```
/** \file equationOfState.h
 *
 * \author Thibault Bridel-Bertomeu
 * \date Sat 28 Mar 2020 09:45:50 PM CET
 */

#ifndef EQUATIONOFSTATE_H
#define EQUATIONOFSTATE_H

#include "main.h"
#ifdef with_petsc
#include "petscinterface.h"
#endif

void primCons(const double pVar[NVAR], double cVar[NVAR]);
void consPrim(const double cVar[NVAR], double pVar[NVAR]);
void consChar(double cVar[NVAR], double charac[3], double pVarRef[NVAR]);
void charCons(double charac[3], double cVar[NVAR], double pVarRef[NVAR]);

void getViscosity(const double pVar[NVAR], double *viscoDyn);

#ifdef with_petsc

/** Conversion conservative to primitive variables */
PetscErrorCode u2w(Vec, Vec*, PetscBool);
PetscErrorCode u2wLocal(Vec, Vec*, PetscBool);
/** Conversion primitive to conservative variables */
PetscErrorCode w2u(Vec, Vec*, PetscBool);
PetscErrorCode w2uLocal(Vec, Vec*, PetscBool);
/** Equation of state for perfect gas, get pressure */
PetscErrorCode pressure_PG(const PetscReal, const EulerNode*,
PetscReal*);
/** Equation of state for perfect gas, get energy */
PetscErrorCode energy_PG(const PetscReal, const EulerNode*, PetscReal,
PetscReal*);
/** Equation of state for perfect gas, get sound velocity */
PetscErrorCode speedOfSound_PG(const PetscReal*, const EulerNode*,
PetscReal*);
/** Law of dynamic viscosity for a perfect gas */
PetscErrorCode lawOfViscosity_PG(const PetscReal, const NSNode*,
PetscReal*);
/** Monitor for Euler system of equations */
PetscErrorCode physicsMonitor_Euler(Model mod, PetscReal time, const
PetscReal *coord, const PetscScalar *xx, PetscReal *f, void *ctx);
/** Monitor for NS system of equations */
PetscErrorCode physicsMonitor_NS(Model mod, PetscReal time, const
PetscReal *coord, const PetscScalar *xx, PetscReal *f, void *ctx);

/*! \def _NavierStokes2DEigenvalues_
  Compute the eigenvalues, given a solution vector in terms of the
conserved variables. The eigenvalues are returned
  as a matrix D whose diagonal values are the eigenvalues. Admittedly,
this is inefficient. The matrix D is stored in
  a row-major format.
*/
#define _NavierStokes2DEigenvalues_(u,D,gamma,dir,dof) \
{ \
    double  rho,vx,vy,e,P,c,vn,vsq; \
```

```
    rho = u[0]; \
    vx  = u[1] / rho; \
    vy  = u[2] / rho; \
    e   = u[3]; \
    vsq = (vx*vx) + (vy*vy); \
    P   = (e - 0.5*rho*vsq) * (gamma-1.0); \
    c   = sqrt(gamma*P/rho); \
    if      (dir == 0) vn = vx; \
    else if (dir == 1) vn = vy; \
    else               vn = 0.0; \
    if (dir == 0) {\
        D[0*dof+0] = vn-c;   D[0*dof+1] = 0;     D[0*dof+2] = 0;
D[0*dof+3] = 0;  \
        D[1*dof+0] = 0;      D[1*dof+1] = vn+c; D[1*dof+2] = 0;
D[1*dof+3] = 0;  \
        D[2*dof+0] = 0;      D[2*dof+1] = 0;     D[2*dof+2] = vn;
D[2*dof+3] = 0;  \
        D[3*dof+0] = 0;      D[3*dof+1] = 0;     D[3*dof+2] = 0;
D[3*dof+3] = vn; \
    } else if (dir == 1) { \
        D[0*dof+0] = vn-c;   D[0*dof+1] = 0;     D[0*dof+2] = 0;
D[0*dof+3] = 0;  \
        D[1*dof+0] = 0;      D[1*dof+1] = vn;    D[1*dof+2] = 0;
D[1*dof+3] = 0;  \
        D[2*dof+0] = 0;      D[2*dof+1] = 0;     D[2*dof+2] = vn+c;
D[2*dof+3] = 0;  \
        D[3*dof+0] = 0;      D[3*dof+1] = 0;     D[3*dof+2] = 0;
D[3*dof+3] = vn; \
    }\
}

/*! \def _NavierStokes2DLeftEigenvectors_
  Compute the left eigenvectors, given a solution vector in terms of the
conserved variables. The eigenvectors are
  returned as a matrix L whose rows correspond to each eigenvector. The
matrix L is stored in the row-major format.
  \n\n
  Reference:
  + Rohde, "Eigenvalues and eigenvectors of the Euler equations in
general geometries", AIAA Paper 2001-2609,
    http://dx.doi.org/10.2514/6.2001-2609
*/
#define _NavierStokes2DLeftEigenvectors_(u,L,gamma,dir,dof) \
{ \
    double  ga = gamma, ga_minus_one=ga-1.0; \
    double  rho,vx,vy,e,P,a,un,ek,vsq; \
    double  nx = 0,ny = 0; \
    rho = u[0]; \
    vx  = u[1] / rho; \
    vy  = u[2] / rho; \
    e   = u[3]; \
    vsq  = (vx*vx) + (vy*vy); \
    P   = (e - 0.5*rho*vsq) * (ga-1.0); \
    ek = 0.5 * (vx*vx + vy*vy); \
    a = sqrt(ga * P / rho); \
    if (dir == 0) { \
        un = vx; \
        nx = 1.0; \
```

```
      L[0*dof+0] = (ga_minus_one*ek + a*un) / (2*a*a); \
      L[0*dof+1] = ((-ga_minus_one)*vx - a*nx) / (2*a*a); \
      L[0*dof+2] = ((-ga_minus_one)*vy - a*ny) / (2*a*a); \
      L[0*dof+3] = ga_minus_one / (2*a*a); \
      L[3*dof+0] = (a*a - ga_minus_one*ek) / (a*a); \
      L[3*dof+1] = (ga_minus_one*vx) / (a*a); \
      L[3*dof+2] = (ga_minus_one*vy) / (a*a); \
      L[3*dof+3] = (-ga_minus_one) / (a*a); \
      L[1*dof+0] = (ga_minus_one*ek - a*un) / (2*a*a); \
      L[1*dof+1] = ((-ga_minus_one)*vx + a*nx) / (2*a*a); \
      L[1*dof+2] = ((-ga_minus_one)*vy + a*ny) / (2*a*a); \
      L[1*dof+3] = ga_minus_one / (2*a*a); \
      L[2*dof+0] = (vy - un*ny) / nx; \
      L[2*dof+1] = ny; \
      L[2*dof+2] = (ny*ny - 1.0) / nx; \
      L[2*dof+3] = 0.0; \
  } else if (dir == 1) {  \
      un = vy;  \
      ny = 1.0; \
      L[0*dof+0] = (ga_minus_one*ek+a*un) / (2*a*a); \
      L[0*dof+1] = ((1.0-ga)*vx - a*nx) / (2*a*a); \
      L[0*dof+2] = ((1.0-ga)*vy - a*ny) / (2*a*a); \
      L[0*dof+3] = ga_minus_one / (2*a*a); \
      L[3*dof+0] = (a*a-ga_minus_one*ek) / (a*a); \
      L[3*dof+1] = ga_minus_one*vx / (a*a); \
      L[3*dof+2] = ga_minus_one*vy / (a*a); \
      L[3*dof+3] = (1.0 - ga) / (a*a); \
      L[2*dof+0] = (ga_minus_one*ek-a*un) / (2*a*a); \
      L[2*dof+1] = ((1.0-ga)*vx + a*nx) / (2*a*a); \
      L[2*dof+2] = ((1.0-ga)*vy + a*ny) / (2*a*a); \
      L[2*dof+3] = ga_minus_one / (2*a*a); \
      L[1*dof+0] = (un*nx-vx) / ny; \
      L[1*dof+1] = (1.0 - nx*nx) / ny; \
      L[1*dof+2] = - nx; \
      L[1*dof+3] = 0; \
   } \
}

/*! \def _NavierStokes2DRightEigenvectors_
  Compute the right eigenvectors, given a solution vector in terms of the
conserved variables. The eigenvectors are
  returned as a matrix R whose columns correspond to each eigenvector.
The matrix R is stored in the row-major format.
  \n\n
  Reference:
  + Rohde, "Eigenvalues and eigenvectors of the Euler equations in
general geometries", AIAA Paper 2001-2609,
    http://dx.doi.org/10.2514/6.2001-2609
*/
#define _NavierStokes2DRightEigenvectors_(u,R,gamma,dir,dof) \
{ \
    double  ga = gamma, ga_minus_one = ga-1.0; \
    double  rho,vx,vy,e,P,un,ek,a,h0,vsq; \
    double  nx = 0,ny = 0; \
    rho = u[0]; \
    vx  = u[1] / rho; \
    vy  = u[2] / rho; \
    e   = u[3]; \
```

```
  vsq = (vx*vx) + (vy*vy); \
  P   = (e - 0.5*rho*vsq) * (ga-1.0); \
  ek  = 0.5 * (vx*vx + vy*vy); \
  a   = sqrt(ga * P / rho); \
  h0  = a*a / ga_minus_one + ek; \
  if (dir == 0) { \
      un = vx; \
      nx = 1.0; \
      R[0*dof+0] = 1.0; \
      R[1*dof+0] = vx - a*nx; \
      R[2*dof+0] = vy - a*ny; \
      R[3*dof+0] = h0 - a*un; \
      R[0*dof+3] = 1.0; \
      R[1*dof+3] = vx; \
      R[2*dof+3] = vy; \
      R[3*dof+3] = ek; \
      R[0*dof+1] = 1.0; \
      R[1*dof+1] = vx + a*nx; \
      R[2*dof+1] = vy + a*ny; \
      R[3*dof+1] = h0 + a*un; \
      R[0*dof+2] = 0.0; \
      R[1*dof+2] = ny; \
      R[2*dof+2] = -nx; \
      R[3*dof+2] = vx*ny - vy*nx; \
  } else if (dir == 1) { \
      un = vy; \
       ny = 1.0; \
      R[0*dof+0] = 1.0; \
      R[1*dof+0] = vx - a*nx; \
      R[2*dof+0] = vy - a*ny; \
      R[3*dof+0] = h0 - a*un; \
      R[0*dof+3] = 1.0; \
      R[1*dof+3] = vx; \
      R[2*dof+3] = vy; \
      R[3*dof+3] = ek; \
      R[0*dof+2] = 1.0; \
      R[1*dof+2] = vx + a*nx; \
      R[2*dof+2] = vy + a*ny; \
      R[3*dof+2] = h0 + a*un; \
      R[0*dof+1] = 0; \
      R[1*dof+1] = ny; \
      R[2*dof+1] = -nx; \
      R[3*dof+1] = vx*ny-vy*nx; \
  } \
}

#endif

#endif
```