Soluciones Numéricas para Dinámica de Fluidos con Free*FEM*++

XXXIII CONGRESO MATEMÁTICA CAPRICORNIO

UNIVERSIDAD DE ANTOFAGASTA - CHILE

Pitágoras Pinheiro de Carvalho
pitagorascarvalho@gmail.com

Coordenação de Matemática
Universidade Estadual do Piauí - UESPI

06 / 08 / 2025

(Premero día)

XXXIII COMCA

Índice

Principales Lenguajes de Programación

- **Python:** Conocido por su sintaxis clara y su amplia biblioteca, Python es popular en áreas como ciencia de datos, inteligencia artificial, desarrollo web y aprendizaje automático.
- **Java:** Un lenguaje robusto y versátil, ampliamente utilizado en aplicaciones empresariales, sistemas distribuidos y desarrollo de aplicaciones Android.
- **JavaScript:** El lenguaje estándar para el desarrollo web, responsable de la interactividad y la dinámica de las páginas, además de ser utilizado en frameworks como Node.js para el desarrollo backend.
- **C++:** Un lenguaje de alto rendimiento, utilizado en videojuegos, sistemas operativos y aplicaciones científicas que requieren velocidad.

Índice

FreeFEM++

Lenguaje orientado a formulación variacional para resolver ecuaciones diferenciales parciales mediante el método de elementos finitos. Permite simulaciones 1D/2D o 3D con flexibilidad y precisión.

- Define dominios de forma paramétrica
- Soporta diversos tipos de mallas y elementos finitos
- Permite visualizaciones y exportaciones interactivas
- Sintaxis similar a la notación matemática

Tabla Comparativa

Software	Método	Licencia	Flexibilidad	Rendimiento	Pre/Post	Uso Ideal
FreeFEM++	MEF	Open-source	5★	4★	Bueno	Investigación, MEF avanzado
MATLAB	MEF/FDM	Comercial	1★	2★	Excelente	Prototipado rápido
OpenFOAM	FVM	Open-source	4★	4★	Externo	CFD industrial
ANSYS Fluent	FVM	Comercial	4★	5★	Avanzado	Industria
COMSOL	MEF	Comercial	3★	3★	Excelente	Multifísica / Insdustria
FEniCS	MEF	Open-source	4★	3★	Externo	Educación/ Investigación

FreeFEM++ se destaca como una excelent opción para uso personal en investigación científica de Elementos Finitos (MEF) debido a su excepcional flexibilidad, acceso total al código y robusto soporte para la implementación de métodos MEF complejos y avanzados.

Recomendación final

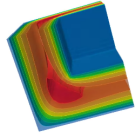
Use FreeFEM++ cuando necesite **control total** sobre las formulaciones numéricas.

FreeFEM++

freefem.org

FREEFEM DOCUMENTATION COMMUNITY MODULES SOURCE CODE GALLERY EVENTS TRY IT ONLINE DONATE

```
load "msh3"

// Parameters
int nn = 20; // Mesh quality

// Mesh
int[int] labs = [1, 2, 2, 1, 1, 2]; // Label numbers
mesh3 Th = cube(nn, nn, nn, label=labs);
// Remove the ]0.5,1[^3 domain of the cube
Th = trunc(Th, (x < 0.5) | (y < 0.5) | (z < 0.5), la

// Fespace
fespace Vh(Th, P1);
Vh u, v;

// Macro
macro Grad(u) [dx(u), dy(u), dz(u)] //

// Define the weak form and solve
solve Poisson(u, v, solver=CG)
    = int3d(Th)(
          Grad(u)' * Grad(v)
    )
    -int3d(Th)(
          1 * v
    )
    + on(1, u=0)
    ;

// Plot
plot(u, nbiso=15);
```



A high level multiphysics finite element software

FreeFEM offers a fast interpolation algorithm and a language for the manipulation of data on multiple meshes.

Examples of Associated book:

- PDE-constrained optimization within FreeFEM
- Mathematics and Finite Element Discretizations of Incompressible Navier-Stokes Flows

v4.15
Release notes

Download

All platforms (LGPL 3.0)

FreeFEM++-cs

FreeFEM-cs is an integrated environment for FreeFEM (a C++-like computer language dedicated to the finite element method, developed at the Jacques-Louis Lions Laboratory, Sorbonne Université). FreeFEM-cs provides an intuitive graphical interface to FreeFEM users.

FreeFEM-cs adds the following extra features to FreeFEM :

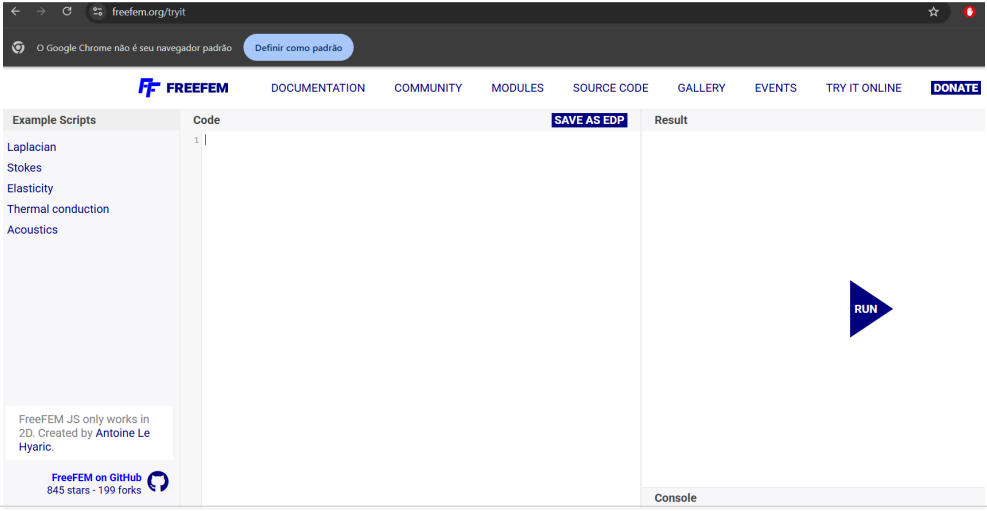
- Integrated interface aimed at making users comfortable
- Color-coded editor where undefined variables are highlighted in red
- FreeFEM compilation errors are linked back to the EDP source code
- Integrated graphics area for 2d and 3d

FreeFEM-cs follows a client/server design : the client deals with user interaction, the server runs FreeFEM (hence its name : "cs" as in "Client and Server"). More details can be found in the Introduction to FreeFem++-cs.

The FreeFEM-cs licence is LGPL.

Bibliographic citation : Antoine Le Hyaric. Introduction to FreeFem++-cs. 2015. <hal-01169630> (bibtex)

FreeFEM++ (Online)



FreeFEM++ (Javascript)

FreeFEM Javascript Example

Antoine Le Hyaric [1]

FreeFEM-cs FreeFEM and Javascript FreeFEM

Here is an example of how to include a runnable FreeFEM script into an HTML page. For more information about how this is done, please refer to the FreeFEM-js project. This is the basis for the Try It Online website.

- The `<head>` section of the HTML document should contain

```
<script src="pub/jquery-1.11.2.min.js"></script>
<script src="pub/FileSaver.js/FileSaver.min.js"></script>
<script src="bc/freefem.js" type="text/javascript"></script>
<script src="ffapi.js" type="text/javascript"></script>
```

- All script parts should be contained in HTML elements of `class="ffjs"`, but they don't need to be contiguous. The program listing can be interrupted anywhere to add more explanations as HTML, Latex (i.e. MathJax), pictures, etc. Some overly complex script details can be hidden from view in a hidden HTML element (if they keep the `ffjs` class). Some lines of the FreeFEM may be modifiable (eg the following program text area can be widened to enter any FreeFEM script).

```
<textarea class="ffjs">
  mesh Th=square(10,10);
  plot(Th);
</textarea>
```

Visualizadores:

- **Python**
- **MATLAB**
- **GMSH**
- **Paraview**
- **Otros...**

Circular Parameterized Mesh


```
// Create a circular mesh

border C00(t=0,2*pi){x=5*cos(t); y=5*sin(t); label=1;} // DOMAIN

plot(C00(100), cmm="Boundary", wait=true); // Plot only the domain
```

Note: In FreeFEM, the domain Ω is assumed to lie to the left of its boundary.

- $\partial\Omega$ is analytically described through a parametric equation for $x(t)$ and $y(t)$.

Circular Parameterized Mesh


```
// Create a circular mesh

border C00(t=0,2*pi){x=5*cos(t); y=5*sin(t); label=1;} // DOMAIN

plot(C00(100), cmm="Boundary", wait=true); // Plot only the domain
```

Note: In FreeFEM, the domain Ω is assumed to lie to the left of its boundary.

- $\partial\Omega$ is analytically described through a parametric equation for $x(t)$ and $y(t)$.
- When defining a boundary curve, each curve must be completely specified, and intersections between curves are only allowed at endpoints (initial or final points).

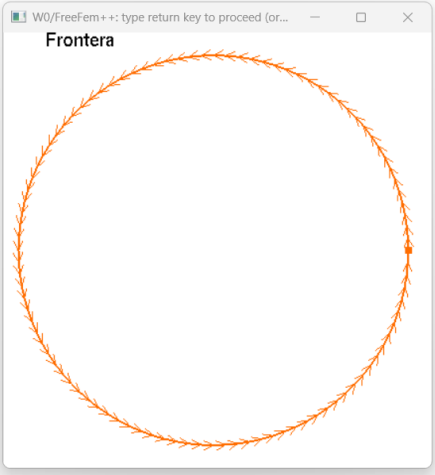
Circular Parameterized Mesh


```
// Create a circular mesh

border C00(t=0,2*pi){x=5*cos(t); y=5*sin(t); label=1;} // DOMAIN

plot(C00(100), cmm="Boundary", wait=true); // Plot only the domain
```

Note: In FreeFEM, the domain Ω is assumed to lie to the left of its boundary.

- $\partial\Omega$ is analytically described through a parametric equation for $x(t)$ and $y(t)$.
- When defining a boundary curve, each curve must be completely specified, and intersections between curves are only allowed at endpoints (initial or final points).
- The keyword `label` can be added to define a group of boundaries, which is useful for applying boundary conditions.

Plots



Circular Mesh Generation


```
load "medit" // Post-visualizer MEDIT
border C00(t=0,2*pi){x=5*cos(t); y=5*sin(t); label=1;} // DOMAIN
plot(C00(100), wait=true); // Plot only the domain
mesh Th = buildmesh(C00(100)); // Construct the mesh with 100 points on the boundary
plot(Th, cmm="Mesh FREEFEM", wait=true); // Plot the mesh
medit("Mesh MEDIT", Th); // Visualize the mesh using MEDIT
```

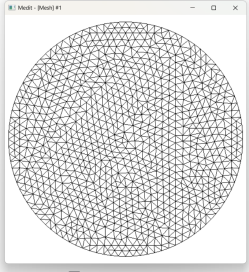


Figura: **Medit**

Circular Mesh Generation


```
load "medit" // Post-visualizer MEDIT
border C00(t=0,2*pi){x=5*cos(t); y=5*sin(t); label=1;} // DOMAIN
plot(C00(100), wait=true); // Plot only the domain
mesh Th = buildmesh(C00(100)); // Construct the mesh with 100 points on the boundary
plot(Th, cmm="Mesh FREEFEM", wait=true); // Plot the mesh
medit("Mesh MEDIT", Th); // Visualize the mesh using MEDIT
```

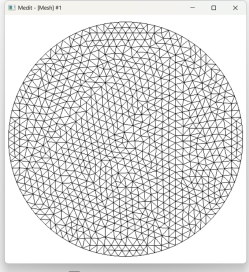


Figura: **Medit**

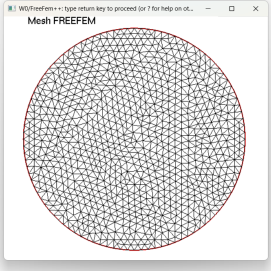
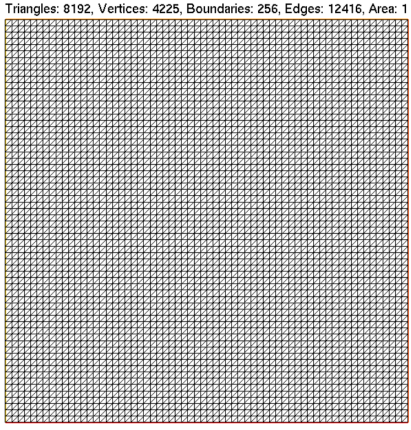


Figura: **FreeFEM++**

Mesh Statistics in FreeFEM++


```
// Create a rectangular mesh
int n = 50; // Number of vertices
real Lx = 1.0, Ly = 1.0; // Dimensions

mesh Th = square(n, n, [x*Lx, y*Ly]); // Domain
int T = Th.nt; // Number of triangles
int V = Th.nv; // Number of vertices
int B = Th.nbe; // Boundary edges
int E = (3*T + B)/2; // Total edges
real area = int2d(Th)(1.0); // Area

string legend = "Triangles: " + T +
  ", Vertices: " + V + ", Boundaries: " + B +
  ", Edges: " + E + ", Area: " + area;

plot(Th, cmm=legend, wait=true); // Mesh with data
```



Mesh generated by FreeFEM++

Dominio cuadrado


```
// Create a square mesh
int n = 2^6; // Number of vertices
real Lx = 1.0, Ly = 1.0; // Dimensions

mesh Th = square(n, n, [x*Lx, y*Ly]); // Mesh
     Domain
int T = Th.nt; // Number of triangles
int V = Th.nv; // Number of vertices
int B = Th.nbe; // Boundary edges
int E = (3*T + B)/2; // Total edges
real area = int2d(Th)(1.0); // Area

string legend = "Triangles: " + T +
  ", Vertices: " + V + ", Boundaries: " + B +
  ", Edges: " + E + ", Area: " + area;

plot(Th, cmm=legend, wait=true); // Plot Mesh
```

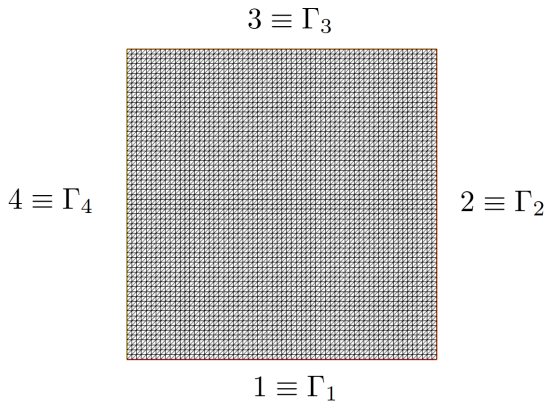


Figura: **Nb** vertices =4225 , **Nb** triangles = 8192 , **Nb** boundary edges 256, **Area** = 1.

Índice

XXXIII COMCA
Congreso Matemática Capricornio
Universidad de Antofagasta

Functions


```
load "medit" // Pós-Visualizador MEDIT

int n = 100; // Número de Vértices (ou nodes en la frontera)

border C00(t=0,2*pi){ x=cos(t); y=sin(t); } // DOMINIO CIRCULAR

mesh Th = buildmesh( C00(n) ); // Malla con Dominio

plot(Th, wait=true); // Plota la malla

func ff= sin(pi*x)*sin(pi*y); // Function

medit("Function", Th, ff, order=1); // Visualización de la function con MEDIT
```

Plots

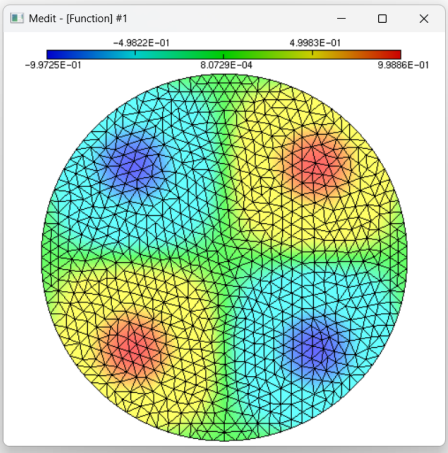


Figura: $\mathbf{m} + \mathbf{b}$

Plots

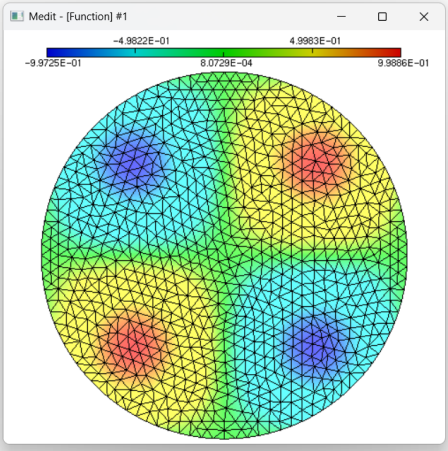


Figura: **m** + **b**

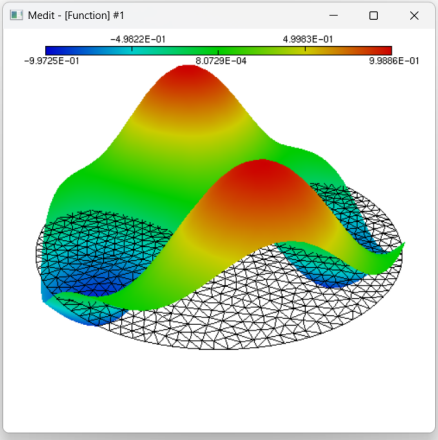


Figura: **k** $\mapsto$ **shift** + **k**

Índice

Índice

Formulación clásica

Dada una función $f \in L^2(\Omega)$, se busca encontrar $u \in V$, tal que:

$$\begin{cases} -\Delta u = f & \text{en} \quad \Omega, \\ u = 0 & \text{sobre} \quad \partial\Omega. \end{cases} \tag{1}$$

Notación

- $u := u(x, y)$ es la solución, $f := f(x, y)$ una fuente externa dada.
- $V := H_0^1(\Omega)$ espacio de Sobolev de funciones que se anulan en la frontera $\partial\Omega$.

Hipótesis sobre el dominio

- $\Omega \subset \mathbb{R}^2$ es un dominio abierto, acotado y de frontera regular.
- $\partial\Omega$ representa la frontera de Ω, asumida suave.

Formulación Débil

La **formulación débil** consiste en encontrar $u \in V$ tal que:

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V, \quad v = 0 \text{ en } \partial\Omega.$$

Formulación Débil

La **formulación débil** consiste en encontrar $u \in V$ tal que:

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V, \quad v = 0 \text{ en } \partial\Omega.$$

Etapas para convertir a un problema numérico

- V es de dimensión infinita, lo que imposibilita una solución numérica directa.

Formulación Débil

La **formulación débil** consiste en encontrar $u \in V$ tal que:

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V, \quad v = 0 \text{ en } \partial\Omega.$$

Etapas para convertir a un problema numérico

- V es de dimensión infinita, lo que imposibilita una solución numérica directa.
- Se realiza una reducción a un subespacio de dimensión finita $V_h \subset V$.

Formulación Débil

La **formulación débil** consiste en encontrar $u \in V$ tal que:

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V, \quad v = 0 \text{ en } \partial\Omega.$$

Etapas para convertir a un problema numérico

- V es de dimensión infinita, lo que imposibilita una solución numérica directa.
- Se realiza una reducción a un subespacio de dimensión finita $V_h \subset V$.
- Se discretiza el problema y se transforma en un sistema finito.

Formulación Débil

La **formulación débil** consiste en encontrar $u \in V$ tal que:

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V, \quad v = 0 \text{ en } \partial\Omega.$$

Etapas para convertir a un problema numérico

- V es de dimensión infinita, lo que imposibilita una solución numérica directa.
- Se realiza una reducción a un subespacio de dimensión finita $V_h \subset V$.
- Se discretiza el problema y se transforma en un sistema finito.
- Se emplea el **M**étodo de los **E**lementos **F**initos para obtener una solución numerica computable.

Etapa 1: Metodo de Galerkin

En el método de Galerkin, se escoge un espacio finito dimensional (basis de elementos finitos)

$$\text{span}[\varphi_1, \varphi_2, \ldots, \varphi_M] = V_h \subset V, \quad \boxed{(dim(V_h) = M)}$$

Etapa 1: Metodo de Galerkin

En el método de Galerkin, se escoge un espacio finito dimensional (basis de elementos finitos)

$$\text{span}[\varphi_1, \varphi_2, \ldots, \varphi_M] = V_h \subset V, \qquad \boxed{(dim(V_h) = M)}$$

y se busca una solución aproximada $\Big(\underbrace{u_h}_{\in V_h} \approx \underbrace{u}_{\in V} \Big)$ *con*

$$u_h = \sum_{j=1}^{M} u_j \varphi_j, \qquad u_h(P_j) \equiv 0, \quad \forall\, P_j \in \partial\Omega.$$

de modo que $u_h \in V_h$ que satisfacen

$$\int_\Omega \nabla u_h \cdot \nabla v_h \, dx = \int_\Omega f v_h \, dx, \qquad \forall v_h \in V_h \,.$$

Etapa 2: Dominio Ω y Malla Ω_h

Sea $\Omega \subset \mathbb{R}^2$ (o $\mathbb{R}^3$) un dominio abierto y acotado.

Una **malla triangular** $\mathcal{T}_h$ de Ω es un conjunto finito de triángulos K tal que:

$$\Omega_h := \bigcup_{K \in \mathcal{T}_h} K\,,$$

con:

- Cada K es un triángulo no degenerado;
- Para cualesquiera $K_i, K_j \in \mathcal{T}_h$, con $i \neq j$, se cumple:

$$K_i \cap K_j \in \{\emptyset,\ \text{vértice común},\ \text{arista común}\};$$

- $\Omega_h \equiv \Omega$ (Dominios poligonales) o $\Omega_h \approx \Omega$ (Dominios no poligonales).

Hipótesis sobre la Malla

Para garantizar una buena calidad numérica, la malla $\mathcal{T}_h$ debe satisfacer:

Regularidad (casi-uniformidad)

Existe $\sigma > 0$ tal que, para todo $K \in \mathcal{T}_h$,

$$\frac{\text{radio del mayor círculo inscrito en } K}{\text{diámetro de } K} \geq \sigma.$$

Parámetro de malla

$$h := \max_{K \in \mathcal{T}_h} \text{diam}(K).$$

Las aproximaciones $V_h(\Omega_h,\, P_\ell)$

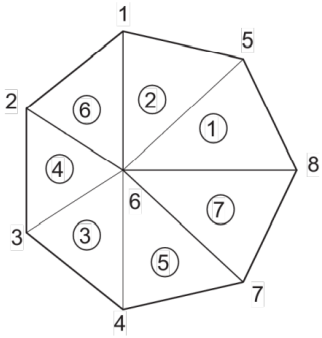
Los espacios $V_h(\Omega_h,\, P_\ell)$ dependen de la malla:

$$V_h(\Omega_h,\, P_\ell) = \left\{ w(x,y) \;\middle|\; w(x,y) = \sum_{k=1}^{M} w_k\, \phi_k(x,y),\; w_k \textbf{ son números reales} \right\} \qquad (2)$$

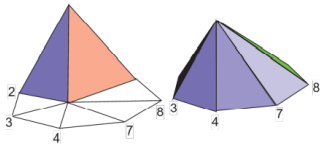
- M es el número de grados de libertad (i.e. la dimensión del espacio V_h);
- Los coeficientes w_k se denominan *grados de libertad* de w;
- ϕ_k son las funciones base (o funciones de forma) en el **MEF**;
- El objetivo es encontrar para $u,\, v \in V$ aproximaciones $u_h,\, v_h \in V_h$, es decir:

$$u(x,y) \approx u_h(x,y) = \sum_{k=1}^{M} u_k\, \phi_k(x,y) \quad .$$

Definindo ϕ_k



(a) mesh Th

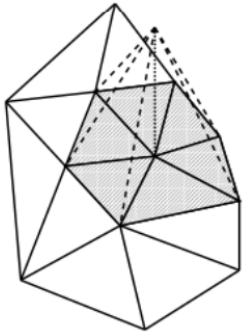


(b) Graph of ϕ_1 (left) and ϕ_6 (right)

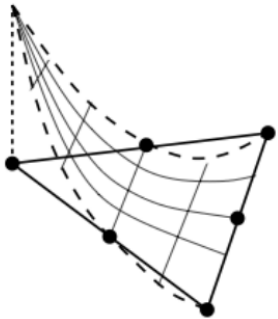
La forma más directa de definir ϕ_k es usando las coordenadas baricéntricas $\lambda_i(x, y)$, $i = 1, 2, 3$, de un punto $q = (x, y) \in T$, con: $\sum_i \lambda_i = 1$, $\sum_i \lambda_i \vec{q}_i = \vec{q}$, donde q_i son los vértices de T.

Así, $\phi_k|_T = \lambda_k$.

Base P_1 y P_2



(a) P1 basis function



(b) P2 basis function

Ejemplo: $f(x,y) = \sin(\pi x)\sin(\pi y)$

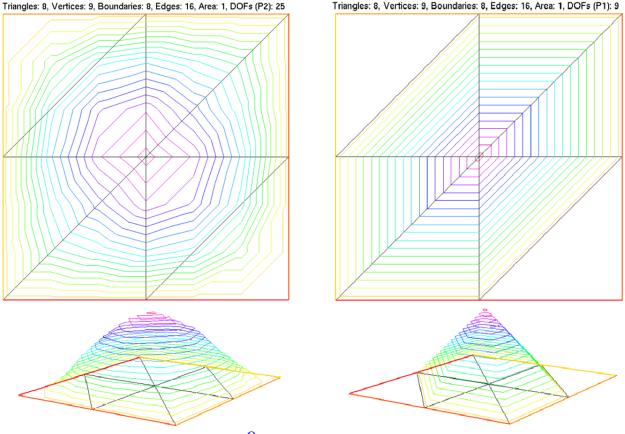


Figura: State $f \equiv \sum_{k=1}^{9} f_k\, \phi_k(x,y)$, con base $P2$ y $P1$.

Galerkin vs. MEF: Resumen de las Diferencias

Galerkin (Genérico)

- Base V_h genérica, del tipo:
  - Polinomios globales
  - Funciones trigonométricas
  - Wavelets
- Sin relación explícita entre V_h y Ω.

MEF (Galerkin + Geometría)

- Base $V_h \equiv V_h(\Omega_h, P_\ell)$
- Funciones con:
  - Soporte local
  - Continuidad controlada (C^0, C^1, etc...)
- Relación directa entre V_h y Ω_h.

Caso Poisson

Formulación Variacional Discreta:

$$\int_\Omega \nabla u_h \cdot \nabla v_h \, dx = \int_\Omega f v_h \, dx, \quad \forall v_h \in V_h \quad (\textbf{Galerkin})$$

$$\int_{\Omega_h} \nabla u_h \cdot \nabla v_h \, dx = \int_{\Omega_h} f v_h \, dx, \quad \forall v_h \in V_h(\Omega_h, P_\ell) \quad (\textbf{MEF}).$$

Solvers FREEFEM++

- `solver = LU, CG, Crout, Cholesky, GMRES, sparsesolver, UMFPACK ...`
- El solucionador predeterminado es **GMRES**.
- El modo de almacenamiento de la matriz del sistema lineal subyacente depende del tipo de solucionador elegido:
  - Para `LU`, la matriz es *sky-line* no simétrica.
  - Para `Crout`, la matriz es *sky-line* simétrica.
  - Para `Cholesky`, la matriz es *sky-line* simétrica definida positiva.
  - Para `CG`, la matriz es dispersa, simétrica y definida positiva.
  - Para `GMRES`, `sparsesolver` o `UMFPACK`, la matriz es simplemente dispersa.

Parte 1: Película de jabón - Resolución y visualización

Película de jabón con contorno fijo (Dirichlet homogéneo).
La fuerza actuante es la gravedad, para simplificar, asumimos que $f = -1$.

Código inicial: malla, espacio funcional y formulación del problema.

```
int nn = 50;
func f = -1;
func ue = (x^2 + y^2 - 1)/4; // solución exacta
border a(t=0, 2*pi){x = cos(t); y = sin(t); label = 1;} // Dominio parametrizado
mesh disk = buildmesh(a(nn)); // Malla
plot(disk); // Plot el Dominio Mallado

// Espacio de Soluciones aproximadas
fespace Vh(disk, P1); // Vh espacio de Elementos Finitos & P1 Polinomios Interpoladores
Vh u, v; // Elementos de Vh
```

Parte 1: Película de jabón


```
// Problem
problem laplace (u, v)
       = int2d(disk)( dx(u)*dx(v) + dy(u)*dy(v) ) // Bilinear form

       - int2d(disk)( f*v ) // Linear form

       + on(1, u=0) // Boundary condition
       ;

laplace; // Resolve el problema

plot(u, value = true, wait = true); // Visualizar la solución en cada atualizacion
```

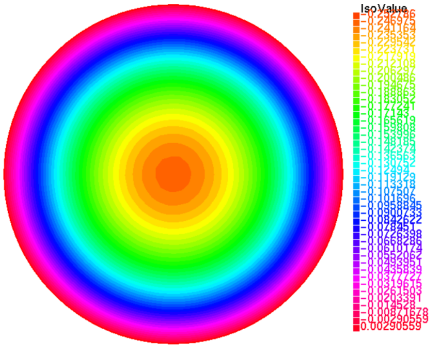
Resolución y cálculo del error


```
Vh err = u - ue; // Definir la variable que computa el error

plot(err, value = true, wait = true); // Plota el error

// Error con norma L2
cout << "error L2 = " << sqrt(int2d(disk)(err^2)) << endl;

// Error con norma H10
cout << "error H10 = " << sqrt(int2d(disk)((dx(u)-x/2)^2) + int2d(disk)((dy(u)-y/2)^2)) << endl;
```

$$\text{Error en la norma L2} : \|u - u_h\|_{L^2(\Omega)} = \left(\int_\Omega |u(x) - u_h(x)|^2 \, dx \right)^{1/2}.$$

$$\text{Error en la norma H1} : \|u - u_h\|_{H^1(\Omega)} = \left(\int_\Omega |\nabla u(x) - \nabla u_h(x)|^2 \, dx \right)^{1/2}.$$

Plots Poisson



Plots Poisson

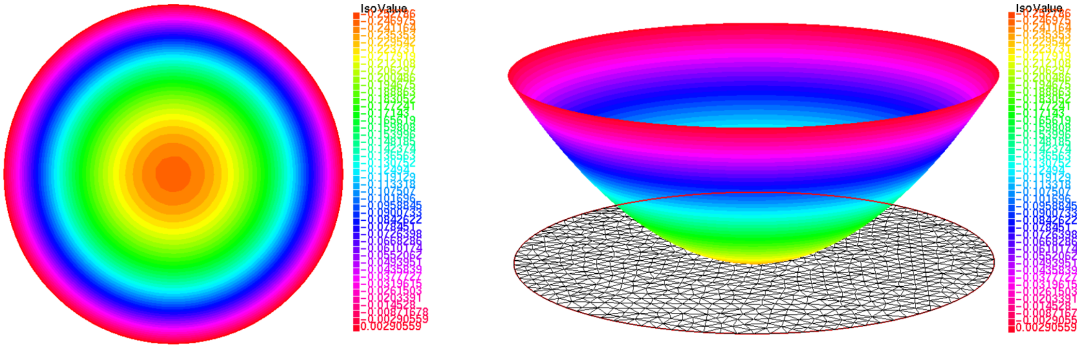


Figura: State u

error L2 = 9.19452e-05 , error H10 = 0.0183127

Armazenar os dados


```
load "iovtk" // para los formatos vtk y vtu
real[int] errL2(10), errH1(10);
for (int i = 1; i < 10; i++)
{
errL2[i-1] = sqrt(int2d(disk)( (u - ue)^2 )); // Calcula el error en la norma L2
errH1[i-1] = sqrt( int2d(disk)( (u - ue)^2 + (dx(u) - dx(ue))^2 + (dy(u) - dy(ue))^2 ) ); //
     Calcula el error en la norma H1

ofstream file1("ErrL2.txt");
   file1 << errL2(0:i) << endl; // Salva los errores L2 en .txt

ofstream file2("ErrH1.txt");
   file2 << errH1(0:i) << endl; // Salva los errores H1 en .txt
}
savevtk("disk.vtk", disk); // SALVA MALLA (PARAVIEW)
int[int] ffolder=[1,1];
savevtk("ustate.vtu", disk, u, dataname="Solucion", order=ffolder);// SALVA SOLUCION ( PARAVIEW)
```

Índice

Referências Bibliográficas

- F. Hecht. New development in Free*Fem*++. J. Numer. Math., 20(3-4):251–265, 2012.
- Roger Temam, Navier-Stokes equations: theory and numerical analysis. 1977.
- Roger Temam, Navier–Stokes equations: theory and numerical analysis. Vol. 343. American Mathematical Society, 2024.
- Roland Glowinski. Finite element methods for incompressible viscous flow. Handbook of numerical analysis, v. 9, p. 3-1176, 2003.
- Roberts, J. E., Thomas, J. M. (1991). Mixed and hybrid methods. Elsevier.

¡Gracias por su presencia en la parte 1!

¿Preguntas?