A metric-based mesh adaptation for LES flow calculations

B. Sauvage

Motivation

Inspired by the dynamic Germano analysis, Toosi and Larson proposed a method for adapting the mesh to LES formulation. In order to minimize

$$\mathcal{F}_{i,\widehat{\overline{\Delta}}} = \frac{\partial}{\partial x_j}\left(\tau^{model}_{ij,\widehat{\overline{\Delta}}}(\widehat{\overline{u}}) - \widehat{\tau^{model}_{ij,\overline{\Delta}}(\overline{u})} - \widehat{\overline{u}_i\overline{u}_j} + \widehat{\overline{u}}_i\widehat{\overline{u}}_j\right),$$

we consider the directional test filters $\widehat{\cdot}^{\mathbf{n}}$, in direction $\mathbf{n}_x$, $\mathbf{n}_y$ and $\mathbf{n}_z$, that gives us

$$\widehat{\overline{\mathcal{F}}}_i^{(\mathbf{n})}(\mathbf{x}) = \frac{\partial}{\partial x_j}\left(\tau^{model}_{ij}(\widehat{\overline{u}}^{(\mathbf{n})}) - \widehat{\tau^{model}_{ij}(\overline{u})}^{(\mathbf{n})} - \widehat{\overline{u}_i\overline{u}_j}^{(\mathbf{n})} + \widehat{\overline{u}}_i^{(\mathbf{n})}\widehat{\overline{u}}_j^{(\mathbf{n})}\right),$$

and we minimize instead the error functional

$$e(\overline{\Delta}_{\mathbf{n}_x}, \overline{\Delta}_{\mathbf{n}_y}, \overline{\Delta}_{\mathbf{n}_z}) = \int_\Omega \left(\langle \widehat{\overline{\mathcal{F}}}_i^{(\mathbf{n}_x)}, \widehat{\overline{\mathcal{F}}}_i^{(\mathbf{n}_x)}\rangle + \langle \widehat{\overline{\mathcal{F}}}_i^{(\mathbf{n}_y)}, \widehat{\overline{\mathcal{F}}}_i^{(\mathbf{n}_y)}\rangle + \langle \widehat{\overline{\mathcal{F}}}_i^{(\mathbf{n}_z)}, \widehat{\overline{\mathcal{F}}}_i^{(\mathbf{n}_z)}\rangle\right)^{\frac{1}{2}} d\mathbf{x}\,.$$

to improve the mesh size in each direction.

Notations

Riemannian metric : symmetric positive matrix 3 × 3 field $(\mathcal{M}(\mathbf{x}))_{x\in\Omega}$

$$\mathcal{M} : \mathbf{x} \in \Omega \mapsto \mathcal{M}(\mathbf{x}) = \mathcal{R}(\mathbf{x})\Lambda(\mathbf{x})^t\mathcal{R}(\mathbf{x}),$$

where

$$\Lambda(\mathbf{x}) = \begin{pmatrix} \lambda_1(\mathbf{x}) & & \\ & \lambda_2(\mathbf{x}) & \\ & & \lambda_3(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} h_1^{-2}(\mathbf{x}) & & \\ & h_2^{-2}(\mathbf{x}) & \\ & & h_3^{-2}(\mathbf{x}) \end{pmatrix},$$

$\mathcal{R}(\mathbf{x})$ is an orthonormal matrix provinding the local orientation of mesh stretching, $h_i(\mathbf{x})$ the local mesh sizes along the principal directions $\mathbf{p}_{1,\mathcal{M}}, \mathbf{p}_{1,\mathcal{M}}, \mathbf{p}_{1,\mathcal{M}}$ of $\mathcal{M}$:

$$\mathbf{p}_{k,\mathcal{M}}(\mathbf{x}) = \mathcal{R}(\mathbf{x})e_k{}^t\mathcal{R}(\mathbf{x}), \quad e_1, e_2, e_3 \text{ the three Cartesian unitary vectors.}$$

Case of a metric-based anisotropic mesh (and compressible LES)

This time we show that the error source term becomes in the compressible case

$$\widehat{\overline{\mathcal{F}}}_i^{(\mathbf{n}_k)}(\mathbf{x}) = \frac{\partial}{\partial x_j}\left((C_s\overline{\Delta}^2)\widehat{\overline{\rho}|\tilde{S}|\tilde{P}_{ij}}^{(\mathbf{n}_k)} - C_s(\widehat{\Delta}^{(\mathbf{n}_k)})^2\,\widehat{\overline{\rho}}^{(\mathbf{n}_k)}\,\widehat{|\tilde{S}|}^{(\mathbf{n}_k)}\,\widehat{\tilde{P}}_{ij}^{(\mathbf{n}_k)} \right.$$

$$\left. -\widehat{\overline{\rho}\tilde{u}_i\tilde{u}_j}^{(\mathbf{n}_k)} + \frac{1}{\widehat{\overline{\rho}}^{(\mathbf{n}_k)}}\left(\widehat{\overline{\rho}\tilde{u}_i}^{(\mathbf{n}_k)}\widehat{\overline{\rho}\tilde{u}_j}^{(\mathbf{n}_k)}\right) + \frac{1}{3}\left(\widehat{\overline{\rho}\tilde{u}_\ell\tilde{u}_\ell}^{(\mathbf{n}_k)} - \frac{1}{\widehat{\overline{\rho}}^{(\mathbf{n}_k)}}\left(\widehat{\overline{\rho}\tilde{u}_\ell}^{(\mathbf{n}_k)}\widehat{\overline{\rho}\tilde{u}_\ell}^{(\mathbf{n}_k)}\right)\right)\delta_{ij}\right),$$

Let note $\mathcal{G}_k = \overline{\Delta}^{-2}\left(\langle\widehat{\overline{\mathcal{F}}}_i^{(\mathbf{n}_k)}, \widehat{\overline{\mathcal{F}}}_i^{(\mathbf{n}_k)}\rangle\right)^{\frac{1}{2}}$, we shall find the metric $\mathcal{M}(\mathbf{x})$ which minimizes

$$\begin{cases} e(\Delta_1,\Delta_2,\Delta_3) = \int_\Omega \left(\mathcal{G}_1^2\Delta_1^4 + \mathcal{G}_2^2\Delta_2^4 + \mathcal{G}_3^2\Delta_3^4\right)^{\frac{1}{2}} d\mathbf{x}, \\ \mathcal{C}(\mathcal{M}) = \int_\Omega \overline{\Delta}^3(h_1h_2h_3\Delta_1\Delta_2\Delta_3)^{-1}d\mathbf{x} = N. \end{cases}$$

Case of a metric-based anisotropic mesh (and compressible LES)

Finally the solution of our problem is

$$\Delta_k^{opt}(\mathbf{x}) = (\mathcal{G}_1(\mathbf{x})\mathcal{G}_2(\mathbf{x})\mathcal{G}_3(\mathbf{x}))^{\frac{1}{6}} (\mathcal{G}_k(\mathbf{x}))^{-\frac{1}{2}} \left(\int_\Omega K^{\frac{3}{5}}(\mathbf{x}')d\mathbf{x}'\right)^{\frac{1}{3}} K(\mathbf{x})^{-\frac{1}{5}} N^{-\frac{1}{3}},$$

with

$$K(\mathbf{x}) = \left(\sum_{k=1}^{3} \mathcal{G}_k^2(\mathbf{x}) \, (\mathcal{G}_1(\mathbf{x})\mathcal{G}_2(\mathbf{x})\mathcal{G}_3(\mathbf{x}))^{\frac{2}{3}} (\mathcal{G}_k(\mathbf{x}))^{-2}\right)^{\frac{1}{2}}.$$

and the optimal metric writes

$$\mathcal{M}^{opt}(\mathbf{x}) = \mathcal{R}_M(\mathbf{x}) \begin{pmatrix} \left(\frac{h_1^M \Delta_1}{\overline{\Delta}}\right)^{-2}(\mathbf{x}) & & \\ & \left(\frac{h_2^M \Delta_2}{\overline{\Delta}}\right)^{-2}(\mathbf{x}) & \\ & & \left(\frac{h_3^M \Delta_3}{\overline{\Delta}}\right)^{-2}(\mathbf{x}) \end{pmatrix} {}^t\mathcal{R}_M(\mathbf{x}).$$

First implementation explanation

The following is a brief explanation of the implementation of the expression :

$$\widehat{\overline{\mathcal{F}}_i}^{(\mathbf{n}_k)}(\mathbf{x}) = \frac{\partial}{\partial x_j}\Bigg((C_s\overline{\Delta}^2)\widehat{\overline{\rho}|\tilde{S}|\tilde{P}_{ij}}^{(\mathbf{n}_k)} - C_s(\widehat{\Delta}^{(\mathbf{n}_k)})^2\,\widehat{\overline{\rho}}^{(\mathbf{n}_k)}\,\widehat{|\tilde{S}|}^{(\mathbf{n}_k)}\widehat{\tilde{P}_{ij}}^{(\mathbf{n}_k)}$$

$$-\widehat{\overline{\rho}\tilde{u}_i\tilde{u}_j}^{(\mathbf{n}_k)} + \frac{1}{\widehat{\overline{\rho}}^{(\mathbf{n}_k)}}\left(\widehat{\overline{\rho}\tilde{u}_i}^{(\mathbf{n}_k)}\widehat{\overline{\rho}\tilde{u}_j}^{(\mathbf{n}_k)}\right) + \frac{1}{3}\left(\widehat{\overline{\rho}\tilde{u}_\ell\tilde{u}_\ell}^{(\mathbf{n}_k)} - \frac{1}{\widehat{\overline{\rho}}^{(\mathbf{n}_k)}}\left(\widehat{\overline{\rho}\tilde{u}_\ell}^{(\mathbf{n}_k)}\widehat{\overline{\rho}\tilde{u}_\ell}^{(\mathbf{n}_k)}\right)\right)\delta_{ij}\Bigg),$$

with

$$\widehat{w(is)}^{(\mathbf{n}_k)} = \sum_{jt\ni is}\sum_{js\in jt} vol(jt)\left|\left\langle \frac{\mathbf{isjs}}{|\mathbf{isjs}|}, \mathbf{n}_k\right\rangle\right| w(js)\left(\sum_{jt\ni is}\sum_{js\in jt} vol(jt)\left|\left\langle \frac{\mathbf{isjs}}{|\mathbf{isjs}|}, \mathbf{n}_k\right\rangle\right|\right)^{-1}.$$

Code(1) (L24-L76)

Dans la première partie de notre première fonction on calcule les quantités LES

$$\tilde{S}_{ij} = \frac{1}{2}\left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i}\right),$$

$$\tilde{P}_{ij} = 2\tilde{S}_{ij} - \frac{2}{3}\tilde{S}_{kk}\delta_{ij},$$

et

$$|\tilde{S}| = \sqrt{2\tilde{S}_{ij}\tilde{S}_{ij}}.$$

A noter que la boucle est sur la totalité des éléments, il suffit de boucler sur les éléments contenants i. A voir comment la fonction sera intégrée dans le code.

```
for (sizet k = 0; k < nbElements; k++) {
    pbEF3D->getGradBaseFunct(k, &ax, &ay, &az);
    vol     = element[k]->getVolume();
    id1     = element[k]->point[0]->itsID;
    id2     = element[k]->point[1]->itsID;
    id3     = element[k]->point[2]->itsID;
    id4     = element[k]->point[3]->itsID;

    dudx    = ax[0]*vx[id1] + ax[1]*vx[id2] + ax[2]*vx[id3] + ax[3]*vx[id4];
    dudy    = ay[0]*vx[id1] + ay[1]*vx[id2] + ay[2]*vx[id3] + ay[3]*vx[id4];
    dudz    = az[0]*vx[id1] + az[1]*vx[id2] + az[2]*vx[id3] + az[3]*vx[id4];

    dvdx    = ax[0]*vy[id1] + ax[1]*vy[id2] + ax[2]*vy[id3] + ax[3]*vy[id4];
    dvdy    = ay[0]*vy[id1] + ay[1]*vy[id2] + ay[2]*vy[id3] + ay[3]*vy[id4];
    dvdz    = az[0]*vy[id1] + az[1]*vy[id2] + az[2]*vy[id3] + az[3]*vy[id4];

    dwdx    = ax[0]*vz[id1] + ax[1]*vz[id2] + ax[2]*vz[id3] + ax[3]*vz[id4];
    dwdy    = ay[0]*vz[id1] + ay[1]*vz[id2] + ay[2]*vz[id3] + ay[3]*vz[id4];
    dwdz    = az[0]*vz[id1] + az[1]*vz[id2] + az[2]*vz[id3] + az[3]*vz[id4];

    rhoL    = rho[id1] + rho[id2] + rho[id3] + rho[id4];

    //div      = dudx + dvdy + dwdz;

    Sxx     = 0.5 * (dudx + dudx);
    Sxy     = 0.5 * (dudy + dvdx);
    Sxz     = 0.5 * (dudz + dwdx);

    Syx     = 0.5 * (dvdx + dudy);
    Syy     = 0.5 * (dvdy + dvdy);
    Syz     = 0.5 * (dvdz + dwdy);

    Szx     = 0.5 * (dwdx + dudz);
    Szy     = 0.5 * (dwdy + dvdz);
    Szz     = 0.5 * (dwdz + dwdz);

    volume[k]   = domaine.element[k]->getVolume();
    //lenghtLES   = pow(volume[k], 2./3.);

    normeS[k]   = sqrt(2*(Sxx*Sxx + Syy*Syy + Szz*Szz
                + 2*Sxy*Sxy + 2*Sxz*Sxz + 2*Syz*Syz));

    Pkk[k] = Sxx + Syy + Szz;
    Pxx[k] = 2 * Sxx - 2 * us3 * Pkk;
    Pyy[k] = 2 * Syy - 2 * us3 * Pkk;
    Pzz[k] = 2 * Szz - 2 * us3 * Pkk;
    Pxy[k] = 2 * Sxy;
    Pxz[k] = 2 * Sxz;
    Pyx[k] = 2 * Syx;
    Pyz[k] = 2 * Syz;
    Pzx[k] = 2 * Szx;
    Pzy[k] = 2 * Szy;

}//end element loop
```

Code(2) (L83-L139)

Dans une deuxième partie on ramène les valeurs LES sur les sommets !

La boucle sur i n'est surement pas nécessaire, le principal c'est d'être au point i et ses voisins.

On calcule les valeurs LES sur les voisins car on va calculer une divergence à la fin.

```
for (i = 0; i < domaine.nbPoints; i++) {

    double NbNeighbors = domaine.areteFromNode[i].size();
    double VolCell[NbNeighbors + 1];
    double VolCellTest[NbNeighbors + 1];
    double normeScell[NbNeighbors + 1];
    double Pxxcell[NbNeighbors + 1], Pxycell[NbNeighbors + 1],
           Pxzcell[NbNeighbors + 1];
    double Pyxcell[NbNeighbors + 1], Pyycell[NbNeighbors + 1],
           Pyzcell[NbNeighbors + 1];
    double Pzxcell[NbNeighbors + 1], Pzycell[NbNeighbors + 1],
           Pzzcell[NbNeighbors + 1];
    double pM_1[NbNeighbors + 1][3], pM_2[NbNeighbors + 1][3],
           pM_3[NbNeighbors + 1][3];
    double n_1[NbNeighbors + 1][3], n_2[NbNeighbors + 1][3],
           n_3[NbNeighbors + 1][3];

    //Liste des voisins:
    double idn[NbNeighbors + 1];
    idn[0] = i;
    for(sizet l = 0; l < NbNeighbors; l++){
      if (domaine.areteFromNode[i][l]->p1->itsID == i) {
        idn[l+1] = domaine.areteFromNode[i][l]->p2->itsID;
      }
      if (domaine.areteFromNode[i][l]->p2->itsID == i) {
        idn[l+1] = domaine.areteFromNode[i][l]->p1->itsID;
      }
    }

    for (sizet l = 0; l < NbNeighbors+1; l++){
      for (sizet k = 0; k < domaine.ver2Tet[idn[l]].size(); k++) {

            Tetk = mesh.ver2Tet[idn[l]][k]->itsRef;


            VolCell[idn[l]] += 0.25 * volume[Tetk];
            VolCellTest[idn[l]] += volume[Tetk];

            normeScell[idn[l]] += volume[Tetk] * normeS[Tetk];

            Pxxcell[idn[l]] += volume[Tetk] * Pxx[Tetk];
            Pyycell[idn[l]] += volume[Tetk] * Pyy[Tetk];
            Pzzcell[idn[l]] += volume[Tetk] * Pzz[Tetk];
            Pxycell[idn[l]] += volume[Tetk] * Pxy[Tetk];
            Pxzcell[idn[l]] += volume[Tetk] * Pxz[Tetk];
            Pyxcell[idn[l]] += volume[Tetk] * Pyx[Tetk];
            Pyzcell[idn[l]] += volume[Tetk] * Pyz[Tetk];
            Pzxcell[idn[l]] += volume[Tetk] * Pzx[Tetk];
            Pzycell[idn[l]] += volume[Tetk] * Pzy[Tetk];
      }

      normeScell[idn[l]] = normeScell[idn[l]]/VolCellTest[idn[l]];
      Pxxcell[idn[l]] = Pxxcell[idn[l]]/VolCellTest[idn[l]];
      Pyycell[idn[l]] = Pyycell[idn[l]]/VolCellTest[idn[l]];
      Pzzcell[idn[l]] = Pzzcell[idn[l]]/VolCellTest[idn[l]];
      Pxycell[idn[l]] = Pxycell[idn[l]]/VolCellTest[idn[l]];
      Pxzcell[idn[l]] = Pxzcell[idn[l]]/VolCellTest[idn[l]];
      Pyxcell[idn[l]] = Pyxcell[idn[l]]/VolCellTest[idn[l]];
      Pyzcell[idn[l]] = Pyzcell[idn[l]]/VolCellTest[idn[l]];
      Pzxcell[idn[l]] = Pzxcell[idn[l]]/VolCellTest[idn[l]];
      Pzycell[idn[l]] = Pzycell[idn[l]]/VolCellTest[idn[l]];
```

Code(3) (L142-L173)

On calcule les directions principales

$$\mathbf{p}_{k,\mathcal{M}}(\mathbf{x}) = \mathcal{R}(\mathbf{x})\,\mathbf{e}_k{}^t\mathcal{R}(\mathbf{x}),$$

puis

$$\mathbf{n}_k(\mathbf{x}) = 2\overline{\Delta}\mathbf{p}_{k,\mathcal{M}}(\mathbf{x}).$$

eigVec[] est une entrée de notre fonction, qui n'est pas encore bien définie.

```
/*
On Calcul :

        [eigVec[0] eigVec[3] eigVec[6]]        [eigVec[0] eigVec[1] eigVec[2]]
p_k,M = [eigVec[1] eigVec[4] eigVec[7]]*e_k*[eigVec[3] eigVec[4] eigVec[5]]
        [eigVec[2] eigVec[5] eigVec[8]]        [eigVec[6] eigVec[7] eigVec[8]]

*/

pM_1[idn[1]][0] = eigVec[0] * eigVec[0] + eigVec[1] * eigVec[3]
                + eigVec[2] * eigVec[6];
pM_1[idn[1]][1] = eigVec[0] * eigVec[1] + eigVec[1] * eigVec[4]
                + eigVec[2] * eigVec[7];
pM_1[idn[1]][2] = eigVec[0] * eigVec[2] + eigVec[1] * eigVec[5]
                + eigVec[2] * eigVec[8];

pM_2[idn[1]][0] = eigVec[3] * eigVec[0] + eigVec[4] * eigVec[3]
                + eigVec[5] * eigVec[6];
pM_2[idn[1]][1] = eigVec[3] * eigVec[1] + eigVec[4] * eigVec[4]
                + eigVec[5] * eigVec[7];
pM_2[idn[1]][2] = eigVec[3] * eigVec[2] + eigVec[4] * eigVec[5]
                + eigVec[5] * eigVec[8];

pM_3[idn[1]][0] = eigVec[6] * eigVec[0] + eigVec[7] * eigVec[3]
                + eigVec[8] * eigVec[6];
pM_3[idn[1]][1] = eigVec[6] * eigVec[1] + eigVec[7] * eigVec[4]
                + eigVec[8] * eigVec[7];
pM_3[idn[1]][2] = eigVec[6] * eigVec[2] + eigVec[7] * eigVec[5]
                + eigVec[8] * eigVec[8];

n_1[idn[1]][0] = 2 * VolCell[idn[1]] * pM_1[idn[1]][0];
n_1[idn[1]][1] = 2 * VolCell[idn[1]] * pM_1[idn[1]][1];
n_1[idn[1]][2] = 2 * VolCell[idn[1]] * pM_1[idn[1]][2];

n_2[idn[1]][0] = 2 * VolCell[idn[1]] * pM_2[idn[1]][0];
n_2[idn[1]][1] = 2 * VolCell[idn[1]] * pM_2[idn[1]][1];
n_2[idn[1]][2] = 2 * VolCell[idn[1]] * pM_2[idn[1]][2];

n_3[idn[1]][0] = 2 * VolCell[idn[1]] * pM_3[idn[1]][0];
n_3[idn[1]][1] = 2 * VolCell[idn[1]] * pM_3[idn[1]][1];
n_3[idn[1]][2] = 2 * VolCell[idn[1]] * pM_3[idn[1]][2];
```

Code(4) (Fonction TestFilter)

Extraction des arêtes **isjs** et calcul de |**isjs**|.
Cette partie sera enlevée de la fonction car on à besoin de ce calcul une seule fois (par point) et non pas à chaque appel de la fonction.

```
for (sizet k = 0; k < NbElement; k++) {

    Tetk = mesh.ver2Tet[id][k]->itsRef;
    volume[k]   = domaine.element[Tetk]->getVolume();

    CArete **elt2EdgesLoc;
    elt2EdgesLoc = &(pbAR3D->domaine->elt2Edges[Tetk][0]);

    double comptedge = 0;
    for (sizet j = 0; j < 6; j++) {
      is1    = elt2EdgesLoc[j]->p1->itsID;
      is2    = elt2EdgesLoc[j]->p2->itsID;
      if(is1 == id){
        idLoc[comptedge] = elt2EdgesLoc[j]->p2->itsID;
        jsx[comptedge] = elt2EdgesLoc[j]->p2->x;
        jsy[comptedge] = elt2EdgesLoc[j]->p2->y;
        jsz[comptedge] = elt2EdgesLoc[j]->p2->z;

        isjsx[comptedge] = jsx[comptedge] - isx;
        isjsy[comptedge] = jsy[comptedge] - isy;
        isjsz[comptedge] = jsz[comptedge] - isz;

        Normeisjs[comptedge] = sqrt(isjsx[comptedge]*isjsx[comptedge]
                             + isjsy[comptedge]*isjsy[comptedge]
                             + isjsz[comptedge]*isjsz[comptedge]);

        comptedge += 1;
      }
      if(is2 == id){
        idLoc[comptedge] = elt2EdgesLoc[j]->p1->itsID;
        jsx[comptedge] = elt2EdgesLoc[j]->p1->x;
        jsy[comptedge] = elt2EdgesLoc[j]->p1->y;
        jsz[comptedge] = elt2EdgesLoc[j]->p1->z;

        isjsx[comptedge] = jsx[comptedge] - isx;
        isjsy[comptedge] = jsy[comptedge] - isy;
        isjsz[comptedge] = jsz[comptedge] - isz;

        Normeisjs[comptedge] = sqrt(isjsx[comptedge]*isjsx[comptedge]
                             + isjsy[comptedge]*isjsy[comptedge]
                             + isjsz[comptedge]*isjsz[comptedge]);

        comptedge += 1;
      }
    }//end loop edges
```

Code(5) (Fonction TestFilter)

On calcule

$$\sum_{js \in jt} \left| \left\langle \frac{\mathbf{isjs}}{|\mathbf{isjs}|}, \mathbf{n}_k \right\rangle \right|, \quad \sum_{js \in jt} \left| \left\langle \frac{\mathbf{isjs}}{|\mathbf{isjs}|}, \mathbf{n}_k \right\rangle \right| w(js),$$

puis

$$\widehat{w(is)}^{(\mathbf{n}_k)} = \sum_{jt \ni is} \sum_{js \in jt} vol(jt) \left| \left\langle \frac{\mathbf{isjs}}{|\mathbf{isjs}|}, \mathbf{n}_k \right\rangle \right| w(js)$$
$$\times \left(\sum_{jt \ni is} \sum_{js \in jt} vol(jt) \left| \left\langle \frac{\mathbf{isjs}}{|\mathbf{isjs}|}, \mathbf{n}_k \right\rangle \right| \right)^{-1}.$$

```
//Somme sur js
  if (FilterId == 1) {
    SclaProd[k] = sqrt(((isjsx[0] * n_1[id][0] + isjsy[0] * n_1[id][1]
                + isjsz[0] * n_1[id][2])/Normeisjs[0])**2)
                    + sqrt(((isjsx[1] * n_1[id][0] + isjsy[1] * n_1[id][1]
                    + isjsz[1] * n_1[id][2])/Normeisjs[1])**2)
                    + sqrt(((isjsx[2] * n_1[id][0] + isjsy[2] * n_1[id][1]
                    + isjsz[2] * n_1[id][2])/Normeisjs[2])**2);

    SclaProdVar[k] = sqrt(((isjsx[0] * n_1[id][0] + isjsy[0] * n_1[id][1]
                   + isjsz[0] * n_1[id][2])/Normeisjs[0])**2) * Var[IdLoc[0]]
                   + sqrt(((isjsx[1] * n_1[id][0] + isjsy[1] * n_1[id][1]
                   + isjsz[1] * n_1[id][2])/Normeisjs[1])**2) * Var[IdLoc[1]]
                   + sqrt(((isjsx[2] * n_1[id][0] + isjsy[2] * n_1[id][1]
                   + isjsz[2] * n_1[id][2])/Normeisjs[2])**2) * Var[IdLoc[2]];
  }
  if (FilterId == 2) {
    SclaProd[k] = sqrt(((isjsx[0] * n_2[id][0] + isjsy[0] * n_2[id][1]
                + isjsz[0] * n_2[id][2])/Normeisjs[0])**2)
                + sqrt(((isjsx[1] * n_2[id][0] + isjsy[1] * n_2[id][1]
                + isjsz[1] * n_2[id][2])/Normeisjs[1])**2)
                + sqrt(((isjsx[2] * n_2[id][0] + isjsy[2] * n_2[id][1]
                + isjsz[2] * n_2[id][2])/Normeisjs[2])**2);

    SclaProdVar[k] = sqrt(((isjsx[0] * n_2[id][0] + isjsy[0] * n_2[id][1]
                   + isjsz[0] * n_2[id][2])/Normeisjs[0])**2) * Var[IdLoc[0]]
                   + sqrt(((isjsx[1] * n_2[id][0] + isjsy[1] * n_2[id][1]
                   + isjsz[1] * n_2[id][2])/Normeisjs[1])**2) * Var[IdLoc[1]]
                   + sqrt(((isjsx[2] * n_2[id][0] + isjsy[2] * n_2[id][1]
                   + isjsz[2] * n_2[id][2])/Normeisjs[2])**2) * Var[IdLoc[2]];
  }
  if (FilterId == 3) {
    SclaProd[k] = sqrt(((isjsx[0] * n_3[id][0] + isjsy[0] * n_3[id][1]
                + isjsz[0] * n_3[id][2])/Normeisjs[0])**2)
                + sqrt(((isjsx[1] * n_3[id][0] + isjsy[1] * n_3[id][1]
                + isjsz[1] * n_3[id][2])/Normeisjs[1])**2)
                + sqrt(((isjsx[2] * n_3[id][0] + isjsy[2] * n_3[id][1]
                + isjsz[2] * n_3[id][2])/Normeisjs[2])**2);

    SclaProdVar[k] = sqrt(((isjsx[0] * n_3[id][0] + isjsy[0] * n_3[id][1]
                   + isjsz[0] * n_3[id][2])/Normeisjs[0])**2) * Var[IdLoc[0]]
                   + sqrt(((isjsx[1] * n_3[id][0] + isjsy[1] * n_3[id][1]
                   + isjsz[1] * n_3[id][2])/Normeisjs[1])**2) * Var[IdLoc[1]]
                   + sqrt(((isjsx[2] * n_3[id][0] + isjsy[2] * n_3[id][1]
                   + isjsz[2] * n_3[id][2])/Normeisjs[2])**2) * Var[IdLoc[2]];
  }
  //somme sur elements voisins

  FilterVar += (volume[k] * SclaProdVar[k]) / (volume[k] * SclaProd[k]);

 }// end loop elements

 return FilterVar;

}
```

Code(6) (L195-L230)

On calcule les quantités filtrées à l'aide de notre fonction.

```
PxxFk[idn[l]] = TestFilter(...);
PyyFk[idn[l]] = TestFilter(...);
PzzFk[idn[l]] = TestFilter(...);
PxyFk[idn[l]] = TestFilter(...);
PxzFk[idn[l]] = TestFilter(...);
PyxFk[idn[l]] = TestFilter(...);
PyzFk[idn[l]] = TestFilter(...);
PzxFk[idn[l]] = TestFilter(...);
PzyFk[idn[l]] = TestFilter(...);

rhoSPxxFk[idn[l]] = TestFilter(...);
rhoSPyyFk[idn[l]] = TestFilter(...);
rhoSPzzFk[idn[l]] = TestFilter(...);
rhoSPxyFk[idn[l]] = TestFilter(...);
rhoSPxzFk[idn[l]] = TestFilter(...);
rhoSPyxFk[idn[l]] = TestFilter(...);
rhoSPyzFk[idn[l]] = TestFilter(...);
rhoSPzxFk[idn[l]] = TestFilter(...);
rhoSPzyFk[idn[l]] = TestFilter(...);

normeScellFk[idn[l]] = TestFilter(...);
rhoFk[idn[l]] = TestFilter(...);
rhoVxFk[idn[l]] = TestFilter(...);
rhoVyFk[idn[l]] = TestFilter(...);
rhoVzFk[idn[l]] = TestFilter(...);

rhoVxVxFk[idn[l]] = TestFilter(...);
rhoVyVyFk[idn[l]] = TestFilter(...);
rhoVzVzFk[idn[l]] = TestFilter(...);
rhoVxVyFk[idn[l]] = TestFilter(...);
rhoVxVzFk[idn[l]] = TestFilter(...);
rhoVyVzFk[idn[l]] = TestFilter(...);

SommeDiag[idn[l]] = rhoVxVxFk[idn[l]] + rhoVyVyFk[idn[l]] + rhoVzVzFk[idn[l]]
                    - (1/rhoFk[idn[l]]) * (rhoVxFk[idn[l]]*rhoVxFk[idn[l]]
                    + rhoVyFk[idn[l]]*rhoVyFk[idn[l]]
                    + rhoVzFk[idn[l]]*rhoVzFk[idn[l]]);
SommeDiag[idn[l]] = us3 * SommeDiag[idn[l]];
```

Code(7) (L232-L252)

On calcule

$$
(C_s\overline{\Delta}^2)\widehat{\overline{\rho}|\tilde{S}|\tilde{P}_{ij}}^{(\mathbf{n}_k)} - C_s(\widehat{\Delta}^{(\mathbf{n}_k)})^2\,\widehat{\overline{\rho}}^{(\mathbf{n}_k)}\,\widehat{|\tilde{S}|}^{(\mathbf{n}_k)}\widehat{\tilde{P}_{ij}}^{(\mathbf{n}_k)}
- \widehat{\overline{\rho}\tilde{u}_i\tilde{u}_j}^{(\mathbf{n}_k)} + \frac{1}{\widehat{\overline{\rho}}^{(\mathbf{n}_k)}}\left(\widehat{\overline{\rho}\tilde{u}_i}^{(\mathbf{n}_k)}\widehat{\overline{\rho}\tilde{u}_j}^{(\mathbf{n}_k)}\right)
+ \frac{1}{3}\left(\widehat{\overline{\rho}\tilde{u}_\ell\tilde{u}_\ell}^{(\mathbf{n}_k)} - \frac{1}{\widehat{\overline{\rho}}^{(\mathbf{n}_k)}}\left(\widehat{\overline{\rho}\tilde{u}_\ell}^{(\mathbf{n}_k)}\widehat{\overline{\rho}\tilde{u}_\ell}^{(\mathbf{n}_k)}\right)\right)\delta_{ij}.
$$

```
IF_11[idn[l]] = Cs*VolCell[idn[l]]*VolCell[idn[l]]*rhoSPxxFk[idn[l]]
      - Cs*VolCellTest[idn[l]]*rhoFk[idn[l]]*normeScellFk[idn[l]]*PxxFk[idn[l]]
      - rhoVxVxFk[idn[l]] + (1/rhoFk[idn[l]]) * (rhoVxFk[idn[l]]*rhoVxFk[idn[l]])
      + SommeDiag[idn[l]];
IF_22[idn[l]] = Cs*VolCell[idn[l]]*VolCell[idn[l]]*rhoSPyyFk[idn[l]]
      - Cs*VolCellTest[idn[l]]*rhoFk[idn[l]]*normeScellFk[idn[l]]*PyyFk[idn[l]]
      - rhoVyVyFk[idn[l]] + (1/rhoFk[idn[l]]) * (rhoVyFk[idn[l]]*rhoVyFk[idn[l]])
      + SommeDiag[idn[l]];
IF_33[idn[l]] = Cs*VolCell[idn[l]]*VolCell[idn[l]]*rhoSPzzFk[idn[l]]
      - Cs*VolCellTest[idn[l]]*rhoFk[idn[l]]*normeScellFk[idn[l]]*PzzFk[idn[l]]
      - rhoVzVzFk[idn[l]] + (1/rhoFk[idn[l]]) * (rhoVzFk[idn[l]]*rhoVzFk[idn[l]])
      + SommeDiag[idn[l]];

IF_12[idn[l]] = Cs*VolCell[idn[l]]*VolCell[idn[l]]*rhoSPxyFk[idn[l]]
      - Cs*VolCellTest[idn[l]]*rhoFk[idn[l]]*normeScellFk[idn[l]]*PxyFk[idn[l]]
      - rhoVxVyFk[idn[l]] + (1/rhoFk[idn[l]]) * (rhoVxFk[idn[l]]*rhoVyFk[idn[l]]);
IF_13[idn[l]] = Cs*VolCell[idn[l]]*VolCell[idn[l]]*rhoSPxzFk[idn[l]]
      - Cs*VolCellTest[idn[l]]*rhoFk[idn[l]]*normeScellFk[idn[l]]*PxzFk[idn[l]]
      - rhoVxVzFk[idn[l]] + (1/rhoFk[idn[l]]) * (rhoVxFk[idn[l]]*rhoVzFk[idn[l]]);

IF_21[idn[l]] = Cs*VolCell[idn[l]]*VolCell[idn[l]]*rhoSPyxFk[idn[l]]
      - Cs*VolCellTest[idn[l]]*rhoFk[idn[l]]*normeScellFk[idn[l]]*PyxFk[idn[l]]
      - rhoVyVxFk[idn[l]] + (1/rhoFk[idn[l]]) * (rhoVxFk[idn[l]]*rhoVyFk[idn[l]]);
IF_23[idn[l]] = Cs*VolCell[idn[l]]*VolCell[idn[l]]*rhoSPyzFk[idn[l]]
      - Cs*VolCellTest[idn[l]]*rhoFk[idn[l]]*normeScellFk[idn[l]]*PyzFk[idn[l]]
      - rhoVyVzFk[idn[l]] + (1/rhoFk[idn[l]]) * (rhoVyFk[idn[l]]*rhoVzFk[idn[l]]);

IF_31[idn[l]] = Cs*VolCell[idn[l]]*VolCell[idn[l]]*rhoSPzxFk[idn[l]]
      - Cs*VolCellTest[idn[l]]*rhoFk[idn[l]]*normeScellFk[idn[l]]*PzxFk[idn[l]]
      - rhoVzVxFk[idn[l]] + (1/rhoFk[idn[l]]) * (rhoVxFk[idn[l]]*rhoVzFk[idn[l]]);
IF_32[idn[l]] = Cs*VolCell[idn[l]]*VolCell[idn[l]]*rhoSPzyFk[idn[l]]
      - Cs*VolCellTest[idn[l]]*rhoFk[idn[l]]*normeScellFk[idn[l]]*PzyFk[idn[l]]
      - rhoVzVyFk[idn[l]] + (1/rhoFk[idn[l]]) * (rhoVzFk[idn[l]]*rhoVyFk[idn[l]]);
```

Code(8) (L260-L286)

Pour finir on calcule

$$\widehat{\widehat{\mathcal{F}_i}}^{(\mathbf{n}_k)}(\mathbf{x}).$$

La divergence est d'abord calculée sur chaque élément qui contient i, puis en se ramène à la cellule (i).

```
for (sizet k = 0; k < domaine.ver2Tet[i].size(); k++) {
  pbEF3D->getGradBaseFunct(k, &ax, &ay, &az);
  vol     = element[k]->getVolume();
  id1     = element[k]->point[0]->itsID;
  id2     = element[k]->point[1]->itsID;
  id3     = element[k]->point[2]->itsID;
  id4     = element[k]->point[3]->itsID;

  dIF1dx = ax[0]*IF_11[id1] + ax[1]*IF_11[id2] + ax[2]*IF_11[id3]
         + ax[3]*IF_11[id4];
  dIF1dy = ay[0]*IF_12[id1] + ay[1]*IF_12[id2] + ay[2]*IF_12[id3]
         + ay[3]*IF_12[id4];
  dIF1dz = az[0]*IF_13[id1] + az[1]*IF_13[id2] + az[2]*IF_13[id3]
         + az[3]*IF_13[id4];

  dIF2dx = ax[0]*IF_21[id1] + ax[1]*IF_21[id2] + ax[2]*IF_21[id3]
         + ax[3]*IF_21[id4];
  dIF2dy = ay[0]*IF_22[id1] + ay[1]*IF_22[id2] + ay[2]*IF_22[id3]
         + ay[3]*IF_22[id4];
  dIF2dz = az[0]*IF_23[id1] + az[1]*IF_23[id2] + az[2]*IF_23[id3]
         + az[3]*IF_23[id4];

  dIF3dx = ax[0]*IF_31[id1] + ax[1]*IF_31[id2] + ax[2]*IF_31[id3]
         + ax[3]*IF_31[id4];
  dIF3dy = ay[0]*IF_32[id1] + ay[1]*IF_32[id2] + ay[2]*IF_32[id3]
         + ay[3]*IF_32[id4];
  dIF3dz = az[0]*IF_33[id1] + az[1]*IF_33[id2] + az[2]*IF_33[id3]
         + az[3]*IF_33[id4];

  F1_nk[i] += 0.25 * vol * (dIF1dx + dIF1dy + dIF1dz);
  F2_nk[i] += 0.25 * vol * (dIF2dx + dIF2dy + dIF2dz);
  F3_nk[i] += 0.25 * vol * (dIF2dx + dIF2dy + dIF2dz);
}
F1_nk[i] = F1_nk[i] / VolCell[i];
F2_nk[i] = F2_nk[i] / VolCell[i];
F3_nk[i] = F3_nk[i] / VolCell[i];
```