after ',i3,' steps,  t =',e10.5,
     &        '  info =',i3,/)
c
 write(10,1010) tend,info,dtv(3),dtv(4),dtv(5),
     &  cflv(3),cflv(4),nv(2)
 1010    format('tend =',d15.4,/,
     &       'info =',i5,/,'smallest dt =',d15.4,/,'largest dt =',
```

```
     &       d15.4,/,'last dt =',d15.4,/,'largest cfl =',
     &         d15.4,/,'last cfl =',d15.4,/,'steps taken =',i4,/)
c
c        # check to see if an error occured:
 if (info .ne. 0) go to 200
  100    continue
c
  200 continue
c     # dump information needed for a possible restart:
      call dump2(maxmx,maxmy,meqn,mwaves,mbc,mx,my,
     &           q,tend,dx,dy,dtv,cflv,nv,method,mthlim,info)
      write(11,1101) nout
 1101 format(i5)
      stop
      end
```

4 Normal Roe–Riemann solver (rpn2iso.f)

```
c
c
c     =====================================================
      subroutine rpn2iso(ixy,maxm,meqn,mwaves,mbc,mx,ql,qr,auxl,auxr,
     &                  wave,s,amdq,apdq)
c     =====================================================
c
c     # Roe-solver for isothermal gas flow
c     # solve Riemann problems along one slice of data.
c
c     # On input, ql contains the state vector at the left edge of each cell
c     #           qr contains the state vector at the right edge of each cell
c
c     # This data is along a slice in the x-direction if ixy=1
c     #                            or the y-direction if ixy=2.
c     # On output, wave contains the waves, s the speeds,
c     # and amdq, apdq the decomposition of the flux difference
c     #   f(qr(i-1)) - f(ql(i))
c     # into leftgoing and rightgoing parts respectively.
c     # With the Roe solver we have
c     #    amdq  =  A^- \Delta q    and    apdq  =  A^+ \Delta q
c     # where A is the Roe matrix.  An entropy fix can also be incorporated
c     # into the flux differences.
c
c     # Note that the i'th Riemann problem has left state qr(i-1,:)
c     #                                    and right state ql(i,:)
c     # From the basic clawpack routines, this routine is called with ql = qr
c
c
      implicit double precision (a-h,o-z)
c
      dimension wave(1-mbc:maxm+mbc, meqn, mwaves)
      dimension    s(1-mbc:maxm+mbc, mwaves)
      dimension   ql(1-mbc:maxm+mbc, meqn)
```

```
      dimension   qr(1-mbc:maxm+mbc, meqn)
      dimension  apdq(1-mbc:maxm+mbc, meqn)
      dimension  amdq(1-mbc:maxm+mbc, meqn)
c
c     local arrays -- common block comroe is passed to rpt2iso
c     ------------
      parameter (maxm2 = 202)  !# assumes at most 200x200 grid with mbc=2
      dimension delta(3)
      logical efix
      common /param/  c  !# the constant sound speed
      common /comroe/ u(-1:maxm2),v(-1:maxm2)
c
      data efix /.false./    !# no entropy fix for transonic rarefactions
c
      if (-1.gt.1-mbc .or. maxm2 .lt. maxm+mbc) then
         write(6,*) 'need to increase maxm2 in rpA'
         stop
         endif
c
c     # set mu to point to  the component of the system that corresponds
c     # to velocity in the direction of this slice, mv to the orthogonal
c     # velocity:
c
      if (ixy.eq.1) then
          mu = 2
          mv = 3
        else
          mu = 3
          mv = 2
        endif
c
c     # note that notation for u and v reflects assumption that the
c     # Riemann problems are in the x-direction with u in the normal
c     # direction and v in the orthogonal direction, but with the above
c     # definitions of mu and mv the routine also works with ixy=2
c     # and returns, for example, f0 as the Godunov flux g0 for the
c     # Riemann problems u_t + g(u)_y = 0 in the y-direction.
c
c
c     # compute the Roe-averaged variables needed in the Roe solver.
c     # These are stored in the common block comroe since they are
c     # later used in routine rpt2iso to do the transverse wave splitting.
c
      do 10 i = 2-mbc, mx+mbc
c     #  \sqrt\rho LEFT/RIGHT
         rhsqrtl = dsqrt(qr(i-1,1))
         rhsqrtr = dsqrt(ql(i,1))
c     #  divisor
         rhsq2 = rhsqrtl + rhsqrtr
c     #  averaged states
         u(i) = (qr(i-1,mu)*rhsqrtl + ql(i,mu)*rhsqrtr) / rhsq2
         v(i) = (qr(i-1,mv)*rhsqrtl + ql(i,mv)*rhsqrtr) / rhsq2
   10    continue
```

```
c
c
c     # now split the jump in q at each interface into waves
c
c     # find a1 thru a3 (\alpha_i^1, \dots, \alpha_i^3, the `wave strengths'),
c     # the coefficients of the 3 eigenvectors:
c
      do 20 i = 2-mbc, mx+mbc
c     #  jumps
         delta(1) = ql(i,1) - qr(i-1,1)
         delta(2) = ql(i,mu) - qr(i-1,mu)
         delta(3) = ql(i,mv) - qr(i-1,mv)
c     #  wave strengths, \alpha_i^k = R^{-1) delta(k)
c     #  c is the constant sound speed.
         a1 = ((u(i) + c) * delta(1) - delta(2)) / (2.d0 * c)
         a2 = (-2.d0 * c * v(i) * delta(1) + 2.d0 * c * delta(3))
     &        / (2.d0 * c)
         a3 = ((-u(i) + c) * delta(1) + delta(2)) / (2.d0 * c)
c
c     # Compute the waves.
c     # wave 1
c
         wave(i,1,1) = a1
         wave(i,mu,1) = a1*(u(i)-c)
         wave(i,mv,1) = a1*v(i)
         s(i,1) = u(i) - c
c
c     # wave 2
c
         wave(i,1,2) = 0
         wave(i,mu,2) = 0
         wave(i,mv,2) = a2
         s(i,2) = u(i)
c
c     # wave 3
c
         wave(i,1,3) = a3
         wave(i,mu,3) = a3*(u(i)+c)
         wave(i,mv,3) = a3*v(i)
         s(i,3) = u(i) + c
 20   continue
c
c     # no entropy fix
c     ----------------
c
c     # amdq = SUM s*wave   over left-going waves
c     # apdq = SUM s*wave   over right-going waves
c
      do 100 m=1,3
         do 100 i=2-mbc, mx+mbc
            amdq(i,m) = 0.d0
            apdq(i,m) = 0.d0
            do 90 mw=1,mwaves
```

```
            if (s(i,mw) .lt. 0.d0) then
                amdq(i,m) = amdq(i,m) + s(i,mw)*wave(i,m,mw)
              else
                apdq(i,m) = apdq(i,m) + s(i,mw)*wave(i,m,mw)
              endif
   90       continue
  100    continue
      go to 900

  900 continue
      return
      end
```

5 Transverse Roe–Riemann solver (rpt2iso.f)

```
c
c
c     =====================================================
      subroutine rpt2iso(ixy,maxm,meqn,mwaves,mbc,mx,
     &                  ql,qr,aux1,aux2,aux3,
     &                  ilr,asdq,bmasdq,bpasdq)
c     =====================================================
      implicit double precision (a-h,o-z)
c
c     # Riemann solver in the transverse direction for the Euler equations.
c     # Split asdq (= A^* \Delta q, where * = + or -)
c     # into down-going flux difference bmasdq (= B^- A^* \Delta q)
c     #    and up-going flux difference bpasdq (= B^+ A^* \Delta q)

c
c     # Uses Roe averages and other quantities which were
c     # computed in rpn2iso and stored in the common block comroe.
c
      dimension     ql(1-mbc:maxm+mbc, meqn)
      dimension     qr(1-mbc:maxm+mbc, meqn)
      dimension   asdq(1-mbc:maxm+mbc, meqn)
      dimension bmasdq(1-mbc:maxm+mbc, meqn)
      dimension bpasdq(1-mbc:maxm+mbc, meqn)
c
      common /param/  c  !# sound speed
      dimension waveb(3,3),sb(3)
      parameter (maxm2 = 202)  !# assumes at most 200x200 grid with mbc=2

      common /comroe/ u(-1:maxm2),v(-1:maxm2)


c
      if (-1.gt.1-mbc .or. maxm2 .lt. maxm+mbc) then
         write(6,*) 'need to increase maxm2 in rpB'
         stop
         endif
c
      if (ixy.eq.1) then
          mu = 2
```

```
          mv = 3
        else
          mu = 3
          mv = 2
        endif
c
         do 20 i = 2-mbc, mx+mbc
c
c     #  Compute a1--a3, that is \beta^1,\ldots,\beta^3
c
            a1 = ((v(i) + c) * asdq(i,1) - asdq(i,mv)) / (2.d0 * c)
            a2 = (-2.d0 * c * u(i) * asdq(i,1) + 2.d0 * c * asdq(i,mu))
     $           / (2.d0 * c)
            a3 = ((-v(i) + c) * asdq(i, 1) + asdq(i, mv)) / (2.d0 * c)
c
            waveb(1,1) = a1
            waveb(mu,1) = a1*u(i)
            waveb(mv,1) = a1*(v(i)-c)
            sb(1) = v(i) - c
c
            waveb(1,2) = 0
            waveb(mu,2) = a2
            waveb(mv,2) = 0
            sb(2) = v(i)
c
            waveb(1,3) = a3
            waveb(mu,3) = a3*u(i)
            waveb(mv,3) = a3*(v(i)+c)
            sb(3) = v(i) + c
c
c           # compute the flux differences bmasdq and bpasdq
c
            do 10 m=1,meqn
               bmasdq(i,m) = 0.d0
               bpasdq(i,m) = 0.d0
               do 10 mw=1,mwaves
                  bmasdq(i,m) = bmasdq(i,m)
     &                         + dmin1(sb(mw), 0.d0) * waveb(m,mw)
                  bpasdq(i,m) = bpasdq(i,m)
     &                         + dmax1(sb(mw), 0.d0) * waveb(m,mw)
 10            continue
c
 20         continue
c
      return
      end
```

6 Radial initial conditions (ic2rpc.f)

```
c
c
c
```

```
c     =====================================================
      subroutine ic2rp(maxmx,maxmy,meqn,mbc,mx,my,x,y,dx,dy,q)
c     =====================================================
c
c     # Set initial conditions for q.
c     # Riemann problem across a circular interface centered at (x0,y0)
c     # with radius r0.  The state inside is specified by qin and
c     # outside by qout.  The cellave routine is used to determine what
c     # fraction of the cell is in each state and set the value of q
c     # to be the appropriate average of qin and qout.
c     #
c
      implicit double precision (a-h,o-z)
      dimension q(1-mbc:maxmx+mbc, 1-mbc:maxmy+mbc, meqn)
      dimension x(1-mbc:maxmx+mbc),y(1-mbc:maxmy+mbc)
      dimension qin(4),qout(4)
      common/cdisc/ x0,y0,alf,beta,r0,idisc
c
c
c     # set idisc for cellave routines (see function fdisc)
      idisc = 2
c
      write(6,*) 'input x0, y0, and r0 for radial Riemann problem'
      read(5,*) x0,y0,r0
      write(6,*) 'input of rho inside'
      read(5,*) rhoin
      write(6,*) 'input of rho outside'
      read(5,*) rhoout
c
      qin(1) = rhoin
      qin(2) = 0.d0
      qin(3) = 0.d0
c
      qout(1) = rhoout
      qout(2) = 0.d0
      qout(3) = 0.d0
c
c
      do 21 i=1,mx
         do 20 j=1,my
c     # set (xlow,ylow) to lower left corner of grid cell:
            xlow = x(i) - 0.5d0*dx
            ylow = y(j) - 0.5d0*dy
            call cellave(xlow,ylow,dx,dy,win)
c     # win is now the fraction of the cell that lies inside the circle
            do 10 m=1,meqn
               q(i,j,m) = win*qin(m) + (1.d0-win)*qout(m)
 10         continue
 20      continue
 21   continue
      return
      end
```

7 Data files

This is the data file for the radial-symmetric sample. Others look similar.

```
0                             irstrt -*- Fundamental -*-
.05d0   .05d0                 dx, dy
.01d0  1.5d0                  dt, tend
10                            nout
1 1 1                         method(1:3), Var time steps, 1st order, w/ transverse prop
2 2 2                         mthlim(1:3)
1.5d-1                        constant sound speed
0.5d0 0.5d0 0.2d0             x0, y0, r0 (radial ic)
2.d0                          rhoin
0.4d0                         rhoout
```

8 Makefile

This makefile shows the linked-in FORTRAN modules for both types of initial data.

```
### Makefile

all: radial

quadrants: isothermal-quadrants
./isothermal-quadrants < data

radial-no-transverse: isothermal-radial
./isothermal-radial < data-radial-no-transverse

radial: isothermal-radial
./isothermal-radial < data-radial

ISOTHERMAL_F_SOURCES = \
driver.f \
$(CLAWPACK)/2d/claw2/claw2.f \
$(CLAWPACK)/2d/claw2/step2.f \
$(CLAWPACK)/2d/claw2/flux2.f \
$(CLAWPACK)/2d/claw2/dimsp2.f  \
$(CLAWPACK)/2d/claw2/step2ds.f \
$(CLAWPACK)/2d/claw2/copyq2.f \
$(CLAWPACK)/limiters/limiter.f \
$(CLAWPACK)/limiters/philim.f \
../rp/rpn2iso.f \
../rp/rpt2iso.f \
$(CLAWPACK)/2d/bc/bc2ex0.f \
../out2iso.f \
$(CLAWPACK)/2d/misc/dump2.f \
$(CLAWPACK)/2d/misc/rstrt2.f

ISOTHERMAL_QUADRANTS_F_SOURCES = $(ISOTHERMAL_F_SOURCES) \
ic2rp2.f

isothermal-quadrants: $(ISOTHERMAL_QUADRANTS_F_SOURCES)
```

```
f77 $(ISOTHERMAL_QUADRANTS_F_SOURCES) -o isothermal-quadrants

ISOTHERMAL_RADIAL_F_SOURCES = $(ISOTHERMAL_F_SOURCES) \
ic2rpc.f \
$(CLAW)/applications/euler/2d/radial/fdisc.f \
$(CLAWPACK)/2d/misc/cellave.f

isothermal-radial: $(ISOTHERMAL_RADIAL_F_SOURCES)
f77 $(ISOTHERMAL_RADIAL_F_SOURCES) -o isothermal-radial

clean:
-rm *~ *.o isothermal isothermal-radial isothermal-quadrants fort.*
```