FYS-GEO 4500

Lecture Notes #11
Capacity Functions, Source Terms, and Other Unfinished Business…

Where we are today

	week number	date	Topic	Chapter in LeVeque
1	35	30. Aug 2010	introduction to conservation laws, Clawpack	1 & 2
2	36	6. Sep 2010	the Riemann problem, characteristics	3 & 5
3	37	13. Sep 2010	finite volume methods for linear systems	4
4	38	20. Sep 2010	high resolution methods	6
5	*40*	*4. Oct 2010*	boundary conditions, accuracy, variable coeff.	7,8, part 9
6	*40*	*5. Oct 2010*	nonlinear conservation laws, finite volume methods	11 & 12
7	41	11. Oct 2010	nonlinear equations & systems	13 & 14
8	42	18. Oct 2010	finite volume methods for nonlinear systems	14 & 15
9	43	25. Oct 2010	source terms and multidimensions	16,17,18,19
10	44	1. Nov 2010	multidimensional systems	20 & 21
11	45	8. Nov 2010	capacity functions, source terms, project plans	
12	46	15. Nov 2010	student presentations	
13	47	22. Nov 2010	student presentations	
14	48	6. Dec 2010	FINAL REPORTS DUE	

More general conservation laws …

A more general conservation law, with both a capacity function and a source term, can be written in 2D as:

$$\kappa q_t + f(q)_x + g(q)_y = \psi .$$

Here $\kappa = \kappa(x,y,t)$ is the capacity function and $\psi = \psi\left(q,x,y,t\right)$ is the source function. By transformation of variables, one can transform an equation having a capacity function into one having a source term, or *vice versa*. Leveque discusses this in Section 9.0.

Capacity form differencing

See Leveque 6.16 for a fuller discussion. We start with the (1D) equation

$$\kappa q_t + f(q)_x = 0,$$

and come up with a flux-update method in the form

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\kappa_i \Delta x}\left(\mathcal{A}^+ \Delta Q_{i-1/2} + \mathcal{A}^- \Delta Q_{i+1/2}\right) - \frac{\Delta t}{\kappa_i \Delta x}\left(\widetilde{F}^n_{i+1/2} - \widetilde{F}^n_{i-1/2}\right)$$

where the correction flux is

$$\widetilde{F}^n_{i-1/2} = \frac{1}{2}\sum_{p=1}^{m}\left(1 - \frac{\Delta t}{\kappa_{i-1/2}\Delta x}\left|s^p_{i-1/2}\right|\right)\left|s^p_{i-1/2}\right|\widetilde{\mathcal{W}}^p_{i-1/2}$$

using an appropriate limiter. Because the flux is defined at the cell interface, we should define an appropriate average of the capacity function to use at that point, for example,

$$\kappa_{i-1/2} = \frac{\kappa_{i-1} + \kappa_i}{2}.$$

Capacity functions in Clawpack:

Comments in claw1.f (*1-D; see claw2.f for 2-D*):

```
c  Solves a hyperbolic system of conservation laws in one space dimension
c  of the general form
c
c     capa * q_t + A q_x = psi
c
c  The "capacity function" capa(x) and source term psi are optional
c  (see below).

...

c        If method(6) = 0 then there is no capacity function.
c        If method(6) = mcapa > 0  then there is a capacity function and
c            capa(i), the "capacity" of the i'th cell, is assumed to be
c            stored in aux(i,mcapa).
c            In this case we require method(7).ge.mcapa.
```

Code in step1.f :

```
      mcapa = method(6)
      do 10 i=1-mbc,mx+mbc
         if (mcapa.gt.0) then
             if (aux(i,mcapa) .le. 0.d0) then
                write(6,*) 'Error -- capa must be positive'
                stop
                endif
             dtdx(i) = dt / (dx*aux(i,mcapa))
            else
             dtdx(i) = dt/dx
            endif
   10    continue
```

An example of how it’s done:

In claw3/book/chap23/acoustics/claw2ez.data:

```
7              method(6)   = mcapa
9              method(7)   = maux (should agree with parameter in driver)
```

In claw3/book/chap23/acoustics/driver.f

```
     parameter (maux = 9)
```

In claw3/book/chap23/acoustics/setaux.f:

```
c           # compute area of physical cell from four corners:

            xpcorn(5) = xpcorn(1)
            ypcorn(5) = ypcorn(1)
            area = 0.d0
            do ic=1,4
              area = area + 0.5d0 * (ypcorn(ic)+ypcorn(ic+1)) *
     &                     (xpcorn(ic+1)-xpcorn(ic))
              enddo
            aux(i,j,7) = area / (dxc*dyc)
```

An example of how it’s done:

In claw3/book/chap23/acoustics/claw2ez.data:

```
7              method(6)    = mcapa
9              method(7)    = maux (should agree with parameter in driver)
```

In claw3/book/chap23/acoustics/driver.f

```
     parameter (maux = 9)
```

In claw3/book/chap23/acoustics/setaux.f:

```
c           # compute area of physical cell from four corners:

            xpcorn(5) = xpcorn(1)
            ypcorn(5) = ypcorn(1)
            area = 0.d0
            do ic=1,4
              area = area + 0.5d0 * (ypcorn(ic)+ypcorn(ic+1)) *
     &                 (xpcorn(ic+1)-xpcorn(ic))
              enddo
            aux(i,j,7) = area / (dxc*dyc)
```

An example of how it’s done:

In claw3/book/chap23/acoustics/claw2ez.data:

```
7              method(6)    = mcapa
9              method(7)    = maux (should agree with parameter in driver)
```

In claw3/book/chap23/acoustics/driver.f

```
     parameter (maux = 9)
```

In claw3/book/chap23/acoustics/setaux.f:

```
c           # compute area of physical cell from four corners:

            xpcorn(5) = xpcorn(1)
            ypcorn(5) = ypcorn(1)
            area = 0.d0
            do ic=1,4
              area = area + 0.5d0 * (ypcorn(ic)+ypcorn(ic+1)) *
     &                   (xpcorn(ic+1)-xpcorn(ic))
              enddo
            aux(i,j,7) = area / (dxc*dyc)
```

From the clawpack_4.3_userguide.pdf

`auxtype(1:maux)`: If `maux` > 0 then for each component of the auxiliary array, a type must be specified from the following list, depending on what the corresponding component of aux represents:

`"xleft"`	a value associated with the “left” edge of the cell in the x-direction,
`"yleft"`	a value associated with the “left” edge of the cell in the y-direction,
`"zleft"`	a value associated with the “left” edge of the cell in the z -direction ("`zleft`" is an option only in `amr3ez.data`),
`"center"`	a cell-centered value,
`"capacity"`	a cell-centered capacity function.

The `auxtype` array is required for *adaptive refinement* because auxiliary arrays must be handled slightly differently at refinement boundaries depending on how these values are used.

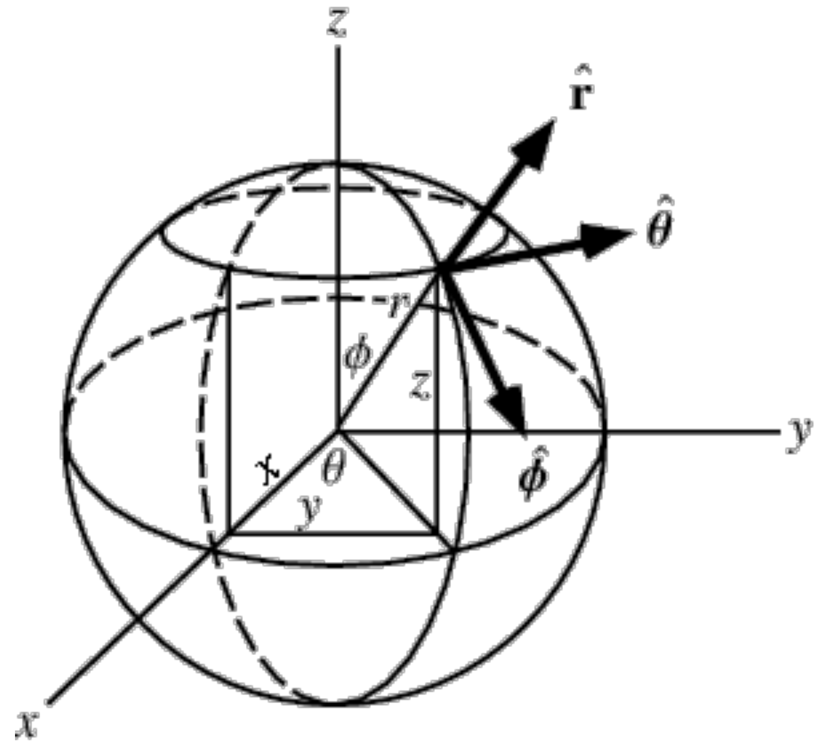
A cell-centered auxiliary value such as the density or impedance in a heterogeneous acoustics problem would have type "`center`". On the other hand, in a variable-coefficient advection problem we may want to store the normal velocity at each edge of the cell. In two dimensions we might use one component of `aux` to store the value $u_{i-1/2,j}$ at the left edge, which would have type "`xleft`", and another component of `aux` to store the value $v_{i-1/2,j}$ at the bottom edge, which would have type "`yleft`".

...

At most one component may have the type "`capacity`" and the value of `mcapa` should be set in a consistent manner. This component is used as a capacity function in capacity-form differencing.

Geometrical source terms

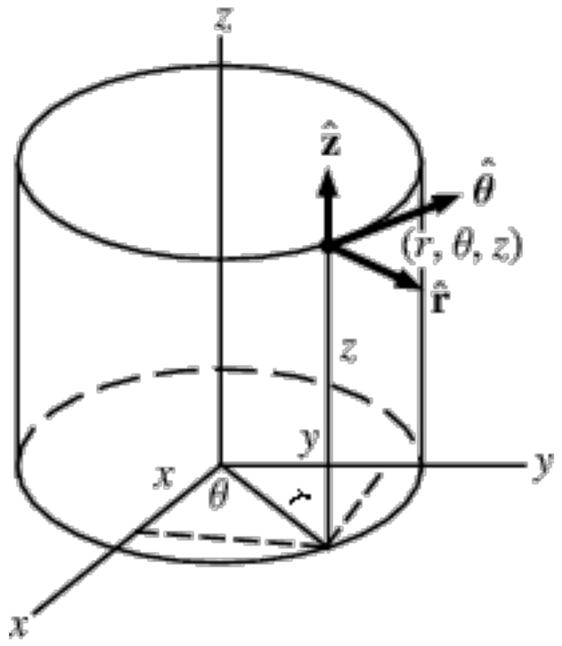
Geometrical source terms are necessary when:



Performing calculations with spherical symmetry

1-D: independent variable r

no variation or motion in θ or ϕ.



Performing calculations with cylindrical symmetry

1-D: independent variable r;

no variation or motion in θ or z.

2-D: independent variables r, z;

no variation or motion in θ.

Euler equations in 1-D radial coordinates:

From Leveque (18.9):

$$\rho_t + (\rho u)_r = -\frac{\alpha}{r}(\rho u)$$

$$(\rho u)_t + \left(\rho u^2 + p\right)_r = -\frac{\alpha}{r}\left(\rho u^2\right)$$

$$E_t + \left((E+p)u\right)_r = -\frac{\alpha}{r}\left((E+p)u\right)$$

where u is the velocity in the radial (r) direction. This system can be used for either cylindrical or spherical symmetry depending on α. In the cylindrical case, α = 1, and in the spherical case α = 2. This is a more precise statement than Leveque makes.

Note that you have to think very carefully about how to define the radial boundary at r = 0. Many codes do this incorrectly! A hint: It pays to be clear about the conservation laws and the fact that you are using a finite volume approach.

You should develop the 2-D axisymmetric form of the Euler equations and implement the system in a Riemann solver for Clawpack

The De Laval nozzle (*Pål*) and the venting problem (*Haakon*) would best be done this way; it would be desirable to do the Sedov problem (*Kristen*) this way also, so that we can generalise it to the case of an explosion near, or eventually in, the ground. The equation of state (*Elvira*) work should also be designed with a Riemann solver for this 2-D axisymmetric case. This could all come together as an important contribution for GeoClaw.

Work together on this! Use all the resources you can find from the internet or books or anywhere.

Mauro's pockmark work cannot be done in the axisymmetric system, however, because even if the pockmark itself is symmetric, the flow will not be.

Alternatively, you could work in a curvilinear system (chapter 23; see also http://kingkong.amath.washington.edu/trac/clawpack/browser/trunk/src/rplib/euler/rpn2euq3.f). This is much more difficult…

To obtain the Euler equations in other coordinate systems it is best to start with the vector equations:

$$\frac{\partial \rho}{\partial t} + \nabla \bullet \left(\rho \vec{u} \right) = 0$$

$$\frac{\partial \rho \vec{u}}{\partial t} + \nabla \bullet \left(\rho \vec{u} \vec{u} \right) + \nabla p = \rho \vec{g}$$

$$\frac{\partial E}{\partial t} + \nabla \bullet \left(\vec{u}(E + p) \right) = -\rho |g| z$$

In this set, the acceleration due to gravity, g, is a vector directed in the negative z direction. This is included as a source term. When these equations are cast into non-Cartesian coordinate systems, the geometrical source terms will also appear on the right-hand sides.

Both geometrical source terms and gravity can be calculated in the same `src2.f` routine.

Another highly useful form of the Euler equations:

$$\frac{\partial \rho}{\partial t} + \nabla \bullet \left(\rho \vec{u} \right) = 0$$

$$\frac{\partial \rho \vec{u}}{\partial t} + \nabla \bullet \left(\rho \vec{u}\vec{u} + \overleftrightarrow{\sigma} \right) = \rho \vec{g}$$

$$\frac{\partial E}{\partial t} + \nabla \bullet \left(E \vec{u} \right) + \nabla \bullet \left(\overleftrightarrow{\sigma} \vec{u} \right) = -\rho \left| g \right| z$$

This version substitutes the stress tensor σ for the pressure, and can be used for materials which support shear stress. The isotropic pressure is one-third the trace of the stress tensor, so for fluids that do not support shear stress, the usual form is recovered.

In this case you need to supplement with a strength model to calculate the stress tensor from the strains that are produced. Leveque treats part of this problem (the elastic equations in a linear elastic medium) in Chapter 22.

The problems you've chosen won't need this form, but it might be worth keeping in mind for future use.

Source terms in Clawpack

Comments in claw1.f (*1-D; see claw2.f for 2-D*):

```
c  In addition, if the equation contains source terms psi, then the user
c  must provide:
c
c    src1               subroutine that solves capa * q_t = psi
c                       over a single time step.
c
c  These routines must be declared EXTERNAL in the main program.
c  For description of the calling sequences, see below.
c
c  Dummy routines b4step1.f and src1.f are available in
c       claw/clawpack/1d/lib
...
c
c         method(5) = 0 if there is no source term psi.  In this case
c                       the subroutine src1 is never called so a dummy
c                       parameter can be given.
c                   = 1 if there is a source term.  In this case
c                       the subroutine src1 must be provided and a
c                       fractional step method is used.
c                       In each time step the following sequence is followed:
c                            call bc to extend data to ghost cells
c                            call step1 to advance hyperbolic eqn by dt
c                            call src1 to advance source terms by dt
c                   = 2 if there is a source term and Strang splitting is to
c                       be used instead of the Godunov splitting above.
c                       In each time step the following sequence is followed:
c                            call bc to extend data to ghost cells
c                            call src1 to advance source terms by dt/2
c                            call step1 to advance hyperbolic equation by dt
c                            call src1 to advance source terms by dt/2
c                       For most problems 1 is recommended rather than 2
c                       since it is less expensive and works essentially as
c                       well on most problems.
```

Source terms in Clawpack

Continued comments in claw1.f (*1-D; see claw2.f for 2-D*):

```
c    src1 = subroutine for the source terms that solves the equation
c               capa * q_t = psi
c           over time dt.
c
c           If method(5)=0 then the equation does not contain a source
c           term and this routine is never called.  A dummy argument can
c           be used with many compilers, or provide a dummy subroutine that
c           does nothing (such a subroutine can be found in
c           claw/clawpack/1d/lib/src1.f)
c
c          The form of this subroutine is
c  -------------------------------------------------
c     subroutine src1(maxmx,meqn,mbc,mx,xlower,dx,q,maux,aux,t,dt)
c     implicit double precision (a-h,o-z)
c     dimension   q(1-mbc:maxmx+mbc, meqn)
c     dimension aux(1-mbc:maxmx+mbc, *)
c  -------------------------------------------------
c      If method(7)=0  or the auxiliary variables are not needed in this solver,
c      then the latter dimension statement can be omitted, but aux should
c      still appear in the argument list.
c
c      On input, q(i,m) contains the data for solving the
c                source term equation.
c      On output, q(i,m) should have been replaced by the solution to
c                 the source term equation after a step of length dt.
c
```

An example of how it's done

From book/chap21/radialdam/1drad/src1.f:

```fortran
c =========================================================
      subroutine src1(maxmx,meqn,mbc,mx,xlower,dx,q,maux,aux,t,dt)
c     src1 = subroutine for the source terms that solves the equation
c                capa * q_t = psi
c            over time dt.
c =========================================================
      implicit real*8(a-h,o-z)
      dimension q(1-mbc:maxmx+mbc, meqn)
c
      common /comrp/ grav
      common /comsrc/ ndim
c
c     # source terms for radial symmetry in shallow water equations
c
c     # ndim should be set in setprob.f
c     # ndim = 2  for cylindrical symmetry
c     # ndim = 3  for spherical symmetry
c
c     # 2-stage Runge-Kutta method
c
      do 10 i=1,mx+mbc
       xcell = xlower + (i-0.5d0)*dx
       qstar1 = q(i,1) - 0.5d0*dt*(ndim-1)/xcell * q(i,2)
       qstar2 = q(i,2) - 0.5d0*dt*(ndim-1)/xcell * q(i,2)**2 / q(i,1)
c
       q(i,1) = q(i,1) - dt*(ndim-1)/xcell * qstar2
       q(i,2) = q(i,2) - dt*(ndim-1)/xcell * qstar2**2 / qstar1
   10    continue
c
      return
      end
```

ndim-1 is α

A more complex example is in myConversions/chap17/stiffburgers/src1.f:

Euler equations in 1-D radial coordinates:

From Leveque (18.9):

$$\rho_t + (\rho u)_r = -\frac{\alpha}{r}(\rho u)$$

$$(\rho u)_t + (\rho u^2 + p)_r = -\frac{\alpha}{r}(\rho u^2)$$

$$E_t + ((E+p)u)_r = -\frac{\alpha}{r}((E+p)u)$$

where u is the velocity in the radial (r) direction. This system can be used for either cylindrical or spherical symmetry depending on α. In the cylindrical case, α = 1, and in the spherical case α = 2. This is a more precise statement than Leveque makes.

Note that you have to think very carefully about how to define the radial boundary at r = 0. Many codes do this incorrectly! A hint: It pays to be clear about the conservation laws and the fact that you are using a finite volume approach.

Boundary sources

Inflow boundary conditions are treated in some codes as location-specific source terms, and so would be solved as just indicated (e.g. in `src2.f`).

In Clawpack, it is more efficient to treat these simply as time-dependent boundary conditions in `bc2.f`, setting the ghost-cell values (of density, pressure and velocity, for example) to the desired time-dependent quantities.

It is important to note, however, that the time applied to the ghost-cell values must be *advanced* with respect to the time at the boundary itself (see Leveque, section 7.2.2, in 1-D):

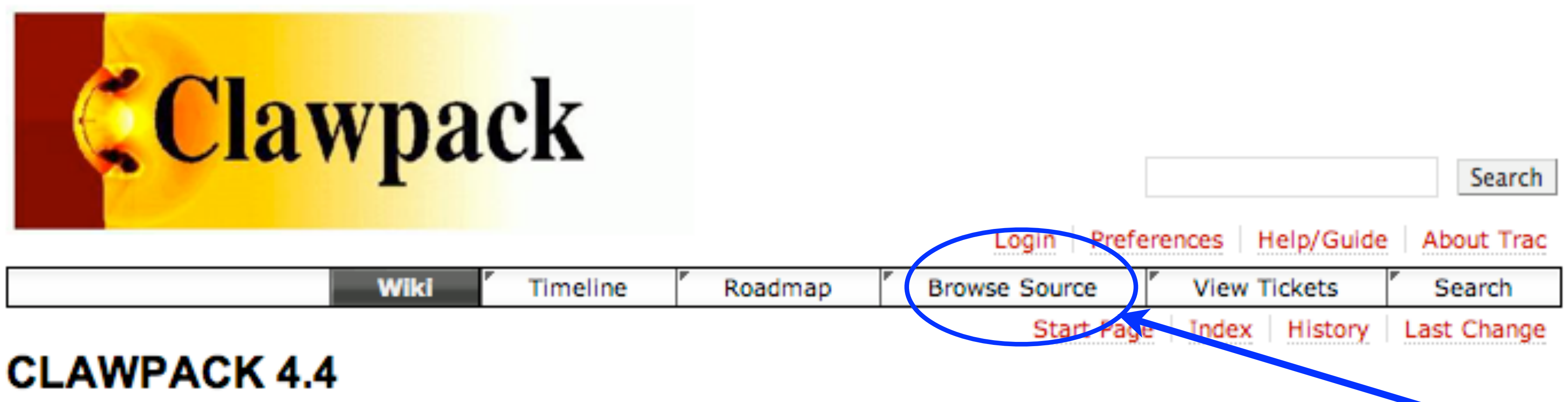
$$Q_0^n = g\left(t_n + \frac{\Delta x}{2u}\right)$$

$$Q_{-1}^n = g\left(t_n + \frac{3\Delta x}{2u}\right)$$

Here g is the function expressing the time dependence desired *at the boundary* and u is the velocity at the boundary.

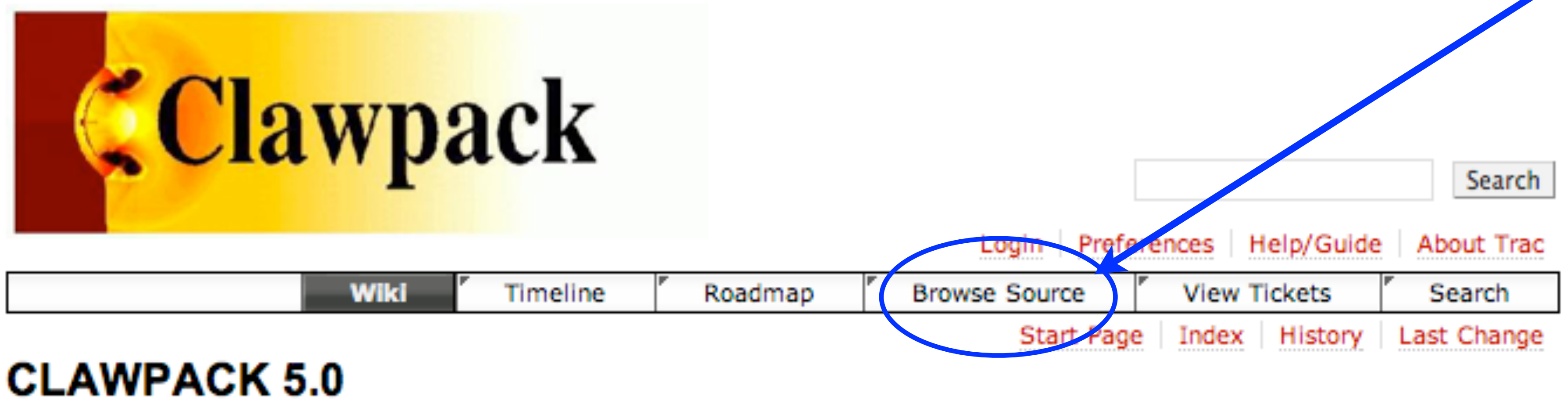
Don’t forget the resources at the Clawpack web site!

http://kingkong.amath.washington.edu/trac/claw4/wiki/

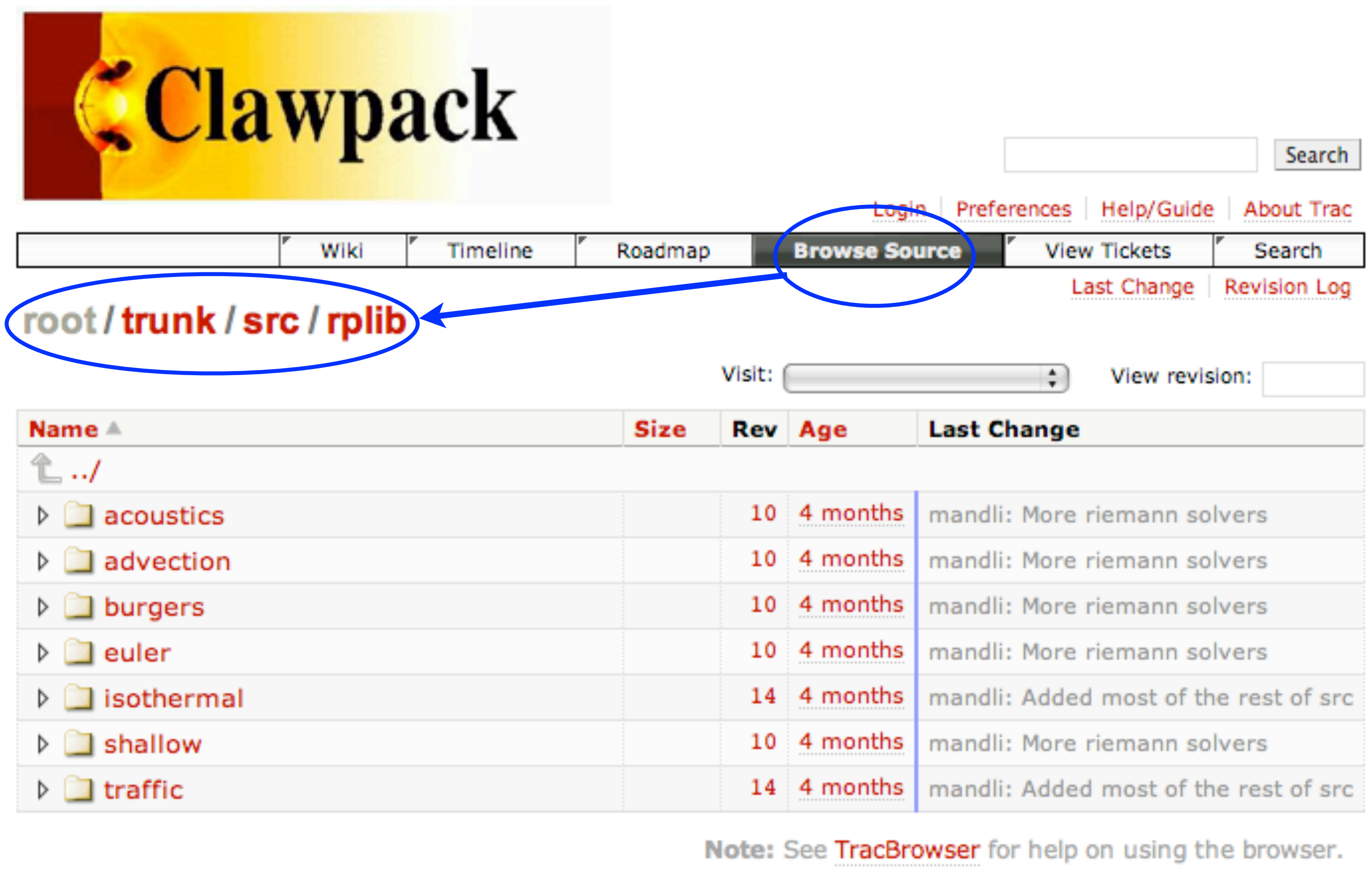

Clawpack
Search
Login | Preferences | Help/Guide | About Trac
Wiki | Timeline | Roadmap | Browse Source | View Tickets | Search
Start Page | Index | History | Last Change
CLAWPACK 4.4

use these!

http://kingkong.amath.washington.edu/trac/clawpack/wiki/


Clawpack
Search
Login | Preferences | Help/Guide | About Trac
Wiki | Timeline | Roadmap | Browse Source | View Tickets | Search
Start Page | Index | History | Last Change
CLAWPACK 5.0

For example, here is a collection of Riemann solvers from Clawpack 5:



root / trunk / src / rplib

Name	Size	Rev	Age	Last Change
../				
acoustics		10	4 months	mandli: More riemann solvers
advection		10	4 months	mandli: More riemann solvers
burgers		10	4 months	mandli: More riemann solvers
euler		10	4 months	mandli: More riemann solvers
isothermal		14	4 months	mandli: Added most of the rest of src
shallow		10	4 months	mandli: More riemann solvers
traffic		14	4 months	mandli: Added most of the rest of src

Note: See TracBrowser for help on using the browser.

Next: project presentations