UCSC | FACULTAD DE INGENIERÍA | DEPARTAMENTO DE MATEMÁTICA Y FÍSICA APLICADAS

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

Ingresar a:

www.menti.com

6988 3769

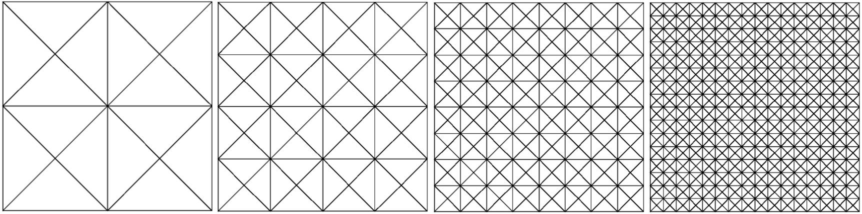
¿Qué entiendes por Método de Elementos Finitos ?

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: $\left\{ \begin{aligned} -\mathtt{D}^{-1}\Delta\, p = f \quad &\text{en} \quad \Omega\,, \\ p = 0 \quad &\text{en} \quad \partial\Omega\,. \end{aligned} \right.$ (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:

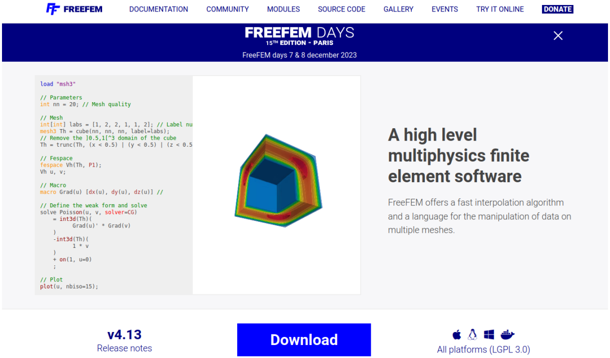


Una respuesta a grandes rasgos ...

- 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:

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

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

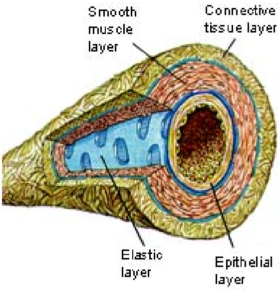
1. Los programas creados con FreeFem++ pueden usarse para resolver problemas en multifísica en 2D y 3D.
2. FreeFem++ es escrito en C++. Puede ser considerado como un lenguaje de programación en sí mismo.
3. Es multiplataforma, se puede instalar en sistemas operativos: MacOS, Windows, y GNU-Linux.
4. 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

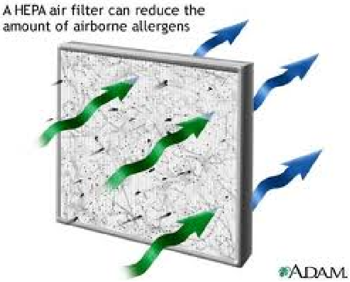
Manos al código ... iniciemos con el cursillo!!!

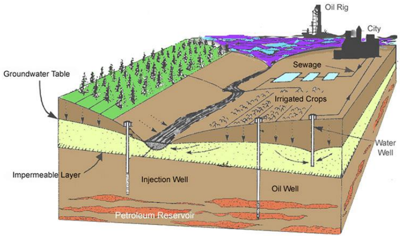
Contenidos del Cursillo

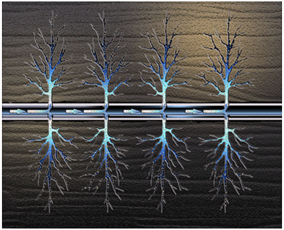
Motivación

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

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

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

$h = \max\limits_{T \in \mathcal{T}_h} h_T$ 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:

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

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

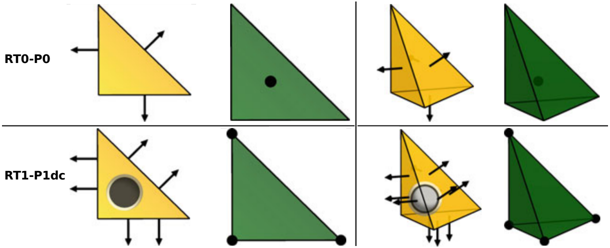
Formulación variacional discreta

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

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

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

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



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

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

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
```

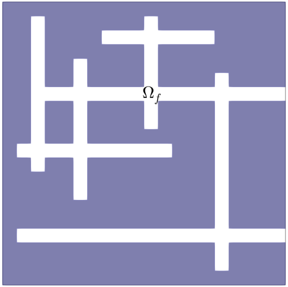
Código FreeFEM++: Darcy-mixto_2D.edp V

```
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

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



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

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

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