Chapter 5
Distributed Lagrange Multiplier for Fluid-Structure Interactions

Daniele Boffi, Frédéric Hecht, and Olivier Pironneau

Abstract In this paper we make preliminary numerical tests to assess the performance of the scheme introduced in Boffi et al. (SIAM J Numer Anal 53(6):2584–2604, 2015) and analyzed in Boffi and Gastaldi (Numer Math 135(3):711–732, 2017) for the approximation of fluid-structure interaction problems. We show how to implement the scheme within the `FreeFem++` framework (Hecht, J Numer Math 20(3–4):251–265, 2012) and validate our code with respect to some known problems and benchmarks. The main conclusion is that a simple implementation can provide quite accurate results for non trivial applications.

5.1 Introduction

The use of a distributed Lagrange multiplier for the modeling and approximation of interface problems has a long history within approaches based on fictitious domain techniques [6].

The applications of this methodology for fluid-structure interaction problems has been rediscovered and discussed in recent research [1, 4] originating from the immersed boundary method [2, 9]. The theoretical properties of this approach are quite good, showing unconditional stability for a semi-implicit time discretization and inf-sup stability for the global saddle point problem under suitable conditions on the underlying meshes. Our formulation and some of the main results about it will be summarized in Sect. 5.2.

In this paper we present a series of numerical tests performed with the help of `FreeFem++` [7]. All results are collected in Sect. 5.3. In all tests, the agreement

D. Boffi (✉)
Dipartimento di Matematica "F. Casorati", University of Pavia, Pavia, Italy
e-mail: daniele.boffi@unipv.it; http://www-dimat.unipv.it/boffi/

F. Hecht · O. Pironneau
Sorbonne Universités, UPMC (Paris VI), Laboratoire Jacques-Louis Lions, Paris, France
e-mail: frederic.hecht@upmc.fr; http://www.ljll.math.jussieu.fr/hecht;
Olivier.Pironneau@upmc.fr; http://www.ljll.math.jussieu.fr/pironneau

D. A. Di Pietro et al. (eds.), *Numerical Methods for PDEs*,
SEMA SIMAI Springer Series 15, https://doi.org/10.1007/978-3-319-94676-4_5

with the analytical solution (if known) or with solutions present in the literature is quite good.

One of the main difficulties in the implementation of any fluid-structure interaction model, consists in the appropriate treatment of the exchange of information between fluid and solid. In our formulation the fluid mesh is fixed, while the solid mesh is defined on a reference configuration and mapped to the actual solid domain via the (unknown) transformation which defines the position of the body. It turns out that some terms in our variational formulation need to combine quantities defined on the fluid and solid meshes. Actually, `FreeFem++` has a built in function that allows the computation of such terms. Our codes are listed in Appendix 1 and some comments are provided in Appendix 2.

5.2 Problem Setting

The model introduced in [4] can deal with co-dimension zero (thick) or co-dimension one (thin) bodies immersed in a fluid of two or three space dimensions. Our numerical tests involve thick bodies in two space dimensions; we recall the related formulation.

Let Ω be a bounded domain in $\mathbb{R}^2$ with Lipschitz continuous boundary. We assume that the domain is partitioned into a fluid part Ω_f and a solid part Ω_s (both subdomains are time dependent). The solid domain Ω_s is the image of a reference domain $\mathcal{B} \subset \mathbb{R}^2$. More precisely, the mapping $\mathbf{X} : \mathcal{B} \to \mathbb{R}^2$ associates to each point $\mathbf{s} \in \mathcal{B}$ its image $\mathbf{x} = \mathbf{X}(\mathbf{s}, t) \in \Omega_s$ at time t. We denote by ρ^f and ρ^s the fluid and solid densities, respectively, by ν the fluid viscosity, and by λ and μ the Lamé constants.

The problem considered in [4] is the following one: given an initial velocity $\mathbf{u}_0 \in (H^1_0(\Omega))^2$, an initial body position $\mathbf{X}_0 \in (W^{1,\infty}(\mathcal{B}))^2$, find velocity and pressure $(\mathbf{u}(t), p(t)) \in (H^1_0(\Omega))^2 \times L^2_0(\Omega)$, body position $\mathbf{X}(t) \in (H^1(\mathcal{B}))^2$, and a Lagrange multiplier $\boldsymbol{\lambda}(t) \in \Lambda$ such that for almost every $t \in]0, T[$ is holds

$$\int_\Omega \Big(\rho^f \mathbb{D}_t \mathbf{u}(t) \cdot \hat{\mathbf{u}} - \hat{p} \nabla \cdot \mathbf{u}(t) - p(t) \nabla \cdot \hat{\mathbf{u}} + \frac{\nu}{2} \mathrm{D}\mathbf{u}(t) : \mathrm{D}\hat{\mathbf{u}}\Big)$$

$$+ \int_{\mathcal{B}} \big(c_1 \mathrm{D}\boldsymbol{\lambda}(t) : \mathrm{D}(\hat{\mathbf{u}}(\mathbf{X}(t))) + c_2 \boldsymbol{\lambda}(t) \cdot \hat{\mathbf{u}}(\mathbf{X}(t))\big) = 0$$

$$\forall (\hat{\mathbf{u}}, \hat{p}) \in (H^1_0(\Omega))^2 \times L^2_0(\Omega)$$

$$\int_{\mathcal{B}} \Big((\rho^s - \rho^f) \partial_{tt} \mathbf{X}(t) \cdot \hat{\mathbf{X}} + \frac{\mu}{2} \mathrm{D}\mathbf{X}(t) : \mathrm{D}\hat{\mathbf{X}} + \lambda \nabla \cdot \mathbf{X}(t) \nabla \cdot \hat{\mathbf{X}}$$

$$- c_1 \mathrm{D}\boldsymbol{\lambda}(t) : \mathrm{D}\hat{\mathbf{X}} - c_2 \boldsymbol{\lambda}(t) \cdot \hat{\mathbf{X}}$$

$$
\begin{aligned}
&+c_1\mathrm{D}\hat{\boldsymbol{\lambda}} : \mathrm{D}(\mathbf{u}(t)(\mathbf{X}(t)) - \partial_t\mathbf{X}(t)) + c_2\hat{\boldsymbol{\lambda}} \cdot (\mathbf{u}(t)(\mathbf{X}(t)) - \partial_t\mathbf{X}(t))\Big) = 0 \\
&\forall(\hat{\mathbf{X}}, \hat{\boldsymbol{\lambda}}) \in (H^1(\mathcal{B}))^2 \times \Lambda, \qquad (5.1)
\end{aligned}
$$

where $\mathbb{D}_t$ is the total derivative and D is the symmetric gradient.

The constants c_1, c_2 and the space Λ are crucial for the definition of the model and the subsequent numerical scheme: in our computations we consider both c_1 and c_2 positive and different from zero (H^1-based Lagrange multiplier), so that the space Λ is $(H^1(\mathcal{B}))^2$.

5.2.1 Numerical Approximation

The time semi-discretization of Problem (5.1) is constructed as follows: in the first equation the total derivative is approximated by the Galerkin-characteristic method (see [10]); the second derivative $\partial_{tt}\mathbf{X}(t)$ in the second equation is approximated by $(\mathbf{X}^{n+1} - 2\mathbf{X}^n + \mathbf{X}^{n-1})/\delta t^2$; $\partial_t\mathbf{X}$ is approximated by $(\mathbf{X}^{n+1} - \mathbf{X}^n)/\delta t$; all other quantities are evaluated implicitly at time $n + 1$ with the following exception. Clearly, there is a problem when a term involving $\hat{\mathbf{u}}(\mathbf{X})$ has to be integrated on $\mathcal{B}$. Treating this term fully implicitly would imply the use of the mapping $\mathbf{X}^{n+1}$ which is not yet available; for this reason we use a semi-implicit scheme where $\hat{\mathbf{u}}(\mathbf{X}^n)$ is used, instead.

In [4, Prop. 3] it has been shown that the resulting semi-discrete scheme is unconditionally stable with respect to the time step δt. The proof is based on a discrete energy estimate which is analogous to the stability estimate for the continuous problem:

$$
\begin{aligned}
&\frac{\rho_f}{2\delta t}\left(\|\mathbf{u}^{n+1}\|_0^2 - \|\mathbf{u}^n\|_0^2\right) + \nu\|\mathbf{D}\mathbf{u}^{n+1}\|_0^2 \\
&\qquad + \frac{\rho_s - \rho_f}{2\delta t}\left(\left\|\frac{\mathbf{X}^{n+1} - \mathbf{X}^n}{\delta t}\right\|_{0,\mathcal{B}}^2 - \left\|\frac{\mathbf{X}^n - \mathbf{X}^{n-1}}{\delta t}\right\|_{0,\mathcal{B}}^2\right) \\
&\qquad + \frac{\mathbf{E}(\mathbf{X}^{n+1}) - \mathbf{E}(\mathbf{X}^n)}{\delta t} \le 0
\end{aligned}
$$

where the energy $\mathbf{E}$ is defined in terms of the energy density $W(\mathbb{F})$ ($\mathbb{F}$ being the deformation gradient)

$$
\mathbf{E}(\mathbf{X}(t)) = \int_{\mathcal{B}} W(\mathbb{F}(\mathbf{s}, t))\, d\mathbf{s}
$$

The numerical approximation of Problem (5.1) is based on a set of four finite element spaces: $V_h \subset (H_0^1(\Omega))^2$ and $Q_h \subset L_0^2(\Omega)$ are inf-sup stable finite element spaces, while $S_h \subset (H^1(\mathcal{B}))^2$ and $\Lambda_h \subset \Lambda$ are finite elements defined in the solid domain. More precisely, V_h and Q_h are finite elements defined according to a triangulation $\mathcal{T}_{h_f}$ of Ω, while S_h and Λ_h are finite elements defined on a mesh $\mathcal{T}_{h_s}$ of the reference solid configuration $\mathcal{B}$.

5.3 Numerical Tests

In all tests the solid is hyper-elastic with Young modulus E, Poisson ratio κ and density ρ^s. The shear modulus is then given by $\mu = E/(1+\kappa)/2$.

The fluid is Newtonian incompressible with density ρ^f and viscosity ν.

5.3.1 Disk Falling in a Liquid

A disk of diameter d is at rest initially centred at $x_c = W/2,\ y_c = H - h$ in a rectangular channel of width W and height H. Only the disk is subject to gravity g, not the fluid. No slip conditions are applied on the walls of the channel.

This test was proposed by Zhao et al. in [12] and more recently by Wang et al. in [11]. Here we chose Wang's values for the parameters:

$$\begin{aligned} &W = 2,\ d = 0.125,\ h = 0.5,\ H = 4,\\ &\rho^s = 1.2,\ \kappa = 0.3,\ \mu = 10^8,\ \rho^f = 1,\ \nu = 1,\ g = 981 \end{aligned} \tag{5.2}$$

The asymptotic vertical velocity is known to be -0.3567. Figure 5.1 shows the evolution of the vertical velocity versus time for four meshes: a coarse mesh with

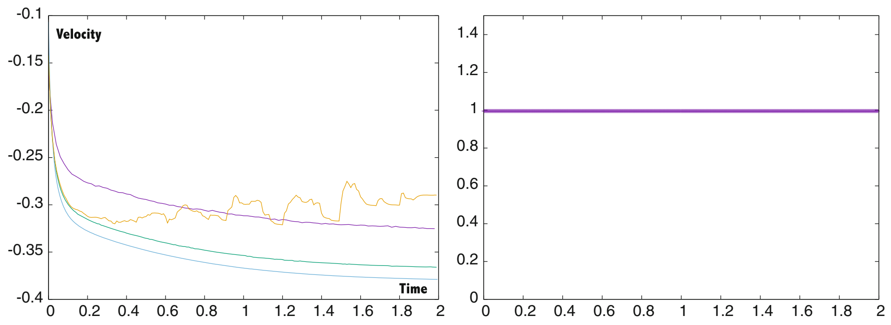

Fig. 5.1 Left: vertical velocity of the solid versus time computed with three body fitted mesh and one non-body fitted mesh. Convergence seems monotone towards a limit curve for the first three meshes; the coarse mesh is the highest, the middle mesh is in the middle and the finest mesh is below. Right: area of the solid divided by $\pi d^2/4$, as a function of time

1182 vertices, a middle mesh with 4693 vertices and a fine mesh with 18,661 vertices. The corresponding time steps are 0.02, 0.01, 0.005.

For these three cases the fluid mesh is modified at each time step to include the fluid-structure interface as an interline curve made of edges of the triangulation. Computation is also made on a fourth mesh with 4693 and the initial fluid-solid interface at time zero but not changed with time. On this fourth mesh precision degrades with time, probably due to mesh interpolations. On the three previous body fitted mesh there is convergence to a limit curve, but the asymptotic value seems to be -0.3788 rather than -0.3567. It could be due to the fact that the fluid model is extended in the solid leading to an error proportional to $\rho^s - \rho^f$. But it could also be the effect of interpolation needed to computed variables earlier defined on the mesh before motion. We are currently intersecting meshes to reduce this interpolation error and preliminary tests (to be published later) point to the direction of a more accurate falling velocity. On the other hand, mass is remarkably preserved as shown on Fig. 5.1-right.

A pressure map at $t = 0.7$ is given on Fig. 5.2-left. Next the same simulation is done with a very soft material having $\mu = 10$. The shape of the solid at $t = 0.7$ is given with a color map of the yy component of the stress on Fig. 5.2-right. The computing time for this last test is 434" on a Core i7-2.5GHz on a single core.

For these two simulations the influence of the coefficients c_1 and c_2 are small, as long as c_2 is not zero. Here both are set at 1. The influence of the degree of the finite element spaces is also surprisingly small. Both the P2/P1 element for velocity pressure or the P1-bubble/P1 element gave the same results. Changing P1 into P2 for the Lagrangian coordinates also didn't make a difference. It seems that the precision of the method is entirely driven by the quadrature formula used for the mixed integrals involving a function on the fluid mesh times a function on the transformed solid mesh.

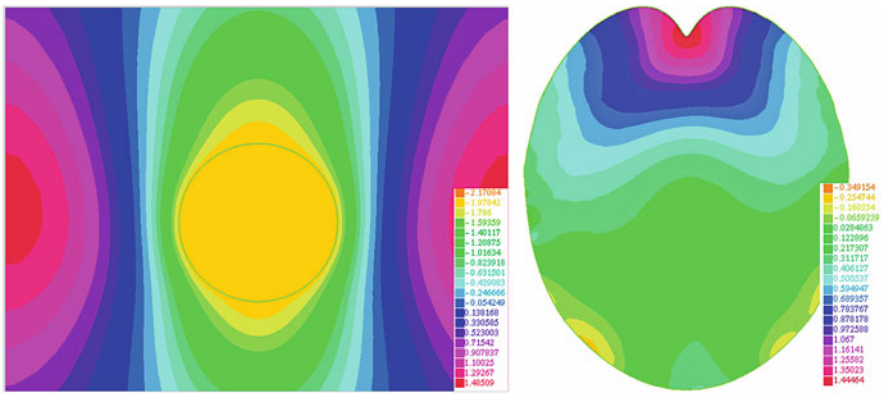

Fig. 5.2 Left: pressure map at $t = 0.7$ close to the solid disk falling in a liquid. Right: the yy component of the stress inside a very soft disk falling in a liquid displayed at $t = 0.7$. The shape is also the result of the numerical simulation

Some integrals in the variational formulation involve piecewise polynomial functions defined on different meshes. We have developed a special quadrature formula in `FreeFem++` (see [7]) to handle them. For instance let u be defined on mesh T_h^u and v be defined on mesh T_h^v obtained from T_h^u by convecting the vertices $q^i \in T_h^u$ with X, namely $X(q^i)$ is a vertex of T_h^v. Then for a triangle T of T_h^v the integral on T of $u \circ X \cdot v$ is approximated by $\sum_{j=1}^{J} u(X(\xi_j))v(\xi_j)\omega_j$ where ξ_j, ω_j are a valid set of quadrature points and coefficients for a quadrature on T (shown in the `FreeFem++` code by a parameter in the integral like `int2d(Ths,qft=qf9pT,mapu=[Xo,Yo])`.

5.3.2 Validation with a Rotating Disk

The purpose of this test is to compare the numerical solution with a semi-analytical solution which can be computed to any desired accuracy.

A cylinder contains a fixed rigid cylindrical rod in its center, a cylindrical layer of hyperelastic material around the rod and the rest is filled with a fluid (see Fig. 5.3). First the system is at rest and then a constant rotation is given to the outer cylinder. This cause the fluid to rotate with an angular velocity which depends on the distance r to the main axis; in turn, because the friction of the fluid at the interface the hyperelastic material will be dragged into a angular velocity ω which is also only a function of r and time . Due to elasticity ω will oscillate with time until numerical dissipation and fluid viscosity damps it.

In a two dimensional cut perpendicular to the main axis, the velocities and displacements are two dimensional as well. Hence the geometry is a ring of inner and outer radii, R_0 and R_1, with hyperelastic material between R_0 and R and fluid between R and R_1. Because of axial symmetry, R is constant, so the geometry does not change.

In this test $R_0 = 3$, $R = 4$, $R_1 = 5$. The solid is an hyperelastic material with $\mu = 100$ and $\lambda = 2\kappa\mu/(1 - 2\kappa)$ with $\kappa = 0.3$ and $\rho^s = 10$. The Newtonian fluid has $\nu = 1$, $\rho^f = 1$. The velocity of the outer cylinder has magnitude 3. As

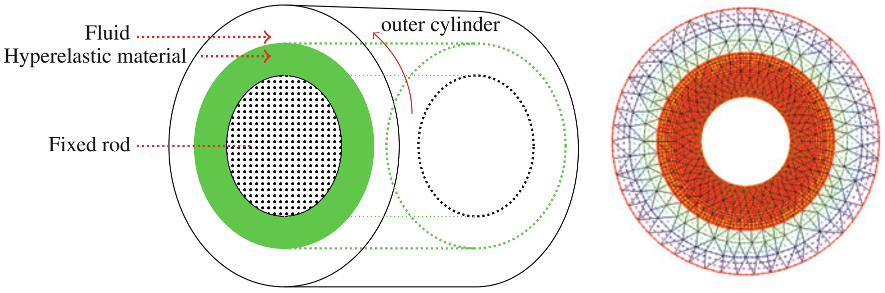

Fig. 5.3 A fluid-structure system inside a rotating cylinder (giving a constant angular velocity to the fluid outer boundary) with a fixed rod in its center. Left: sketch of the system. Right: a 2d calculation showing the velocity vectors at time 0.85 for the coarser mesh

everything is axisymmetric the computation can be done in polar coordinates r, θ, and the fluid-solid system reduces to

$$\rho\partial_t v - \frac{1}{r}\partial_r[\xi^f r\partial_r v + \xi^s r\partial_r d] = 0,$$
$$\partial_t d = v, \ r \in (R_0, R_1), \ \ v_{|R_0} = 0, \ v_{|R_1} = 3, \tag{5.3}$$

with $\rho = \rho^s \mathbf{1}_{r\le R} + \rho^f \mathbf{1}_{r>R}$, $\xi^s = \mu\mathbf{1}_{r\le R}$, $\xi^f = \nu\mathbf{1}_{r>R}$, and with $d(r, 0) = 0$.

In all 2d computations $c_1 = c_2 = 10$ and $\delta t = 0.005$. We have verified that a smaller time step does not improve the precision.

Comparison between this one dimensional approach and the numerical solution of system (5.1) is given on Fig. 5.3—right, at $T = 0.5$ and a coarse mesh with 505 vertices. Then the same is computed on a finer mesh having 1986 vertices and finally with a mesh with 7433 vertices. Results are displayed on Fig. 5.4.

This test has two qualities: (a) the exact solution is easy to compute to any precision; (b) the geometry does not change and quadrature errors are due only to quadrature for integrals involving functions on the same domain but with two different triangulations.

5.3.2.1 Flow Past a Cylinder with a Flagella Attached

This test is known as FLUSTRUK-FSI-3 in [5]. The geometry is shown on Fig. 5.5. The inflow velocity is $\bar{U} = 2$, $\mu = 210^6$ and $\rho^s = \rho^f$. After some time a Karman-Vortex alley develops and the flagella beats accordingly. Results are shown on Figs. 5.5 and 5.6 with a mesh of 9692 vertices and a time step size of 0.0015; the first one displays a snapshot of the velocity vector norms and the second the y-coordinate versus time of the top right corner of the flagella.

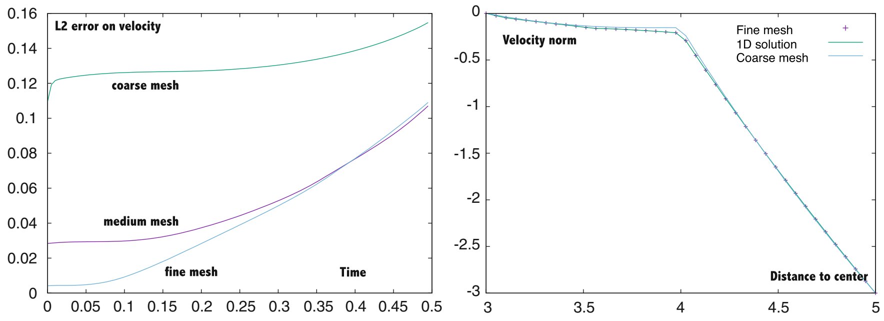

Fig. 5.4 Rotating cylinder. Left: Evolution of the L^2 error versus time for the three meshes. Right: velocities normal to the ray at $\theta = \pi/4$ versus $r - 3$, computed on the coarsest meshes shown in green with continuous line and crosses. The "exact" solution of the one dimensional equation is shown in blue

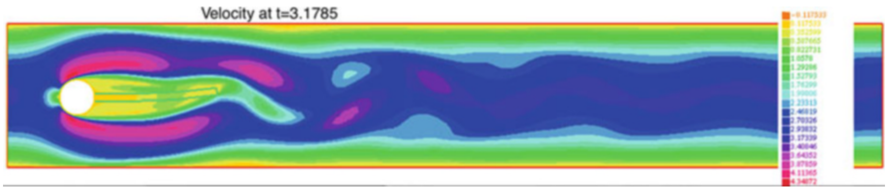

Fig. 5.5 FLUSTRUK-FSI-3 Test. Color map based on the norm of the fluid and solid velocity vectors

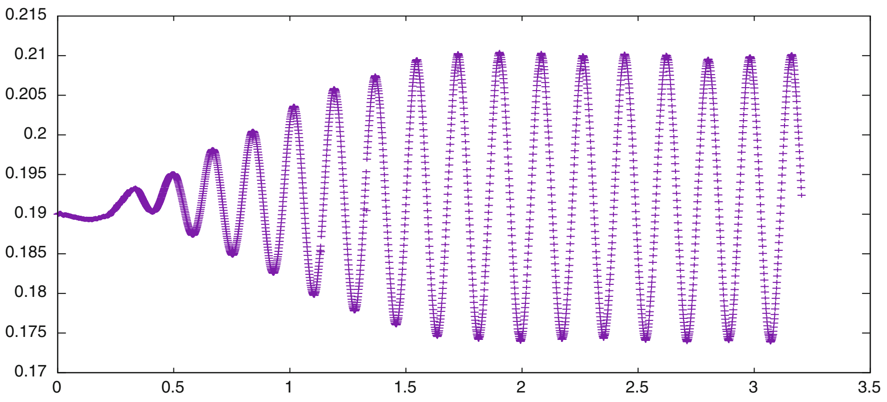

Fig. 5.6 FLUSTRUK-FSI-3 Test. Vertical position of the upper right tip of the flagella versus time shown up to t = 5

These numerical results compare reasonably well with those of [5]. The frequency is 5.6 compared to 5.04 and the maximum amplitude 0.018 compared with 0.032. Amplitude is very sensitive to the method (see [8]).

Appendix 1: FreeFem++ Codes

Listing 5.1 Code for the falling disk

```
int n=20, m=8*n;  // higher for fine mesh

int H=4, W=2; // vertical length of fluid box
real h=0.5, R1=0.125*2, R2=R1, xc=W/2, yc=H-h;  // elliptic radii and
     center of disk
real rhof=1, rhos=1.2, nu=1, penal=1e-9;
// rho, mu, rescaled : divided by 1e6
real  kappa=0.3, /*E=1e4, mu=E/(1+kappa)/2 */ mu=1e1, lambda=2*kappa*mu
     /(1-2*kappa);
real gravity=981;
real T=0.7, dt=0.1/n, dt2=dt*dt;
real c1=1, c2=10; // Lagrange multiplier constants: H1 -> 1,1, L2 -> 0,1

// mesh Thf=square(10*n,H*n,[x,H*y]); // fluid + solid domain
```

```

border a1(t=0,1){x=W*t; y=0;}
border a2(t=0,1){x=W; y=H*t;}
border a3(t=1,0){x=W*t; y=H;}
border a4(t=1,0){x=0;y=H*t;}
border C(t=0,2*pi){x=xc+R1*cos(t); y=yc+R2*sin(t);}
mesh Thsi = buildmesh(C(m)); // Initial solid domain
fespace Whi(Thsi,P1);
Whi Xoi=x,Yoi=y;
real[int] xx(m), yy(m);
int[int] nn(m);
for(int i=0;i<m;i++){xx[i]=xc+R1*cos(2*i*pi/(m)); yy[i]=yc+R2*sin(2*i*pi/(m)); nn[i]=2;}
border D(t=0,1;i){
        int ii = (i+1)%m; real t1 = 1-t; real x1 = xx[i]*t1 + xx[ii]*t;
        real y2 = yy[i]*t1 + yy[ii]*t; x= Xoi(x1,y2); y=Yoi(x1,y2);
}
plot(D(nn));
mesh Ths = buildmesh(D(nn));
mesh Thso=Ths;
mesh Thf= buildmesh(a1(W*n/2)+a2(H*n)+a3(W*n)+a4(H*n)+D(nn));
//plot(a1(10*n)+a2(H*n)+a3(10*n)+a4(H*n)+D(nn));
plot(Thf,Thso, cmm="Initial configuration");


fespace Vh(Thf,P1b);  // velocity space
fespace Qh(Thf,P1);  // pressure space
fespace Wh(Ths,P1b);  // Lagrangian coordinates X,Y space
fespace Lh(Ths,P1);  // Lagrangian multiplier space
fespace Zh(Thf,[P1b,P1b,P1]); // fluid space
fespace Rh(Ths,[P1,P1,P1b,P1b]);// solid space

Vh u,v,uh,vh;
Qh p,ph;
Wh  Xoo=x-0.*(x-xc),Yoo= y-0.*(y-yc); // the X,Y are now the displacements
Rh [lamx,lamy,X,Y],[lamxo,lamyo,Xo,Yo]=[0,0,x,y];//x-(x-xc)/2,y+(y-yc)/2];
Zh [uo,vo,po]=[0,0,0];


macro div(u,v) ( dx(u)+dy(v) ) // EOM
macro Grad(u,v)[[dx(u),dy(u)],[dx(v),dy(v)]] // EOM
macro DD(u,v)  [[2*dx(u),div(v,u)],[div(v,u),2*dy(v)]] // EOM

varf aa([u,v,p],[uh,vh,ph]) =
          int2d(Thf)(rhof*[u,v]'*[uh,vh]/dt- div(uh,vh)*p -div(u,v)*ph
                     + penal*p*ph + nu*trace(DD(uh,vh)'*DD(u,v))/2)
          + on(1,3,u=0,v=0) + on(2,4,u=0) ;

varf bb([lamx,lamy,X,Y],[lamxh,lamyh,Xh,Yh]) =
          int2d(Ths)((rhos-rhof)*(X*Xh+Y*Yh)/dt2
                     - c1*trace(Grad(lamx,lamy)'*Grad(Xh,Yh)) - c2*(lamx*Xh+lamy*Yh)
                     - c1*trace(Grad(lamxh,lamyh)'*Grad(X,Y))/dt - c2*(lamxh*X+lamyh*Y)/dt
                     + mu*trace(DD(X,Y)'*DD(Xh,Yh))/2 + lambda*div(X,Y)*div(Xh,Yh)
                                         + penal*(lamx*lamxh+lamy*lamyh));

varf ab([u,v,p],[lamxh,lamyh,Xh,Yh]) =
          int2d(Ths,qft=qf9pT,mapu=[Xo,Yo])(
                       c1*trace(Grad(lamxh,lamyh)'*(Grad(Xo,Yo)*Grad(u,v)))
                     + c2*(lamxh*u+lamyh*v));

```

```
varf ba([lamx,lamy,X,Y],[uh,vh,ph]) =
          int2d(Ths,qft=qf9pT,mapt=[Xo,Yo])(
                       c1*trace(Grad(lamx,lamy)'*(Grad(Xo,Yo)*Grad(uh,vh
                            )))
                     + c2*(lamx*uh+lamy*vh));

varf rhs1([u,v,p],[uh,vh,ph]) =
          int2d(Thf)(-rhof*gravity*vh * 0 // 0=no gravity in the fluid
                     + rhof*convect([uo,vo],-dt,uo)*uh/dt
                     + rhof*convect([uo,vo],-dt,vo)*vh/dt)
           + on(1,3,u=0,v=0) + on(2,4,u=0,v=0) ;

varf rhs2([lamx,lamy,X,Y],[lamxh,lamyh,Xh,Yh]) =
          int2d(Ths)(-(rhos-rhof)*gravity*Yh + 2*(mu+lambda)*div(Xh,Yh)
                     - c1*trace(Grad(lamxh,lamyh)'*Grad(Xo,Yo))/dt
                     - c2*(lamxh*Xo+lamyh*Yo)/dt
                     - (rhos-rhof)*((Xoo-2*Xo)*Xh+(Yoo-2*Yo)*Yh)/dt2);

for(int i=0;i<T/dt;i++){
//       cout<<"time= "<<i*dt<<endl;
                real[int] RHS1 = rhs1(0,Zh);
                real[int] RHS2 = rhs2(0,Rh);
                matrix AA = aa(Zh,Zh);
                matrix AB = ab(Zh,Rh);
                matrix BA = ba(Rh,Zh);
                matrix BB = bb(Rh,Rh);

                matrix AABB = [ [AA,BA], [AB,BB] ];
                set(AABB,solver=sparsesolver,master=-1);
                real[int] rhs = [RHS1, RHS2];
                real[int] w=AABB^-1*rhs;
                Xoo=Xo; Yoo=Yo; Xoi=Xo; Yoi=Yo;
                            [uo[],lamxo[]] = w;
                Thso = buildmesh(D(nn));
                                Thf= buildmesh(a1(W*n/2)+a2(H*n)+a3(W*n)
                                     +a4(H*n)+D(nn));
                                [uo,vo,po]=[uo,vo,po]; [lamxo,lamyo,Xo,
                                     Yo]=[lamxo,lamyo,Xo,Yo];
                                cout<<i*dt<<" "<<int2d(Thso)(Yo-Yoo)/R1/
                                     R1/pi/dt<<" "<<int2d(Thso)(vo)/R1/R1
                                     /pi<<" area= "<<int2d(Thso)(1.)/R1/
                                     R1/pi<<" velocity "<<endl;
                uh=sqrt(uo*uo+vo*vo);
                plot(Thf, Thso, uh, value=1, fill=0, coef=100, cmm="t="+
                     i*dt+"      Velocity");
}
Thf = buildmesh(a1(W*n/2)+a2(H*n)+a3(W*n)+a4(H*n)+D(nn));
[uo,vo,po]=[uo,vo,po];
plot(vo, value=1, fill=0, coef=100, cmm="t="+T+" Pressure ");
fespace Bh(Thso,P1);
Bh s,sh;
solve bbc(s,sh)= int2d(Thso)(s*sh + 0.01*(dx(s)*dx(sh)+dy(s)*dy(sh)))
-int2d(Thso)(sh*(mu*dy(vo) + lambda*(dx(uo)+dy(vo))));
plot(s);
```

Listing 5.2 Code for the rotating disk test

```
load "MUMPS"
load "pipe"
verbosity=0;

int n=20;  // higher for fine mesh

real R0=1, R1=2, R2=3,  gravity=0, ringvelocity=-3;
real  kappa=0.3, /*E=1e4, mu=E/(1+kappa)/2 */ mu=1e2, lambda=2*kappa*mu
     /(1-2*kappa);
```

```
real rhof=1, rhos=10, nu=1, penal=1e-6;
real c1=10,c2=10,T=0.5, dt=0.005, dt2=dt*dt;

real delta=0.05;
border CF(t=0,2*pi){x=R2*cos(t); y=R2*sin(t);} // fluid
border CS(t=0,2*pi){x=R1*cos(t); y=R1*sin(t);}  // solid
border CC(t=0,2*pi){x=R0*cos(t); y=R0*sin(t);}  // clamped
border C1(t=0,2*pi){x=(R1-delta)*cos(t); y=(R1-delta)*sin(t);}  // solid
border C2(t=0,2*pi){x=(R1-delta/2)*cos(t); y=(R1-delta/2)*sin(t);}  //
     solid
border C3(t=0,2*pi){x=(R1+delta/2)*cos(t); y=(R1+delta/2)*sin(t);}  //
     solid
border C4(t=0,2*pi){x=(R1+delta)*cos(t); y=(R1+delta)*sin(t);}  // solid
mesh Thf = buildmesh(CC(-10*n) /*+C1(10*n)+C2(10*n)+C3(10*n) +C4(10*n)*/
     +CS(10*n)+CF(10*n));
mesh Ths = buildmesh(CC(-10*n)+CS(10*n));
mesh Thso=Ths;
// plot(Thf,Ths);
fespace Vh(Thf,P2);
fespace Qh(Thf,P1);
fespace Wh(Ths,P1);
fespace Lh(Ths,P1);
fespace Zh(Thf,[P2,P2,P1]);
fespace Rh(Ths,[P1,P1,P1,P1]);

Vh u,v,uh,vh;
Qh p,ph;
Wh  Xoo=x,Yoo=y;
Rh [lamx,lamy,X,Y],[lamxo,lamyo,Xo,Yo]=[0,0,x,y];//x-(x-xc)/2,y+(y-yc)
     /2];
Zh [uo,vo,po]=[0,0,0];

macro div(u,v) ( dx(u)+dy(v) ) // EOM
macro Grad(u,v)[[dx(u),dy(u)],[dx(v),dy(v)]] // EOM
macro DD(u,v)  [[2*dx(u),div(v,u)],[div(v,u),2*dy(v)]] // EOM

varf aa([u,v,p],[uh,vh,ph]) =
          int2d(Thf)(rhof*[u,v]'*[uh,vh]/dt- div(uh,vh)*p -div(u,v)*ph
                     + penal*p*ph + nu*trace(DD(uh,vh)'*DD(u,v))/2)
          + on(CC,u=0,v=0) + on(CF,u=-ringvelocity*y/R2,v=ringvelocity*x
               /R2) ;

varf bb([lamx,lamy,X,Y],[lamxh,lamyh,Xh,Yh]) =
          int2d(Ths)((rhos-rhof)*(X*Xh+Y*Yh)/dt2
                     - c1*trace(Grad(lamx,lamy)'*Grad(Xh,Yh)) - c2*(lamx
                         *Xh+lamy*Yh)
                     - c1*trace(Grad(lamxh,lamyh)'*Grad(X,Y))/dt - c2*(
                         lamxh*X+lamyh*Y)/dt
                     + mu*trace(DD(X,Y)'*DD(Xh,Yh))/2 + lambda*div(X,Y)*
                         div(Xh,Yh)
                                         + penal*(lamx*lamxh+lamy*lamyh)
                                             );

varf ab([u,v,p],[lamxh,lamyh,Xh,Yh]) =
          int2d(Ths,qft=qf9pT,mapu=[Xo,Yo])(
                       c1*trace(Grad(lamxh,lamyh)'*(Grad(Xo,Yo)*Grad(u,v
                           )))
                     + c2*(lamxh*u+lamyh*v));

varf ba([lamx,lamy,X,Y],[uh,vh,ph]) =
          int2d(Ths,qft=qf9pT,mapt=[Xo,Yo])(
                       c1*trace(Grad(lamx,lamy)'*(Grad(Xo,Yo)*Grad(uh,vh
                           )))
                     + c2*(lamx*uh+lamy*vh));

varf rhs1([u,v,p],[uh,vh,ph]) =
          int2d(Thf)( rhof*convect([uo,vo],-dt,uo)*uh/dt
                     + rhof*convect([uo,vo],-dt,vo)*vh/dt)
           + on(CC,u=0,v=0) + on(CF,u=-ringvelocity*y/R2,v=ringvelocity*x/R2) ;

varf rhs2([lamx,lamy,X,Y],[lamxh,lamyh,Xh,Yh]) =
          int2d(Ths)( 2*(mu+lambda)*div(Xh,Yh)
                     - c1*trace(Grad(lamxh,lamyh)'*Grad(Xo,Yo))/dt
                     - c2*(lamxh*Xo+lamyh*Yo)/dt
                     - (rhos-rhof)*((Xoo-2*Xo)*Xh+(Yoo-2*Yo)*Yh)/dt2);

////////////////////////////////////////////////////////////
//// semi-analytic solution by solving a 1d problem /////
mesh Th=square(100,1,[R0+(R2-R0)*x,0.1*y]);
fespace WWh(Th,P2,periodic=[[1,x],[3,x]]);
fespace W0(Th,P1dc);
WWh d=0,wo,wh,wold=0;
W0 nnu=nu*(x>R1)+mu*dt*(x<=R1), Rho=rhof*(x>R1)+(rhos-rhof)*(x<=R1);
problem AA1d(wo,wh) = int2d(Th)(Rho*x*wo*wh/dt+x*nnu*dx(wo)*dx(wh) )
                        + int2d(Th)( -Rho*x*wold*wh/dt + mu*(x<=R1)*x*dx(d)*dx(wh) )
                        +on(2,wo=ringvelocity)+on(4,wo=0);// this is the one-d axisymmetric problem
//////////////////////////////////////////

pstream  pgnuplot("gnuplot" );  // prepare gnuplot //////////////
int NT=T/dt,J=40;
real l2error=0, dr = (R2-R0)/(J-1);

/////////////////////////////// time loop /////////////////////////
for(int i=0;i<NT;i++){
//       cout<<"time= "<<i*dt<<endl;
                real[int] RHS1 = rhs1(0,Zh);
                real[int] RHS2 = rhs2(0,Rh);
                matrix AA = aa(Zh,Zh);
                matrix AB = ab(Zh,Rh);
                matrix BA = ba(Rh,Zh);
                matrix BB = bb(Rh,Rh);

                matrix AABB = [ [AA,BA], [AB,BB] ];
                set(AABB,solver=sparsesolver,master=-1);
                real[int] rhs = [RHS1, RHS2];
                real[int] w=AABB^-1*rhs;
                Xoo=Xo; Yoo=Yo;
                            [uo[],lamxo[]] = w;
//                              cout<<i*dt;<<" "<<int2d(Thso)(Yo-Yoo)/R1/R1/pi/dt<<" "<<int2d(Thso)(vo)/R1/R1/pi<<" area= "<<int2d(Thso)(1.)/R1/R1/pi<<" velocity "<<endl;
//                uh=sqrt(uo*uo+vo*vo);
                ///////////////// for error plot ///////////////////////
                AA1d;
                d=d+wo*dt;
                wold=wo;  // this is for the one-d axisymmetric problem
                ofstream f("aux.gp");
                for(int j=0;j<J;j++) {
                        f << j*dr <<"   " << vo(R0+j*dr,0)<<"  "<< wo(R0+j*dr,0.05)<< endl;
                        l2error += (vo(R0+j*dr,0)-wo(R0+j*dr,0.05))^2*dt;
                }
                pgnuplot << " plot [0:2][-3:0.51]'aux.gp' u 1:2 w l,'aux.gp' u 1:3 w l"<< endl;
                cout<<i*dt<<"  "<<sqrt(l2error)/T<<endl;
                flush(pgnuplot);
}
plot(Ths,Thf,[uo,vo],fill=1, coef=0.1, cmm="t="+T, wait=1)      ;

for(int j=0;j<J;j++)
```

```
 cout<< j*dr<<"  "<< vo(R0+j*dr,0)<<"  "<<wo(R0+j*dr,0.05)<<"  "<<vo(R0+
      j*dr,0)-wo(R0+j*dr,0.05) <<endl;
 // copy past the numbers in a file "results.txt", call gnuplot and do
      in the gnuplot terminal window:
 // plot"results.txt"using 1:2,"results.txt"using 1:3 w l
```

Listing 5.3 Code for FSI-3 test

```
verbosity=0;

int n=2, m=2*n, NN=500*m;  // higher for fine mesh

real rhof=1, rhos=1, nu=0.001, penal=1e-9;
// rho, mu, rescaled : divided by 1e6
real  kappa=0.4, E=1e6, mu=2e3, lambda=2*kappa*mu/(1-2*kappa);
real Ubar=2, gravity=0*981;
real T=6, dt=T/NN, dt2=dt*dt;
real c1=1, c2=1; // Lagrange multiplier constants: H1 -> 1,1, L2 -> 0,1

// mesh Thf=square(10*n,H*n,[x,H*y]); // fluid + solid domain

int la1=10, la2=11;
real cx0 = 0.2, cy0 = 0.2; // center of cyl.
real r=0.05, H=0.41, L=2.5; // radius of cylinder, size of domain
real ll=0.35, h2=0.01;//flagella length and half thickness
real la=asin(h2/r), x0=sqrt(r*r-h2*h2);

border fr1(t=0,L){x=t; y=0; label=1;} // outer box begins
border fr2(t=0,H){x=L; y=t; label=2;}
border fr3(t=L,0){x=t; y=H; label=1;}
border fr4(t=H,0){x=0; y=t; label=3;} // outer box ends
border fr5(t=la,2*pi-la){x=cx0+r*cos(-t); y=cy0+r*sin(-t); label=4;}
border br1(t=-la,la){x=cx0+r*cos(-t); y=cy0+r*sin(-t); label=4;} // flag
      begins
border br2(t=0,ll){x=cx0+x0+t; y=cy0-h2;label=la1;}
border br3(t=-h2,h2){x=x0+cx0+ll; y=cy0+t;label=la1;}
border br4(t=ll,0){x=cx0+x0+t; y=cy0+h2;label=la1;}              // flag
      ends

mesh Thf=buildmesh(fr1(20*m)+fr2(3.25*m)+fr3(20*m)+fr4(4*m)+fr5(12*m) +
     br1(m)+br2(12*m)+br3(2*m)+br4(12*m));

mesh Thsi = buildmesh(br1(m)+br2(12*m)+br3(m)+br4(12*m)); // Initial
     solid domain

fespace Whi(Thsi,P1);
Whi Xoi=x,Yoi=y;
real[int] xx(25*m+1), yy(25*m+1);
int[int] nn(25*m);
for(int i=0;i<=25*m;i++){
        if(i<=12*m){ real t=ll*i/(12.0*m); xx[i]=cx0+x0+t; yy[i]=cy0-h2;
              }
                else if(i<=13*m){real t=2*h2*(i-12.0*m)/m-h2; xx[i]=x0+
                     cx0+ll; yy[i]=cy0+t;}
                else { real t=ll-ll*(i-13*m)/(12.0*m); xx[i]=cx0+x0+t;
                     yy[i]=cy0+h2; }
                if(i<25*m) nn[i]=2;
        }
border D(t=0,1;i){
        int ii = (i+1)%(25*m+1); real t1 = 1-t; real x1 = xx[i]*t1 + xx[
             ii]*t;
        real y2 = yy[i]*t1 + yy[ii]*t; x= Xoi(x1,y2); y=Yoi(x1,y2);
        label=la1;
}
plot(br1(m) + D(nn));
mesh Ths = buildmesh(br1(m) + D(nn));
```

```
int nbA;
real xnear=x0+cx0+l1+3*h2, ynear=cy0-3*h2, distmin=10;
plot(Ths);

mesh Thso=Ths;
//mesh Thf= buildmesh(a1(W*n/2)+a2(H*n)+a3(W*n)+a4(H*n)+D(nn));
//plot(a1(10*n)+a2(H*n)+a3(10*n)+a4(H*n)+D(nn));
plot(Thf,Thso, cmm="Initial configuration");


fespace Vh(Thf,P1b);  // velocity space
fespace Qh(Thf,P1);  // pressure space
fespace Wh(Ths,P1b);  // Lagrangian coordinates X,Y space
fespace Lh(Ths,P1);  // Lagrangian multiplier space
fespace Zh(Thf,[P1b,P1b,P1]); // fluid space
fespace Rh(Ths,[P1,P1,P1b,P1b]);// solid space

Vh u,v,uh,vh;
Qh p,ph;
Wh  Xoo=x,Yoo= y; // the X,Y are now the displacements
Rh [lamx,lamy,X,Y],[lamxo,lamyo,Xo,Yo]=[0,0,x,y];//x-(x-xc)/2,y+(y-yc)/2];
Zh [uo,vo,po]=[0,0,0];


macro div(u,v) ( dx(u)+dy(v) ) // EOM
macro Grad(u,v)[[dx(u),dy(u)],[dx(v),dy(v)]] // EOM
macro DD(u,v)  [[2*dx(u),div(v,u)],[div(v,u),2*dy(v)]] // EOM

varf aa([u,v,p],[uh,vh,ph]) =
          int2d(Thf)(rhof*[u,v]'*[uh,vh]/dt- div(uh,vh)*p -div(u,v)*ph
                     + penal*p*ph + nu*trace(DD(uh,vh)'*DD(u,v))/2)
          + on(1,4, u=0,v=0) +  on(3,u=Ubar*y*(H-y)*6/H/H,v=0) ;

varf bb([lamx,lamy,X,Y],[lamxh,lamyh,Xh,Yh]) =
          int2d(Ths)((rhos-rhof)*(X*Xh+Y*Yh)/dt2
                     - c1*trace(Grad(lamx,lamy)'*Grad(Xh,Yh)) - c2*(lamx*Xh+lamy*Yh)
                     - c1*trace(Grad(lamxh,lamyh)'*Grad(X,Y))/dt - c2*(lamxh*X+lamyh*Y)/dt
                     + mu*trace(DD(X,Y)'*DD(Xh,Yh))/2 + lambda*div(X,Y)*div(Xh,Yh)
                                         + penal*(lamx*lamxh+lamy*lamyh))
                                         + on(4,X=x,Y=y);

varf ab([u,v,p],[lamxh,lamyh,Xh,Yh]) =
          int2d(Ths,qft=qf9pT,mapu=[Xo,Yo])(
                       c1*trace(Grad(lamxh,lamyh)'*(Grad(Xo,Yo)*Grad(u,v)))
                     + c2*(lamxh*u+lamyh*v));

varf ba([lamx,lamy,X,Y],[uh,vh,ph]) =
          int2d(Ths,qft=qf9pT,mapt=[Xo,Yo])(
                       c1*trace(Grad(lamx,lamy)'*(Grad(Xo,Yo)*Grad(uh,vh)))
                     + c2*(lamx*uh+lamy*vh));

varf rhs1([u,v,p],[uh,vh,ph]) =
          int2d(Thf)(-rhof*gravity*vh * 0 // 0=no gravity in the fluid
                     + rhof*convect([uo,vo],-dt,uo)*uh/dt
                     + rhof*convect([uo,vo],-dt,vo)*vh/dt)
           + on(1,4, u=0,v=0) +  on(3,u=Ubar*y*(H-y)*6/H/H,v=0) ;

varf rhs2([lamx,lamy,X,Y],[lamxh,lamyh,Xh,Yh]) =
          int2d(Ths)(-(rhos-rhof)*gravity*Yh + 2*(mu+lambda)*div(Xh,Yh)
                     - c1*trace(Grad(lamxh,lamyh)'*Grad(Xo,Yo))/dt
                     - c2*(lamxh*Xo+lamyh*Yo)/dt
                     - (rhos-rhof)*((Xoo-2*Xo)*Xh+(Yoo-2*Yo)*Yh)/dt2)
                                         + on(4,X=x,Y=y);

real t0=0, MMCL=-100, MCL=-100,maxCL=-100,minCL=100;
for(int i=0;i<T/dt;i++){
//       cout<<"time= "<<i*dt<<endl;
                real[int] RHS1 = rhs1(0,Zh);
                real[int] RHS2 = rhs2(0,Rh);
                matrix AA = aa(Zh,Zh);
                matrix AB = ab(Zh,Rh);
                matrix BA = ba(Rh,Zh);
                matrix BB = bb(Rh,Rh);

                matrix AABB = [ [AA,BA], [AB,BB] ];
                set(AABB,solver=sparsesolver,master=-1);
                real[int] rhs = [RHS1, RHS2];
                real[int] w=AABB^-1*rhs;
                Xoo=Xo; Yoo=Yo;
                                Xoi=Xo; Yoi=Yo;
                            [uo[],lamxo[]] = w;
                Thso = buildmesh(br1(m) + D(nn));
                                Thf= buildmesh(fr1(20*m)+fr2(3.25*m)+fr3(20*m)+fr4(4*m)+fr5(12*m) + br1(m) + D(nn));
//                              plot(br1(m) + D(nn));
                                [uo,vo,po]=[uo,vo,po]; [lamxo,lamyo,Xo,Yo]=[lamxo,lamyo,Xo,Yo];
                uh=sqrt(uo*uo+vo*vo);
                plot(Thf, Thso, uh, value=1, fill=1, coef=100, cmm="t="+i*dt+"      Velocity");
                                int nbA;
                                real xnear=x0+cx0+l1+3*h2, ynear=cy0-3*h2, distmin=10;
                                for(int k=0;k<Ths.nv;k++)
                                                if((Thso(k).x-xnear)^2 + (Thso(k).y-ynear)^2 < distmin)
                                                        {distmin=(Thso(k).x-xnear)^2 + (Thso(k).y-ynear)^2; nbA=k;}
                            real CL=Thso(nbA).y;
                            if(minCL>CL) minCL=CL;
                            if(MMCL<MCL && CL<MCL && MCL>0.2 && MMCL>0.2) {
                                        cout<<"ft= "<<1./(i*dt-t0)<<" minCL= "<<minCL
                                        <<" maxCL= "<<MCL<<" (max-min)/2= "<< (MCL-minCL)/2<< endl;
                                        MCL=-100; MMCL=-100;
                                        t0=i*dt; minCL=10;
                                }
                              MMCL=MCL; MCL=CL;
                                cout<<i*dt<<"  "<<Thso(nbA).x<<" "<<Thso(nbA).y<<"  " << int2d(Thso)(1.)<< endl;
}
```

Appendix 2: Some Comments on the Codes

We assume that the reader has a basic experience with `FreeFem++`. The codes are then pretty straightforward to be understood. Given the fluid mesh `Thf`, for instance, the definition of the finite element spaces for the approximation of the Navier–Stokes equation is performed with the following lines

```
fespace Vh(Thf,P1b);
fespace Qh(Thf,P1);
```

in the case of the MINI element [3], or with the following lines

```
fespace Vh(Thf,P2);
fespace Qh(Thf,P1);
```

if the user prefers Taylor–Hood element. Analogous definition can be used for the two finite element spaces based on the solid mesh `Ths`, for instance,

```
fespace Wh(Ths,P1);
fespace Lh(Ths,P1);
```

Like in all fluid-structure interaction problems, one of the crucial parts of the code consists in the evaluation of the terms involving an interaction between the fluid and solid meshes. This occurs in two places of our code: in the assembly of the bilinear form `ab` and of `ba`. Let us look in more detail at the assembly of `ab`, for instance. The bilinear form to be approximated is

$$ab((\mathbf{u},p);(\hat{\boldsymbol{\lambda}},\hat{\mathbf{X}}))=\int_{\mathcal{B}}\Big(c_1\mathrm{D}\hat{\boldsymbol{\lambda}}:\mathrm{D}\mathbf{u}(\mathbf{X})+c_2\hat{\boldsymbol{\lambda}}\cdot\mathbf{u}(\mathbf{X})\Big)\,d\mathbf{s}$$

and the crucial terms involve $\mathbf{u}(\mathbf{X})$ where the finite element function $\mathbf{u}$, defined on the mesh `Thf` has to be evaluated on $\mathbf{X}$ which is defined on the mesh `Ths`. This interpolation problem is naturally solved by using the option `mapu` in the evaluation of the integral as follows:

```
varf ab([u,v,p],[lamxh,lamyh,Xh,Yh]) =
    int2d(Ths,qft=qf9pT,mapu=[Xo,Yo])(
        c1*trace(Grad(lamxh,lamyh)'
           *(Grad(Xo,Yo)*Grad(u,v)))
        + c2*(lamxh*u+lamyh*v));
```

Analogously, when the mapping between the meshes involves the test function, as in the case of the definition of `ba`, the option `mapt` comes into play. In this particular case, the bilinear form

$$ab((\boldsymbol{\lambda},\mathbf{X});(\hat{\mathbf{u}},\hat{p}))=\int_{\mathcal{B}}\big(c_1\mathrm{D}\boldsymbol{\lambda}:\mathrm{D}\hat{\mathbf{u}}(\mathbf{X})+c_2\boldsymbol{\lambda}\cdot\hat{\mathbf{u}}(\mathbf{X})\big)\,d\mathbf{s}$$

can be described with the following code instruction

```
varf ba([lamx,lamy,X,Y],[uh,vh,ph]) =
    int2d(Ths,qft=qf9pT,mapt=[Xo ,Yo])(
        c1*trace(Grad(lamx,lamy)'
           *(Grad(Xo,Yo)*Grad(uh,vh)))
        + c2*(lamx*uh+lamy*vh));
```

References

1. Boffi, D., Gastaldi, L.: A fictitious domain approach with Lagrange multiplier for fluid-structure interactions. Numer. Math. **135**(3), 711–732 (2017)
2. Boffi, D., Gastaldi, L., Heltai, L., Peskin, C.S.: On the hyper-elastic formulation of the immersed boundary method. Comput. Methods Appl. Mech. Eng. **197**(25–28), 2210–2231 (2008)
3. Boffi, D., Brezzi, F., Fortin, M.: Mixed Finite Element Methods and Applications. Springer Series in Computational Mathematics, vol. 44. Springer, New York (2013)
4. Boffi, D., Cavallini, N., Gastaldi, L.: The finite element immersed boundary method with distributed Lagrange multiplier. SIAM J. Numer. Anal. **53**(6), 2584–2604 (2015)
5. Dunne, Th., Rannacher, R., Richter, Th.: Numerical simulation of fluid-structure interaction based on monolithic variational formulations. In: Fundamental Trends in Fluid-Structure Interaction. Contemporary Challenges in Mathematical Fluid Dynamics and its Applications, vol. 1, pp. 1–75. World Scientific Publishing, Hackensack (2010)
6. Girault, V., Glowinski, R.: Error analysis of a fictitious domain method applied to a Dirichlet problem. Jpn. J. Ind. Appl. Math. **12**(3), 487–514 (1995)
7. Hecht, F.: New development in FreeFem++. J. Numer. Math. **20**(3–4), 251–265 (2012)
8. Hecht, F., Pironneau, O.: An energy stable monolithic Eulerian fluid-structure finite element method. Int. J. Numer. Methods Fluids **85**(7), 430–446 (2017)
9. Peskin, C.S.: The immersed boundary method. Acta Numer. **11**, 479–517 (2002)
10. Pironneau, O.: On the transport-diffusion algorithm and its applications to the Navier-Stokes equations. Numer. Math. **38**(3), 309–332 (1981/82)
11. Wang, Y., Jimack, P.K., Walkley, M.A.: A one-field monolithic fictitious domain method for fluid–structure interactions. Comput. Methods Appl. Mech. Eng. **317**, 1146–1168 (2017)
12. Zhao, H., Freund, J.B., Moser, R.D.: A fixed-mesh method for incompressible flow-structure systems with finite solid deformations. J. Comput. Phys. **227**(6), 3114–3140 (2008)