Implementation of HLLC-AUSM low-Mach scheme in a density-based compressible solver in FOAM-extend

Mohammad Hossein Arabnejad

Mechanics and Maritime Sciences/Marine Technology,
Chalmers University of Technology,
Gothenburg, Sweden

2017-09-25

Outline

Incompressible solver vs Compressible Solver

Incompressible solver	does not consider the variation of density in the flow(density is constant)
Compressible Solver	Consider the variation of density in the flow

Pressure-based solver vs Density-based solver

Pressure-based solver	pressure is calculated from pressure correction equation derived from momentum equation and continuity equation.
density-based solver	density is calculated from continuity equation while pressure is calculated from equation of state

Two density-based solver in foam-extend: `dbnsFoam` and `dbnsTurbFoam`

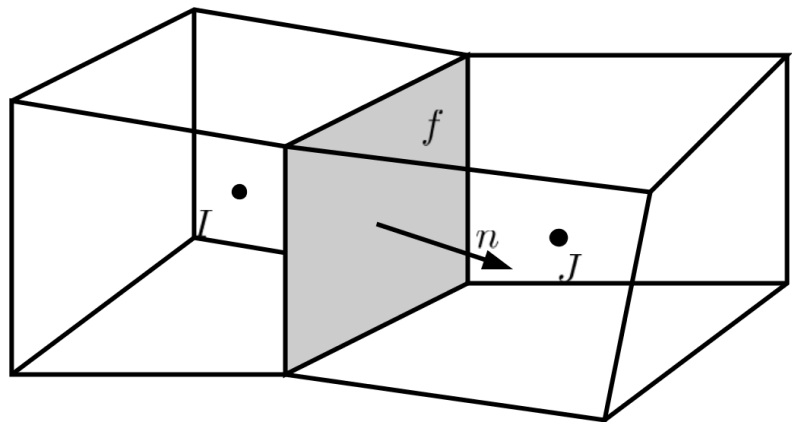
Governing Equations

- Continuity, momentum, and energy equations
- The integral form for the cell I

$$\frac{\partial}{\partial t}\iiint_{Vi}\mathbf{U}\,dV+\iint_{\partial Vi}\vec{\mathbf{F}}\left(\mathbf{U}\right)\cdot\vec{n}\,d\partial V_i=0$$

$$\mathbf{U}=\left[\rho,\rho u,\rho v,\rho w,\rho E\right]^T$$

$$\vec{\mathbf{F}}\left(\mathbf{U}\right)\cdot\vec{n}=\begin{bmatrix}\rho\hat{u}\\ \rho\hat{u}u+pn_x\\ \rho\hat{u}v+pn_y\\ \rho\hat{u}w+pn_z\\ \rho\hat{u}(E+p/\rho)\end{bmatrix}$$



Governing Equations

The average of the conserved variables over the volume of cell i, $|V_i|$, is defined as

$$\bar{\mathbf{U}}_i = \frac{1}{|V_i|} \iiint_{V_i} \mathbf{U}\, dV$$

$$\frac{\partial \bar{\mathbf{U}}_i}{\partial t} + \frac{1}{|V_i|} \sum_{j=1}^{Nf_i} \vec{\mathbf{F}}(\mathbf{U}) \cdot \vec{n_{fi}}\, S_{fi} = 0$$

V_i : Volume of cell i
Nf_i : Number of faces
$\vec{n_{fi}}$: normal vector of face i
S_{fi} : surface area of face i

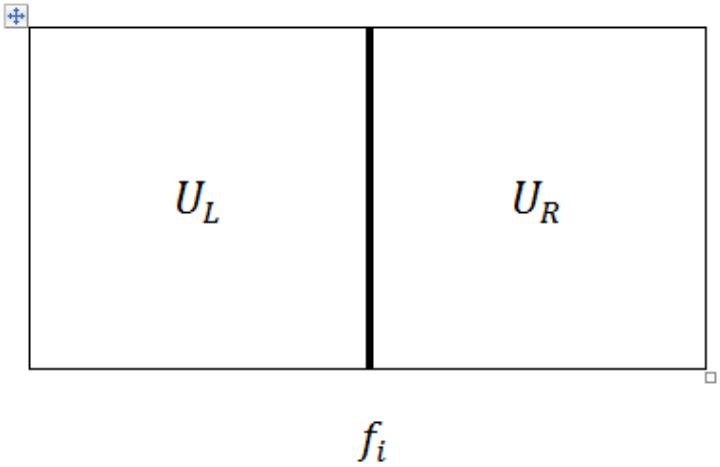
$\vec{\mathbf{F}}(\mathbf{U}) \cdot \vec{n_{fi}}$: flux over the face f_i

Governing Equations

$$\frac{\partial \bar{\mathbf{U}}_i}{\partial t} + \frac{1}{|V_i|} \sum_{j=1}^{Nf_i} \vec{\mathbf{F}}(\mathbf{U}) \cdot \vec{n_{fi}} \, S_{fi} = 0$$

- The equation should be integrated in time $\rightarrow$ Time integration scheme
- The flux over each face, $\vec{\mathbf{F}}(\mathbf{U}) \cdot \vec{n_{fi}}$ should be approximated $\rightarrow$ Flux discretization scheme

Flux discretization scheme



$$\vec{\mathbf{F}}\left(\mathbf{U}_L, \mathbf{U}_R\right) \cdot \vec{n_{fi}}$$

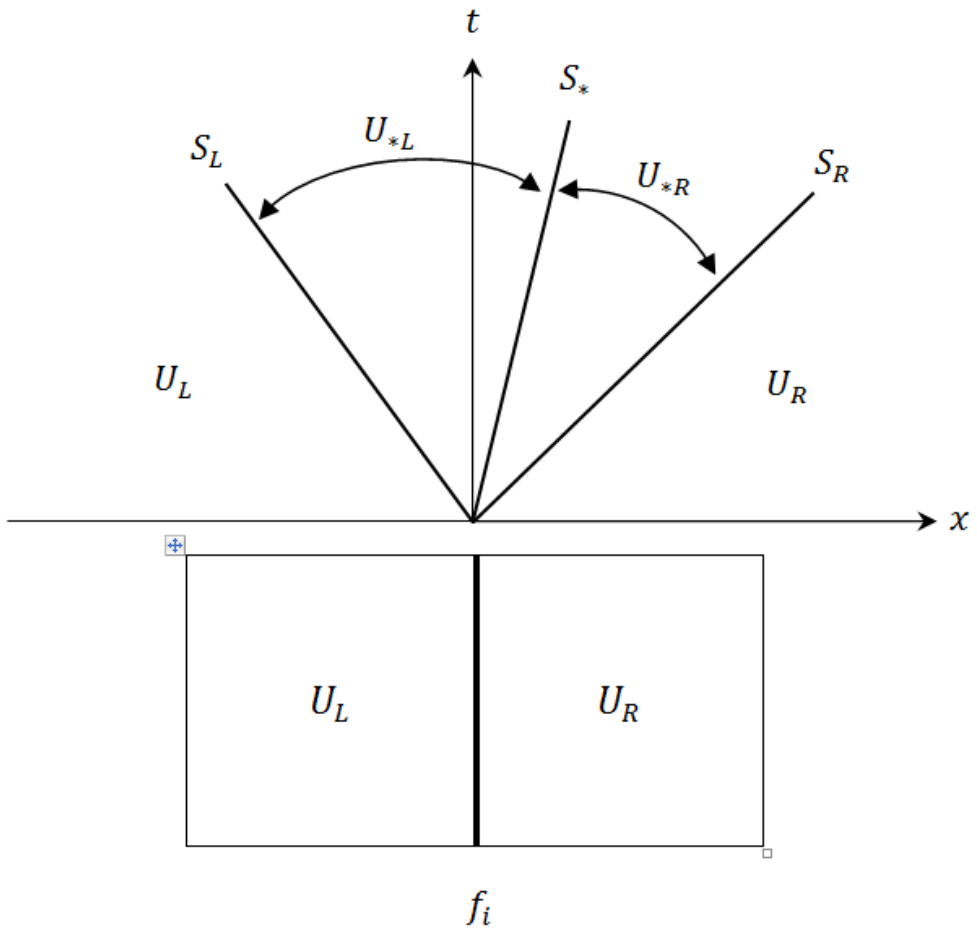
- In dbnsFoam, the flux at each face is calculated by approximate Riemann solvers
- Approximate Riemann solvers: The approximate solution of Riemann problem for the face is used for flux calculation
- Riemann problem : an initial value problem with initial conditions given by

$$U(x,0) = \begin{cases} U_L \text{ for } x \leq 0 \\ U_R \text{ for } x > 0 \end{cases}$$

Flux discretization scheme

- solution of Riemann problem at $x = 0$ is used to calculate the flux through the face

approximate solution of Riemann problem



HLLC-AUSM Scheme

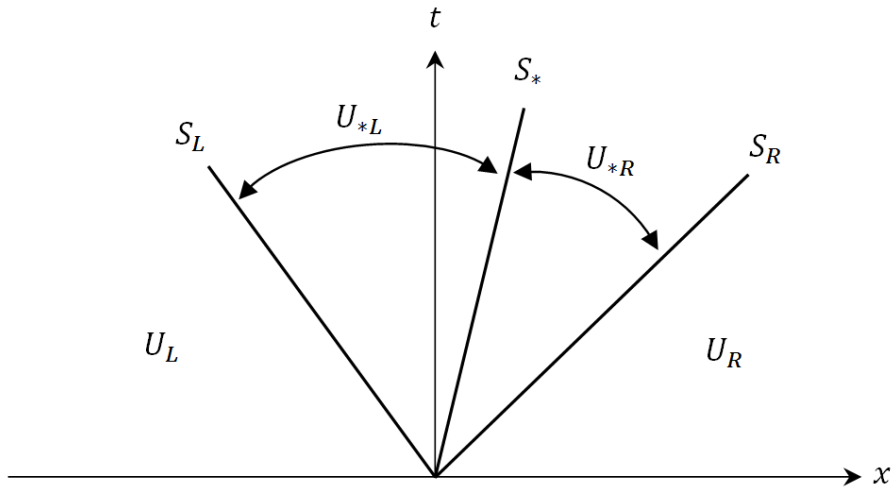
- $\vec{\mathbf{F}}(\mathbf{U}) \cdot \vec{n}_{ij}$ is decomposed into two parts: a convective component and a pressure vector,

$$\vec{\mathbf{F}}(\mathbf{U}) \cdot \vec{n}_{ij} = \dot{m} F_{conv} + F_{pressure}$$

- $F_{pressure} = [0, \bar{p}n_x, \bar{p}n_y, \bar{p}n_z, 0]^T$ is calculated from AUSM scheme
- $\dot{m}$ is calculated from HLLC scheme
- $\mathrm{F}_{conv} = \begin{cases} [1, u_L, v_L, w_L, H_L]^T & for \quad \dot{m} > 0 \\ [1, u_R, v_R, w_R, H_R]^T & for \quad \dot{m} \leq 0 \end{cases}$

HLLC scheme

- The solution of Riemann problem is approximated by three wave structures and four constant regions
- we are looking for the mass flux at $x = 0$ which corresponds to the mass flux through the faces



- $\dot{m} =$

$$\begin{cases} \rho_L u_L & 0 \leq S_L \\ \rho_L u_L + S_L \left(\rho_L \frac{S_L - u_L}{S_L - S_*} - \rho_L \right) & S_L < 0 \leq S_* \\ \rho_R u_R + S_R \left(\rho_R \frac{S_R - u_R}{S_R - S_*} - \rho_R \right) & S_* < 0 \leq S_R \\ \rho_R u_R & S_R < 0 \end{cases}$$

AUSM+-up for all speeds scheme

- the pressure flux part is calculated based on AUSM+-up for all speeds scheme from the following fifth-order polynomial,

$$\bar{p} = \mathcal{P}^{+}_{(5)}(M_L)p_L + \mathcal{P}^{-}_{(5)}(M_R)p_R - K_u\mathcal{P}^{+}_{(5)}(M_L)\mathcal{P}^{-}_{(5)}(M_R)(\rho_L + \rho_R)(u_R$$

- In this polynomial, the pressure functions $\mathcal{P}$ and split Mach numbers $\mathcal{M}$ are defined as

$$\mathcal{P}^{\pm}_{(5)} = \begin{cases} (1/M)\mathcal{M}^{\pm}_{(1)} & |M| \geq 1 \\ \mathcal{M}^{\pm}_{(2)}\left[(\pm 2 - M) \mp 16\gamma M \mathcal{M}^{\mp}_{(2)}\right] & |M| < 1 \end{cases}$$

$$\mathcal{M}^{\pm}_{(1)} = \frac{1}{2}(M \pm |M|)$$

$$\mathcal{M}^{\pm}_{(2)} = \pm\frac{1}{4}(M \pm 1)^2$$

Higher Order Reconstruction

- The numerical flux is calculated using U_L and U_R which are unknown
- The simplest way is to assume that the left and right states at interface are equal to the state at the center of left and right cells.$\rightarrow$ first order
- A second-order spatial accuracy can be achieved by the piece-wise linear reconstruction method

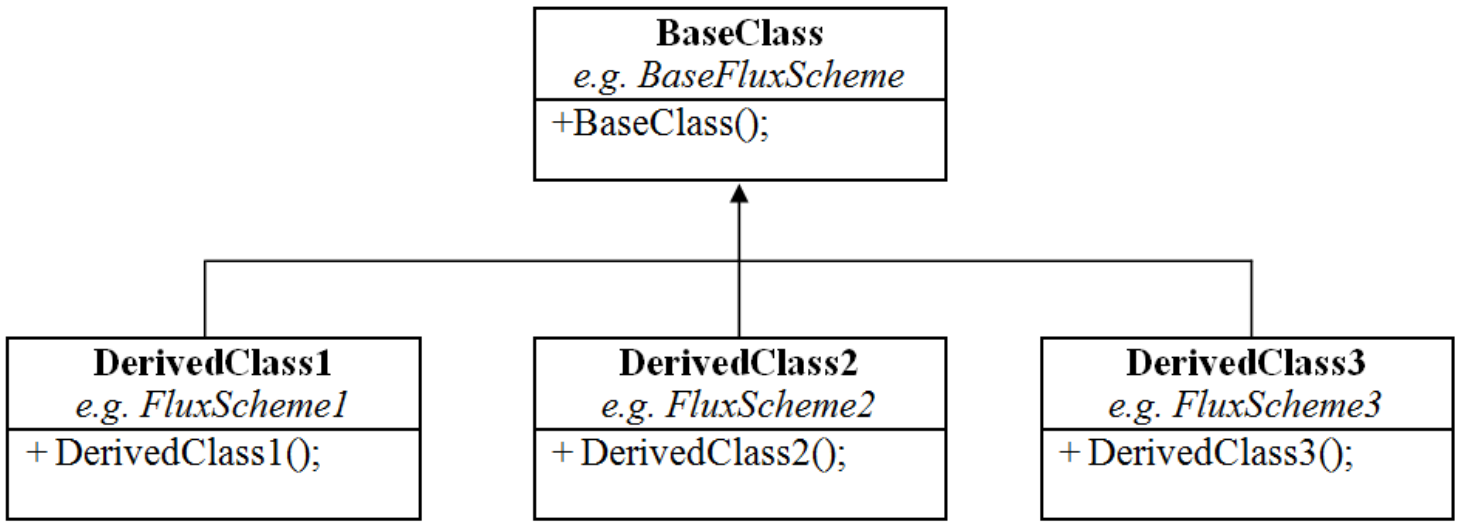
$$\mathbf{U}_L = \bar{\mathbf{U}}_i + \Phi_i \left[(\nabla \mathbf{U})_{cg,i} (\vec{x}_{ij} - \vec{x}_{cg,i}) \right]$$

$$\mathbf{U}_R = \bar{\mathbf{U}}_j + \Phi_j \left[(\nabla \mathbf{U})_{cg,j} (\vec{x}_{ij} - \vec{x}_{cg,j}) \right]$$

- $\Phi \in [0, 1]$ is the limiter function.

On OpenFOAM's Runtime Selection Mechanism

- References
  - `https://openfoamwiki.net/index.php/OpenFOAM_guide/runTimeSelection_mechanism`
- Function



On OpenFOAM's Runtime Selection Mechanism

- a hash table of DerivedClass constructor pointers in BaseClass

Key	Value
DeriveClass1's name	*Pointer to the DeriveClass1()*
DeriveClass2's name	*Pointer to the DeriveClass2()*
DeriveClass3's name	*Pointer to the DeriveClass3()*

- Two macros in BaseClass.H declare and define the hash table of constructors
  - declareRunTimeSelectionTable
  - defineRunTimeSelectionTable

On OpenFOAM's Runtime Selection Mechanism

- a macro in each DerivedClass header file adds the constructor of that DerivedClass to the hash table
  - addToRunTimeSelectionTable
- A selector function defined in BaseClass looks up the constructor in the hash table based on the parameter given during run-time
  - New function

dbnsFoam Solver

- Top-level folder: `$FOAM_SOLVERS/compressible/dbnsFoam`

```
cd $FOAM_SOLVERS/compressible/dbnsFoam
```

- The folder contains

```
createFields.H
dbnsFoam.C
Make
    files
    options
```

dbnsFoam.C

- The main class for flux calculation is the numericFlux class which is included in dbnsFoam.C(line 47):

```
#include "numericFlux.H"
```

- Inside the loop for time integration(line 88), the `computeFlux()` function of the object `dbnsFlux` is called.

```
dbnsFlux.computeFlux();
```

- This function updates the fluxes of all faces using the selected flux scheme and limiter function.
- the object `dbnsFlux` is constructed at the end of the createFields.H file

createFields.H

- In createFields.H file(line 73), an object for the numerical flux is constructed. This is done by the following snippet of code where the function `New` is called.

```
// Create numeric flux
autoPtr<basicNumericFlux> dbnsFluxPtr = basicNumericFlux::New
(
    p,
    U,
    T,
    thermo()
);
basicNumericFlux& dbnsFlux = dbnsFluxPtr();
```

basicNumericFlux class

- The definition and implementation are located in
  `$FOAM_SRC/dbns/basicNumericFlux`
- base class for run-time selectable numerical flux methods
- numerical flux methods are implemented as subclasses of basicNumericFlux class
- basicNumericFlux class includes hash table of pointers to the constructors of basicNumericFlux subclasses(numerical flux methods).
- This hash table is declared and defined by two macros , one in basicNumericFlux.H and one basicNumericFlux.C
  - declareRunTimeSelectionTable
  - defineRunTimeSelectionTable

basicNumericFlux class

- New function : a selector function

```
Foam::autoPtr<Foam::basicNumericFlux> Foam::basicNumericFlux::New
(
    const volScalarField& p,
    const volVectorField& U,
    const volScalarField& T,
    basicThermo& thermo
)
{
    const dictionary& subDict =
        p.mesh().schemesDict().subDict("divSchemes").subDict("dbns")

    word name = word(subDict.lookup("flux")) + "Flux"
        + word(subDict.lookup("limiter")) + "Limiter";

    Info<< "Selecting numericFlux " << name << endl;

    stateConstructorTable::iterator cstrIter =
```

basicNumericFlux class

- New function : a selector function

```
    stateConstructorTable::iterator cstrIter =
        stateConstructorTablePtr_->find(name);

    if (cstrIter == stateConstructorTablePtr_->end())
    {
        FatalErrorIn("basicNumericFlux::New(const fvMesh&)")
            << "Unknown basicNumericFlux type " << name << nl << nl
            << "Valid basicNumericFlux types are:" << nl
            << stateConstructorTablePtr_->sortedToc() << nl
            << exit(FatalError);
    }

    return autoPtr<basicNumericFlux>(cstrIter()(p, U, T, thermo));
}
}
```

numericFlux class

- numericFlux class is the main class for the flux calculation
- The definition and implementation are located in

  `$FOAM_SRC/dbns/numericFlux`
- It is a subclass of basicNumericFlux It is a template class with two templated parameters, `Flux` and `Limiter`.
- In dbnsFoam.C the following member functions of numericFlux are called.
  - `computeFlux()`
  - `rhoFlux()`
  - `rhoUFlux()`
  - `rhoEFlux()`

numericFlux class

- computeFlux()
  - This function constructs the limter functions for pressure, velocity, and temprature fields based on the Limiter parameter.
  - Then, it updates the fluxes rhoFlux_, rhoUFlux_, and rhoEFlux_ for all internal and boundary faces by calling Flux::evaluteFlux

```
Flux::evaluateFlux
(
    rhoFlux_[faceI],
    rhoUFlux_[faceI],
    rhoEFlux_[faceI],
    p_[own] + pLimiter[own]*(deltaRLeft & gradP[own]),
    p_[nei] + pLimiter[nei]*(deltaRRight & gradP[nei]),
    U_[own] + cmptMultiply(ULimiter[own], (deltaRLeft & gradU[own])),
    U_[nei] + cmptMultiply(ULimiter[nei], (deltaRRight & gradU[nei])),
    T_[own] + TLimiter[own]*(deltaRLeft & gradT[own]),
    T_[nei] + TLimiter[nei]*(deltaRRight & gradT[nei]),
    R[own],
    R[nei],
    Cv[own],
    Cv[nei],
    Sf[faceI],
```

numericFlux class

- `rhoFlux()`, `rhoUFlux()`, and `rhoEFlux()`
- These functions return the updated fluxes.

```
//- Return density flux
virtual const surfaceScalarField& rhoFlux() const
{
    return rhoFlux_;
}

//- Return velocity flux
virtual const surfaceVectorField& rhoUFlux() const
{
    return rhoUFlux_;
}

//- Return energy flux
virtual const surfaceScalarField& rhoEFlux() const
{
    return rhoEFlux_;
}
```

numericFlux class

- numericFlux is a template class with two template parameters, Flux and Limiter.
- In order to create an instance of numericFlux class, two template parameters should be provided.
- There are four limiter functions avialable in Foam-extend 4.0. The implementation of these limiter functions are located in

  `$FOAM_SRC/finiteVolume/finiteVolume/gradientLimiters`
- There are five flux schemes available in Foam-extend 4.0. The implementation of these flux schemes are located in

  `$FOAM_SRC/dbns/dbnsFlux`

numericFluxes.H

- The constructors of basicNumericFlux subclasses should be added to the hash table held by basicNumericFlux class.
- This is done in numericFluxes.H by the following macros

```
makeBasicNumericFluxForAllLimiters(rusanovFlux);
makeBasicNumericFluxForAllLimiters(betaFlux);
makeBasicNumericFluxForAllLimiters(roeFlux);
makeBasicNumericFluxForAllLimiters(hllcFlux);
```

makeBasicNumericFluxForAllLimiters(rusanovFlux)

- This macro creates the instances of numericFlux class by setting the flux paramter to rusanovFlux and the limiter parameter to all of the available limiter functions.

```
typedef numericFlux<Flux,Limiter>
    Flux##Limiter;
```

- It adds the pointer to the constructor of each instance into the hash table held by the basicNumericFlux class.

```
addToRunTimeSelectionTable
(
    basicNumericFlux,
    Flux##Limiter,
    state
)
```

- The implementations of different flux schemes are located in

  `$FOAM_SRC/dbns/dbnsFlux`

- There are fives flux schemes in this folder
  - betaFlux
  - hllcALEFlux
  - hllcFlux
  - roeFlux
  - rusanovFlux

hllcFlux Class

- This class has only one member function evaluateFlux

```
void evaluateFlux
(
    scalar& rhoFlux,
    vector& rhoUFlux,
    scalar& rhoEFlux,
    const scalar& pLeft,
    const scalar& pRight,
    const vector& ULeft,
    const vector& URight,
    const scalar& TLeft,
    const scalar& TRight,
    const scalar& RLeft,
    const scalar& RRight,
    const scalar& CvLeft,
    const scalar& CvRight,
    const vector& Sf,
    const scalar& magSf
) const;
```

Implementation of HLLC-AUSM low mach scheme

- Open a new terminal
- Load foam-extend-4.0
- Create a folder with hllcAusmFlux name

```
mkdir hllcAusmFlux
```

- Go to the folder

```
cd hllcAusmFlux
```

- Copy the following files from dbns library

```
cp $FOAM_SRC/dbns/dbnsFlux/hllcFlux/hllcFlux.H .
cp $FOAM_SRC/dbns/dbnsFlux/hllcFlux/hllcFlux.C .
cp $FOAM_SRC/dbns/numericFlux/numericFluxes.C .
```

Implementation of HLLC-AUSM low mach scheme

- Rename hllcFlux.H and hllcFlux.C to hllcAusmFlux.H and hllcAusmFlux.C

```
mv hllcFlux.H hllcAusmFlux.H
mv hllcFlux.C hllcAusmFlux.C
```

- Replace hllcFlux with hllcAusmFlux in the files using the sed command

```
sed -i s/hllcFlux/hllcAusmFlux/g hllcAusmFlux*
sed -i s/hllcFlux/hllcAusmFlux/g numericFluxes.C
```

- Remove the following files in numericFluxes.C

```
#include "rusanovFlux.H"
#include "roeFlux.H"
#include "betaFlux.H"
#include "hllcALEFlux.H"
makeBasicNumericFluxForAllLimiters(rusanovFlux);
makeBasicNumericFluxForAllLimiters(betaFlux);
makeBasicNumericFluxForAllLimiters(roeFlux);
```

Implementation of HLLC-AUSM low mach scheme

- Create `Make` folder containing `files` and `options` files

```
mkdir Make
touch Make/options
touch Make/files
```

- Copy the following lines into `files`

```
hllcAusmFlux.C
numericFluxes.C

LIB = $(FOAM_USER_LIBBIN)/libhllcAusmFlux
```

Implementation of HLLC-AUSM low mach scheme

- Copy the following lines into `options`

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/basic/lnInclude \
    -I$(LIB_SRC)/dbns/lnInclude

LIB_LIBS = \
    -lfiniteVolume \
    -lmeshTools \
    -ldbns
```

Implementation of HLLC-AUSM low mach scheme

- In hllcAusmFlux.C, remove the code in the implementation of evaluateFlux function and copy the following lines.

```
// normal vector
const vector normalVector = Sf/magSf;

// Ratio of specific heat capacities
const scalar kappaLeft = (RLeft + CvLeft)/CvLeft;
const scalar kappaRight = (RRight + CvRight)/CvRight;

// Density
const scalar rhoLeft = pLeft/(RLeft*TLeft);
const scalar rhoRight = pRight/(RRight*TRight);

// DensityVelocity
const vector rhoULeft = rhoLeft*ULeft;
const vector rhoURight = rhoRight*URight;
```

Implementation of HLLC-AUSM low mach scheme

```
// DensityTotalEnergy
const scalar rhoELeft = rhoLeft*(CvLeft*TLeft+0.5*magSqr(UL
const scalar rhoERight = rhoRight*(CvRight*TRight+0.5*magSq

// Compute left and right total enthalpies:
const scalar HLeft = (rhoELeft + pLeft)/rhoLeft;
const scalar HRight = (rhoERight + pRight)/rhoRight;

// Compute qLeft and qRight (q_{l,r} = U_{l,r} \bullet n)
const scalar qLeft = (ULeft & normalVector);
const scalar qRight = (URight & normalVector);

// Speed of sound, for left and right side
const scalar aLeft =
    Foam::sqrt(max(0.0,kappaLeft * pLeft/rhoLeft));
const scalar aRight =
    Foam::sqrt(max(0.0,kappaRight * pRight/rhoRight));
```

Implementation of HLLC-AUSM low mach scheme

```
//  compute signal speeds for face:
    const scalar SLeft  = min(qLeft-aLeft, qRight-aRight);
    const scalar SRight = max(qLeft+aLeft, qRight+aRight);

    const scalar SStar = (rhoRight*qRight*(SRight-qRight)
    - rhoLeft*qLeft*(SLeft - qLeft) + pLeft - pRight )/
    stabilise((rhoRight*(SRight-qRight)-rhoLeft
    *(SLeft-qLeft)),VSMALL);

    //compute HLLC mass flux
    scalar m_dot = 0.0;
     if (pos(SLeft))
    {
        m_dot = rhoLeft*qLeft;
    }
```

Implementation of HLLC-AUSM low mach scheme

```
else if (pos(SStar))
{
    scalar omegaLeft = scalar(1.0)/stabilise((SLeft - SStar
    m_dot = rhoLeft*qLeft + SLeft*(rhoLeft*omegaLeft*(SLeft
}
else if (pos(SRight))
{
    scalar omegaRight = scalar(1.0)/stabilise((SRight - SSt
    m_dot = rhoRight*qRight + SRight*(rhoRight*omegaRight*(
}
else if (neg(SRight))
{
    m_dot = rhoRight*qRight;
}
else
{
    Info << "Error in HLLC Riemann solver" << endl;
}
```

Implementation of HLLC-AUSM low mach scheme

```
//compute pressure from AUSM+-up for all speeds scheme
const scalar Ku    = 0.75;
const scalar aTilde = 0.5*(aLeft+aRight);
const scalar sqrMaDash = (sqr(qLeft)+sqr(qRight))/(2.0*sqr(

const scalar MaInf = 0.01;
const scalar sqrMaZero = min(1.0,max(sqrMaDash,sqr(MaInf)))
const scalar MaZero    = Foam::sqrt(sqrMaZero);

const scalar fa = MaZero*(2.0-MaZero);
const scalar alpha = 3.0/16.0*(-4.0+5.0*sqr(fa));

const scalar MaRelLeft  = qLeft /aTilde;
const scalar MaRelRight = qRight/aTilde;

const scalar magMaRelLeft  = mag(MaRelLeft);
const scalar magMaRelRight = mag(MaRelRight);
```

Implementation of HLLC-AUSM low mach scheme

```
//compute split Mach numbers
const scalar Ma1PlusLeft   = 0.5*(MaRelLeft +magMaRelLeft )
const scalar Ma1MinusRight = 0.5*(MaRelRight-magMaRelRight)

const scalar Ma2PlusLeft   =  0.25*sqr(MaRelLeft +1.0);
const scalar Ma2PlusRight  =  0.25*sqr(MaRelRight+1.0);
const scalar Ma2MinusLeft  = -0.25*sqr(MaRelLeft -1.0);
const scalar Ma2MinusRight = -0.25*sqr(MaRelRight-1.0);

//compute pressure functions
const scalar P5alphaPlusLeft   = ((magMaRelLeft  >= 1.0) ?
    (Ma1PlusLeft/MaRelLeft)    :
    (Ma2PlusLeft  *(( 2.0-MaRelLeft)
     -16.0*alpha*MaRelLeft *Ma2MinusLeft )));
const scalar P5alphaMinusRight = ((magMaRelRight >= 1.0) ?
    (Ma1MinusRight/MaRelRight) :
    (Ma2MinusRight*((-2.0-MaRelRight)
     +16.0*alpha*MaRelRight*Ma2PlusRight)));
```

Implementation of HLLC-AUSM low mach scheme

```
const scalar pU = -Ku*P5alphaPlusLeft*P5alphaMinusRight
*(rhoLeft+rhoRight)*(fa*aTilde)*(qRight-qLeft);
scalar pTilde = pLeft*P5alphaPlusLeft
+ pRight*P5alphaMinusRight + pU;

//compute fluxes
 if(m_dot>0)
{
    rhoFlux = m_dot*magSf;
    rhoUFlux = (m_dot*ULeft+pTilde*normalVector)*magSf;
    rhoEFlux = (m_dot*HLeft)*magSf;
}
else
{
    rhoFlux = m_dot*magSf;
    rhoUFlux = (m_dot*URight+pTilde*normalVector)*magSf;
    rhoEFlux = (m_dot*HRight)*magSf;
}
```

Implementation of HLLC-AUSM low mach scheme

- Compile the code

```
wmake libso
```

Simulation of 1D Riemann problem for gas flow

- Go to run folder

  `run`
- Copy the shockTube tutorial for dbnsFoam to run folder

  `cp -r $FOAM_TUTORIALS/compressible/dbnsFoam/shockTube/ .`
- In controlDict file, the compiled library of HLLC-AUSM scheme must be linked to dbnsFoam solver at run-time.This is done by adding the following line to the end of controlDict file.

  `libs ( "libhllcAusmFlux.so" );`

Simulation of 1D Riemann problem for gas flow

- we should change the dbns subdictionay in fvSchemes file.

```
dbns
{
flux            hllcAusm;
limiter         BarthJespersen;
}
```

- we should create the mesh using blockMesh utility

```
blockMesh >& log&
```

- we should intilize the domain using setField utility

```
setFields >& log&
```

- Then we run dbnsFoam solver

```
dbnsFoam >& log&
```

Simulation of 1D Riemann problem for gas flow

- We can view the results by paraFoam

Thank you for your attention!