BEARCLAW *Hermes* BREACH MODEL

PURPOSE	2D model of *Hermes* spaceship breach
EQUATION	Euler equations $q_t + f(q)_x + f(q)_y = 0$
DOMAIN	$-100 \leqslant x \leqslant 100, -20 \leqslant y \leqslant 20$
INITIAL CONDITION	$p = p_0$ for $x > 0$, $p = p_v \ll p_0$ for $x \leqslant 0$
BOUNDARY CONDITIONS	Characteristic outflow, moving wall

Note. This is a literate programming implementation directly linked to the application source code. Open this document at the command line, within its directory, i.e.,

```
@tarheellinux/hermes:texmacs doc.tm &
```

Links within a light green background `Makefile` directly invoke an editor to modify the specified file. Syntax-highlighted images of the file are automatically updated within this document.

Table of contents

1 Theory

Compute the force exerted by escaping gas in a 1D model of the *Hermes* spaceship. Solve 2D Euler equations in conservative form

$$q_t + f(q)_x + g(q)_y = 0$$

$$q = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix}, f(q) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ \rho u H \end{pmatrix}, g(q) = \begin{pmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ \rho v H \end{pmatrix},$$

through wave propagation algorithm using a linearized Riemann solver based on Roe averages. Set initial conditions to have no variation along y axis. Assume *Hermes* atmosphere is a perfect gas satisfying thermodynamic relations

$$p = \rho R T = \rho(\gamma - 1) c_V T = (\gamma - 1)\rho e = (\gamma - 1)\rho\left(E - \frac{u^2 + v^2}{2}\right),$$

$$H = E + \frac{p}{\rho}.$$

The sound speed is defined as $c^2 = \gamma p / \rho$.

1.1 Euler equation eigenmodes

The eigenmodes of the flux Jacobian matrices

$$A = \frac{\partial f}{\partial q}, B = \frac{\partial g}{\partial q},$$

are required for the Riemann solver. Define the conservative variables and fluxes.

Mathematica

```
In[33]:= q={rho, l, m, epsilon}
```

$\{\rho, l, m, \epsilon\}$

```
In[34]:= p=(gamma-1)(epsilon - (l^2+m^2)/(2 rho))
```

$(\gamma - 1)\left(\epsilon - \frac{l^2 + m^2}{2\,\rho}\right)$

```
In[35]:= H=(epsilon+p)/rho
```

$$\frac{(\gamma - 1)\left(\epsilon - \frac{l^2 + m^2}{2\,\rho}\right) + \epsilon}{\rho}$$

```
In[36]:= f={l, l^2/rho+p, l m/rho, l H}
```

$$\left\{l, (\gamma - 1)\left(\epsilon - \frac{l^2 + m^2}{2\,\rho}\right) + \frac{l^2}{\rho}, \frac{l\,m}{\rho}, \frac{l\left((\gamma - 1)\left(\epsilon - \frac{l^2 + m^2}{2\,\rho}\right) + \epsilon\right)}{\rho}\right\}$$

```
In[37]:= g={m, l m/rho, m^2/rho+p, m H}
```

$$\left\{m, \frac{l\,m}{\rho}, (\gamma - 1)\left(\epsilon - \frac{l^2 + m^2}{2\,\rho}\right) + \frac{m^2}{\rho}, \frac{m\left((\gamma - 1)\left(\epsilon - \frac{l^2 + m^2}{2\,\rho}\right) + \epsilon\right)}{\rho}\right\}$$

```
In[38]:=
```

Compute the Jacobians

```
In[38]:= A=Simplify[Grad[f,q]]
```

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{(\gamma-3)\,l^2+(\gamma-1)\,m^2}{2\,\rho^2} & -\frac{(\gamma-3)\,l}{\rho} & \frac{m-\gamma\,m}{\rho} & \gamma-1 \\ -\frac{l\,m}{\rho^2} & \frac{m}{\rho} & \frac{l}{\rho} & 0 \\ \frac{l\,((\gamma-1)\,l^2+(\gamma-1)\,m^2-\epsilon\,\gamma\,\rho)}{\rho^3} & -\frac{3\,(\gamma-1)\,l^2+(\gamma-1)\,m^2-2\,\epsilon\,\gamma\,\rho}{2\,\rho^2} & -\frac{(\gamma-1)\,l\,m}{\rho^2} & \frac{\gamma\,l}{\rho} \end{pmatrix}$$

```
In[39]:= B=Simplify[Grad[g,q]]
```

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ -\frac{l\,m}{\rho^2} & \frac{m}{\rho} & \frac{l}{\rho} & 0 \\ \frac{(\gamma-1)\,l^2+(\gamma-3)\,m^2}{2\,\rho^2} & \frac{l-\langle\text{gammal}\rangle}{\rho} & -\frac{(\gamma-3)\,m}{\rho} & \gamma-1 \\ \frac{m\,((\gamma-1)\,l^2+(\gamma-1)\,m^2-\epsilon\,\gamma\,\rho)}{\rho^3} & -\frac{(\gamma-1)\,l\,m}{\rho^2} & -\frac{(\gamma-1)\,l^2+3\,(\gamma-1)\,m^2-2\,\epsilon\,\gamma\,\rho}{2\,\rho^2} & \frac{\gamma\,m}{\rho} \end{pmatrix}$$

```
In[40]:=
```

Physical interpretation of the eigenmodes is easier in the primitive variables $\{\rho, u, v, p\}$. Define the transformation rules, and apply them to the flux Jacobian

```
In[23]:= q2p = {l->rho u, m->rho v,
          epsilon->rho c^2/gamma/(gamma-1)+rho(u^2/2+v^2/2)}
```

$$\left\{l \to \rho\,u,\, m \to \rho\,v,\, \epsilon \to \frac{c^2\,\rho}{(\gamma-1)\,\gamma} + \rho\left(\frac{u^2}{2}+\frac{v^2}{2}\right)\right\}$$

```
In[41]:= Au=FullSimplify[A /. q2p]
```

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{1}{2}\,((\gamma-3)\,u^2+(\gamma-1)\,v^2) & -(\gamma-3)\,u & v-\gamma\,v & \gamma-1 \\ -u\,v & v & u & 0 \\ \frac{u\,((\gamma-2)\,(\gamma-1)\,(u^2+v^2)-2\,c^2)}{2\,(\gamma-1)} & \frac{2\,c^2-(\gamma-1)\,((2\,\gamma-3)\,u^2-v^2)}{2\,(\gamma-1)} & -(\gamma-1)\,u\,v & \gamma\,u \end{pmatrix}$$

```
In[42]:= Bu=FullSimplify[B /. q2p]
```

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ -u\,v & v & u & 0 \\ \frac{1}{2}\,((\gamma-1)\,u^2+(\gamma-3)\,v^2) & u-\langle\text{gammau}\rangle & -(\gamma-3)\,v & \gamma-1 \\ \frac{v\,((\gamma-2)\,(\gamma-1)\,(u^2+v^2)-2\,c^2)}{2\,(\gamma-1)} & -(\gamma-1)\,u\,v & \frac{2\,c^2+(\gamma-1)\,(u^2+(3-2\,\gamma)\,v^2)}{2\,(\gamma-1)} & \gamma\,v \end{pmatrix}$$

```
In[43]:=
```

Disturbances in the fluid field values propagate as waves. The eigenvalues of the flux Jacobians are wave speeds

```
In[45]:= lambdaA=FullSimplify[Eigenvalues[Au] /. q2p,Assumptions->{rho>0, c>0}]
```

$$\{c+u, u-c, u, u\}$$

```
In[46]:= lambdaB=FullSimplify[Eigenvalues[Bu] /. q2p,Assumptions->{rho>0, c>0}]
```

$$\{c+v, v-c, v, v\}$$

```
In[47]:=
```

The Euler system has 4 eigenmodes with speeds $\lambda_1 = u+c$, $\lambda_2 = u-c$, $\lambda_3 = u$, $\lambda_4 = u$. Modes 1 and 2 correspond to forward and backward propagating acoustic waves. Modes 3 and 4 correspond to the same eigenvalue and correspond to propagation of a shear wave and a contact discontinuity.

The eigenvectors of the flux Jacobians describe how the conservative variables are coupled in wave propagation. After computation of the eigenvectors, the modes are expressed in a simpler, physically meaningful form using allowable algebraic operations.

```
In[63]:= RA=FullSimplify[Eigenvectors[Au] /. q2p,Assumptions->{rho>0, c>0}];
         Transpose[RA]
```

$$\begin{pmatrix} \frac{2(\gamma-1)}{2c^2+2(\gamma-1)uc+(\gamma-1)(u^2+v^2)} & \frac{2(\gamma-1)}{2c^2-2(\gamma-1)uc+(\gamma-1)(u^2+v^2)} & \frac{2}{u^2-v^2} & \frac{2v}{v^2-u^2} \\ \frac{2(\gamma-1)(c+u)}{2c^2+2(\gamma-1)uc+(\gamma-1)(u^2+v^2)} & \frac{2(\gamma-1)(c-u)}{-2c^2+2(\gamma-1)uc-(\gamma-1)(u^2+v^2)} & \frac{2u}{u^2-v^2} & -\frac{2uv}{u^2-v^2} \\ \frac{2(\gamma-1)v}{2c^2+2(\gamma-1)uc+(\gamma-1)(u^2+v^2)} & \frac{2(\gamma-1)v}{2c^2-2(\gamma-1)uc+(\gamma-1)(u^2+v^2)} & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

```
In[64]:= RA[[1]]=Simplify[RA[[1]]/RA[[1,1]] /. c^2 -> (gamma-1)(h-(u^2+v^2)/2)];
         RA[[2]]=Simplify[RA[[2]]/RA[[2,1]] /. c^2 -> (gamma-1)(h-(u^2+v^2)/2)];
         RA[[4]]=Simplify[RA[[4]]/RA[[3,1]]];
         RA[[3]]=Simplify[RA[[3]]/RA[[3,1]]];
         Transpose[RA]
```

$$\begin{pmatrix} 1 & 1 & 1 & -v \\ c+u & u-c & u & -uv \\ v & v & 0 & \frac{1}{2}(u^2-v^2) \\ h+cu & h-cu & \frac{1}{2}(u^2-v^2) & 0 \end{pmatrix}$$

```
In[65]:= RA[[4]]=Simplify[RA[[4]] + v RA[[3]]]; Transpose[RA]
```

$$\begin{pmatrix} 1 & 1 & 1 & 0 \\ c+u & u-c & u & 0 \\ v & v & 0 & \frac{1}{2}(u^2-v^2) \\ h+cu & h-cu & \frac{1}{2}(u^2-v^2) & \frac{1}{2}v(u^2-v^2) \end{pmatrix}$$

```
In[66]:= RA[[4]]=Simplify[RA[[4]]/RA[[4,3]]]; Transpose[RA]
```

$$\begin{pmatrix} 1 & 1 & 1 & 0 \\ c+u & u-c & u & 0 \\ v & v & 0 & 1 \\ h+cu & h-cu & \frac{1}{2}(u^2-v^2) & v \end{pmatrix}$$

```
In[67]:= RA[[3]]=FullSimplify[RA[[3]] + v RA[[4]]]; Transpose[RA]
```

$$\begin{pmatrix} 1 & 1 & 1 & 0 \\ c+u & u-c & u & 0 \\ v & v & v & 1 \\ h+cu & h-cu & \frac{1}{2}(u^2+v^2) & v \end{pmatrix}$$

```
In[70]:= XA=RA; Transpose[XA]
```

$$\begin{pmatrix} 1 & 1 & 1 & 0 \\ c+u & u-c & u & 0 \\ v & v & v & 1 \\ h+cu & h-cu & \frac{1}{2}(u^2+v^2) & v \end{pmatrix}$$

```
In[71]:=
```

2 Makefile

The `Makefile` contains application-specific instructions to produce an **xbear** executable.

```
#===============================================================================
# BEARCLAW Boundary Embedded Adaptive Refinement Conservation LAW package
#===============================================================================
# (c) Copyright Sorin Mitran, 2017
# Department of Mathematics
# University of North Carolina at Chapel Hill
# mitran@unc.edu
# ------------------------------------------------------------------------------
# This code may be freely used for educational and research purposes.
# For any other use please contact the author.
# ------------------------------------------------------------------------------
# File:            Makefile
# Purpose:         Build 2D Euler equation Hermes breach model
# Contains:
# Revision History: Ver. 1.0 Feb. 2017 Sorin Mitran
# ------------------------------------------------------------------------------
#
default: xbear

include $(BEARCLAW)/Makefile.inc

PROBLEM_SOURCES = \
  nodeinfodef.f90 \
  problem.f90

PROBLEM_OBJECTS = \
  nodeinfodef.o \
  problem.o

METHOD_SOURCES = \
  $(BEARCLAWLIB)/wavebear.f90

METHOD_MODULES = \
  $(BEARCLAWLIB)/wavebear.mod

METHOD_OBJECTS = \
  $(BEARCLAWLIB)/wavebear.o

nodeinfodef.o: nodeinfodef.f90
 $(FC) -I$(DIR_MODULES) -o nodeinfodef.o nodeinfodef.f90

problem.o: problem.f90 physflux1.f90
 $(FC) -I$(DIR_MODULES) -o problem.o problem.f90

include $(BEARCLAW)/Makefile.targets
```

3 Input files for Hermes 2D model with multiple root-level grids

Multiple root-level grids are defined in order to model the *Hermes* substructures.

The number of BEARCLAW root level grids is specified in `bear.data`. Each grid is defined through a `grid`*nnn*`.data` file. The first grid is defined in `grid.data` (the most common case), additional grids are defined by `grid`*nnn*`.data` with *nnn* starting from 002 to the number of grids. The relative positions of each grid have to be specified; this is done through a global numbering system (i, j).

3.1 Problem dependent parameters

Define in `setprob.data` the state inside the *Hermes*, and that approximating vacuum conditions. The data in this file is read in by `problem.f90:setprob` and is typically stored as global variables defined in `problem.f90:GlobalVariables`.

```
T               Entropy fix (efix)
1.4d0             gamma
0.              xBreach
42390             mHermes (kg)
1.013000000000000d5 1.225000000000000d0 0.000000000000000d0 0.000000000000000d0 Hermes atmosphere
0.010000000000000d5 0.045600000000000d0 0.000000000000000d0 0.000000000000000d0 Vacuum approximation
      p (Pa)            rho (kg/m^3)       u (m/s)           v (m/s)

Mass calculation:

Hermes is a L=100m long x D=5m diameter cylinder with d=5mm thick walls made of Al, rhoAl=2700 kg/m^3
  mHull = rhoAl*L*(pi*D)*d = 21195 kg
  mPayload = mHull

  mHermes ~= 42390 kg
```

3.2 Global run parameters

The `bear.data` file contains two sections.

1. bear.data:RunFlags sets flags affecting various global execution options

```
!:RunFlags:!   | Variable       | Description
!=============================================================================
F    0        Restart, Frame    Resume from checkpoint data dump
F             LevelEqSets       Solve different equations  on grid levels
F             LevelMethods      Apply different algorithms on grid levels
F             SaveAtFixedTimes  F=maintain CFL, T=save data at desired times
F             MaintainAuxArrays Treat aux similarily to q in MPI runs
F             InitialAMRonly    Generate initial AMR structure and stop
T             OutputStyleParams Outputstyle line contains additional formatting
!=============================================================================
```

2. bear.data:RunParameters sets parameters affecting various global execution options

```
!:RunParameters:!
!=============================================================================
14              nRootGrids        Number of root-level grids
3               MaxLevels         Maximum number of grid refinement levels
2 2 2 2 2       CoarsenRatio      ... of child grid to obtain parent spacing
4               MinimumGridPoints ... along one dimension
1               TimeStepMethod    0 fixed dt, 1 variable dt
0.d0            t0                initial time (if not Restart)
1.5d0           tfinal            final time
4     0.5       MaxCFLRetry, rCFL Try reducing CFL by this ratio this many times
3               OutputStyle       1 AMRCLAW, 2 TECPLOT, 3 HDF, 9 GnuPlot, 11 VTK
300             OutputFrames      Number of data checkpoints
T T T T T T     OutputLevel       Level output flag
!=============================================================================
```

3.3 Root-level grid parameters

Define the individual grids with minimal effort through use of the Python template facility.

1. grid.template:GridParameters defines the grid geometry and boundaries. Variables that change with each grdi are prefaced by a dollar sign.

```
!:GridParameters:! Variable       Description
!=============================================================================
2             nDim          Grid spatial dimensions
4             MaxLevel      Max grid refinement levels for this grid
500           mx            Cells in x direction
4             my            Cells in y direction
1 100         mGlobal(1)    Global index extents of this grid (x-direction)
40 50         mGlobal(2)    Global index extents of this grid (y-direction)
-10.d0        xlower        Left edge of computational domain
0.d0          xupper        Right edge of computational domain
0.0d0         ylower        Bottom edge of computational domain
4.0d0         yupper        Top edge of computational domain
2             mbc           Number of ghost cells at each boundary
1             mthbc(1)      Left   boundary condition code
11            mthbc(2)      Right  boundary condition code
3             mthbc(3)      Bottom boundary condition code
3             mthbc(4)      Top    boundary condition code
0.15d-3       dtv(1)        Initial time step (constant dt TimeStepMethod=0)
1.0d99        dtv(2)        Max allowable time step
1.00d0        cflv(1)       Max allowable Courant number
0.90d0        cflv(2)       Desired Courant number
0.1           cflv(3)       Time step relaxation parameter
!=============================================================================
```

2. grid.template:MultiphysicsParameters defines the conservation laws solved on this grid. These are identical for all grids in the computation

```
!:MultiphysicsParameters:! - one value if LevelEqSets==F else (>=MaxLevel) values
!================================================================================
! NrVars          = Number of primary field variables
4
! Output style parameters
0 1 1 0
! nEquationSet     = Equation set for these fields
1
! maux            = Number of auxilliary fields
0
!================================================================================
```

3. grid.template:NumericalSchemeParameters defines the numerical scheme on this grid, same for all grids in the computation.

```
!:NumericalSchemeParameters:! one value if LevelMethods==F else (>=MaxLevel) values
!================================================================================
0             method(1)   = (reserved)
2             method(2)   = convergence order
2             method(3)   = transverse convergence order
0             method(4)   = verbosity of wavebear output
0             method(5)   = source term splitting
0             method(6)   = 0 split q differences, 1 split flux differences
0             method(7)   = radius of slab around current 1D array of cells

4             mwaves      = number of waves in each Riemann solution
3 3 3 3         mthlim(mw)  = limiter for each wave  (mw=1,mwaves)
!================================================================================
```

4. grid.template:UserRootLevelParameters defines additional grid parameters

```
!:UserRootLevelParameters:!
!================================================================================
! (none for this application)
!================================================================================
```

Define arrays for the variables that reflect geometry in Fig. 1. It's convenient to define a grid number array to keep visual track of grid dimension assignment.

```
Python] nG    =[  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14];
Python] mx    =[100,200,200,250,150, 50,200, 50, 50, 50,150, 50,200,500];
Python] my    =[100, 50,100,100,200,200,200,150,100,150,200,200,200,100];
Python] mGx1  =[  1,101,101,301,401,551,601,551,551,551,401,551,601,601];
Python] mGx2  =[100,300,300,550,550,600,800,600,600,600,550,600,800,1100];
Python] mGy1  =[351,451,351,351,501,501,501,451,351,201,  1,  1,  1,351];
Python] mGy2  =[450,500,450,450,800,800,800,600,450,350,200,200,200,450];
Python] xlower=[  0, 10, 10, 30, 40, 55, 60, 55, 55, 55, 40, 55, 60, 60];
Python] xupper=[ 10, 30, 30, 55, 55, 60, 80, 60, 60, 60, 55, 60, 80,110];
Python] ylower=[ 35, 45, 35, 35, 60, 60, 60, 45, 35, 20,  0,  0,  0, 35];
Python] yupper=[ 45, 50, 45, 45, 80, 80, 80, 60, 45, 35, 20, 20, 20, 45];
```

```
Python] mthbc1=[  1, 10,999,999, 10,999,999, 10,999, 10, 10,999,999,999];
Python] mthbc2=[999, 11,999,999,999,999, 11, 11,999, 11,999,999, 11, 11];
Python] mthbc3=[  3,999,  3,  3,  3,999,  3,999,999,999,  3,  3,  3,  3];
Python] mthbc4=[  3,  3,999,  3,  3,  3,  3,999,999,999,  3,999,  3,  3];
Python] nG    =[  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14];
        nGrids=len(nG); print nGrids

   14

Python]
```

Use the Python template module.

```
Python] from string import Template;
```

Write a loop to replace template variables with values defined above and write resulting grid*nnn*.data files.

```
Python] s=open("grid.template","r").read(); t=Template(s);
        for i in range(nGrids):
          g=t.safe_substitute(mx=str(mx[i]),my=str(my[i]),mGx1=str(mGx1[i]),
        mGx2=str(mGx2[i]),mGy1=str(mGy1[i]),mGy2=str(mGy2[i]),xlower=str(xlower[i]),
        xupper=str(xupper[i]),ylower=str(ylower[i]),yupper=str(yupper[i]),
        mthbc1=str(mthbc1[i]),mthbc2=str(mthbc2[i]),mthbc3=str(mthbc3[i]),
        mthbc4=str(mthbc4[i]) );
          i1=i+1; fname = 'grid%3.3d.data' % i1
          fgrid = open(fname, 'w');
          fgrid.write(g);
          fgrid.close();
Python]
```

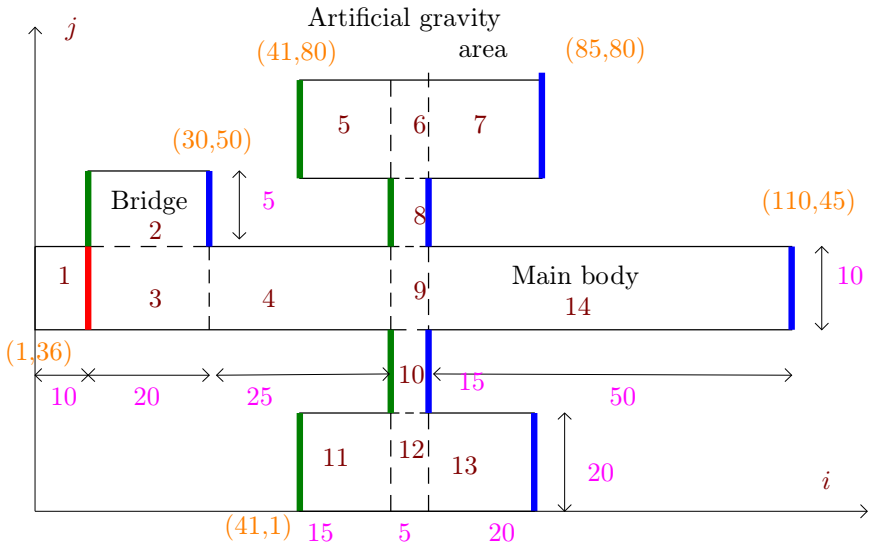


Figure 1. Hermes schematic model showing root level grids and boundary conditions: (red) breach location, (green) forward moving wall, (blue) rear moving wall. Dimensions are shown in meters (magenta). Global indexing coordinate pairs of cell centers (assuming cell size of 1m) are shown in orange.

4 Problem definition module

The `problem.f90` file defines a Fortran module that encapsulates problem physics.

4.1 Global variables

`problem.f90:GlobalVariables` specifies variables globally defined throughout the module

```
!:GlobalVariables:! Problem specific global variables
! Parameters
INTEGER, PARAMETER :: OK=0
REAL (KIND=qPrec), PARAMETER :: zero=0., half=0.5d0, one=1.d0, two=2.d0
! Variables
LOGICAL efix                              ! Flag: efix=T to apply entropy fix
REAL (KIND=qPrec) :: gamma,gamma1               ! Adiabatic coefficient gamma, gamma-1
REAL (KIND=xPrec) :: xBreach                  ! Breach location
REAL (KIND=qPrec) :: vHermes,mHermes            ! Hermes velocity, mass
REAL (KIND=qPrec), DIMENSION(2) :: pQ,rhoQ,uQ,vQ ! Initial values

!=======
CONTAINS
!=======
```

4.2 Set problem parameters

`problem.f90:setprob` reads problem-specific parameters

```
!:setprob:! Read problem global data
SUBROUTINE setprob
  INTEGER i
  ! Read gas parameters from file
  OPEN(UNIT=7,FILE='setprob.data',STATUS='old',FORM='formatted')
  READ(7,*)efix                        ! set to T to apply entropy fix in Riemann solver
  READ(7,*) gamma                        ! adiabatic constant
  gamma1 = gamma - one
  READ(7,*) xBreach                      ! position of Hermes breach
  READ(7,*) mHermes
  DO i=1,2
    READ(7,*)pQ(i),rhoQ(i),uQ(i),vQ(i)    ! initial values (primitive variables)
  END DO
  CLOSE(7)
  ! Initialize Hermes velocity, reaction force of escaping gas
  vHermes=zero
END SUBROUTINE setprob
```

4.3 Field variable initialization

problem.f90:qinit initialized the field variables

```
!qinit:! Field variable initialization
SUBROUTINE qinit(Info)
 TYPE (NodeInfo) :: Info  ! Data associated with this grid
 ! Internal declarations
 INTEGER i,iQ
 REAL (KIND=xPrec) :: x
 ! iQ=1 inside Hermes, at right; iQ=2 vacuum approximation
 x=Info%Xlower(1)+Info%dX(1)/2.0
 DO i=1,Info%mX(1)
  IF (x >= xBreach) THEN
    iQ = 1
  ELSE
    iQ = 2
  ENDIF
  Info%q(i,:,1,1,1) = rhoQ(iQ)
  Info%q(i,:,1,1,2) = rhoQ(iQ)*uQ(iQ)
  Info%q(i,:,1,1,3) = rhoQ(iQ)*vQ(iQ)
  Info%q(i,:,1,1,4) = pQ(iQ)/gamma1 + 0.5d0*rhoQ(iQ)*(uQ(iQ)**2 + vQ(iQ)**2)
  x=x+Info%dX(1)
 END DO
END SUBROUTINE qinit
```

4.4 Boundary conditions

problem.f90:problemBC defines application-specific boundary conditions, different from the predefined codes: 1=outflow, 2=periodic, 3=wall.

```
!:problemBC:!  User-defined boundary conditions
SUBROUTINE problemBC(Info)
 TYPE (NodeInfo) :: Info  ! Data associated with this grid
 ! Internal declarations
 INTEGER ibc,mx,mbc,r,rmbc
 SELECT CASE(Info%bcnow)
  CASE (1)
   SELECT CASE(Info%mthbc(1))
    CASE (10) ! Moving solid wall boundary condition at left
     mx=Info%mX(1); mbc=Info%mbc; r=Info%r; rmbc=r*mbc
     DO ibc=1,rmbc
      Info%q(1-ibc,:,1,1,:) = Info%q(ibc,:,1,1,:)
     END DO
     ! Negate the normal velocity (assumed to 2'nd component):
     DO ibc=1,rmbc
      Info%q(1-ibc,:,1,1,2) = 2*vHermes - Info%q(ibc,:,1,1,2)
     END DO
   END SELECT
  CASE (2)
   SELECT CASE(Info%mthbc(2))
    CASE (11) ! Moving solid wall boundary condition at right
     mx=Info%mX(1); mbc=Info%mbc; r=Info%r; rmbc=r*mbc
     DO ibc=1,rmbc
      Info%q(mx+ibc,:,1,1,:) = Info%q(mx+1-ibc,:,1,1,:)
     END DO
     ! Negate the normal velocity (assumed to 2'nd component):
     DO ibc=1,rmbc
      Info%q(mx+ibc,:,1,1,2) = 2*vHermes - Info%q(mx+1-ibc,:,1,1,2)
     END DO
   END SELECT
 END SELECT
END SUBROUTINE problemBC
```

4.5 Actions to take after each time step

The force acting on wall perpendicular to the x-direction is computed in `problem.f90:afterstep` and used to determine the change in *Hermes* velocity.

```
!:afterstep:! Actions after a time step
SUBROUTINE afterstep(Info)
  TYPE (NodeInfo) :: Info
  ! Internal declarations
  INTEGER j,mx,my
  REAL (KIND=qPrec) :: rho,rhou,rhov,rhoE,p,F
  ! Check if this is a leaf node grid with a moving wall boundary condition on right
  IF ((Info%nSubGrids .EQ. 0) .AND. (Info%mthbc(2) .EQ. 10)) THEN
    ! Yes, moving wall at left edge. Compute force, update velocity
    my = Info%mX(2); F=0.
    DO j=1,my
      rho  = Info%q(1,j,1,1,1);
      rhou = Info%q(1,j,1,1,2);
      rhov = Info%q(1,j,1,1,3);
      rhoE = Info%q(1,j,1,1,4);
      p = gamma1*(rhoE - (rhou**2+rhov**2)/(2.*rho))
      F = F - p*Info%dX(2)
    ENDDO
    ! Increment change in Hermes velocity produced by each leaf grid
    vHermes = vHermes + Info%dt*F/mHermes
  ENDIF
  ! Check if this is a leaf node grid with a moving wall boundary condition on right
  IF ((Info%nSubGrids .EQ. 0) .AND. (Info%mthbc(2) .EQ. 11)) THEN
    ! Yes, moving wall at right edge. Compute force, update velocity
    mx = Info%mX(1); my = Info%mX(2); F=0.
    DO j=1,my
      rho  = Info%q(mx,j,1,1,1);
      rhou = Info%q(mx,j,1,1,2);
      rhov = Info%q(mx,j,1,1,3);
      rhoE = Info%q(mx,j,1,1,4);
      p = gamma1*(rhoE - (rhou**2+rhov**2)/(2.*rho))
      F = F + p*Info%dX(2)
    ENDDO
    ! Increment change in Hermes velocity produced by each leaf grid
    vHermes = vHermes + Info%dt*F/mHermes
  ENDIF
END SUBROUTINE afterstep
```

4.6 Physical fluxes

The `physflux1.f90:physflux` routine defines the problem physics through specification of the fluxes and solution of the Riemann problem

4.6.1 Local variable declarations

The quantities $\Delta q, \alpha$ that arise in the Riemann problem solution are stored in `delta, coef` defined in physflux1.f90:physfluxInternalDeclarations. These arrays are allocated as needed. The `SAVE` attribute maintains the values in `delta, coef` between successive request calls to `physflux`.

```
!:physfluxInternalDeclarations:!
INTEGER, SAVE :: mbc,mx,nq,mwav,NrVars,mwaves,iError,j,mu,mv,mw,i
INTEGER, POINTER, DIMENSION(:) :: iCell,iEdge,iLft,iRgt
REAL (KIND=qPrec), ALLOCATABLE, SAVE, DIMENSION(:) :: delta,coef
! Cell center quantities
REAL (KIND=qPrec), ALLOCATABLE, SAVE, DIMENSION (:) :: sqrtrho,pres, &
          u,v,c,h,rho
! Roe averaged quantities
REAL (KIND=qPrec), ALLOCATABLE, SAVE, DIMENSION (:) :: uR,vR,hR,cR, &
          rhosq2,g1c2,u2v2,euv
! Shorter local names
REAL (KIND=qPrec), POINTER, DIMENSION (:,:) :: Apdq,Amdq,Asdq,BpAsdq,BmAsdq, &
                                  speed,q1D
REAL (KIND=qPrec), POINTER, DIMENSION (:,:,:) :: wave
REAL (KIND=qPrec) :: rho1,rhou1,en1,p1,c1,rho2,rhou2,en2,p2,c2
REAL (KIND=qPrec) :: s0,s1,s2,s3,sfract,df(5),rhov1,rhov2
```

4.6.2 Local aliases for `Info` fields

It is convenient to have shorter names for the `Info` structure fields, as in `wave=>Info%wave` - the local `wave` pointer references the `Info%wave` array. Define these pointers in physflux1.f90:physfluxInfo

```
!:physfluxInfo:! Extract information from Info structure
! Current number of interior cells, boundary cells, field variables, waves
mx=Info%mXnow; mbc=Info%mbc; NrVars=Info%NrVars; mwaves=Info%mwaves
! Associate local names with Info components
q1D=>Info%q1D
Apdq=>Info%Apdq; Amdq=>Info%Amdq; speed=>Info%speed; wave=>Info%wave
Asdq=>Info%Asdq; BpAsdq=>Info%BpAsdq; BmAsdq=>Info%BmAsdq
iEdge=>Info%I1D; iCell=>Info%I1Dcells; iLft=>Info%I1Dleft; iRgt=>Info%I1Dright
```

4.6.3 Switching between Riemann problem directions

The eigenmodes for $A = \partial f / \partial q$ and $B = \partial g / \partial q$ can be obtained from one another through index permutation as defined in physflux1.f90:physfluxNormalDirection.

```
!:physfluxNormalDirection:! Define permutation to reuse eigenmode code
IF (ixy==1) THEN
  mu = 2; mv = 3
ELSE
  mu = 3; mv = 2
END IF
SELECT CASE (irequest)
```

4.6.4 Riemann problem solution

Code must be written to solve the normal Riemann problem along one-dimensional slices of the grid in response to `RequestNormalWaves`, and the transverse problem in response to `RequestTransverseWaves`.

The `physflux` routine is also called with requests `Initialize, Finalize, RequestFluxes` by the `wavebear` solver module.

Initialize The code in `physflux1.f90:physfluxInitialize` allocates space needed for cell-centered quantities (`u, v, h, c`), Roe-average quantities (`uR, vR, hR, cR`), and various work vectors.

```
!:physfluxInitialize:!
CASE (Initialize)
  ! Allocate local work space
  ALLOCATE(delta(NrVars),coef(mwaves),                    &
        sqrtrho(1-mbc:mx+mbc),pres(1-mbc:mx+mbc),rho(1-mbc:mx+mbc),          &
        u(1-mbc:mx+mbc),v(1-mbc:mx+mbc),h(1-mbc:mx+mbc),c(1-mbc:mx+mbc),     &
        uR(2-mbc:mx+mbc),vR(2-mbc:mx+mbc),hR(2-mbc:mx+mbc),cR(2-mbc:mx+mbc), &
        rhosq2(2-mbc:mx+mbc),g1c2(2-mbc:mx+mbc),u2v2(2-mbc:mx+mbc),          &
        euv(2-mbc:mx+mbc),                                &
        STAT=iError)
  IF (iError/=OK) THEN
    PRINT *,'Cannot allocate work space in problem module, physflux'
    STOP
  END IF
```

Finalize The code in `physflux1.f90:physfluxFinalize` releases space previously allocated by the response to `Initialize`.

```
!:physfluxFinalize:!
CASE (Finalize)
  ! DeAllocate local work space
  DEALLOCATE( delta,coef,sqrtrho,pres,rho,u,v,h,c,uR,vR,hR,cR,rhosq2,g1c2, &
          u2v2,euv,STAT=iError)
  IF (iError/=OK) THEN
    PRINT *,'Cannot deallocate work space in problem module, physflux'
    STOP
  END IF
```

RequestFluxes Code in `physflux1.f90:physfluxRequestFluxes` evaluate the physical fluxes based on cell-centered quantities.

```
!:physfluxRequestFluxes:!
CASE (RequestFluxes)
  IF (MINVAL(q1D(1-mbc:mx+mbc,1))<=zero) THEN
    PRINT *,'Error: Negative or zero density in physflux.'
    STOP
  END IF
  ! Compute cell centered quantities required in Riemann solve & entropy fix
  rho(iCell) = 1./q1D(iCell,1)
  u(iCell) = q1D(iCell,mu)*rho(iCell)
  v(iCell) = q1D(iCell,mv)*rho(iCell)
  rho(iCell) = q1D(iCell,1)
  sqrtrho(iCell) = SQRT(rho(iCell))
  pres(iCell) = gamma1*( q1D(iCell,4) - &
                half*(q1D(iCell,mu)*u(iCell) + q1D(iCell,mv)*v(iCell) ) )
  c = gamma*pres/rho
  IF (MINVAL(c)<=zero) THEN
    PRINT *,'Error: Non-physical centered sound velocity in physflux, RequestFluxes.'
    STOP
  END IF
  c = SQRT(c)
  h(iCell) = (q1D(iCell,4) + pres(iCell)) / rho(iCell)
  ! Compute the physical fluxes at cell centers
  f(iCell,1)  = q1D(iCell,mu)
  f(iCell,mu) = f(iCell,1)*u(iCell)+pres(iCell)
  f(iCell,mv) = f(iCell,1)*v(iCell)
  f(iCell,4)  = f(iCell,1)*h(iCell)
```

RequestNormalWaves The solution of the Riemann problem in the normal direction is computed in a succession of stages:

physflux1.f90:physfluxRequestNormalWaves Initialization of fluctuations and waves

```
!:physfluxRequestNormalWaves:!
CASE (RequestNormalWaves)
  Apdq=0.; Amdq=0.; speed=0.; wave=0.
```

physflux1.f90:ComputeRoeAverages computes the intermediate state at which the flux Jacobians are evaluated.

```
!:ComputeRoeAverages:!
rhosq2(iEdge) = one / (sqrtrho(iLft)+sqrtrho(iRgt))
uR(iEdge) = (q1D(iLft,mu)/sqrtrho(iLft) + q1D(iRgt,mu)/sqrtrho(iRgt)) * rhosq2(iEdge)
vR(iEdge) = (q1D(iLft,mv)/sqrtrho(iLft) + q1D(iRgt,mv)/sqrtrho(iRgt)) * rhosq2(iEdge)
hR(iEdge) = ((q1D(iLft,4)+pres(iLft))/sqrtrho(iLft)   +       &
             (q1D(iRgt,4)+pres(iRgt))/sqrtrho(iRgt)       ) * &
            rhosq2(iEdge)
u2v2=uR**2+vR**2
cR = gamma1*(hR-half*u2v2)
IF (MINVAL(cR)<=zero) THEN
  PRINT *,'Error: Negative or zero Roe average sound velocity in physflux.'
  STOP
END IF
g1c2 = gamma1/cR
cR = SQRT(cR)
euv = hR - u2v2
```

physflux1.f90:ConstructEigenmodes defines the propagating waves resulting from the flux Jacobian eigensystem

```
!:ConstructEigenmodes:!
! Backward acoustic
speed(iEdge,1) = uR(iEdge) - cR(iEdge)
wave(iEdge,1,1) = one
wave(iEdge,mu,1) = speed(iEdge,1)
wave(iEdge,mv,1) = vR(iEdge)
wave(iEdge,4,1) = hR(iEdge) - uR(iEdge)*cR(iEdge)
! Shear
speed(iEdge,2) = uR(iEdge)
wave(iEdge,1,2) = zero
wave(iEdge,mu,2) = zero
wave(iEdge,mv,2) = one
wave(iEdge,4,2) = vR(iEdge)
! Entropy
speed(iEdge,3) = uR(iEdge)
wave(iEdge,1,3) = one
wave(iEdge,mu,3) = uR(iEdge)
wave(iEdge,mv,3) = vR(iEdge)
wave(iEdge,4,3) = half*u2v2(iEdge)
! Forward acoustic
speed(iEdge,4) = uR(iEdge) + cR(iEdge)
wave(iEdge,1,4) = one
wave(iEdge,mu,4) = speed(iEdge,4)
wave(iEdge,mv,4) = vR(iEdge)
wave(iEdge,4,4) = hR(iEdge) + uR(iEdge)*cR(iEdge)
! Tracer
IF (mwaves==5) THEN
  speed(iEdge,5) = uR(iEdge)
  wave(iEdge,1:4,5) = zero
  wave(iEdge,5,5)   = one
END IF
```

`physflux1.f90:DecomposeJump` solves the linearized Riemann problem at each cell interface

```
!:DecomposeJump:! in q onto eigenbases
DO j=2-mbc,mx+mbc
 ! Find coef(1) thru coef(4), the coefficients of the 4 eigenvectors
 delta(1) = q1D(j,1) - q1D(j-1,1)
 delta(2) = q1D(j,mu) - q1D(j-1,mu)
 delta(3) = q1D(j,mv) - q1D(j-1,mv)
 delta(4) = q1D(j,4) - q1D(j-1,4)
 coef(3) = g1c2(j) * (euv(j)*delta(1) + uR(j)*delta(2) + vR(j)*delta(3) - delta(4))
 coef(2) = delta(3) - vR(j)*delta(1)
 coef(4) = (delta(2) + (cR(j)-uR(j))*delta(1) - cR(j)*coef(3)) / (two*cR(j))
 coef(1) = delta(1) - coef(3) - coef(4)
 IF (NrVars==5) THEN
  ! Tracer variable
  coef(5) = q1D(j,5) - q1D(j-1,5)
 END IF
 DO mw=1,mwaves
  wave(j,1:NrVars,mw) = coef(mw)*wave(j,1:NrVars,mw)
 END DO
END DO
```

`physflux1.f90:GodunovFlux` evaluates the numerical flux at the interface between two finite volume cells $i, i+1$ as the physical flux evaluated at the intermediate state

$$F_i^n = f\left(q_{i+1/2}^{\uparrow}\right)$$

with $q_{i+1/2}^{\uparrow}$ the solution to the Riemann problem at the $x_{i+1/2}$ interface. Through linearization of the Riemann problem, a continuous rarefaction fan has been replaced by a propagating rarefaction discontinuity that violates the thermodynamic condition of non-decreasing entropy. Only part of the jump associated with the rarefaction shock actually affects $q_{i+1/2}^{\uparrow}$. A sequence of checks is required to compute this correction (the "entropy fix"). Since this is computationally costly and only encountered in a small part of the finite volume cells within the computation, the entropy fix is an option specified in `setprob.data` through the logical flag `efix`.

```
!:GodunovFlux:! for first-order approximation
IF (efix) THEN
```

physflux1.f90:EntropyFix checks each wave to determine if only a fraction of the decomposed jump should actually affect cells to the left.

```
!:EntropyFix:!
! Compute flux differences amdq and apdq.
! First compute amdq as sum of s*wave for left going waves.
! Incorporate entropy fix by adding a modified fraction of wave
! if s should change sign.
DO j=2-mbc,mx+mbc
  ! 1-wave. Check for fully supersonic case
  s0=u(j-1)-c(j-1)
  IF ( s0>=zero .AND. speed(j,1) > zero ) THEN
    ! Everything is right-going
    Amdq(j,1:NrVars) = zero
    CYCLE
  END IF
  ! u-c to right of 1-wave
  rho1 = q1D(j-1,1) + wave(j,1,1)
  rhou1 = q1D(j-1,mu) + wave(j,mu,1)
  rhov1 = q1D(j-1,mv) + wave(j,mv,1)
  en1 = q1D(j-1,4) + wave(j,4,1)
  p1 = gamma1*(en1  - 0.5*(rhou1**2+rhov1**2)/rho1)
  IF ((p1<zero) .OR. (rho1<zero)) THEN
    PRINT *,'Negative pressure/density between 1-wave and 2,3-waves'
    CYCLE
    STOP
  END IF
  c1 = SQRT(gamma*p1/rho1)
  s1 = rhou1/rho1 - c1
  IF ( s0<zero .AND. s1>zero ) THEN
    ! Transonic rarefaction in the 1-wave
    sfract = s0 * (s1-speed(j,1)) / (s1-s0)
  ELSE IF (speed(j,1) < zero) THEN
    ! Left-going 1-wave
    sfract=speed(j,1)
  ELSE
    ! Right-going 1-wave
    sfract=zero  ! Never should reach this instruction
  END IF
  Amdq(j,1:NrVars) = sfract*wave(j,1:NrVars,1)

  ! Check 2,3-wave (contact+shear discontinuities)
  IF (speed(j,2) >= zero) CYCLE ! 2- and 3- wave are right-going
  Amdq(j,1:NrVars) = Amdq(j,1:NrVars) + &
                uR(j)*(wave(j,1:NrVars,2) + wave(j,1:NrVars,3))
  ! Check 4-wave. Compute u+c to left of 4-wave
  rho2 = q1D(j,1) - wave(j,1,4)
  rhou2 = q1D(j,mu) - wave(j,mu,4)
  rhov2 = q1D(j,mv) - wave(j,mv,4)
  en2 = q1D(j,4) - wave(j,4,4)
  p2 = gamma1*(en2 - 0.5d0*(rhou2**2+rhov2**2)/rho2)
  IF ((p2<=zero) .OR. (rho2<=zero)) THEN
    PRINT *,'Negative pressure/density between 2,3-waves and 4-wave'
    CYCLE
    STOP
  END IF
  c2 = SQRT(gamma*p2/rho2)
  s2 = rhou2/rho2 + c2; s3=u(j)+c(j)
  IF ( s2 < zero .AND. s3 > zero ) THEN
    ! Transonic rarefaction in the 4-wave
    sfract = s2 * (s3-speed(j,4)) / (s3-s2)
  ELSE IF (speed(j,4) < zero) THEN
    ! Left-going 4-wave
    sfract = speed(j,4)
  ELSE
    ! Right-going 4-wave
    CYCLE
  END IF
  Amdq(j,1:NrVars) = Amdq(j,1:NrVars) + sfract*wave(j,1:NrVars,4)
END DO
```

physflux1.f90:RightGoingFluxes subsequently evaluates the correction in the right-going fluxes

```
!:RightGoingFluxes:!
DO j=2-mbc,mx+mbc
  df=zero
  DO mwav=1,mwaves
    df(1:NrVars) = df(1:NrVars) + speed(j,mwav) * wave(j,1:NrVars,mwav)
  END DO
  Apdq(j,1:NrVars) = df(1:NrVars) - Amdq(j,1:NrVars)
 END DO
!===
ELSE
!===
```

physflux1.f90:NoEntropyFix directly uses the jump decomposition provided by the linearized Riemann solver to evaluate fluctuations that update cell averages, without correcting for possible rarefaction fans (and thus allowing small violations of the entropy condition in the computation).

```
!:NoEntropyFix:!
DO mw=1,mwaves
  DO j=2-mbc,mx+mbc
    IF (speed(j,mw) < zero) THEN
      Amdq(j,1:NrVars) = Amdq(j,1:NrVars) +  &
                  speed(j,mw) * wave(j,1:NrVars,mw)
    ELSE
      Apdq(j,1:NrVars) = Apdq(j,1:NrVars) +  &
                  speed(j,mw) * wave(j,1:NrVars,mw)
    END IF
  END DO
 END DO
END IF
```

RequestTransverseWaves Part of the fluctuation computed in the normal direction Riemann problem actually travels to adjacent cells in the transverse direction. This component is computed in physflux1.f90:physfluxRequestTransverseWaves

```
!:physfluxRequestTransverseWaves:!
CASE (RequestTransverseWaves)
  BpAsdq=0.; BmAsdq=0.; speed=0.; wave=0.
  ! Construct the transverse eigenbasis
  ! Backward acoustic
  speed(iEdge,1) = vR(iEdge) - cR(iEdge)
  wave(iEdge,1,1) = one
  wave(iEdge,mu,1) = uR(iEdge)
  wave(iEdge,mv,1) = speed(iEdge,1)
  wave(iEdge,4,1) = hR(iEdge) - vR(iEdge)*cR(iEdge)
  ! Shear
  speed(iEdge,2) = vR(iEdge)
  wave(iEdge,1,2) = zero
  wave(iEdge,mu,2) = one
  wave(iEdge,mv,2) = zero
  wave(iEdge,4,2) = uR(iEdge)
  ! Entropy
  speed(iEdge,3) = vR(iEdge)
  wave(iEdge,1,3) = one
  wave(iEdge,mu,3) = uR(iEdge)
  wave(iEdge,mv,3) = vR(iEdge)
  wave(iEdge,4,3) = half*u2v2(iEdge)
  ! Forward acoustic
  speed(iEdge,4) = vR(iEdge) + cR(iEdge)
  wave(iEdge,1,4) = one
  wave(iEdge,mu,4) = uR(iEdge)
  wave(iEdge,mv,4) = speed(iEdge,4)
  wave(iEdge,4,4) = hR(iEdge) + vR(iEdge)*cR(iEdge)
  IF (mwaves==5) THEN
    ! Tracer
    speed(iEdge,5) = vR(iEdge)
    wave(iEdge,1:4,5) = zero
    wave(iEdge,5,5) = one
  END IF
  ! Decompose fluctuation in Asdq onto eigenbasis
  DO j=2-mbc,mx+mbc
    ! Find coef(1) thru coef(4), the coefficients of the 4 eigenvectors
    delta(1:NrVars) = Asdq(j,1:NrVars)
    coef(3) = g1c2(j) * (euv(j)*delta(1) + uR(j)*delta(mu) + vR(j)*delta(mv) - delta(4))
    coef(2) = delta(mu) - uR(j)*delta(1)
    coef(4) = (delta(mv) + (cR(j)-vR(j))*delta(1) - cR(j)*coef(3)) / (two*cR(j))
    coef(1) = delta(1) - coef(3) - coef(4)
    IF (NrVars==5) THEN
      ! Tracer variable
      coef(5) = delta(5)
    END IF
    DO mw=1,mwaves
      wave(j,1:NrVars,mw) = coef(mw)*wave(j,1:NrVars,mw)
    END DO
  END DO
  ! Compute the transverse fluctuations
  DO mwav=1,mwaves
    DO j=2-mbc,mx+mbc
      IF (speed(j,mwav) < zero) THEN
        BmAsdq(j,1:NrVars) = BmAsdq(j,1:NrVars) +  &
                             speed(j,mwav) * wave(j,1:NrVars,mwav)
      ELSE
        BpAsdq(j,1:NrVars) = BpAsdq(j,1:NrVars) +  &
                             speed(j,mwav) * wave(j,1:NrVars,mwav)
      END IF
    END DO
  END DO
```

5 Results

The code is compiled and executed in a separate terminal window.

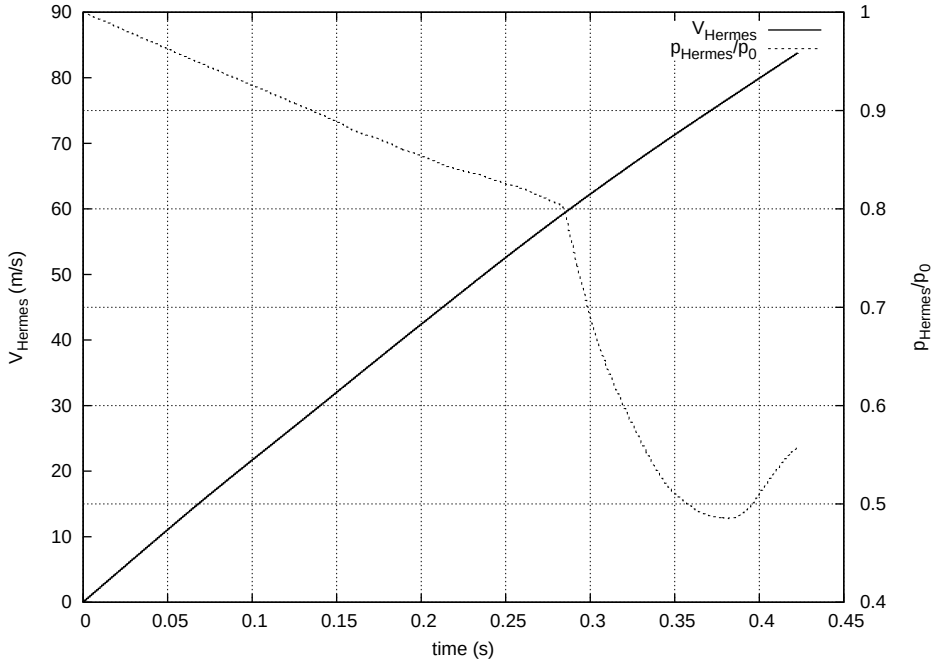
```
   Shell session inside TeXmacs pid = 2590

Shell] pwd

   /home/student/courses/MATH762/homework/hw1solution/1dshocktube

Shell] make xbear

   /usr/bin/gfortran -c -Dgfortran -fcray-pointer -O3 -g -DHDF -DNOMKL -I/home/
   student/bearclaw/lib -o problem.o problem.f90
   /usr/bin/gfortran   nodeinfodef.o problem.o  \
   /home/student/bearclaw/lib/treeops.o /home/student/bearclaw/lib/cutcell.o /home/
   student/bearclaw/lib/fixup.o /home/student/bearclaw/lib/beario.o /home/student/
   bearclaw/lib/wavebear.o /home/student/bearclaw/lib/linsolve.o /home/student/
```

```
bearclaw/lib/infofieldutils.o /home/student/bearclaw/lib/serial_exec.o /home/
student/bearclaw/lib/bearez.o /home/student/bearclaw/lib/setbc.o /home/student/
bearclaw/lib/driver.o -L/usr/lib64/hdf  -lmfhdf -ldf -ljpeg -lz -L/usr/lib64 -
lhdf5_fortran -lhdf5  -o xbear
```

```
Shell] exo-open --launch TerminalEmulator xbear
Shell]
```

The code saves a time history of the *Hermes* velocity and pressure drop in `vhermes.data`, rendered below using Gnuplot

```
GNUplot] set grid x y2;
         set xlabel("time (s)"); set ylabel("V_{Hermes} (m/s)");
         set y2label("p_{Hermes}/p_0");
         set autoscale y; set autoscale y2;
         set ytics nomirror; set y2tics;
         plot "vhermes.data" using 1:2 axes x1y1 title "V_{Hermes}" w l,
         "vhermes.data" using 1:3 axes x1y2 title "p_{Hermes}/p_0" w l
```




```
GNUplot]
```

The OpenDX script `hermes.net` produces animations of the time evolution of the field variables. Choose `File->Save Image ...->Continuous Saving` to save the animation to `anim.miff`. Frames from the resulting animation can be extracted using `convert` and combined into a single image using `montage`.

```
Shell] rm --force anim.miff; exo-open --launch TerminalEmulator dx -execute -program
       hermes.net
Shell] rm --force hermes???.png; convert anim.miff[0,10,30,45,60,80] hermes%03d.png
Shell] montage hermes???.png -tile 2x3 -geometry +0+0 hermes.png
Shell]
```

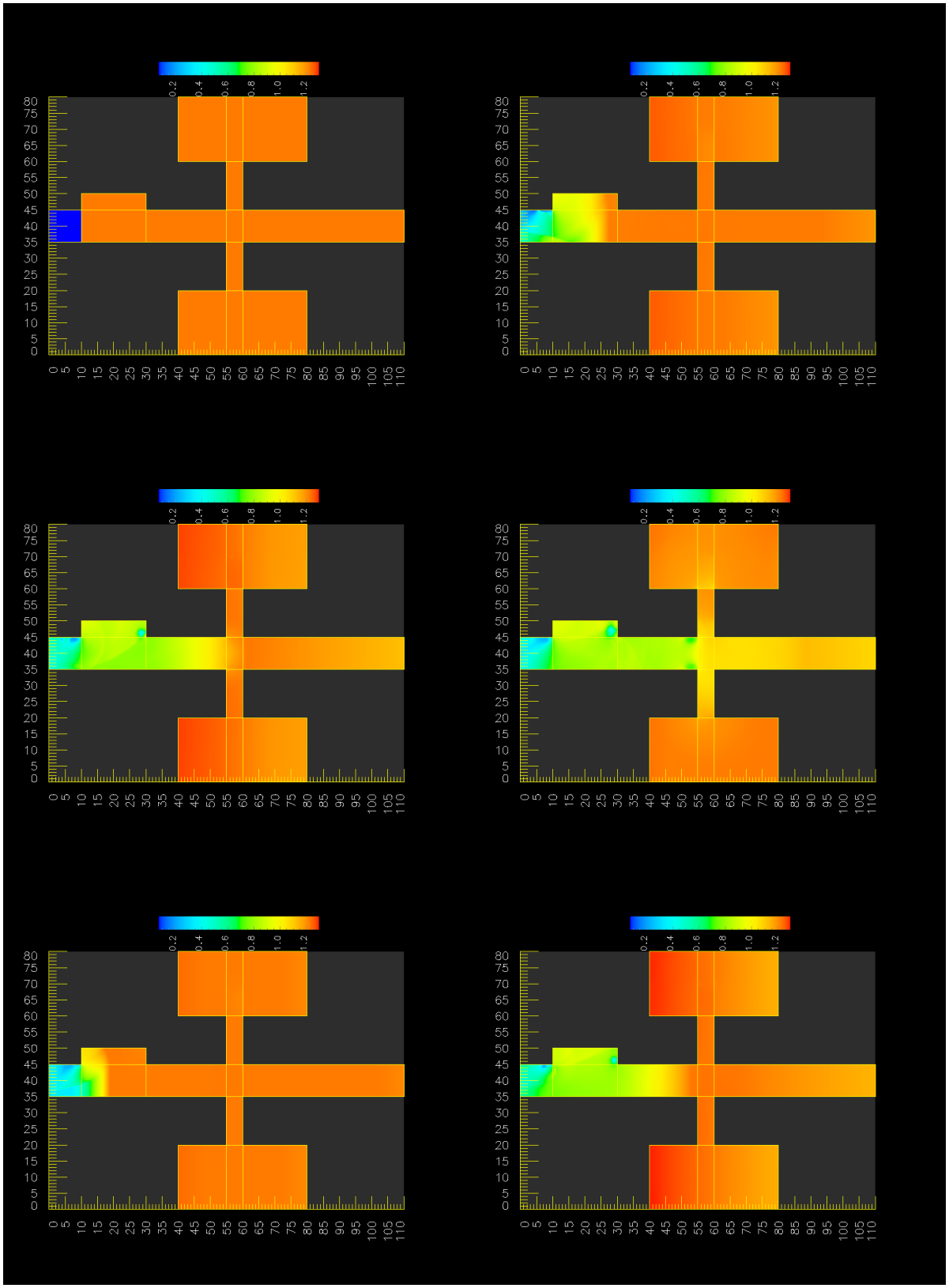


Figure 2. Evolution of density field with time in *Hermes*.

The OpenDX script `schlieren.net` produces an animation of $\log|\nabla\rho|$ as would be visualized using the Schlieren technique superimposed with the velocity variables. A Schlieren visualization shows the acoustic and vorticity waves in a compressible fluid.

```
Shell] rm --force schlieren.miff; exo-open --launch TerminalEmulator dx -execute -
       program schlieren.net
Shell] rm --force schlieren???.png; convert schlieren.miff[0,10,30,45,60,80]
       schlieren%03d.png
Shell] montage schlieren???.png -tile 2x3 -geometry +0+0 schlieren.png
Shell]
```

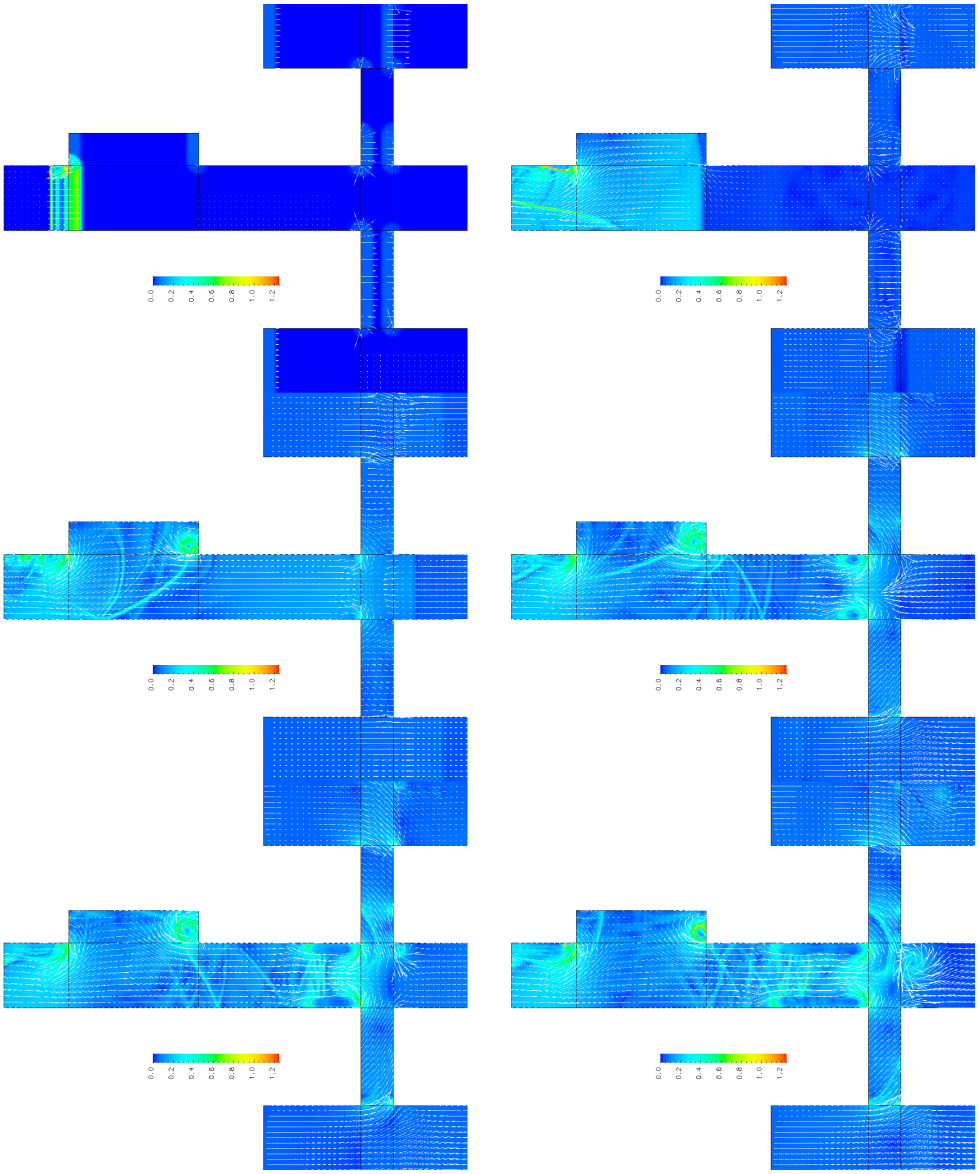


Figure 3. Schlieren visualization of *Hermes* breach.