Introducción a la Programación de Elementos Finitos con FreeFem++ y su Aplicación en Mecánica de Fluidos

Sergio Caucao

Departamento de Matemática y Física Aplicadas (DMFA)

Grupo de Investigación en Análisis Numérico y Cálculo Científico (GIANuC^2)

XXXVI Jornada de Matemática de la Zona Sur

Universidad Católica de Temuco

24, 25, y 26 de Abril, 2024

Recolectando datos ...

Ingresar a:

www.menti.com

6988 3769

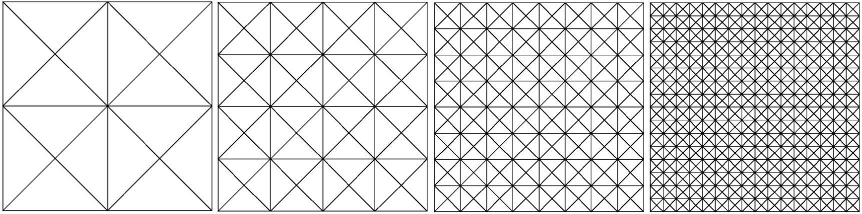
¿Qué entiendes por Método de Elementos Finitos ?

Una respuesta con un poco de historia ...

- El Método de los Elementos Finitos (MEF) es una técnica numérica usada ampliamente en el contexto de ingeniería y aplicaciones varias.
- El MEF es capaz de manejar sistemas complejos para los cuales no puede encontrarse soluciones analíticas explícitas.
- Fue desarrollado por primera vez en 1943 por Richard Courant, quien utilizó el método de Ritz del análisis numérico y la minimización del cálculo variacional para obtener soluciones aproximadas a sistemas vibrantes.
- En un artículo publicado en 1956 por M. J. Turner, R. W. Clough, H. C. Martin, y L. J. Topp, se estableció una definición más amplia del método y análisis numérico. El artículo se focalizaba en la **rigidez y la deflexión de estructuras complejas**. De aquí se acuñan los términos: Matriz de Rigidez y Vector de carga.

Una respuesta a grandes rasgos ...

- Dada una EDP, por ejemplo: $\left\{ \begin{array}{rcl} -\mathtt{D}^{-1}\Delta\, p = f & \text{en} & \Omega\,, \\ p = 0 & \text{en} & \partial\Omega\,. \end{array} \right.$ (Ecuación de Darcy)
- Obtener su formulación variacional a nivel continuo en H y discreto en H_h.
- Particionar/dividir el dominio donde se quiere resolver la EDP, en elementos más pequeños y finitos:



Una respuesta a grandes rasgos ...

- Escoger espacios discretos (polinomiales) H_h contenidos en su contraparte continua H.

- Considerar funciones bases φ_n, con n la cantidad de incógnitas, tales que, cualquier función en dicho espacio de dimensión finita se pueda escribir como una combinación lineal de ellas.

- En particular, $p_h = \sum_{i=1}^{n} \alpha_i \, \varphi_i$, con α_i incógnitas a calcular.

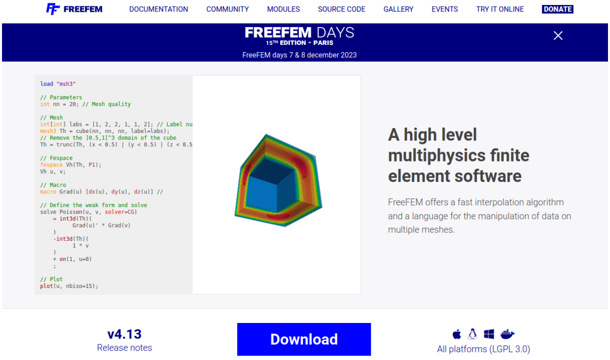
- Mediante cálculos locales (en cada triángulo de la malla) y un proceso de ensamble de la información, generar la matriz $\mathbf{A}$ y el vector $\mathbf{F}$ globales.

- Así, el problema original se reduce a resolver el sistema de ecuaciones lineales:

$$\mathbf{A}\,\boldsymbol{\alpha} = \mathbf{F}, \text{ con } \boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)\,.$$

Programación del Método de Elementos Finitos con FreeFem++

FreeFem++ es un toolbox para resolver ecuaciones diferenciales parciales. Creado y desarrollado desde 1987. Su sitio oficial es: http://www.freefem.org.



Creador: **Frédéric Hecht**

FreeFem++: Algunos Datos Importantes

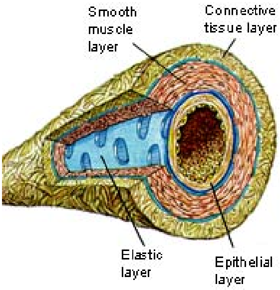
1. Los programas creados con FreeFem++ pueden usarse para resolver problemas en multifísica en 2D y 3D.
2. FreeFem++ es escrito en C++. Puede ser considerado como un lenguaje de programación en sí mismo.
3. Es multiplataforma, se puede instalar en sistemas operativos: MacOS, Windows, y GNU-Linux.
4. Para escribir los códigos, podemos utilizar cualquier editor de texto plano: vi, vim, nano, gedit, xed, notepad, etc.
5. En linux-mint/ubuntu, lo podemos instalar como: sudo apt-get install FreeFem++.
6. En general, lo podemos descargar desde el sitio oficial: http://www.freefem.org
7. Alternativamente, podemos descargar un entorno integrado para FreeFem++: FreeFem++-cs

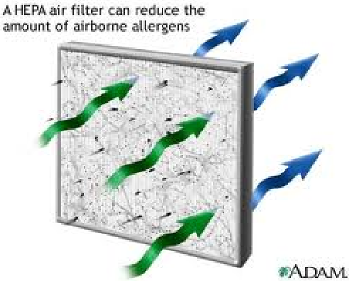
Manos al código ... iniciemos con el cursillo!!!

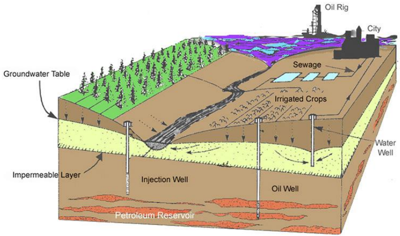
Contenidos del Cursillo

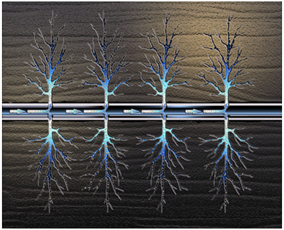
Motivación

Algunos fenómenos físicos en mecánica de fluidos









Notaciones en 2D: $\Delta p = \dfrac{\partial^2 p}{\partial x^2} + \dfrac{\partial^2 p}{\partial y^2}$, $\nabla p = \left(\dfrac{\partial p}{\partial x}, \dfrac{\partial p}{\partial y}\right)^{\mathrm{t}}$, y $\mathrm{div}(\mathbf{u}) = \dfrac{\partial u_1}{\partial x} + \dfrac{\partial u_2}{\partial y}$

Ecuaciones de Darcy: presión p – velocidad $\mathbf{u}$ y presión p

$$-\mathtt{D}^{-1}\,\Delta\, p = f \quad \text{en} \quad \Omega \Longleftrightarrow \left\{ \begin{array}{l} \mathtt{D}\,\mathbf{u} + \nabla p = \mathbf{0} \quad \text{en} \quad \Omega, \\ \mathrm{div}(\mathbf{u}) = f \quad \text{en} \quad \Omega, \end{array} \right. , \quad p = 0 \quad \text{en} \quad \Gamma\,.$$

Ecuaciones de Stokes: velocidad $\mathbf{u}$ y presión p

$$-\nu\,\Delta\,\mathbf{u} + \nabla p = \mathbf{f} \quad \text{en} \quad \Omega, \quad \mathrm{div}(\mathbf{u}) = 0 \quad \text{en} \quad \Omega, \quad \mathbf{u} = \mathbf{0} \quad \text{en} \quad \Gamma\,.$$

Ecuaciones de Brinkman–Forchheimer: velocidad $\mathbf{u}$ y presión p

$$-\nu\,\Delta\,\mathbf{u} + \mathtt{D}\,\mathbf{u} + \mathtt{F}\,|\mathbf{u}|\,\mathbf{u} + \nabla p = \mathbf{f} \quad \text{en} \quad \Omega, \quad \mathrm{div}(\mathbf{u}) = 0 \quad \text{en} \quad \Omega, \quad \mathbf{u} = \mathbf{0} \quad \text{en} \quad \Gamma\,.$$

Ecuación de Darcy:
Formulación Primal

Problema modelo

$$-\mathtt{D}^{-1}\,\Delta p = f \quad \text{en} \quad \Omega\,, \qquad p = 0 \quad \text{en} \quad \partial\Omega\,.$$

- p: presión
- $\mathtt{D}$: número de Darcy (permeabilidad)
- $n = 2, 3$: dimensión del espacio
- $\Omega \subseteq \mathrm{R}^n$: dominio poligonal/poliedral
- $\partial\Omega$: frontera de Ω
- $f \in L^2(\Omega)$

Multiplicando por una función test apropiada e integrando por partes, deducimos que

$$-\mathtt{D}^{-1}\int_\Omega \Delta p\, q = -\mathtt{D}^{-1}\int_{\partial\Omega}\frac{\partial p}{\partial n}\cancel{q}\,ds + \mathtt{D}^{-1}\int_\Omega \nabla p\cdot\nabla q = \mathtt{D}^{-1}\int_\Omega \nabla p\cdot\nabla q\,.$$

Formulación variacional: Hallar $p \in H_0^1(\Omega)$ tal que

$$a(p,q) := \mathtt{D}^{-1}\int_\Omega \nabla p\cdot\nabla q \;=\; \int_\Omega f\,q =: F(q) \qquad \forall\, q \in H_0^1(\Omega)\,.$$

Tiene solución?

Método de Elementos Finitos

Espacio de Elementos Finitos ($H_h \subset H^1(\Omega)$):

$$H_h := \left\{ q_h \in C(\bar{\Omega}) : \quad q_h|_T \in \mathbb{P}_k(T) \quad \forall\, T \in \mathcal{T}_h,\, k \geq 1 \right\}$$

$$H_{h,0} := H_h \cap H_0^1(\Omega)$$

Problema discreto: Hallar $p_h \in H_{h,0}$ tal que

$$a(p_h, q_h) := \mathtt{D}^{-1} \int_\Omega \nabla p_h \cdot \nabla q_h = \int_\Omega f\, q_h =: F(q_h) \qquad \forall\, q_h \in H_{h,0}\,.$$

P1: Tiene solución?

P2: Como calcular p_h?

P3: $\|p - p_h\|_{1,\Omega} \leq C(p)\, h^k$,

$h = \max_{T \in \mathcal{T}_h} h_T$ y h_T la mayor longitud entre dos puntos del triángulo T

Fórmula de tasa de convergencia

- Dadas dos particiones/mallas del dominio de estudio Ω, con tamaños de mallas h_{n-1} y h_n, respectivamente, se tienen las estimaciones del error

$$\|p - p_{h_{n-1}}\|_{1,\Omega} \leq C_{n-1}(p)\, h_{n-1}^k \quad \text{y} \quad \|p - p_{h_n}\|_{1,\Omega} \leq C_n(p)\, h_n^k\,,$$

  donde k es el orden de convergencia del método dependiendo de la norma con la que es medida el error y el grado polinomial utilizado para dicha aproximación.

- De lo anterior, deducimos que

$$\frac{\|p - p_{h_n}\|_{1,\Omega}}{\|p - p_{h_{n-1}}\|_{1,\Omega}} \approx \left(\frac{h_n}{h_{n-1}}\right)^k \,.$$

- Así, el órden de convergencia del método puede ser calculado como:

$$k \approx \log\left(\frac{\|p - p_{h_n}\|_{1,\Omega}}{\|p - p_{h_{n-1}}\|_{1,\Omega}}\right) \Big/ \log\left(\frac{h_n}{h_{n-1}}\right) \,.$$

- Llevemos a FreeFem++ nuestro primer ejemplo ...

Ejemplo 2D: Tasa de Convergencia para Formulación Primal

Solución para ilustrar tasa de convergencia teórica

- $\Omega := (0,1) \times (0,1)$
- $\mathtt{D} = 1$
- $p(x,y) = \cos(\pi\, x)\sin(\pi\, y)$
- $p_x(x,y) = -\pi\, \sin(\pi\, x)\sin(\pi\, y)$
- $p_y(x,y) = \pi\cos(\pi\, x)\cos(\pi\, y)$
- $f = -\mathtt{D}^{-1}\,(p_{xx}(x,y) + p_{yy}(x,y))$

Código FreeFEM++: Darcy-primal_2D.edp I

```
//
// This code solves the 2D Darcy problem in primal formulation
// with nonhomogeneous Dirichlet boundary condition
//  - (1/D)*\Delta p = f in \Omega,  p = pD on \Gamma.
//
// Global information
load "iovtk";      // for saving data in paraview format
load "UMFPACK64";  // UMFPACK solver
load "Element_P3";
//------------------------------------------------------------------------------------
//                Initial parameters
//------------------------------------------------------------------------------------
//----- Global parameters
int nref = 5;
real[int] H(nref);    // mesh size
real[int] DOF(nref);  // degrees of freedom
real[int] nbT(nref);  // degrees of freedom

//----- error
real[int] H1p(nref);

//----- rate of convergence
real[int] pH1rate(nref-1);
//------------------------------------------------------------------------------------
//                   Global data
//------------------------------------------------------------------------------------
//--- Data RHS
func p   = cos(pi*x)*sin(pi*y);
func px  = -pi*sin(pi*x)*sin(pi*y);
func py  =  pi*cos(pi*x)*cos(pi*y);
```

Código FreeFEM++: Darcy-primal_2D.edp II

```
func pxx = -(pi^2)*cos(pi*x)*sin(pi*y);
func pyy = -(pi^2)*cos(pi*x)*sin(pi*y);

real D = 1.;
func f   = -(1./D)*(pxx + pyy);

//--- Macros
macro gp [px,py] //
macro grad(qh) [dx(qh),dy(qh)] //
//----------------------------------------------------------------------------------
//                    Defining The Domain
//----------------------------------------------------------------------------------
for(int n = 0; n < nref; n++){

int size = 2^(n + 2); // space discretization
int Gamma = 11;

border GammaD1(t=0,1){x=t; y=0; label = Gamma;};
border GammaD2(t=0,1){x=1; y=t; label = Gamma;};
border GammaD3(t=1,0){x=t; y=1; label = Gamma;};
border GammaD4(t=1,0){x=0; y=t; label = Gamma;};

mesh Th = buildmesh(GammaD1(size) + GammaD2(size) + GammaD3(size) + GammaD4(size));
//plot(Th,wait=true);
//----------------------------------------------------------------------------------
//             Finite element spaces
//----------------------------------------------------------------------------------
fespace Hhp(Th,P1);

```

Código FreeFEM++: Darcy-primal_2D.edp III

```
fespace Vh(Th,P1); // discrete space to compute the meshsize
//-----------------------------------------------------------------------------------
//                  Defining the bilinear forms and RHS
//-----------------------------------------------------------------------------------
Hhp ph;
//----- bilinear forms
varf a(ph,qh) = int2d(Th)( (1./D)*(grad(ph)'*grad(qh)) ) + on(Gamma,ph=p);

//----- RHS
varf rhs(ph,qh) = int2d(Th)( f*qh ) + on(Gamma,ph=p);
//-----------------------------------------------------------------------------------
//                  Building matrices and Load vector
//-----------------------------------------------------------------------------------
matrix A = a(Hhp,Hhp);
//
real[int] RHS = rhs(0,Hhp);
//
set(A,solver = sparsesolver);

//----- calculating the solution
real[int] sol = A^-1*RHS;

//----- exporting data
ph[] = sol;

//-----  calculating the errors
H1p[n] = sqrt(int2d(Th)( (p - ph)^2 + (gp - grad(ph))'*(gp - grad(ph)) ));

//----- for the meshsize in Omega
```

Código FreeFEM++: Darcy-primal_2D.edp IV

```
Vh h = hTriangle;
H[n] = h[].max;
DOF[n] = Hhp.ndof;
nbT[n] = Th.nt;

//----- exporting to Praraview
savevtk("Data_Paraview_2D/Darcy-primal_aprox"+n+".vtk",Th,ph,dataname="ph");
savevtk("Data_Paraview_2D/Darcy-primal_exact"+n+".vtk",Th,p,dataname="p");
}
//----------------------------------------------------------------------------------
//                    showing the tables
//----------------------------------------------------------------------------------
cout << " p error in H1 = " << H1p <<endl;
for(int n =1; n < nref; n++)
pH1rate[n-1] = log(H1p[n]/H1p[n-1]) / log(H[n]/H[n-1]);
cout << " convergence rate p in H1 = " << pH1rate <<endl;

cout << " mesh size = " << H <<endl;
cout << " degrees of freedom = " << DOF <<endl;
cout << " number of Triangles = " << nbT <<endl;
```

Observaciones

- El grado polinomial se puede modificar en la opción: `fespace Hhp(Th,P2)`
- Código simple de migrar a 3D ... hagámoslo!!!

Ejemplo 3D: Tasa de Convergencia para Formulación Primal

Solución para ilustrar tasa de convergencia teórica

- $\Omega := (0,1) \times (0,1) \times (0,1)$
- $\mathtt{D} = 1$
- $p(x,y,z) = \cos(\pi\, x) \sin(\pi\, y) \exp(z)$
- $p_x(x,y,z) = -\pi\, \sin(\pi\, x) \sin(\pi\, y) \exp(z)$
- $p_y(x,y,z) = \pi \cos(\pi\, x) \cos(\pi\, y) \exp(z)$
- $p_z(x,y,z) = \cos(\pi\, x) \sin(\pi\, y) \exp(z)$
- $f = -\mathtt{D}^{-1} \left(p_{xx}(x,y,z) + p_{yy}(x,y,z) + p_{zz}(x,y,z) \right)$

Implementación Numérica en FreeFEM++: Darcy-primal_3D.edp I

```
//
// This code solves the 3D Darcy problem in primal formulation
// with nonhomogeneous Dirichlet boundary condition
//  - (1/D)*\Delta p = f in \Omega,  p = pD on \Gamma.
//
// Global information
load "msh3";
load "medit";
load "iovtk";
load "UMFPACK64";
include "cube.idp";
//----------------------------------------------------------------------------------
//                Initial parameters
//----------------------------------------------------------------------------------
//----- Global parameters
int nref = 5;
real[int] H(nref);    // mesh size
real[int] DOF(nref);  // degrees of freedom
real[int] nbT(nref);  // degrees of freedom

//----- error
real[int] H1p(nref);

//----- rate of convergence
real[int] pH1rate(nref-1);
//----------------------------------------------------------------------------------
//                   Global data
//----------------------------------------------------------------------------------
//--- Data RHS
```

Implementación Numérica en FreeFEM++: Darcy-primal_3D.edp II

```
func p   =  cos(pi*x)*sin(pi*y)*exp(z);
func px  = -pi*sin(pi*x)*sin(pi*y)*exp(z);
func py  =  pi*cos(pi*x)*cos(pi*y)*exp(z);
func pz  =  cos(pi*x)*sin(pi*y)*exp(z);
func pxx = -(pi^2)*cos(pi*x)*sin(pi*y)*exp(z);
func pyy = -(pi^2)*cos(pi*x)*sin(pi*y)*exp(z);
func pzz =  cos(pi*x)*sin(pi*y)*exp(z);

real D = 1.;
func f   = -(1./D)*(pxx + pyy + pzz);

//--- Macros
macro gp [px,py,pz] //
macro grad(qh) [dx(qh),dy(qh),dz(qh)] //
//-------------------------------------------------------------------------------------
//                   Defining The Domain
//-------------------------------------------------------------------------------------
for(int n = 0; n < nref; n++){

int size = n^2 + n + 2.; // space discretization
int Gamma = 11;

int[int] NN = [size,size,size]; //  the number of step in each direction
real[int,int] BB = [[0,1],[0,1],[0,1]];
int[int,int] LL = [[Gamma,Gamma],[Gamma,Gamma],[Gamma,Gamma]]; // left,right,front, back, down
      , top
mesh3 Th = Cube(NN,BB,LL);

//medit("cube",Th);
```

Implementación Numérica en FreeFEM++: Darcy-primal_3D.edp III

```
//---------------------------------------------------------------------------------
//            Finite element spaces
//---------------------------------------------------------------------------------
fespace Hhp(Th,P13d);

fespace Vh(Th,P13d); // discrete space to compute the meshsize
//---------------------------------------------------------------------------------
//                  Defining the bilinear forms and RHS
//---------------------------------------------------------------------------------
Hhp ph;
//----- bilinear forms
varf a(ph,qh) = int3d(Th)( (1./D)*(grad(ph)'*grad(qh)) ) + on(Gamma,ph=p);

//----- RHS
varf rhs(ph,qh) = int3d(Th)( f*qh ) + on(Gamma,ph=p);
//---------------------------------------------------------------------------------
//                  Building matrices and Load vector
//---------------------------------------------------------------------------------
matrix A = a(Hhp,Hhp);
//
real[int] RHS = rhs(0,Hhp);
//
set(A,solver = sparsesolver);

//----- calculating the solution
real[int] sol = A^-1*RHS;

//----- exporting data
ph[] = sol;
```

Implementación Numérica en FreeFEM++: Darcy-primal_3D.edp IV

```

//-----  calculating the errors
H1p[n] = sqrt(int3d(Th)( (p - ph)^2 + (gp - grad(ph))'*(gp - grad(ph)) ));

//----- for the meshsize in Omega
Vh h = hTriangle;
H[n] = h[].max;
DOF[n] = Hhp.ndof;
nbT[n] = Th.nt;

//----- exporting to Praraview
savevtk("Data_Paraview_3D/Darcy-primal_aprox"+n+".vtk",Th,ph,dataname="ph");
savevtk("Data_Paraview_3D/Darcy-primal_exact"+n+".vtk",Th,p,dataname="p");
}
//------------------------------------------------------------------------------------
//                    showing the tables
//------------------------------------------------------------------------------------
cout << " p error in H1 = " << H1p <<endl;
for(int n =1; n < nref; n++)
pH1rate[n-1] = log(H1p[n]/H1p[n-1]) / log(H[n]/H[n-1]);
cout << " convergence rate p in H1 = " << pH1rate <<endl;

cout << " mesh size = " << H <<endl;
cout << " degrees of freedom = " << DOF <<endl;
cout << " number of Tetrahedra = " << nbT <<endl;
```

Ecuación de Darcy: Formulación Mixta

Formulación Mixta

Problema de Darcy en formulación mixta

$$\mathbf{u} = -\mathtt{D}^{-1}\nabla p \quad \text{en} \quad \Omega, \quad \operatorname{div}(\mathbf{u}) = f \quad \text{en} \quad \Omega, \quad p = p_D \quad \text{en} \quad \Gamma$$

$$\mathbf{H}(\operatorname{div};\Omega) := \left\{ \mathbf{v} = (v_1, v_2) \in \mathbf{L}^2(\Omega) : \quad \operatorname{div}(\mathbf{v}) \in L^2(\Omega) \right\}, \text{ con } \operatorname{div}(\mathbf{v}) = \frac{\partial v_1}{\partial x} + \frac{\partial v_2}{\partial y}.$$

Formulación variacional

Hallar $(\mathbf{u}, p) \in \mathbf{H}(\operatorname{div};\Omega) \times L^2(\Omega)$ tal que:

$$\begin{aligned} \mathtt{D}\int_\Omega \mathbf{u}\cdot\mathbf{v} - \int_\Omega p \operatorname{div}(\mathbf{v}) &= -\langle \mathbf{v}\cdot\mathbf{n}, p_D\rangle_\Gamma & \forall\, \mathbf{v} \in \mathbf{H}(\operatorname{div};\Omega), \\ -\int_\Omega q \operatorname{div}(\mathbf{u}) &= -\int_\Omega f\, q & \forall\, q \in L^2(\Omega) \end{aligned}$$

G.N. GATICA, *A Simple Introduction to the Mixed Finite Element Method. Theory and Applications*. Springer Briefs in Mathematics. Springer, Cham, 2014.

Formulación variacional discreta

Hallar $(\mathbf{u}_h, p_h) \in \mathbf{H}_h^{\mathbf{u}} \times H_h^p$ tal que:

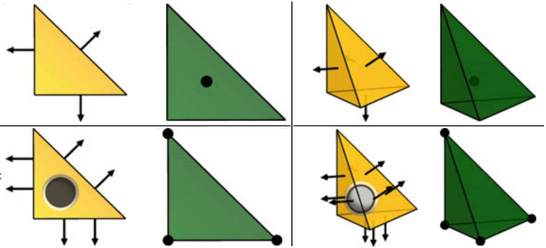
$$\begin{aligned} \mathtt{D} \int_\Omega \mathbf{u}_h \cdot \mathbf{v}_h - \int_\Omega p_h \, \mathrm{div}(\mathbf{v}_h) &= -\langle \mathbf{v}_h \cdot \mathbf{n}, p_D \rangle_\Gamma & \forall\, \mathbf{v}_h \in \mathbf{H}_h^{\mathbf{u}}, \\ -\int_\Omega q_h \, \mathrm{div}(\mathbf{u}_h) &= -\int_\Omega f\, q_h & \forall\, q_h \in H_h^p. \end{aligned}$$

$$\mathbf{H}_h^{\mathbf{u}} := \Big\{ \mathbf{v}_h \in \mathbf{H}(\mathrm{div};\Omega) : \quad \mathbf{v}_h|_T \in \mathbf{RT}_k(T) \quad \forall\, T \in \mathcal{T}_h \Big\},$$

$$H_h^p := \Big\{ q_h \in L^2(\Omega) : \quad q_h|_T \in \mathrm{P}_k(T) \quad \forall\, T \in \mathcal{T}_h \Big\}.$$

RT0-P0

RT1-P1dc



Tasas de convergencia

P1: Tiene solución?

P2: Como calcular $\mathbf{u}_h, p_h$?

P3: $\|\mathbf{u} - \mathbf{u}_h\|_{\mathrm{div};\Omega} + \|p - p_h\|_{0,\Omega} \leq C(\mathbf{u}, p)\, h^{k+1}$, con $k \geq 0$

Manos al código ... iniciemos con nuestro tercer programa ...

Ejemplo 2D: Tasa de Convergencia para Formulación Mixta

Solución para ilustrar tasa de convergencia teórica

- $\Omega := (0,1) \times (0,1)$
- $\mathtt{D} = 1$
- $p(x,y) = \cos(\pi\, x)\sin(\pi\, y)$
- $p_x(x,y) = -\pi\; \sin(\pi\, x)\sin(\pi\, y)$
- $p_y(x,y) = \pi\cos(\pi\, x)\cos(\pi\, y)$
- $\mathbf{u}(x,y) = -\mathtt{D}^{-1}\,\big(p_x(x,y), p_y(x,y)\big)^{\mathrm{t}}$
- $f = -\mathtt{D}^{-1}\,(p_{xx}(x,y) + p_{yy}(x,y))$

Código FreeFEM++: Darcy-mixto_2D.edp I

```
//
// This code solves the 2D Darcy problem in mixed formulation with
// nonhomogeneous Dirichlet boundary conditions
//     u = -(1/D)*\nabla p \qin \Omega, div(u) = f in \Omega,
//                    p = pD on \Gamma.
//
// Global information
load "iovtk";           // for saving data in paraview format
load "UMFPACK64";   // UMFPACK solver
load "Element_Mixte";
//------------------------------------------------------------------------------------
//                Initial parameters
//------------------------------------------------------------------------------------
//----- Global parameters
int nref = 5;
real[int] H(nref);    // mesh size
real[int] DOF(nref);  // degrees of freedom
real[int] nbT(nref);  // degrees of freedom

//----- errors
real[int] uerror(nref);
real[int] perror(nref);

//----- rate of convergence
real[int] urate(nref-1);
real[int] prate(nref-1);
//------------------------------------------------------------------------------------
//                   Global data
//------------------------------------------------------------------------------------
```

Código FreeFEM++: Darcy-mixto_2D.edp II

```
//--- Data RHS
func p   = cos(pi*x)*sin(pi*y);
func px  = -pi*sin(pi*x)*sin(pi*y);
func py  =  pi*cos(pi*x)*cos(pi*y);
func pxx = -(pi^2)*cos(pi*x)*sin(pi*y);
func pyy = -(pi^2)*cos(pi*x)*sin(pi*y);
//
real D = 1.;
func f   = -(1./D)*(pxx + pyy);
//
//----- Macros
macro u [-(1./D)*px,-(1./D)*py] //

macro uh [uh1,uh2] //
macro vh [vh1,vh2] //

macro norm [N.x,N.y] //
macro div(vh) ( dx(vh[0]) + dy(vh[1]) ) //
//------------------------------------------------------------------------------------
//                   Defining The Domain
//------------------------------------------------------------------------------------
for(int n = 0; n < nref; n++){
//
int size = 2^(n + 2); // space discretization
int Gamma = 11;

border GammaD1(t=0,1){x=t; y=0; label = Gamma;};
border GammaD2(t=0,1){x=1; y=t; label = Gamma;};
border GammaD3(t=1,0){x=t; y=1; label = Gamma;};
```

Código FreeFEM++: Darcy-mixto_2D.edp III

```
border GammaD4(t=1,0){x=0; y=t; label = Gamma;};

mesh Th = buildmesh(GammaD1(size) + GammaD2(size) + GammaD3(size) + GammaD4(size));
//---------------------------------------------------------------------------------
//            Finite element spaces
//---------------------------------------------------------------------------------
fespace Hhu(Th,RT0);
fespace Hhp(Th,P0);

fespace Vh(Th,P1); // discrete space to compute the meshsize
//---------------------------------------------------------------------------------
//                  Defining the bilinear forms
//---------------------------------------------------------------------------------
Hhu uh; Hhp ph;
//----- bilinear forms
varf a(uh,vh)   = int2d(Th)( D*(uh'*vh) );
varf b([ph],vh) = int2d(Th)( -(ph*div(vh)) );

//----- RHS
varf rhs1(uh,vh) = int1d(Th,Gamma)( -p*(vh'*norm) );
varf rhs2(ph,qh) = int2d(Th)( -(f*qh) );
//---------------------------------------------------------------------------------
//                  Building matrices
//---------------------------------------------------------------------------------
matrix A = a(Hhu,Hhu);
matrix B = b(Hhp,Hhu);

matrix M;{
M = [[  A, B],
```

Código FreeFEM++: Darcy-mixto_2D.edp IV

```
     [ B', 0]];}
//--------------------------------------------------------------------------------
//                      Load vector
//--------------------------------------------------------------------------------
real[int] RHS1 = rhs1(0,Hhu);
real[int] RHS2 = rhs2(0,Hhp);

real[int] L = [RHS1, RHS2];

set(M,solver = sparsesolver);

//----- calculating the solution
real[int] sol = M^-1*L;

//----- exporting data
uh1[] = sol(0:Hhu.ndof - 1);
ph[]  = sol(Hhu.ndof:Hhu.ndof + Hhp.ndof - 1);

//-----  calculating the errors
uerror[n] = sqrt(int2d(Th)( (u - uh)'*(u - uh) + (f - div(uh))^2 ));
perror[n] = sqrt(int2d(Th)( (p - ph)^2 ));

//----- for the meshsize in Omega
Vh h = hTriangle;
H[n] = h[].max;
DOF[n] = Hhu.ndof + Hhp.ndof;
nbT[n] = Th.nt;

//----- exporting to Praraview
```

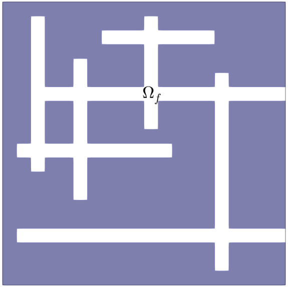
Código FreeFEM++: Darcy-mixto_2D.edp V

```
savevtk("Data_Paraview_2D/Darcy-mixto_aprox"+n+".vtk",Th,[uh1,uh2,0],ph,dataname="uh ph");
savevtk("Data_Paraview_2D/Darcy-mixto_exact"+n+".vtk",Th,[px,py,0],p,dataname="u p");
}
//----------------------------------------------------------------------------------
//                    showing the tables
//----------------------------------------------------------------------------------
cout << " u error in Hdiv = " << uerror <<endl;
for(int n = 1; n < nref; n++)
urate[n-1] = log(uerror[n]/uerror[n-1]) / log(H[n]/H[n-1]);
cout << " convergence rate u in Hdiv = " << urate <<endl;

cout << " p error in L2 = " << perror <<endl;
for(int n =1; n < nref; n++)
prate[n-1] = log(perror[n]/perror[n-1]) / log(H[n]/H[n-1]);
cout << " convergence rate p in L2 = " << prate <<endl;

cout << " mesh size = " << H <<endl;
cout << " degrees of freedom = " << DOF <<endl;
cout << " number of Triangles = " << nbT <<endl;
```

Fluido en un medio poroso 2D con fracturas

Parámetros

- $\Omega = (-1,1)^2$
- $\mathtt{D} = \begin{cases} 10 & \text{en} \quad \overline{\Omega} \setminus \Omega_{\mathrm{f}} \\ 1 & \text{en} \quad \Omega_{\mathrm{f}} \end{cases}$



$$p = \begin{cases} -0.5\,(y-1) & \text{en} \quad \Gamma_{\text{left}}\,, \\ -0.5\,(x-1) & \text{en} \quad \Gamma_{\text{bottom}}\,, \end{cases}$$

$$p = 0 \quad \text{en} \quad \Gamma_{\text{right}} \cup \Gamma_{\text{top}}\,,$$

Figure: Izquierda: dominio computacional. Derecha: condiciones de contorno.

Código FreeFEM++: Darcy-mixto-fracture_2D.edp I

```
//
// This code solves the Darcy problem in mixed formulation with
// nonhomogeneous Dirichlet boundary conditions in a fracture domain
//     u = -(1/D)*\nabla p in \Omega, div(u) = f in \Omega,
//                          p = pD on \Gamma.
//
// Global information
load "iovtk";            // for saving data in paraview format
load "UMFPACK64";    // UMFPACK solver
load "Element_Mixte";
//------------------------------------------------------------------------------------
//                    Global data
//------------------------------------------------------------------------------------
//----- Global parameters
real H, DOF, nbT;

//--- Data RHS
real D1 = 10;
real D2 = 1.;

func pLeft = -0.5*(y-1.);
func pBottom = -0.5*(x-1.);
func f = 0.;

//----- Macros
macro uh [uh1,uh2] //
macro vh [vh1,vh2] //

macro norm [N.x,N.y] //
```

Código FreeFEM++: Darcy-mixto-fracture_2D.edp II

```
macro div(vh) ( dx(vh[0]) + dy(vh[1]) ) //
//------------------------------------------------------------------------------------
//                   Defining The Domain
//------------------------------------------------------------------------------------
mesh Th = readmesh("Fracture_network-mesh.msh");
// Labels setting:
// 33: region outside the fracture
// 34: region inside the fracture
// 1: bottom boundary
// 22: right and top boundaries
// 4: left boundary
//------------------------------------------------------------------------------------
//            Finite element spaces
//------------------------------------------------------------------------------------
fespace Hhu(Th,RT0);
fespace Hhp(Th,P0);

fespace Vh(Th,P1); // discrete space to compute the meshsize
//------------------------------------------------------------------------------------
//                  Defining the bilinear forms
//------------------------------------------------------------------------------------
Hhu uh; Hhp ph;
//----- bilinear forms
varf a(uh,vh)   = int2d(Th,33)( D1*(uh'*vh) ) + int2d(Th,34)( D2*(uh'*vh) );
varf b([ph],vh) = int2d(Th)( -(ph*div(vh)) );

//----- RHS
varf rhs1(uh,vh) = int1d(Th,1)( -pBottom*(vh'*norm) ) + int1d(Th,4)( -pLeft*(vh'*norm) );
varf rhs2(ph,qh) = int2d(Th)( -(f*qh) );
```

Código FreeFEM++: Darcy-mixto-fracture_2D.edp III

```
//------------------------------------------------------------------------------------
//                    Building matrices
//------------------------------------------------------------------------------------
matrix A = a(Hhu,Hhu);
matrix B = b(Hhp,Hhu);

matrix M;{
M = [[  A, B],
     [ B', 0]];}
//------------------------------------------------------------------------------------
//                        Load vector
//------------------------------------------------------------------------------------
real[int] RHS1 = rhs1(0,Hhu);
real[int] RHS2 = rhs2(0,Hhp);

real[int] L = [RHS1, RHS2];

set(M,solver = sparsesolver);

//----- calculating the solution
real[int] sol = M^-1*L;

//----- exporting data
uh1[] = sol(0:Hhu.ndof - 1);
ph[]  = sol(Hhu.ndof:Hhu.ndof + Hhp.ndof - 1);

//----- for the meshsize in Omega
Vh h = hTriangle;
H = h[].max;
```

Código FreeFEM++: Darcy-mixto-fracture_2D.edp IV

```
DOF = Hhu.ndof + Hhp.ndof;
nbT = Th.nt;

//----- exporting to Praraview
savevtk("Data_Paraview_2D/Darcy-mixto-fracture_aprox.vtk",Th,[uh1,uh2,0],ph,dataname="uh ph");
//------------------------------------------------------------------------------------
//                    showing the tables
//------------------------------------------------------------------------------------
cout << " mesh size = " << H <<endl;
cout << " degrees of freedom = " << DOF <<endl;
cout << " number of Triangles = " << nbT <<endl;
```

Observaciones

- Podemos cargar mallas creadas con malladores externos, como `Gmsh` o `Triangle`, siempre y cuando tengan el formato `msh` soportado por FreeFEM++
- Como este ejemplo no tiene solución analítica, realizamos un estudio cualitativo de los resultados obtenidos. Usemos ParaView!!!

ParaView como una herramienta para crear imágenes de alta calidad

https://www.paraview.org

Ecuaciones de Brinkman–Forchheimer

Formulación velocidad-presión con $\rho \in [3,4]$:

$$-\nu\,\Delta\mathbf{u} + \mathtt{D}\,\mathbf{u} + \mathtt{F}\,|\mathbf{u}|^{\rho-2}\mathbf{u} + \nabla p = \mathbf{f} \quad \text{en} \quad \Omega, \quad \operatorname{div}(\mathbf{u}) = 0 \quad \text{en} \quad \Omega,$$

$$\mathbf{u} = \mathbf{u}_{\mathrm{D}} \quad \text{en} \quad \Gamma, \quad \int_\Omega p = 0\,.$$

$$\boldsymbol{\sigma} := \nu\nabla\mathbf{u} - p\,\mathbf{I} \quad \text{and} \quad \operatorname{div}(\mathbf{u}) = 0 \quad \text{en} \quad \Omega \Longleftrightarrow \operatorname{tr}(\boldsymbol{\sigma}) = -n\,p \quad \text{en} \quad \Omega\,,$$

de donde

$$p = -\frac{1}{n}\operatorname{tr}(\boldsymbol{\sigma}), \quad \text{and} \quad \boldsymbol{\sigma}^{\mathrm{d}} := \boldsymbol{\sigma} - \frac{1}{n}\operatorname{tr}(\boldsymbol{\sigma})\mathbf{I}\,.$$

Formulación mixta (Pseudoesfuerzo-velocidad)

$$\frac{1}{\nu}\boldsymbol{\sigma}^{\mathrm{d}} = \nabla\mathbf{u} \quad \text{en} \quad \Omega, \quad \mathtt{D}\,\mathbf{u} + \mathtt{F}\,|\mathbf{u}|^{\rho-2}\mathbf{u} - \mathbf{div}(\boldsymbol{\sigma}) = \mathbf{f} \quad \text{en} \quad \Omega,$$

$$\mathbf{u} = \mathbf{u}_{\mathrm{D}} \quad \text{en} \quad \Gamma, \quad \int_\Omega \operatorname{tr}(\boldsymbol{\sigma}) = 0$$

S. Caucao and I. Yotov, *A Banach space mixed formulation for the unsteady Brinkman–Forchheimer equations*. IMA Journal of Numerical Analysis, vol. 41, 4, pp. 2708-2743, (2021).

Consideremos $\varrho \in [4/3, 3/2]$ tal que $1/\rho + 1/\varrho = 1$

- $\mathbb{H}(\mathbf{div}_\varrho;\Omega) := \left\{ \boldsymbol{\tau} \in \mathbf{L}^2(\Omega) : \quad \mathbf{div}(\boldsymbol{\tau}) \in \mathbf{L}^\varrho(\Omega) \right\}$
- $\mathbb{H}_0(\mathbf{div}_\varrho;\Omega) := \left\{ \boldsymbol{\tau} \in \mathbb{H}(\mathbf{div}_\varrho;\Omega) : \quad \int_\Omega \mathrm{tr}(\boldsymbol{\tau}) = 0 \right\}$
- Notar que $\boldsymbol{\sigma} \in \mathbb{H}_0(\mathbf{div}_\varrho;\Omega)$.

Formulación variacional

Hallar $(\boldsymbol{\sigma}, \mathbf{u}) \in \mathbb{H}_0(\mathbf{div}_\varrho;\Omega) \times \mathbf{L}^\rho(\Omega)$ tal que:

$$
\begin{aligned}
\frac{1}{\nu}\int_\Omega \boldsymbol{\sigma}^{\mathrm{d}} : \boldsymbol{\tau}^{\mathrm{d}} + \int_\Omega \mathbf{u}\cdot\mathbf{div}(\boldsymbol{\tau}) &= \langle \boldsymbol{\tau}\mathbf{n}, \mathbf{u}_D \rangle_\Gamma \,, \\
\int_\Omega \mathbf{v}\cdot\mathbf{div}(\boldsymbol{\sigma}) - \mathtt{D}\int_\Omega \mathbf{u}\cdot\mathbf{v} - \mathtt{F}\int_\Omega |\mathbf{u}|^{\rho-2}\mathbf{u}\cdot\mathbf{v} &= -\int_\Omega \mathbf{f}\cdot\mathbf{v}\,,
\end{aligned}
$$

para todo $(\boldsymbol{\tau}, \mathbf{v}) \in \mathbb{H}_0(\mathbf{div}_\varrho;\Omega) \times \mathbf{L}^\rho(\Omega)$.

Formulación variacional discreta

Hallar $(\boldsymbol{\sigma}_h, \mathbf{u}_h) \in \mathbb{H}^{\boldsymbol{\sigma}}_{h,0} \times \mathbf{H}^{\mathbf{u}}_h$ tal que:

$$\begin{aligned} \frac{1}{\nu}\int_\Omega \boldsymbol{\sigma}_h^{\mathrm{d}} : \boldsymbol{\tau}_h^{\mathrm{d}} + \int_\Omega \mathbf{u}_h \cdot \mathbf{div}(\boldsymbol{\tau}_h) &= \langle \boldsymbol{\tau}_h \mathbf{n}, \mathbf{u}_D \rangle_\Gamma \,, \\ \int_\Omega \mathbf{v}_h \cdot \mathbf{div}(\boldsymbol{\sigma}_h) - \mathtt{D}\int_\Omega \mathbf{u}_h \cdot \mathbf{v}_h - \mathtt{F}\int_\Omega |\mathbf{u}_h|^{\rho-2}\mathbf{u}_h \cdot \mathbf{v}_h &= -\int_\Omega \mathbf{f} \cdot \mathbf{v}_h \,, \end{aligned}$$

para todo $(\boldsymbol{\tau}_h, \mathbf{v}_h) \in \mathbb{H}^{\boldsymbol{\sigma}}_{h,0} \times \mathbf{H}^{\mathbf{u}}_h$.

$$\mathbb{H}^{\boldsymbol{\sigma}}_h := \Big\{ \boldsymbol{\tau}_h \in \mathbb{H}(\mathbf{div}_\varrho;\Omega) : \quad \mathbf{c}^{\mathrm{t}} \boldsymbol{\tau}_h|_T \in \mathbf{RT}_k(T) \quad \forall\, \mathbf{c} \in \mathrm{R}^n, \quad \forall\, T \in \mathcal{T}_h \Big\},$$

$$\mathbb{H}^{\boldsymbol{\sigma}}_{h,0} := \mathbb{H}^{\boldsymbol{\sigma}}_h \cap \mathbb{H}_0(\mathbf{div}_\varrho;\Omega)\,,$$

$$\mathbf{H}^u_h := \Big\{ \mathbf{v}_h \in \mathbf{L}^\rho(\Omega) : \quad \mathbf{v}_h|_T \in \mathbf{P}_k(T) \quad \forall\, T \in \mathcal{T}_h \Big\}.$$

Formulación variacional discreta

Hallar $(\boldsymbol{\sigma}_h, \mathbf{u}_h, \lambda_h) \in \mathbb{H}_h^{\boldsymbol{\sigma}} \times \mathbf{H}_h^{\mathbf{u}} \times \mathrm{R}$ tal que:

$$\begin{aligned}
&\frac{1}{\nu}\int_\Omega \boldsymbol{\sigma}_h^{\mathrm{d}} : \boldsymbol{\tau}_h^{\mathrm{d}} + \int_\Omega \mathbf{u}_h \cdot \mathbf{div}(\boldsymbol{\tau}_h) + \lambda_h \int_\Omega \mathrm{tr}(\boldsymbol{\tau}_h) && = \langle \boldsymbol{\tau}_h \mathbf{n}, \mathbf{u}_D \rangle_\Gamma \,, \\
&\int_\Omega \mathbf{v}_h \cdot \mathbf{div}(\boldsymbol{\sigma}_h) - \mathtt{D}\int_\Omega \mathbf{u}_h \cdot \mathbf{v}_h - \mathtt{F}\int_\Omega |\mathbf{u}_h|^{\rho-2}\mathbf{u}_h \cdot \mathbf{v}_h && = -\int_\Omega \mathbf{f} \cdot \mathbf{v}_h \,, \\
&\eta_h \int_\Omega \mathrm{tr}(\boldsymbol{\sigma}_h) && = 0 \,,
\end{aligned}$$

para todo $(\boldsymbol{\tau}_h, \mathbf{v}_h, \eta_h) \in \mathbb{H}_h^{\boldsymbol{\sigma}} \times \mathbf{H}_h^{\mathbf{u}} \times \mathrm{R}$.

Notar que:

$$p = -\frac{1}{n}\,\mathrm{tr}(\boldsymbol{\sigma}) \quad \text{y} \quad \nabla \mathbf{u} = \frac{1}{\nu}\,\boldsymbol{\sigma}^{\mathrm{d}} \,.$$

Variables postprocesadas

$$p_h = -\frac{1}{n}\,\mathrm{tr}(\boldsymbol{\sigma}_h) \quad \text{y} \quad \mathbf{G}_h = \frac{1}{\nu}\,\boldsymbol{\sigma}_h^{\mathrm{d}} \,.$$

Estrategia Iterativa de Picard

Dado $\mathbf{u}_h^0 \in \mathbf{H}_h^{\mathbf{u}}$, para $m \geq 1$, hallar $(\boldsymbol{\sigma}_h^m, \mathbf{u}_h^m, \lambda_h^m) \in \mathbb{H}_h^{\boldsymbol{\sigma}} \times \mathbf{H}_h^{\mathbf{u}} \times \mathrm{R}$ tal que:

$$\frac{1}{\nu}\int_\Omega (\boldsymbol{\sigma}_h^m)^{\mathrm{d}} : \boldsymbol{\tau}_h^{\mathrm{d}} + \int_\Omega \mathbf{u}_h^m \cdot \mathbf{div}(\boldsymbol{\tau}_h) + \lambda_h^m \int_\Omega \mathrm{tr}(\boldsymbol{\tau}_h) = \langle \boldsymbol{\tau}_h \mathbf{n}, \mathbf{u}_D \rangle_\Gamma \ ,$$

$$\int_\Omega \mathbf{v}_h \cdot \mathbf{div}(\boldsymbol{\sigma}_h^m) - \mathtt{D}\int_\Omega \mathbf{u}_h^m \cdot \mathbf{v}_h - \mathtt{F}\int_\Omega |\mathbf{u}_h^{m-1}|^{\rho-2}\mathbf{u}_h^m \cdot \mathbf{v}_h = -\int_\Omega \mathbf{f} \cdot \mathbf{v}_h \,,$$

$$\eta_h \int_\Omega \mathrm{tr}(\boldsymbol{\sigma}_h^m) = 0 \,,$$

para todo $(\boldsymbol{\tau}_h, \mathbf{v}_h, \eta_h) \in \mathbb{H}_h^{\boldsymbol{\sigma}} \times \mathbf{H}_h^{\mathbf{u}} \times \mathrm{R}$.

Ejemplo 2D: Tasa de Convergencia para Formulación Mixta

Solución para ilustrar tasa de convergencia teórica

- $\Omega := (0,1) \times (0,1)$
- $\nu = 1$, $\mathtt{D} = 1$, $\mathtt{F} = 10$, y $\rho = 3$
- $p(x,y) = \cos(\pi\, x)\sin\left(\dfrac{\pi}{2}\, y\right)$
- $\mathbf{u}(x,y) = \begin{pmatrix} \sin(\pi\, x)\cos(\pi\, y) \\ -\cos(\pi\, x)\sin(\pi\, y) \end{pmatrix}$
- $\mathbf{f} = \mathtt{D}\,\mathbf{u} + \mathtt{F}\,|\mathbf{u}|\mathbf{u} - \mathbf{div}(\boldsymbol{\sigma})$

Código FreeFEM++: BF-mixto_2D-Picard.edp I

```
//
// This code solves a 2D mixed finite element method
// for the Brinkman-Forchheimer equations
//     sig = nu*\nabla u - p*I \qin \Omega, D*u + F*|u|**(rho-2)*u - div(sig) = f in \Omega,
//                       u = uD on \Gamma,   int_Omega tr(sig) = 0.
//
// Global information
load "iovtk";          // for saving data in paraview format
load "UMFPACK64";      // UMFPACK solver
load "Element_Mixte"; // for using RT1
//------------------------------------------------------------------------------------
//                Initial parameters
//------------------------------------------------------------------------------------
//----- Global parameters
int nref = 4;
real tol;

real[int] H(nref);
real[int] DOF(nref);
real[int] nbT(nref);
real[int] iterations(nref);

//----- errors
real[int] sigerror(nref);
real[int] uerror(nref);
real[int] perror(nref);
real[int] Guerror(nref);

//----- rate of convergence
```

Código FreeFEM++: BF-mixto_2D-Picard.edp II

```
real[int] sigrate(nref-1);
real[int] urate(nref-1);
real[int] prate(nref-1);
real[int] Gurate(nref-1);
//-----------------------------------------------------------------------------------
//                   Global data
//-----------------------------------------------------------------------------------
real nu = 1.;
real pp = 3.;
real qq = pp/(pp-1);
real Fc = 10.;
real D  = 1.;

func p  =  cos(pi*x)*sin((pi/2.)*y);
func px = -pi*sin(pi*x)*sin((pi/2.)*y);
func py =  (pi/2.)*cos(pi*x)*cos((pi/2.)*y);

func u1   =  sin(pi*x)*cos(pi*y);
func u2   = -cos(pi*x)*sin(pi*y);
func u1x  =  pi*cos(pi*x)*cos(pi*y);
func u1y  = -pi*sin(pi*x)*sin(pi*y);
func u2x  =  pi*sin(pi*x)*sin(pi*y);
func u2y  = -u1x;
func u1xx = -(pi^2)*sin(pi*x)*cos(pi*y);
func u1yy = -(pi^2)*sin(pi*x)*cos(pi*y);
func u2xx =  (pi^2)*cos(pi*x)*sin(pi*y);
func u2yy =  (pi^2)*cos(pi*x)*sin(pi*y);

func sig1 = nu*u1x - p;
```

Código FreeFEM++: BF-mixto_2D-Picard.edp III

```
func sig2 = nu*u1y;
func sig3 = nu*u2x;
func sig4 = nu*u2y - p;

func Divsig1 = nu*(u1xx + u1yy) - px;
func Divsig2 = nu*(u2xx + u2yy) - py;

func fbnorm = sqrt(u1^2 + u2^2);
func f1 = (D + Fc*pow(fbnorm,pp-2) )*u1 - Divsig1;
func f2 = (D + Fc*pow(fbnorm,pp-2) )*u2 - Divsig2;

//----- Global macros
macro u [u1,u2] //
macro Gu [u1x,u1y,u2x,u2y] //
macro sig [sig1,sig2,sig3,sig4] //
macro Divsig [Divsig1,Divsig2] //
macro F [f1,f2] //

macro sigh [sigh1,sigh2,sigh3,sigh4] //
macro tauh [tauh1,tauh2,tauh3,tauh4] //

macro uh [uh1,uh2] //
macro vh [vh1,vh2] //
macro wh [wh1,wh2] //

macro norm [N.x,N.y] //

macro fb(vh) ( sqrt(vh[0]^2 + vh[1]^2) ) //
macro tr(tauh) (tauh[0] + tauh[3]) //
```

Código FreeFEM++: BF-mixto_2D-Picard.edp IV

```
macro dev(tauh) [0.5*(tauh[0] - tauh[3]),tauh[1],tauh[2],0.5*(tauh[3] - tauh[0])] //
macro Div(tauh) [dx(tauh[0]) + dy(tauh[1]),dx(tauh[2]) + dy(tauh[3])] //

//----- Post-processing formulae
macro ph(tauh) ( -0.5*(tauh[0] + tauh[3]) ) //
macro Guh(tauh) [1./(2.*nu)*(tauh[0]-tauh[3]),(1./nu)*tauh[1],(1./nu)*tauh[2],1./(2.*nu)*(tauh
      [3]-tauh[0])] //
//------------------------------------------------------------------------------------
//                   Defining the domain
//------------------------------------------------------------------------------------
for(int n = 0; n < nref; n++){

int size  = 2^(n + 2.);

int GammaD = 11;

border Gamma1(t=0,1){x=t; y=0; label = GammaD;};
border Gamma2(t=0,1){x=1; y=t; label = GammaD;};
border Gamma3(t=1,0){x=t; y=1; label = GammaD;};
border Gamma4(t=1,0){x=0; y=t; label = GammaD;};

mesh Th = buildmesh(Gamma1(size) + Gamma2(size) + Gamma3(size) + Gamma4(size));
//------------------------------------------------------------------------------------
//            Finite element spaces
//------------------------------------------------------------------------------------
fespace Hhsig(Th,[RT0,RT0]);
fespace Hhu(Th,[P0,P0]);

fespace Vh(Th,P1);
```

Código FreeFEM++: BF-mixto_2D-Picard.edp V

```
//-----------------------------------------------------------------------------------
//                   Defining the bilinear forms
//-----------------------------------------------------------------------------------
Hhsig sigh;
Hhu uh, wh;
//----- bilinear forms
varf a11(sigh,tauh) = int2d(Th)( (1./nu)*(dev(sigh)'*dev(tauh)) );
varf a12(uh,tauh)   = int2d(Th)( uh'*Div(tauh) );
varf a22(uh,vh)     = int2d(Th)( -(D + Fc*pow(fb(wh),pp-2))*(uh'*vh) );

varf lm(sigh,tauh)  = int2d(Th)( tr(tauh) );

//----- RHS
varf rhs1(sigh,tauh) = int1d(Th,GammaD)( u'*([[tauh[0],tauh[1]],[tauh[2],tauh[3]]]*norm) );
varf rhs2(uh,vh)     = int2d(Th)( -(F'*vh) );
//-----------------------------------------------------------------------------------
//                       Stiff matrix
//-----------------------------------------------------------------------------------
matrix A11 = a11(Hhsig,Hhsig);
matrix A12 = a12(Hhu,Hhsig);
real[int] LM = lm(0,Hhsig);

//----- RHS
real[int] RHS1 = rhs1(0,Hhsig);
real[int] RHS2 = rhs2(0,Hhu);
real[int] L = [RHS1,RHS2,0];
//-----------------------------------------------------------------------------------
//                      Picard iteration
//-----------------------------------------------------------------------------------
```

Código FreeFEM++: BF-mixto_2D-Picard.edp VI

```
wh = [0.,0.];

int itt = 0.;
tol = 10.;
real[int] solt(Hhsig.ndof+Hhu.ndof+1); solt = 0.;

while((tol > 1e-6) && (itt < 30)){
    itt = itt + 1.;

    matrix A22 = a22(Hhu,Hhu);
    matrix M;{
    M = [[  A11, A12, LM],
        [ A12', A22,  0],
        [  LM',   0,  0]];}

    set(M,solver = sparsesolver);
    real[int] sol = M^-1*L;

    wh1[] = sol(Hhsig.ndof:Hhsig.ndof + Hhu.ndof-1);

//----- computing tol
    real[int] diff = sol - solt;
    tol = sqrt(diff'*diff)/sqrt(sol'*sol);
    cout << " tolerance = " << tol << endl;

//----- updating data for the next step
  solt  = sol;
}
iterations[n] = itt;
```

Código FreeFEM++: BF-mixto_2D-Picard.edp VII

```

//----- Approximation of the solution
sigh1[] = solt(0:Hhsig.ndof-1);
uh1[]   = solt(Hhsig.ndof:Hhsig.ndof + Hhu.ndof-1);

//----- calculating the errors
sigerror[n] = sqrt(int2d(Th)( (sig - sigh)'*(sig - sigh) )) + pow(int2d(Th)( pow((Divsig - Div
      (sigh))'*(Divsig - Div(sigh)),qq/2.) ),1./qq);
uerror[n]   = pow(int2d(Th)( pow((u - uh)'*(u - uh),pp/2.) ),1./pp);

perror[n]   = sqrt(int2d(Th)( square(p - ph(sigh)) ));
Guerror[n]  = sqrt(int2d(Th)( (Gu - Guh(sigh))'*(Gu - Guh(sigh)) ));

//----- for the meshsize in Omega
Vh h = hTriangle;
H[n] = h[].max;

nbT[n] = Th.nt;
DOF[n] = Hhsig.ndof + Hhu.ndof;
//----- exporting to Praraview
savevtk("Data_Paraview_2D/BF_aprox"+n+".vtk",Th,[sigh1,sigh2,0],[sigh3,sigh4,0],[uh1,uh2,0],ph
      (sigh),(1./nu)*sqrt( (0.5*(sigh[0]-sigh[3]))^2 + sigh[1]^2 + sigh[2]^2 + (0.5*(sigh[3]-
      sigh[0]))^2 ),dataname="sig1h sig2h uh ph mGuh");
savevtk("Data_Paraview_2D/BF_exact"+n+".vtk",Th,[sig1,sig2,0],[sig3,sig4,0],[u1,u2,0],p,sqrt(
      (u1x)^2 + (u1y)^2 + (u2x)^2 + (u2y)^2 ),dataname="sig1 sig2 u p mGu");
}
//----------------------------------------------------------------------------------
//                    showing the tables
//----------------------------------------------------------------------------------
```

Código FreeFEM++: BF-mixto_2D-Picard.edp VIII

```
cout << " sigerror = " << sigerror <<endl;
for(int n = 1; n < nref; n++)
sigrate[n-1] = log(sigerror[n-1]/sigerror[n]) / log(H[n-1]/H[n]);
cout <<" convergence rate sig in Hdiv-q = "<< sigrate <<endl;

cout << " uerror = " << uerror <<endl;
for(int n = 1; n < nref; n++)
urate[n-1] = log(uerror[n-1]/uerror[n]) / log(H[n-1]/H[n]);
cout << " convergence rate u in Lp = " << urate <<endl;

cout << " perror = " << perror <<endl;
for(int n = 1; n < nref; n++)
prate[n-1] = log(perror[n-1]/perror[n]) / log(H[n-1]/H[n]);
cout << " convergence rate p in L2 = " << prate <<endl;

cout << " Guerror = " << Guerror <<endl;
for(int n = 1; n < nref; n++)
Gurate[n-1] = log(Guerror[n-1]/Guerror[n]) / log(H[n-1]/H[n]);
cout << " convergence rate velocity gradient in L2 = " << Gurate <<endl;

cout << " mesh size Omega = " << H <<endl;
cout << " DOF Omega = " << DOF <<endl;
cout << " Number of triangles = " << nbT <<endl;
cout << " Newton iterations = " << iterations <<endl;
```

Estrategia Iterativa de Newton

Dado $\mathbf{u}_h^0 \in \mathbf{H}_h^{\mathbf{u}}$, para $m \geq 1$, hallar $(\boldsymbol{\sigma}_h^m, \mathbf{u}_h^m, \lambda_h^m) \in \mathbb{H}_h^{\boldsymbol{\sigma}} \times \mathbf{H}_h^{\mathbf{u}} \times \mathrm{R}$ tal que:

$$\frac{1}{\nu}\int_\Omega (\boldsymbol{\sigma}_h^m)^{\mathrm{d}} : \boldsymbol{\tau}_h^{\mathrm{d}} + \int_\Omega \mathbf{u}_h^m \cdot \mathbf{div}(\boldsymbol{\tau}_h) + \lambda_h^m \int_\Omega \mathrm{tr}(\boldsymbol{\tau}_h) = \langle \boldsymbol{\tau}_h \mathbf{n}, \mathbf{u}_D \rangle_\Gamma \,,$$

$$\int_\Omega \mathbf{v}_h \cdot \mathbf{div}(\boldsymbol{\sigma}_h^m) - \mathtt{D}\int_\Omega \mathbf{u}_h^m \cdot \mathbf{v}_h \; - \; \mathtt{F}(\rho-2)\int_\Omega |\mathbf{u}_h^{m-1}|^{\rho-4}(\mathbf{u}_h^{m-1}\cdot\mathbf{u}_h^m)(\mathbf{u}_h^{m-1}\cdot\mathbf{v}_h)$$

$$-\,\mathtt{F}\int_\Omega |\mathbf{u}_h^{m-1}|^{\rho-2}\mathbf{u}_h^m \cdot \mathbf{v}_h = -\int_\Omega \mathbf{f}\cdot\mathbf{v}_h \; - \; \mathtt{F}(\rho-2)\int_\Omega |\mathbf{u}_h^{m-1}|^{\rho-2}\mathbf{u}_h^{m-1}\cdot\mathbf{v}_h \,,$$

$$\eta_h \int_\Omega \mathrm{tr}(\boldsymbol{\sigma}_h^m) = 0\,,$$

para todo $(\boldsymbol{\tau}_h, \mathbf{v}_h, \eta_h) \in \mathbb{H}_h^{\boldsymbol{\sigma}} \times \mathbf{H}_h^{\mathbf{u}} \times \mathrm{R}$.

Corramos el ejemplo anterior con nuestra estrategia iterativa de Newton

Código FreeFEM++: BF-mixto_2D-Newton.edp I

```
//
// This code solves a 2D mixed finite element method
// for the Brinkman-Forchheimer equations
//     sig = nu*\nabla u - p*I \qin \Omega, D*u + F*|u|**(rho-2)*u - div(sig) = f in \Omega,
//                     u = uD on \Gamma,   int_Omega tr(sig) = 0.
//
// Global information
load "iovtk";          // for saving data in paraview format
load "UMFPACK64";      // UMFPACK solver
load "Element_Mixte"; // for using RT1
//-----------------------------------------------------------------------------------
//                Initial parameters
//-----------------------------------------------------------------------------------
//----- Global parameters
int nref = 4;
real tol;

real[int] H(nref);
real[int] DOF(nref);
real[int] nbT(nref);
real[int] iterations(nref);

//----- errors
real[int] sigerror(nref);
real[int] uerror(nref);
real[int] perror(nref);
real[int] Guerror(nref);

//----- rate of convergence
```

Código FreeFEM++: BF-mixto_2D-Newton.edp II

```
real[int] sigrate(nref-1);
real[int] urate(nref-1);
real[int] prate(nref-1);
real[int] Gurate(nref-1);
//-----------------------------------------------------------------------------------
//                   Global data
//-----------------------------------------------------------------------------------
real nu = 1.;
real pp = 3.;
real qq = pp/(pp-1);
real Fc = 10.;
real D  = 1.;

func p  =  cos(pi*x)*sin((pi/2.)*y);
func px = -pi*sin(pi*x)*sin((pi/2.)*y);
func py =  (pi/2.)*cos(pi*x)*cos((pi/2.)*y);

func u1   =  sin(pi*x)*cos(pi*y);
func u2   = -cos(pi*x)*sin(pi*y);
func u1x  =  pi*cos(pi*x)*cos(pi*y);
func u1y  = -pi*sin(pi*x)*sin(pi*y);
func u2x  =  pi*sin(pi*x)*sin(pi*y);
func u2y  = -u1x;
func u1xx = -(pi^2)*sin(pi*x)*cos(pi*y);
func u1yy = -(pi^2)*sin(pi*x)*cos(pi*y);
func u2xx =  (pi^2)*cos(pi*x)*sin(pi*y);
func u2yy =  (pi^2)*cos(pi*x)*sin(pi*y);

func sig1 = nu*u1x - p;
```

Código FreeFEM++: BF-mixto_2D-Newton.edp III

```
func sig2 = nu*u1y;
func sig3 = nu*u2x;
func sig4 = nu*u2y - p;

func Divsig1 = nu*(u1xx + u1yy) - px;
func Divsig2 = nu*(u2xx + u2yy) - py;

func fbnorm = sqrt(u1^2 + u2^2);
func f1 = (D + Fc*pow(fbnorm,pp-2) )*u1 - Divsig1;
func f2 = (D + Fc*pow(fbnorm,pp-2) )*u2 - Divsig2;

//----- Global macros
macro u [u1,u2] //
macro Gu [u1x,u1y,u2x,u2y] //
macro sig [sig1,sig2,sig3,sig4] //
macro Divsig [Divsig1,Divsig2] //
macro F [f1,f2] //

macro sigh [sigh1,sigh2,sigh3,sigh4] //
macro tauh [tauh1,tauh2,tauh3,tauh4] //

macro uh [uh1,uh2] //
macro vh [vh1,vh2] //
macro wh [wh1,wh2] //

macro norm [N.x,N.y] //

macro fb(vh) ( sqrt(vh[0]^2 + vh[1]^2) ) //
macro tr(tauh) (tauh[0] + tauh[3]) //
```

Código FreeFEM++: BF-mixto_2D-Newton.edp IV

```
macro dev(tauh) [0.5*(tauh[0] - tauh[3]),tauh[1],tauh[2],0.5*(tauh[3] - tauh[0])] //
macro Div(tauh) [dx(tauh[0]) + dy(tauh[1]),dx(tauh[2]) + dy(tauh[3])] //

//----- Post-processing formulae
macro ph(tauh) ( -0.5*(tauh[0] + tauh[3]) ) //
macro Guh(tauh) [1./(2.*nu)*(tauh[0]-tauh[3]),(1./nu)*tauh[1],(1./nu)*tauh[2],1./(2.*nu)*(tauh
      [3]-tauh[0])] //
//-----------------------------------------------------------------------------------
//                    Defining the domain
//-----------------------------------------------------------------------------------
for(int n = 0; n < nref; n++){

int size  = 2^(n + 2.);

int GammaD = 11;

border Gamma1(t=0,1){x=t; y=0; label = GammaD;};
border Gamma2(t=0,1){x=1; y=t; label = GammaD;};
border Gamma3(t=1,0){x=t; y=1; label = GammaD;};
border Gamma4(t=1,0){x=0; y=t; label = GammaD;};

mesh Th = buildmesh(Gamma1(size) + Gamma2(size) + Gamma3(size) + Gamma4(size));
//-----------------------------------------------------------------------------------
//             Finite element spaces
//-----------------------------------------------------------------------------------
fespace Hhsig(Th,[RT0,RT0]);
fespace Hhu(Th,[P0,P0]);

fespace Vh(Th,P1);
```

Código FreeFEM++: BF-mixto_2D-Newton.edp V

```
//--------------------------------------------------------------------------------------
//                  Defining the bilinear forms
//--------------------------------------------------------------------------------------
Hhsig sigh;
Hhu uh, wh;
//----- bilinear forms
varf a11(sigh,tauh) = int2d(Th)( (1./nu)*(dev(sigh)'*dev(tauh)) );
varf a12(uh,tauh)   = int2d(Th)( uh'*Div(tauh) );
varf a22(uh,vh)     = int2d(Th)( -(D + Fc*pow(fb(wh),pp-2))*(uh'*vh) - (pp-2)*Fc*pow(fb(wh),pp
     -4)*(wh'*uh)*(wh'*vh) );

varf lm(sigh,tauh)  = int2d(Th)( tr(tauh) );

//----- RHS
varf rhs1(sigh,tauh) = int1d(Th,GammaD)( u'*([[tauh[0],tauh[1]],[tauh[2],tauh[3]]]*norm) );
varf rhs2(uh,vh)     = int2d(Th)( -(F'*vh) - (pp-2)*Fc*pow(fb(wh),pp-2)*(wh'*vh) );
//--------------------------------------------------------------------------------------
//                      Stiff matrix
//--------------------------------------------------------------------------------------
matrix A11 = a11(Hhsig,Hhsig);
matrix A12 = a12(Hhu,Hhsig);
real[int] LM = lm(0,Hhsig);

//----- RHS
real[int] RHS1 = rhs1(0,Hhsig);
//--------------------------------------------------------------------------------------
//                    Picard iteration
//--------------------------------------------------------------------------------------
wh = [0.,1e-6];
```

Código FreeFEM++: BF-mixto_2D-Newton.edp VI

```

int itt = 0.;
tol = 10.;
real[int] solt(Hhsig.ndof+Hhu.ndof+1); solt = 0.;

while((tol > 1e-6) && (itt < 30)){
    itt = itt + 1.;

    real[int] RHS2 = rhs2(0,Hhu);
    real[int] L = [RHS1,RHS2,0];

    matrix A22 = a22(Hhu,Hhu);
    matrix M;{
    M = [[  A11, A12, LM],
        [ A12', A22,  0],
        [  LM',   0,  0]];}

    set(M,solver = sparsesolver);
    real[int] sol = M^-1*L;

    wh1[] = sol(Hhsig.ndof:Hhsig.ndof + Hhu.ndof-1);

//----- computing tol
    real[int] diff = sol - solt;
    tol = sqrt(diff'*diff)/sqrt(sol'*sol);
    cout << " tolerance = " << tol << endl;

//----- updating data for the next step
  solt  = sol;
```

Código FreeFEM++: BF-mixto_2D-Newton.edp VII

```
}
iterations[n] = itt;

//----- Approximation of the solution
sigh1[] = solt(0:Hhsig.ndof-1);
uh1[]   = solt(Hhsig.ndof:Hhsig.ndof + Hhu.ndof-1);

//----- calculating the errors
sigerror[n] = sqrt(int2d(Th)( (sig - sigh)'*(sig - sigh) )) + pow(int2d(Th)( pow((Divsig - Div(sigh))'*(Divsig - Div(sigh)),qq/2.) ),1./qq);
uerror[n]   = pow(int2d(Th)( pow((u - uh)'*(u - uh),pp/2.) ),1./pp);

perror[n]   = sqrt(int2d(Th)( square(p - ph(sigh)) ));
Guerror[n]  = sqrt(int2d(Th)( (Gu - Guh(sigh))'*(Gu - Guh(sigh)) ));

//----- for the meshsize in Omega
Vh h = hTriangle;
H[n] = h[].max;

nbT[n] = Th.nt;
DOF[n] = Hhsig.ndof + Hhu.ndof;
//----- exporting to Praraview
savevtk("Data_Paraview_2D/BF_aprox"+n+".vtk",Th,[sigh1,sigh2,0],[sigh3,sigh4,0],[uh1,uh2,0],ph(sigh),(1./nu)*sqrt( (0.5*(sigh[0]-sigh[3]))^2 + sigh[1]^2 + sigh[2]^2 + (0.5*(sigh[3]-sigh[0]))^2 ),dataname="sig1h sig2h uh ph mGuh");
savevtk("Data_Paraview_2D/BF_exact"+n+".vtk",Th,[sig1,sig2,0],[sig3,sig4,0],[u1,u2,0],p,sqrt((u1x)^2 + (u1y)^2 + (u2x)^2 + (u2y)^2 ),dataname="sig1 sig2 u p mGu");
}
//--------------------------------------------------------------------
```

Código FreeFEM++: BF-mixto_2D-Newton.edp VIII

```
//                    showing the tables
//----------------------------------------------------------------------------------
cout << " sigerror = " << sigerror <<endl;
for(int n = 1; n < nref; n++)
sigrate[n-1] = log(sigerror[n-1]/sigerror[n]) / log(H[n-1]/H[n]);
cout <<" convergence rate sig in Hdiv-q = "<< sigrate <<endl;

cout << " uerror = " << uerror <<endl;
for(int n = 1; n < nref; n++)
urate[n-1] = log(uerror[n-1]/uerror[n]) / log(H[n-1]/H[n]);
cout << " convergence rate u in Lp = " << urate <<endl;

cout << " perror = " << perror <<endl;
for(int n = 1; n < nref; n++)
prate[n-1] = log(perror[n-1]/perror[n]) / log(H[n-1]/H[n]);
cout << " convergence rate p in L2 = " << prate <<endl;

cout << " Guerror = " << Guerror <<endl;
for(int n = 1; n < nref; n++)
Gurate[n-1] = log(Guerror[n-1]/Guerror[n]) / log(H[n-1]/H[n]);
cout << " convergence rate velocity gradient in L2 = " << Gurate <<endl;

cout << " mesh size Omega = " << H <<endl;
cout << " DOF Omega = " << DOF <<endl;
cout << " Number of triangles = " << nbT <<endl;
cout << " Newton iterations = " << iterations <<endl;
```

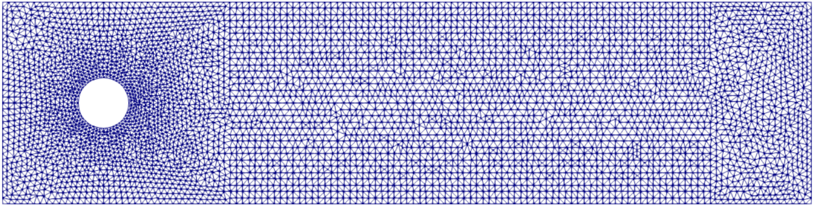
Fluido en un canal rectangular 2D con obstáculos

Parámetros

- $\Omega = (0, 1.6) \times (0, 0.4) \setminus \Omega_c$, where

$$\Omega_c = \left\{ (x, y) : \quad (x - 0.2)^2 + (y - 0.2)^2 < 0.05^2 \right\}$$

- $\nu = 1e - 3$, $\mathtt{D} = 1$, $\mathtt{F} = 10$, $\rho = 4$ y $\mathbf{f} = \mathbf{0}$



$$\mathbf{u} = \Big(-0.5\, y\, (y - 0.4), 0 \Big)^{\mathrm{t}} \text{ en } \Gamma_{\mathrm{left}} \cup \Gamma_{\mathrm{right}}\,, \quad \text{y} \quad \mathbf{u} = \mathbf{0} \text{ en } \Gamma_{\mathrm{top}} \cup \Gamma_{\mathrm{bottom}}$$

Código FreeFEM++: BF-mixto_2D-rectangle-holes.edp I

```
//
// This code solves a 2D mixed finite element method
// for the Brinkman-Forchheimer equations
//     sig = nu*\nabla u - p*I \qin \Omega, D*u + F*|u|**(rho-2)*u - div(sig) = f in \Omega,
//                     u = uD on \Gamma,   int_Omega tr(sig) = 0.
//
// Global information
load "iovtk";          // for saving data in paraview format
load "UMFPACK64";      // UMFPACK solver
load "Element_Mixte"; // for using RT1
//-----------------------------------------------------------------------------------
//                Initial parameters
//-----------------------------------------------------------------------------------
//----- Global parameters
real tol, H, DOF, nbT, iterations;

//-----------------------------------------------------------------------------------
//                   Global data
//-----------------------------------------------------------------------------------
real nu = 1e-3;
real pp = 4.;
real qq = pp/(pp-1);
real Fc = 10.;
real D  = 1.;

func u1   = -0.5*y*(y - 0.4);
func u2   = 0.;

func fbnorm = sqrt(u1^2 + u2^2);
```

Código FreeFEM++: BF-mixto_2D-rectangle-holes.edp II

```
func f1 = 0;
func f2 = 0;

//----- Global macros
macro u [u1,u2] //
macro F [f1,f2] //

macro sigh [sigh1,sigh2,sigh3,sigh4] //
macro tauh [tauh1,tauh2,tauh3,tauh4] //

macro uh [uh1,uh2] //
macro vh [vh1,vh2] //
macro wh [wh1,wh2] //

macro norm [N.x,N.y] //

macro fb(vh) ( sqrt(vh[0]^2 + vh[1]^2) ) //
macro tr(tauh) (tauh[0] + tauh[3]) //
macro dev(tauh) [0.5*(tauh[0] - tauh[3]),tauh[1],tauh[2],0.5*(tauh[3] - tauh[0])] //
macro Div(tauh) [dx(tauh[0]) + dy(tauh[1]),dx(tauh[2]) + dy(tauh[3])] //

//----- Post-processing formulae
macro ph(tauh) ( -0.5*(tauh[0] + tauh[3]) ) //
macro Guh(tauh) [1./(2.*nu)*(tauh[0]-tauh[3]),(1./nu)*tauh[1],(1./nu)*tauh[2],1./(2.*nu)*(tauh
      [3]-tauh[0])] //
//------------------------------------------------------------------------------------
//                    Defining the domain
//------------------------------------------------------------------------------------
int size  = 32;
```

Código FreeFEM++: BF-mixto_2D-rectangle-holes.edp III

```

border Gamma1(t=0,1.6){x=t; y=0; label = 1;};
border Gamma2(t=0,0.4){x=1.6; y=t; label = 2;};
border Gamma3(t=1.6,0){x=t; y=0.4; label = 3;};
border Gamma4(t=0.4,0){x=0; y=t; label = 4;};
border Gamma5(t=0,2*pi){x=0.05*cos(t)+0.2; y=0.05*sin(t)+0.2; label = 5;};

mesh Th = buildmesh(Gamma1(4*size) + Gamma2(size) + Gamma3(4*size) + Gamma4(size) + Gamma5(-1.5*size) );
//-----------------------------------------------------------------------------------
//             Finite element spaces
//-----------------------------------------------------------------------------------
fespace Hhsig(Th,[RT1,RT1]);
fespace Hhu(Th,[P1dc,P1dc]);

fespace Vh(Th,P1);
//-----------------------------------------------------------------------------------
//                   Defining the bilinear forms
//-----------------------------------------------------------------------------------
Hhsig sigh;
Hhu uh, wh;
//----- bilinear forms
varf a11(sigh,tauh) = int2d(Th)( (1./nu)*(dev(sigh)'*dev(tauh)) );
varf a12(uh,tauh)   = int2d(Th)( uh'*Div(tauh) );
varf a22(uh,vh)     = int2d(Th)( -(D + Fc*pow(fb(wh),pp-2))*(uh'*vh) - (pp-2)*Fc*pow(fb(wh),pp-4)*(wh'*uh)*(wh'*vh) );

varf lm(sigh,tauh)  = int2d(Th)( tr(tauh) );

```

Código FreeFEM++: BF-mixto_2D-rectangle-holes.edp IV

```
//----- RHS
varf rhs1(sigh,tauh) = int1d(Th,2,4)( u'*([[tauh[0],tauh[1]],[tauh[2],tauh[3]]]*norm) );
varf rhs2(uh,vh)     = int2d(Th)( -(F'*vh) - (pp-2)*Fc*pow(fb(wh),pp-2)*(wh'*vh) );
//--------------------------------------------------------------------------------------
//                      Stiff matrix
//--------------------------------------------------------------------------------------
matrix A11 = a11(Hhsig,Hhsig);
matrix A12 = a12(Hhu,Hhsig);
real[int] LM = lm(0,Hhsig);

//----- RHS
real[int] RHS1 = rhs1(0,Hhsig);
//--------------------------------------------------------------------------------------
//                    Picard iteration
//--------------------------------------------------------------------------------------
wh = [0.,1e-6];

int itt = 0.;
tol = 10.;
real[int] solt(Hhsig.ndof+Hhu.ndof+1); solt = 0.;

while((tol > 1e-6) && (itt < 30)){
    itt = itt + 1.;

    real[int] RHS2 = rhs2(0,Hhu);
    real[int] L = [RHS1,RHS2,0];

    matrix A22 = a22(Hhu,Hhu);
    matrix M;{
```

Código FreeFEM++: BF-mixto_2D-rectangle-holes.edp V

```
    M = [[  A11, A12, LM],
        [ A12', A22,  0],
        [  LM',   0,  0]];}

    set(M,solver = sparsesolver);
    real[int] sol = M^-1*L;

    wh1[] = sol(Hhsig.ndof:Hhsig.ndof + Hhu.ndof-1);

//----- computing tol
    real[int] diff = sol - solt;
    tol = sqrt(diff'*diff)/sqrt(sol'*sol);
    cout << " tolerance = " << tol << endl;

//----- updating data for the next step
  solt  = sol;
}
iterations = itt;

//----- Approximation of the solution
sigh1[] = solt(0:Hhsig.ndof-1);
uh1[]   = solt(Hhsig.ndof:Hhsig.ndof + Hhu.ndof-1);

//----- for the meshsize in Omega
Vh h = hTriangle;
H = h[].max;

nbT = Th.nt;
DOF = Hhsig.ndof + Hhu.ndof;
```

Código FreeFEM++: BF-mixto_2D-rectangle-holes.edp VI

```
//----- exporting to Praraview
savevtk("Data_Paraview_2D/BF_aprox.vtk",Th,[sigh1,sigh2,0],[sigh3,sigh4,0],[uh1,uh2,0],ph(sigh
      ),(1./nu)*sqrt( (0.5*(sigh[0]-sigh[3]))^2 + sigh[1]^2 + sigh[2]^2 + (0.5*(sigh[3]-sigh
      [0]))^2 ),dataname="sig1h sig2h uh ph mGuh");
//------------------------------------------------------------------------------------
//                    showing the tables
//------------------------------------------------------------------------------------
cout << " mesh size Omega = " << H <<endl;
cout << " DOF Omega = " << DOF <<endl;
cout << " Number of triangles = " << nbT <<endl;
cout << " Newton iterations = " << iterations <<endl;
```

Funding:

Agencia Nacional de Investigación y Desarrollo
Ministerio de Ciencia, Tecnología, Conocimiento e Innovación
Gobierno de Chile

Fondecyt 11220393

Gracias por su atención!!!

correo: scaucao@ucsc.cl

web: scaucao.github.io