ELEMENTOS FINITOS PARA LA ECUACIÓN DE STOKES 2D EN LA FORMULACIÓN CORRIENTE - VORTICIDAD

LUIS FERNANDO MEJIA RODRIGUEZ

UNIVERSIDAD DEL VALLE
FACULTAD DE CIENCIAS
PROGRAMA ACADÉMICO DE MATEMÁTICAS
SANTIAGO DE CALI
2011

ELEMENTOS FINITOS PARA LA ECUACIÓN DE STOKES 2D EN LA FORMULACIÓN CORRIENTE - VORTICIDAD

LUIS FERNANDO MEJIA RODRIGUEZ

Trabajo de grado presentado como requisito para optar al título de Matemático

Director
JAIRO DUQUE, Dr. rer. nat.

UNIVERSIDAD DEL VALLE
FACULTAD DE CIENCIAS
PROGRAMA ACADÉMICO DE MATEMÁTICAS
SANTIAGO DE CALI
2011

UNIVERSIDAD DEL VALLE
FACULTAD DE CIENCIAS
PROGRAMA ACADÉMICO DE MATEMÁTICAS

LUIS FERNANDO MEJIA RODRIGUEZ,1982

ELEMENTOS FINITOS PARA LA ECUACIÓN DE STOKES 2D EN LA FORMULACIÓN CORRIENTE - VORTICIDAD

Palabras Claves: El problema de Stokes, Método de elementos finitos, Formulación débil en la función corriente y función vorticidad.

SANTIAGO DE CALI
2011

Nota de Aceptación

_____

_____

_____

Jairo Duque, Dr. rer. nat.
Director de Tesis

_____

Jurado

Santiago de Cali, Mayo de 2011

Agradecimientos

Agradezco a todas las personas que hicieron parte en mi formación académica y personal, sin ninguna duda no estaría aquí de no ser por ellos, agradezco muy especialmente a mi familia que durante este largo camino fue mi apoyo incondicional, a mi mamá y papá por su amor y educación, a mis queridos hermanos por no dejarme solo y ser una voz de ánimo, a mis amigos que durante toda mi formación me dierón la mano y palabras de fortaleza para enfrentar tantos problemas, a mi abuela que aunque ya no está con nosotros nunca saldrá de nuestros corazones, sus palabras y su sonrisa vivirán por siempre, a todos los profesores que participarón de mi formación por su gran esfuerzo. Un agradecimiento muy especial a el profesor Jairo Duque por su ayuda, por su paciencia, sus enseñanzas y consejos, me son de gran utilidad para la vida y por último a mi muy amada Ana María por haber estado todos estos años a mi lado por su amor y amistad, por sus palabras, por ser la niña de mis sueños.

Resumen

En este trabajo se implementa una solución numérica del problema de Stokes en las variables corriente - vorticidad mediante la técnica de los elementos finitos. La aproximación numérica de la solución se lleva a cabo usando elementos finitos bilineales en una región rectangular, considerando un flujo con pequeño número de Reynolds. Además, se plantea la formulacón débil del problema en el contexto del Teorema de Brezzi.

Índice general

Introducción

En este trabajo se implementará un método de elementos finitos para el problema de Stokes, que clásicamente modela el flujo de fluidos viscosos incompresibles; en este trabajo se consideran flujos con números de Reynolds pequeños.

Sea Ω un subconjunto abierto y acotado de $\mathbb{R}^2$ con frontera suave $\partial\Omega = \Gamma$ y sean $\mathbf{u} : \Omega \to \mathbb{R}^2$ el campo de velocidades y $p : \Omega \to \mathbb{R}$ la presión del fluido en cada punto del dominio; el problema de Stokes en forma vectorial está dado por:

$$\begin{aligned} -\frac{1}{Re}\Delta\mathbf{u} + \nabla p &= \mathbf{f} \text{ en } \Omega \\ \operatorname{div}(\mathbf{u}) &= 0 \text{ en } \Omega \\ \mathbf{u} &= \mathbf{u_0} \text{ sobre } \Gamma \end{aligned} \tag{1}$$

Aquí $\mathbf{f}$ es la fuerza que actúa sobre el fluido, por ejemplo $\mathbf{f}$ puede ser la gravedad y $\mathbf{u_0}$ es un campo de velocidades conocido, además $\frac{1}{Re}$ es una constante dinámica que es inversamente proporcional al número de Reynolds. El número de Reynolds Re está definido como $Re = \frac{l.k}{a}$, donde a es la viscosidad del fluido, l una longitud característica que depende de la geometría del problema y k la velocidad promedio del fluido [4]. Al ser este modelo un sistema de ecuaciones diferenciales parciales cuyas soluciones son difíciles de encontrar en general, se propone plantear el problema de Stokes en las variables corriente y vorticidad, donde la vorticidad en el contexto de la mecánica de fluidos es el rotacional del campo de velocidades; se define como $w = \nabla \times \mathbf{u} = \frac{\partial u_2}{\partial x} - \frac{\partial u_1}{\partial y}$ y la función corriente ψ se define de manera implícita como $\nabla\psi = (-u_2, u_1)$ donde ψ está relacionada con el hecho de que las líneas a lo largo de las cuales ψ es constante, son líneas de corriente es decir, líneas en el campo de flujo que son tangentes a las velocidades en todas partes [9]. De esta manera se cambia el problema de Stokes por:

$$\begin{aligned} w + \triangle\psi &= 0 \text{ en } \Omega \\ -\frac{1}{Re}\triangle w &= rot(\mathbf{f}) \text{ en } \Omega \\ \psi &= cte \text{ sobre } \Gamma. \end{aligned} \tag{2}$$

donde $rot(\mathbf{f}) = \frac{\partial f_2}{\partial x} - \frac{\partial f_1}{\partial y}$ y $\mathbf{f} = (f_1, f_2)$ y las condiciones de frontera para la función corriente ψ pueden ser $\psi = cte$ y $\frac{\partial \psi}{\partial n}$ conocidas en Γ, donde $\frac{\partial \psi}{\partial n}$ es la derivada normal a lo largo de Γ. Este cambio de variable es útil ya que no se tiene que tratar con la divergencia de $\mathbf{u}$, que es un inconveniente al intentar implementar un método de elementos finitos para el problema de Stokes; la restricción $div(\mathbf{u}) = 0$ introduce en la discretización del problema dificultades numéricas. En la diagonal del sistema matricial que se obtiene en la formulación con elementos finitos aparece un bloque de ceros que hace que métodos iterativos, como el de Jacobi o Gauss-Seidel, no se puedan usar. En la nueva formulación esta dificultad no aparece, pero a cambio se pierde la información explícita del campo de velocidades $\mathbf{u}$ y la presión p . Para visualizar el flujo es necesario obtener las líneas de corriente a partir de la función ψ. Finalmente, se implementará un método de elementos finitos para aproximar de manera numérica la solución del nuevo problema, en la formulación de las variables $w - \psi$.

Capítulo 1

El Modelo matemático de Stokes

Normalmente la descripción matemática del flujo de un fluido Newtoniano se realiza con las *ecuaciones de Navier-Stokes*. En un fluido newtoniano se considera que los esfuerzos son proporcionales a la velocidad de deformación y al coeficiente de viscosidad. Mediante el reemplazo de las ecuaciones constitutivas de un fluido newtoniano en las ecuaciones de conservación (masa y momento) se obtienen las *ecuaciones de Navier-Stokes*, las cuales están dadas por (para una completa descripción de la derivación de estas ecuaciones véase [6], [7], [2]):

$$\rho(\frac{\partial(\mathbf{u})}{\partial t} + (\mathbf{u}\cdot\nabla)(\mathbf{u})) = -\nabla p + \mathbf{f} + \tfrac{1}{\mathrm{Re}}\,\Delta\mathbf{u}, \tag{1.1}$$

las variables $\mathbf{u}$ y p en las *ecuaciones de Navier-Stokes* corresponden al vector de distribución de velocidades y a la presión del flujo respectivamente. Los parámetros ρ y μ corresponden a la densidad y a la viscosidad dinámica del fluido. Esta ecuación junto con la ecuación de continuidad:

$$\frac{\partial\rho}{\partial t} + \operatorname{div}(\rho\mathbf{u}) = 0, \tag{1.2}$$

describen el movimiento de un flujo viscoso incompresible. En el caso analizado en este trabajo se propone un flujo de fluido estacionario, así las ecuaciones (1.1) y (1.2) se reducen a

$$\rho(\mathbf{u}\cdot\nabla\mathbf{u}) = -\nabla p + \mathbf{f} + \tfrac{1}{\mathrm{Re}}\Delta\mathbf{u}, \tag{1.3}$$

$$\operatorname{div}(\rho\mathbf{u}) = 0, \tag{1.4}$$

el número de Reynolds Re está definido como $Re = \frac{l.k}{a}$, donde a es la viscosidad del fluido, l es una longitud característica que depende de la geometría del problema y k la velocidad promedio del fluido. Cuando el número de Reynolds es muy bajo, es decir, cuando la velocidad k o la longitud característica l del problema son muy bajos o la viscosidad a del fluido es alta, los términos de viscosidad $\frac{1}{\mathrm{Re}}\Delta\mathbf{u}$ son muy grandes y la velocidad es muy pequeña, así los términos no lineales $\mathbf{u}\cdot\nabla\mathbf{u}$ son despreciables frente a los términos viscosos. En el caso analizado en este trabajo, se consideran números de Re pequeños (menores que 10^3), así las ecuaciones (1.3) y (1.4) se reducen a

$$\frac{1}{\mathrm{Re}}\Delta\mathbf{u} - \nabla p + \mathbf{f} = 0 \tag{1.5}$$

$$\mathrm{div}\,(\rho\mathbf{u}) = 0, \tag{1.6}$$

donde $\mathbf{f}$ es el término proveniente de fuerzas externas que actúan sobre el fluido. Además suponiendo que ρ es constante se tiene que las ecuaciones (1.5) y (1.6) se reducen a

$$-\frac{1}{\mathrm{Re}}\Delta\mathbf{u} + \nabla p = \mathbf{f} \tag{1.7}$$

$$\mathrm{div}\,(\mathbf{u}) = 0. \tag{1.8}$$

Las ecuaciones (1.7) y (1.8) son conocidas como el problema de Stokes que clásicamente modelan el flujo de fluidos viscosos incompresibles.

Capítulo 2

El problema de Stokes en las variables corriente-vorticidad

El problema de Stokes que se describe en la ecuación (1) es un sistema de ecuaciones diferenciales parciales que en general no es fácil de resolver. En este trabajo se propone un cambio de las variables originales $\mathbf{u}$ y p por las variables corriente y vorticidad, para ello se introduce la vorticidad y la función corriente ψ definidas de manera implícita como

$$w = \frac{\partial u_2}{\partial x} - \frac{\partial u_1}{\partial y} \tag{2.1}$$

$$\nabla\psi = (-u_2, u_1). \tag{2.2}$$

Por la ecuación (2.2) tenemos que

$$\frac{\partial \psi}{\partial x} = -u_2 \tag{2.3}$$

$$\frac{\partial \psi}{\partial y} = u_1 \tag{2.4}$$

Al reemplazar (2.3) y (2.4) en (2.1) tenemos que:

$$w = \frac{\partial u_2}{\partial x} - \frac{\partial u_1}{\partial y} = -\frac{\partial^2 \psi}{\partial x^2} - \frac{\partial^2 \psi}{\partial y^2} = -(\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2}) = -\Delta\psi$$

por lo tanto

$$w + \Delta\psi = 0 \tag{2.5}$$

De otro lado al tomar el rotacional en ambos lados de la ecuación (1.7) se obtiene:

$$\begin{aligned}
rot(-\frac{1}{\mathrm{Re}}\Delta\mathbf{u}+\nabla p) &= rot(f)\\
rot\begin{pmatrix} -\frac{1}{\mathrm{Re}}\Delta u_1+\partial_x p\\ -\frac{1}{\mathrm{Re}}\Delta u_2+\partial_y p\end{pmatrix} &= rot(f)\\
\frac{\partial(-\frac{1}{\mathrm{Re}}\Delta u_2+\partial_y p)}{\partial x}-\frac{\partial(-\frac{1}{\mathrm{Re}}\Delta u_1+\partial_x p)}{\partial y} &= rot(f)\\
\frac{\partial(-\frac{1}{\mathrm{Re}}\Delta u_2)}{\partial x}+\frac{\partial(\partial_y p)}{\partial x}-\frac{\partial(-\frac{1}{\mathrm{Re}}\Delta u_1)}{\partial y}-\frac{\partial(\partial_x p)}{\partial y} &= rot(f)\\
\frac{\partial(-\frac{1}{\mathrm{Re}}\Delta u_2)}{\partial x}-\frac{\partial(-\frac{1}{\mathrm{Re}}\Delta u_1)}{\partial y} &= rot(f)\\
-\frac{1}{\mathrm{Re}}\partial_x\Delta u_2+\frac{1}{\mathrm{Re}}\partial_y\Delta u_1 &= rot(f)\\
-\frac{1}{\mathrm{Re}}\Delta(\partial_x u_2-\partial_y u_1) &= rot(f)
\end{aligned}$$

Usando (2.1) en esta última ecuación obtenemos:

$$-\frac{1}{\mathrm{Re}}\Delta w = rot(f). \tag{2.6}$$

De lo anterior, el problema de Stokes en las nuevas variables está dado por las ecuaciones (2.5), (2.6), es decir,

$$\begin{aligned}
w+\triangle\psi &= 0 \text{ en } \Omega\\
-\frac{1}{Re}\triangle w &= rot(f) \text{ en } \Omega.
\end{aligned} \tag{2.7}$$

Como el campo de velocidades $\mathbf{u}$ se conoce en la frontera de la región, entonces $\nabla\psi$ también, y por lo tanto $\partial_n\psi$. En algunos casos se conoce ψ en la frontera, por ejemplo, cuando esta parte se considera como una linea de corriente o cuando el flujo es tangente a la frontera de la región. También para determinar ψ de forma única en la condición $\nabla\psi=(-u_2,u_1)$ se puede fijar el valor de ψ en la frontera.

Formulación variacional

Sea $W^{1,2}(\Omega)$ el espacio de Sobolev dado por el conjunto de funciones cuadrado integrables con derivadas hasta orden uno cuadrado integrables, véase [5] y $W_0^{1,2}(\Omega)=$

$\{u \in W^{1,2}(\Omega) \colon u\,|_{\partial\Omega} = 0\}$ el conjunto de funciones en el espacio de Sobolev $W^{1,2}(\Omega)$ que además son cero en la frontera de Ω. Sean $v, q : \Omega \to \mathbb{R}$ funciones de prueba en $W_0^{1,2}(\Omega)$ y $W^{1,2}(\Omega)$, respectivamente. Multiplicando las ecuaciones del sistema (2.7) por v y q obtenemos:

$$\begin{aligned} w(x)q(x) + \triangle\psi(x)q(x) &= 0 \\ -\frac{1}{Re}\triangle w(x)v(x) &= rot(f(x))v(x) \end{aligned}$$

Luego se integran las ecuaciones en Ω respecto de x:

$$\begin{aligned} -\int_\Omega \triangle\psi(x)q(x)dx &= \int_\Omega w(x)q(x)dx \\ -\frac{1}{Re}\int_\Omega \triangle w(x)v(x)dx &= \int_\Omega rot(f(x))v(x)dx \end{aligned}$$

Aplicando integración por partes en los términos que contienen las derivadas de orden superior, se obtiene [3]:

$$\begin{aligned} -\int_{\partial\Omega} \nabla\psi(x)n(x)q(x)d\sigma + \int_\Omega \nabla\psi(x)\nabla q(x)dx &= \int_\Omega w(x)q(x)dx \\ -\frac{1}{Re}\int_{\partial\Omega} \nabla w(x)n(x)v(x)d\sigma + \frac{1}{Re}\int_\Omega \nabla w(x)\nabla v(x)dx &= \int_\Omega rot(f(x))v(x)dx \end{aligned}$$

La formulación variacional para el problema (2.7) consiste en encontrar $\psi, w \in W^{1,2}(\Omega)$ tales que:

$$\begin{aligned} \int_\Omega \nabla\psi(x)\nabla q(x)dx &= \int_\Omega w(x)q(x)dx + \int_{\partial\Omega} \nabla\psi(x)n(x)q(x)d\sigma, \quad \forall q \in W^{1,2}(\Omega) \\ \frac{1}{Re}\int_\Omega \nabla w(x)\nabla v(x)dx &= \int_\Omega rot(f(x))v(x)dx, \quad \forall v \in W_0^{1,2}(\Omega) \end{aligned}$$

Capítulo 3

Existencia y Unicidad de la solución

A continuación se formula el problema variacional en el contexto de los espacios de Hilbert. Para demostrar la existencia de las soluciones débiles de este problema, se hace uso del teorema de Brezzi que se cita a continuación:

Teorema 1 (Formulación mixta)**.** *Sean Z y M dos espacios de Hilbert. Sean además $a(w,\varphi) : M \times M \to \mathbb{R}$ y $b(\xi,\varphi) : Z \times M \to \mathbb{R}$ dos formas bilineales continuas tales que $b(\cdot,\cdot)$ satisface la llamada condición $inf sup$:*

$$\exists \beta > 0, \inf_{\xi \in Z} \sup_{\varphi \in M} \frac{b(\xi,\varphi)}{\|\xi\|_Z \|\varphi\|_M} \geq \beta \tag{3.1}$$

y la forma bilineal $a(\cdot,\cdot)$ es elíptica en el kernel derecho K de $b(\cdot,\cdot)$:

$$K = \{\varphi \in M, \forall \xi \in Z, b(\xi,\varphi) = 0\} \tag{3.2}$$

$$\exists \alpha > 0, \forall \varphi \in K, a(\varphi,\varphi) \geq \alpha \|\varphi\|_M \tag{3.3}$$

entonces para cada par $(\rho,\sigma) \in M \times Z$ el problema variacional mixto: encontrar $\psi \in Z$, $w \in M$ tales que

$$a(w,\varphi) + b(\psi,\varphi) = \langle \rho,\varphi \rangle_M \ \ \forall \varphi \in M \tag{3.4}$$

$$b(\xi,w) = \langle \sigma,\xi \rangle_Z \ \ \forall \xi \in Z \tag{3.5}$$

tiene una única solución $(\psi,w) \in Z \times M$ que depende continuamente del dato (ρ,σ):

$$\exists c > 0, \|\psi\|_Z + \|w\|_M \leq c(\|\rho\|_M + \|\sigma\|_Z) \tag{3.6}$$

Consideremos el problema de Stokes en su formulación corriente - vorticidad (2.7) introduciendo los siguientes espacios de Hilbert

$$M(\Omega) = \{\varphi \in L^2(\Omega), \Delta\varphi \in H^{-1}(\Omega)\} \tag{3.7}$$

donde $H^{-1}(\Omega)$ es el espacio dual topológico de $H_0^1(\Omega)$ con la norma asociada

$$H^{-1}(\Omega) \ni \Theta \to \|\Theta\|_{-1,\Omega} = \sup_{v \in H_0^1(\Omega)} \frac{\langle \Theta, v \rangle}{\|\nabla v\|_{0,\Omega}}. \tag{3.8}$$

La norma en el espacio $M(\Omega)$ está definida por la relación:

$$M(\Omega) \ni \varphi \to \|\varphi\|_M = (\|\varphi\|_{0,\Omega}^2 + \|\Delta\varphi\|_{-1,\Omega}^2)^{\frac{1}{2}} \tag{3.9}$$

Si $\mathbf{f}$ está dada en el espacio $(L^2(\Omega))^2$, la formulación variacional del problema (2.7) es la siguiente: encontrar $\psi \in H_0^1(\Omega)$, $w \in M(\Omega)$ tales que

$$\int_\Omega \nabla\psi(x)\nabla\varphi(x)dx - \int_\Omega w(x)\varphi(x)dx = 0, \quad \forall\varphi \in M(\Omega)$$
$$\frac{1}{Re}\int_\Omega \nabla w(x)\nabla\xi(x)dx = \int_\Omega rot(\mathbf{f}(\mathbf{x}))\xi(x)dx, \quad \forall\xi \in H_0^1(\Omega)$$

En la presente formulación variacional la integral de frontera $\int_{\partial\Omega} \nabla\psi(x)n(x)q(x)d\sigma$ se anula, puesto que en los experimentos numéricos se supone que la corriente ψ se conoce en la frontera de la región. Está formulación variacional se puede escribir como un problema de punto de silla usando las formas bilineales

$$a(\psi, \varphi) = \int_\Omega \nabla\psi(x)\nabla\varphi(x)dx \tag{3.10}$$
$$b(w, \xi) = \int_\Omega \nabla w(x)\nabla\xi(x)dx. \tag{3.11}$$

Así la formulación variacional del problema es encontrar $\psi \in H_0^1(\Omega)$, $w \in M(\Omega)$ tales que:

$$a(\psi, \varphi) - b(w, \varphi) = 0 \quad \forall\varphi \in M(\Omega) \tag{3.12}$$
$$\frac{1}{Re}b(\xi, w) = (rot(f), \xi)_0 \quad \forall\xi \in H_0^1(\Omega) \tag{3.13}$$

Este problema de acuerdo con el teorema de de Brezzi, tiene única solución, véase [5].

Las demostraciones de que las formas bilineales (3.10) y (3.11) satisfacen las hipótesis del Teorema de Brezzi se pueden consultar en [9]. A continuación se agrega la demostración de que la forma bilineal $a(\cdot,\cdot)$ es elíptica en el kernel derecho K de $b(\cdot,\cdot)$ realizada por [9] :

El kernel K definido en (3.2) puede ser evaluado y escrito como

$$K = \left\{\varphi \in L^2(\Omega), \Delta\varphi = 0 \text{ en } H^1(\Omega)\right\}.$$

Entonces en L^2 el producto escalar es claramente elíptico (con $\alpha = 1$) sobre K para la norma (3.9) en el espacio $M(\Omega)$

Capítulo 4

El método de los elementos finitos

El método de los elementos finitos es un método numérico para aproximar soluciones de ecuaciones diferenciales parciales. Consiste en una terna (K, P, Q) donde P es un conjunto de funciones (pueden ser lineales, bilineales, cuadráticas, etc) definidas en un dominio K en $\mathbb{R}^n$ donde n puede ser 2 o 3 y Q son los grados de libertad, es decir, es un conjunto de valores que nos indica cómo obtener las funciones en P. En general la implementación numérica del problema se realiza en tres pasos, [1]:

- Partición del dominio
- Discretización de la ecuación integral
- Solución del problema discreto resultante

Para este caso, K es un rectángulo de vértices b_1, b_2, b_3, b_4,

$$P = \{u\colon K \to \mathbb{R} \mid u|_K \text{ es bilineal}\},$$
$$Q = \{u(b_1), u(b_2), u(b_3), u(b_4)\},$$

donde Q es el conjunto de valores de las funciones de P en los vértices del cuadrilátero. Sea $T_h = \cup_{i=1}^{m} K_i$ una partición del dominio, véase [4], donde K_i son rectángulos. Las funciones ψ y w se aproximan mediante funciones $\widetilde{\psi}, \widetilde{w} \in V_h$ donde V_h es el espacio de elementos finitos definido como $V_h = \{v\colon T_h \to \mathbb{R} \mid v|_K \in P \quad \forall k \in T_h\}$ es decir,

$$\psi(x) \approx \widetilde{\psi}(x)$$
$$w(x) \approx \widetilde{w}(x).$$

Puesto que el espacio de elementos finitos V_h tiene dimensión finita [4], existe una base $\{\varphi_1(x), \varphi_2(x), \varphi_3(x).....\varphi_N(x)\}$ de V_h para la cual

$$\widetilde{\psi}(x) = \sum_{i=1}^{N} \alpha_i \varphi_i(x)$$
$$\widetilde{w}(x) = \sum_{i=1}^{N} \beta_i \varphi_i(x)$$

para algunos coeficientes α_i, β_i con $i = 1, 2, 3, ..N$. En general se escogen los elementos de la base con la siguiente propiedad: Sean $a_1, a_2, a_3, ..., a_n$ los nodos de la partición T_h entonces

$$\varphi_i(a_j) = \delta_{ji}, \forall i, j = 1, 2, 3, ..., N.$$

4.1. Discretización del problema variacional

Para discretizar el problema (2.7) se reemplaza en la formulación variacional la función corriente ψ por $\widetilde{\psi}$ y la vorticidad w por $\widetilde{w}$, además se reemplazan las funciones de prueba v y q por funciones en el espacio de elementos finitos, es decir $q(x) = \varphi_j(x)$ y $v(x) = \varphi_j(x)$, $j = 1, \ldots, N$, esto es

$$-\int_\Omega \sum_{i=1}^{N} \beta_i \varphi_i(x)\varphi_j(x)dx + \int_\Omega \nabla(\sum_{i=1}^{N} \alpha_i \varphi_i(x))\nabla\varphi_j(x)dx = \int_{\partial\Omega} \nabla\psi(x)n(x)\varphi_j(x)d\sigma$$
$$\frac{1}{Re}\int_\Omega \nabla(\sum_{i=1}^{N} \beta_i \varphi_i(x))\nabla\varphi_j(x)dx = \int_\Omega rot(f(x))\varphi_j(x)dx$$

Operando se tiene el sistema de ecuaciones lineales para las incógnitas $\alpha = (\alpha_1, \alpha_2, ..., \alpha_N)$, $\beta = (\beta_1, \beta_2, ..., \beta_N)$.

$$-\int_\Omega \sum_{i=1}^{N} \beta_i \varphi_i(x)\varphi_j(x)dx + \int_\Omega \sum_{i=1}^{N} \alpha_i \nabla\varphi_i(x)\nabla\varphi_j(x)dx = \int_{\partial\Omega} \nabla\psi(x)n(x)\varphi_j(x)d\sigma$$
$$\frac{1}{Re}\int_\Omega \sum_{i=1}^{N} \beta_i \nabla\varphi_i(x)\nabla\varphi_j(x)dx = \int_\Omega rot(f(x))\varphi_j(x)dx$$

Por lo tanto la matriz de rigidez del sistema está dada por

$$\begin{bmatrix} [B_1]_{N\times N} & [A]_{N\times N} \\ [B_2]_{N\times N} & [0]_{N\times N} \end{bmatrix} \begin{bmatrix} [\beta]_{1\times N} \\ [\alpha]_{1\times N} \end{bmatrix} = \begin{bmatrix} [F]_{1\times N} \\ [G]_{1\times N} \end{bmatrix}$$

donde

$$\begin{aligned} [B_1]_{ij} &= \int_\Omega \varphi_i(x)\varphi_j(x)dx \\ [B_2]_{ij} &= \frac{1}{Re}\int_\Omega \nabla\varphi_i(x)\nabla\varphi_j(x)dx \\ [A]_{ij} &= \int_\Omega \nabla\varphi_i(x)\nabla\varphi_j(x)dx \\ [F]_{1j} &= \int_{\partial\Omega} \nabla\psi(x)n(x)\varphi_j(x)d\sigma \\ [G]_{1j} &= \int_\Omega rot(f(x))\varphi_j(x)dx \end{aligned}$$

Además usando el hecho de que las funciones bases φ_i tienen soporte compacto y $\Omega = T_h$, véase [4], se obtiene

$$\begin{aligned} [B_2]_{ij} &= \frac{1}{Re}\int_{\cup_{i=1}^m K_i} \nabla\varphi_i(x)\nabla\varphi_j(x)dx = \frac{1}{Re}\sum_{i=1}^N \int_{K_i} \nabla\varphi_i(x)\nabla\varphi_j(x)dx \\ [B_2]_{ij} &= \frac{1}{Re}\int_{supp(\varphi_i)\cap supp(\varphi_j)\neq\phi} \nabla\varphi_i(x)\nabla\varphi_j(x)dx \\ [B_1]_{ij} &= \int_{\cup_{i=1}^m K_i} \varphi_i(x)\varphi_j(x)dx = \sum_{i=1}^N \int_{K_i} \varphi_i(x)\varphi_j(x)dx \\ [B_1]_{ij} &= \int_{supp(\varphi_i)\cap supp(\varphi_j)\neq\phi} \varphi_i(x)\varphi_j(x)dx \\ [A]_{ij} &= \int_{\cup_{i=1}^m K_i} \nabla\varphi_i(x)\nabla\varphi_j(x)dx = \sum_{i=1}^N \int_{K_i} \nabla\varphi_i(x)\nabla\varphi_j(x)dx \\ [A]_{ij} &= \int_{supp(\varphi_i)\cap supp(\varphi_j)\neq\phi} \nabla\varphi_i(x)\nabla\varphi_j(x)dx. \end{aligned}$$

Estas integrales se calculan de manera numérica usando una cuadratura de Gauss.

4.2. Implementación numérica Deal.II

La implementación numérica del problema de Stokes en las variables corriente-vorticidad se realizó usando las rutinas del paquete deal.II, que es un conjunto de librerías de C++ dirigido a la solución numérica de las ecuaciones en derivadas parciales utilizando elementos finitos. El objetivo principal de deal.II es permitir el rápido desarrollo de modernos códigos de elementos finitos, utilizando entre otros aspectos, las mallas de adaptación y una amplia variedad de clases de herramientas de uso frecuente en el programa de elementos finitos. Escribir este tipo de programas es una tarea no trivial, y programas exitosos tienden a ser muy grandes y complejos. Se cree que esto se hace mejor con una biblioteca de programas que se encarga de los detalles del manejo de la malla y el refinamiento, la manipulación de los grados de libertad, de las mallas de entrada y salida de los resultados en formatos gráficos, etc. Para este trabajo se usa la version 6.2.1 para más información sobre Deal.II consulte la presente dirección http://www.dealii.org/

Stokes 2D Este programa está escrito para resolver el problema de Stokes 2D en las variables corriente - vorticidad cuyas ecuaciones están representadas en (2.7); la formulación débil para este problema se describió en la sección 3. En la primera parte del código se define la clase `MixedLaplaceProblem`, que contiene las rutinas que ejecutan los pasos que el método de elementos finitos necesita para resolver numéricamente el problema: una rutina para hacer la triangulación del domino, definir el elemento finito que se usa, ensamblar las matrices de rigidez, resolver numéricamente el sistema de ecuaciones lineales y finalmente exportar los resultados para ser procesados con un visualizador gráfico.

```
template <int dim>
class MixedLaplaceProblem
{
  public:
    MixedLaplaceProblem ();
    ~MixedLaplaceProblem ();
    void run ();

  private:
    void make_grid_and_dofs ();
    void setup_system ();
    void assemble_system ();
    void solve ();
    void compute_errors () const;
    void output_results () const;
```

```
Triangulation<dim>   triangulation;
FESystem<dim>        fe;
DoFHandler<dim>      dof_handler;
ConstraintMatrix     hanging_node_constr

SparsityPattern      sparsity_pattern;
SparseMatrix<double> system_matrix;

Vector<double>       solution;
Vector<double>       system_rhs;
```

En esta parte del código se define el rotacional de la función del lado derecho, es decir de las fuerzas externas que actúan sobre el fluido.

```
template <int dim>
class RotRightHandSide :  public Function<dim>
{
  public:
    RotRightHandSide ();

    virtual void vector_value (const Point<dim> &p,
                               Vector<double>   &values) const;

    virtual void vector_value_list (const std::vector<Point<dim> > &points,
                                    std::vector<Vector<double> >   &value_list) const;
```

Esta clase define en un vector los valores de la solución en la frontera, condición de Dirichlet.

```
template <int dim>
class BoundaryValues : public Function<dim>
{
    public:
    BoundaryValues  () : Function<dim>(dim) {}

    virtual void vector_value (const Point<dim> &p,
                               Vector<double>   &value) const;
```

Este código define las componente del rotacional de la función f del lado derecho de la ecuación.

```
template <int dim>
 inline
 void RotRightHandSide<dim>::vector_value (const Point<dim> &p,
                                           Vector<double>   &values) const
{
  values(0) = 0;
  values(1) = 0;
```

Aquí se dan los valores de vorticidad y la corriente en la frontera de la región.

```
void
BoundaryValues<dim>::vector_value (const Point<dim> &p,
                                   Vector<double>   &values) const
{
  Assert (values.size() == dim,
          ExcDimensionMismatch (values.size(), dim));
values(0) =20*p[0]*p[0]*p[0] - 60*p[0]*p[1]*p[1];
values(1) =5*p[0]*p[1]*p[1]*p[1]*p[1] - p[0]*p[0]*p[0]*p[0]*p[0];
```

Esta clase define el valor exacto de la solución en la vorticidad y la corriente, que se usa para encontrar el error y determinar la convergencia del método.

```
template <int dim>
class ExactSolution : public Function<dim>
{
  public:
    ExactSolution () : Function<dim>(dim) {}

    virtual void vector_value (const Point<dim> &p,
                               Vector<double>   &value) const;
};
```

```
template <int dim>
void
ExactSolution<dim>::vector_value (const Point<dim> &p,
                                  Vector<double>   &values) const
{
  Assert (values.size() == dim,
          ExcDimensionMismatch (values.size(), dim));


  values(0) =20*p[0]*p[0]*p[0] - 60*p[0]*p[1]*p[1];
  values(1) =5*p[0]*p[1]*p[1]*p[1]*p[1] - p[0]*p[0]*p[0]*p[0]*p[0];
```

En esta parte del código se indica que elemento se va a usar para cada una de las variables, por ejemplo este código usa elementos bilineales para ambas variables

```
template <int dim>
MixedLaplaceProblem<dim>::MixedLaplaceProblem ()
        :
        dof_handler (triangulation),
        fe (FE_Q<dim>(1), dim)
```

Aquí se indica el número de veces que se refina y las coordenadas del rectángulo además, se pide que imprima el total de celdas y grados de libertad

```
template <int dim>
void MixedLaplaceProblem<dim>::make_grid_and_dofs ()
{
  GridGenerator::hyper_cube (triangulation, 0, 1);
  triangulation.refine_global (5);

std::cout << "   Number of active cells: "
          << triangulation.n_active_cells()
          << std::endl
          << "   Total number of cells: "
          << triangulation.n_cells()
          << std::endl;

  dof_handler.distribute_dofs (fe);

  std::cout << "   Number of degrees of freedom: "
          << dof_handler.n_dofs()
          << std::endl;
```

Lo siguiente es el ensamble del sistema, aquí se discretizan las ecuaciones de Stokes en su forma variacional

```
void MixedLaplaceProblem<dim>::assemble_system ()
{


  QGauss<dim>  quadrature_formula(2);
  QGauss<dim-1> face_quadrature_formula(2);


  FEValues<dim> fe_values (fe, quadrature_formula,
                           update_values   | update_gradients |
                                     update_quadrature_points | update_JxW_values);

 FEFaceValues<dim> fe_face_values (fe, face_quadrature_formula,
                                   update_values    | update_normal_vectors |
                                   update_q_points  | update_JxW_values);


  const unsigned int   dofs_per_cell = fe.dofs_per_cell;
  const unsigned int   n_q_points    = quadrature_formula.size();
  const unsigned int   n_face_q_points = face_quadrature_formula.n_quadrature_po



  FullMatrix<double>   cell_matrix (dofs_per_cell, dofs_per_cell);
  Vector<double>       cell_rhs (dofs_per_cell);

  std::vector<unsigned int> local_dof_indices (dofs_per_cell);



  RotRightHandSide<dim>      Rotright_hand_side;
  BoundaryValues<dim>    BoundaryValues;
  const ExactSolutionpsi<dim> exact_solutionpsi;
  std::vector<Vector<double> > rhs_values (n_q_points,
                                           Vector<double>(dim));

unsigned int comp_i, comp_j;
typename DoFHandler<dim>::active_cell_iterator cell = dof_handler.begin_active()
                                                      endc = dof_handler.end();
for (; cell!=endc; ++cell)
    {
    cell_matrix = 0;
        cell_rhs = 0;

        fe_values.reinit (cell);
        Rotright_hand_side.vector_value_list (fe_values.get_quadrature_points(),
                                              rhs_values);

  for (unsigned int q=0; q<n_q_points; ++q)
        for (unsigned int i=0; i<dofs_per_cell; ++i)
          {
        comp_i = fe.system_to_component_index(i).first;
            for (unsigned int j=0; j<dofs_per_cell; ++j)
              {
```

```
        // fi * fj
if ( (comp_i==1) && (comp_j==0) )
    cell_matrix(i,j) +=  -fe_values.shape_value(i,q) *
                         fe_values.shape_value(j,q) *
                         fe_values.JxW(q);


      }
 cell_rhs(i) += fe_values.shape_value(i,q) *
                     rhs_values[q](comp_i) * fe_values.JxW(q);
  }
```

en esta parte del código se resuelve la derivada normal de ψ para poder calcular la integral de frontera $\int_{\partial\Omega} \nabla\psi(x)n(x)q(x)d\sigma$

```
     face<GeometryInfo<dim>::faces_per_cell;
     ++face)
   if (cell->at_boundary(face))
     {
      fe_face_values.reinit (cell, face);


      for (unsigned int q=0; q<n_face_q_points; ++q)
   {
         double neumann_value
           = (exact_solutionpsi.gradient (fe_face_values.quadrature_point(
             fe_face_values.normal_vector(q));


   for (unsigned int i=0; i<dofs_per_cell; ++i) {
           comp_i = fe.system_to_component_index(i).first;
    if (comp_i==1)
           cell_rhs(i) += (neumann_value *
                           fe_face_values.shape_value(i,q) *
                           fe_face_values.JxW(q));
    }
     }
```

El siguiente código asigna a las incógnitas del sistema en la frontera de la región el valor que el usario proporciona en la función `BoundaryValues`. Para distinguir las incógnitas en la frontera de aquellas en el interior del dominio se asigna a las aristas en la frontera un marcador con valor cero.

```
std::map<unsigned int,double> boundary_values;
VectorTools::interpolate_boundary_values (dof_handler,
                                          0,
                                          BoundaryValues,
                                          boundary_values,std::vector<bool>(false,true));
MatrixTools::apply_boundary_values (boundary_values,
                                    system_matrix,
                                    solution,
                                    system_rhs);
```

En esta parte se resuelve el sistema de ecuaciones lineales que se obtiene con la discretización del método de elementos finitos.

```
template <int dim>
void MixedLaplaceProblem<dim>::solve ()
{
  SolverControl           solver_control (1000, 1e-8);
  SolverGMRES<>              gmres (solver_control);

  PreconditionJacobi<> preconditioner;
  preconditioner.initialize(system_matrix, 1.0);

  gmres.solve (system_matrix, solution, system_rhs,
       preconditioner);
  std::cout << solver_control.last_step()
            << " Solver iterations to obtain convergence."
            << std::endl;
  hanging_node_constraints.distribute (solution);
```

En lo siguiente se calcula el error de la aproximación en $L^2(\Omega)$

```
void MixedLaplaceProblem<dim>::compute_errors () const
{
  const ComponentSelectFunction<dim>
      w_mask (0, 2);
  const ComponentSelectFunction<dim>
    psi_mask(1, 2);
       ExactSolutionpsi<dim> exact_solutionpsi;
  ExactSolution<dim> exact_solution;
  Vector<double> cellwise_errors (triangulation.n_active_cells());


  QGauss<2>     quadrature(4);

  VectorTools::integrate_difference (dof_handler, solution, exact_solution,
                                     cellwise_errors, quadrature,
                                      VectorTools::L2_norm,
                                     &w_mask);
  const double w_l2_error = cellwise_errors.l2_norm();

  VectorTools::integrate_difference (dof_handler, solution, exact_solution,
                                     cellwise_errors, quadrature,
                                      VectorTools::L2_norm,
                                     &psi_mask);
  const double psi_l2_error = cellwise_errors.l2_norm();

  std::cout << "Errors: ||e_w||_L2 = " << w_l2_error
        << ",   ||e_psi||_L2 = " << psi_l2_error
```

En esta parte definimos el formato en el cual queremos exportar la aproximación de la solución para su posterior visualización.

```
void MixedLaplaceProblem<dim>::output_results () const
{
  std::vector<std::string> solution_names;
  switch (dim)
    {
      case 2:
            solution_names.push_back ("w");

            solution_names.push_back ("psi");
            break;


      default:
            Assert (false, ExcNotImplemented());
    }


  DataOut<dim> data_out;

  data_out.attach_dof_handler (dof_handler);
  data_out.add_data_vector (solution, solution_names);

  data_out.build_patches (1);

  std::ofstream output ("solution.gmv");
  data_out.write_gmv (output);
}
```

Capítulo 5

Validación numérica

5.1. Test de Bercovier–Engelman (condición Dirichlet nula)

A continuación consideramos el Test de Bercovier–Engelman, el cual resuelve las ecuaciones de Stokes (1) en $\Omega = (0,1)^2$. En este caso se tiene $Re = 1$, además $\mathbf{u} = 0$ sobre $\partial\Omega$ y la función del lado derecho $\mathbf{f}(x,y)$ tiene componentes$f_1(x,y), f_2(x,y)$ definidos de la siguiente forma, véase [8] y [1]:

$$\begin{aligned} f_1(x,y) &= g(x,y) + y - 0.5 \\ f_2(x,y) &= -g(y,x) + x - 0.5 \end{aligned}$$

donde

$$g(x,y) = 256(x^2(x-1)^2(12y-6) + y(y-1)(2y-1)(12x^2 - 12x + 2))$$

Las componentes de la velocidad $\mathbf{u}(x,y)$ son $u_1(x,y), u_2(x,y)$ y la solución analítica del sistema es

$$\begin{aligned} u_1(x,y) &= -256y(y-1)(2y-1)x^2(x-1)^2 \\ u_2(x,y) &= -u_1(y,x) \\ p(x,y) &= (x-0.5)(y-0.5). \end{aligned}$$

Por cálculos directos se encuentra que la solución analítica en las nuevas variables vorticidad y corriente está dada por

$$w = 256(y^2(y-1)^2(6x^2-6x+1) + x^2(x-1)^2(6y^2-6y+1))$$
$$\psi = -128x^2y^2(x-1)^2(y-1)^2$$

Las gráficas de las soluciones analíticas y aproximadas para el test de Bercovier-Engelman se presentan en el mismo dominio Ω con la intensión de que el lector pueda comparar los resultados de manera gráfica, se presenta primero la vorticidad analítica y luego la aproximada figuras 5.1 y 5.2, además se presentan estas mismas figuras junto con sus curvas de nivel figuras 5.3 y 5.4, la malla usada figuras 5.5 y 5.6 y sus formas tridimensionales simultáneamente con las curvas de nivel figuras 5.7 y 5.8. De forma análoga se presentan las soluciones de la corriente analítica y aproximada figuras 5.9, 5.10, 5.11, 5.12, 5.13, 5.14 con la diferencia de que no se muestra la malla usada ya que es la misma malla de la vorticidad.

Los colores en las figuras indican el valor de las funciones en esos puntos del dominio, la barra vertical a la izquierda de las figuras muestra el valor de cada color usado en la figura.

Se observa en la Figura 5.2 que la vorticidad es mayor en las fronteras, este comportamiento de la función era de esperarse ya que cuando el fluido toca la frontera cambia de dirección. En las figuras 5.5 y 5.6 se puede observar cómo se distribuye la malla usada en el dominio Ω, en la Figura 5.10 se observa que la corriente forma circunferencias concéntricas lo que da una idea del campo de velocidades $\mathbf{u}$. Las gráficas para las funciones analíticas y aproximadas son:

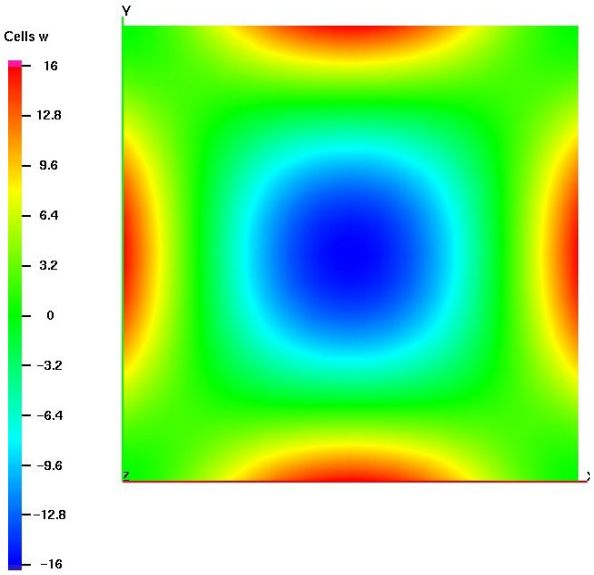

Figura 5.1: Vorticidad analítica

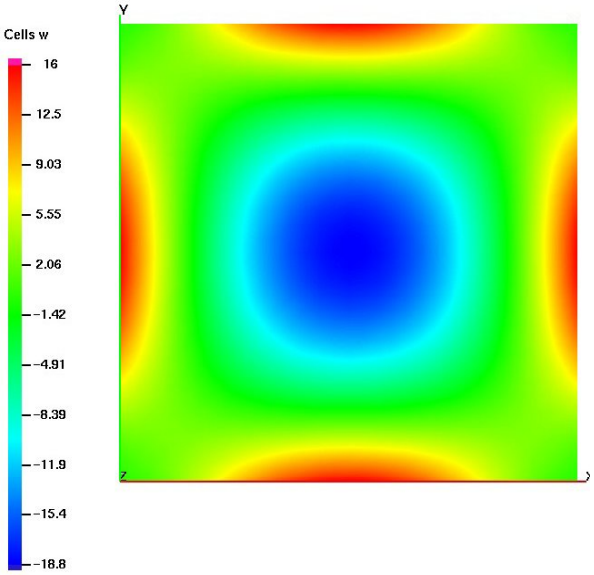

Figura 5.2: Vorticidad aproximada

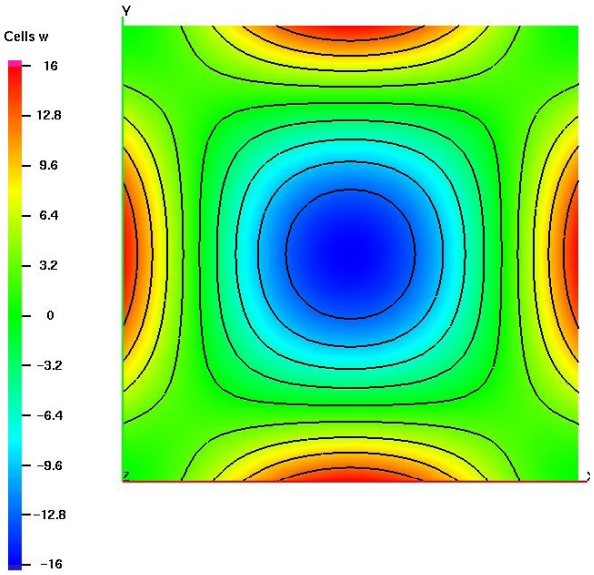

Figura 5.3: Curvas de nivel de la vorticidad analítica

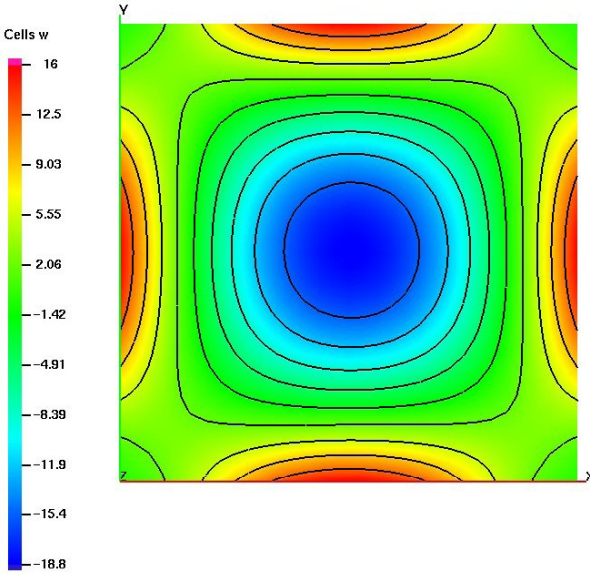

Figura 5.4: Curvas de nivel de la vorticidad aproximada

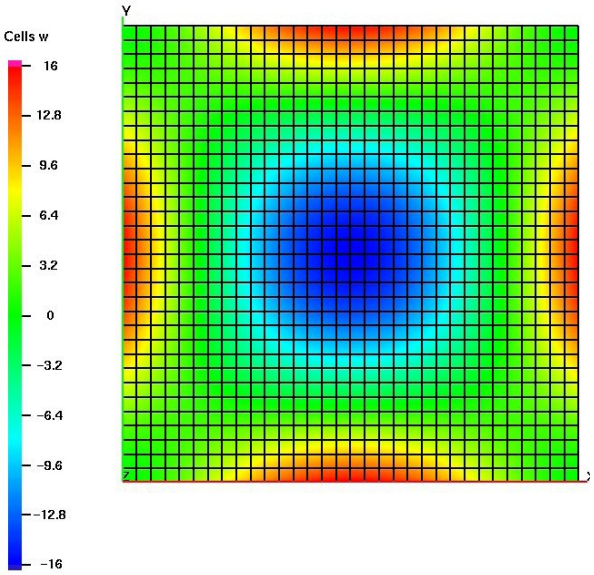

Figura 5.5: Vorticidad analítica y la malla usada en el método

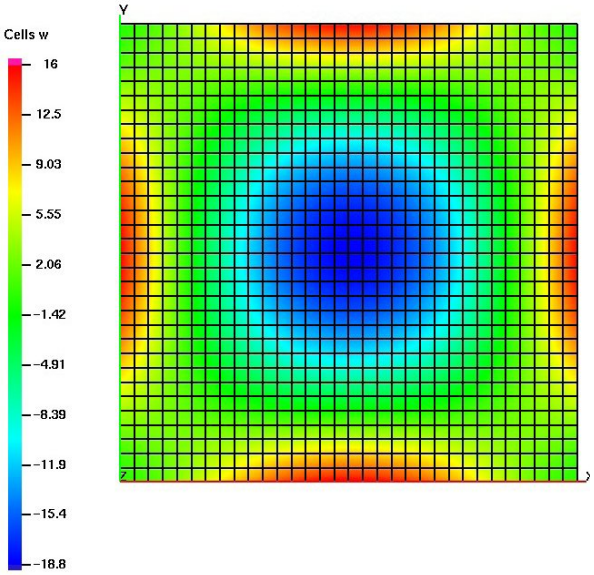

Figura 5.6: Vorticidad aproximada y la malla usada en el método

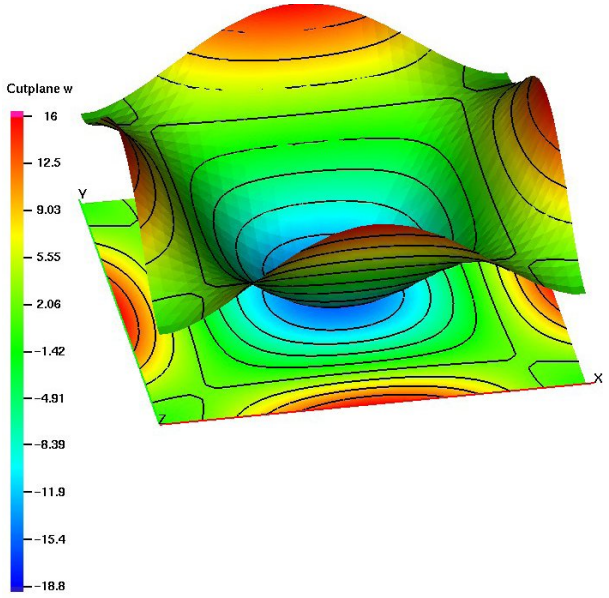

Figura 5.7: Curvas de nivel para la vorticidad analítica 3D

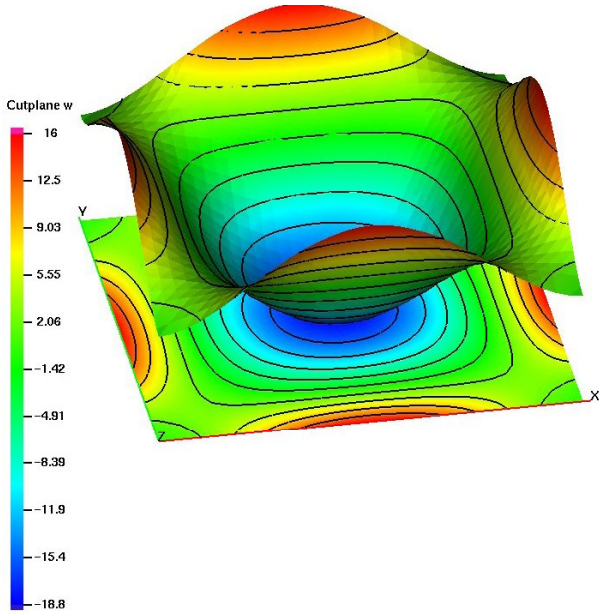

Figura 5.8: Curvas de nivel para la vorticidad aproximada en 3D

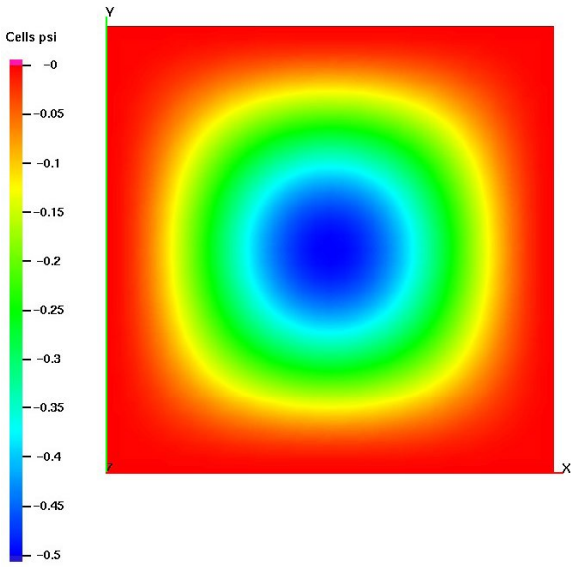

Figura 5.9: Corriente analítica

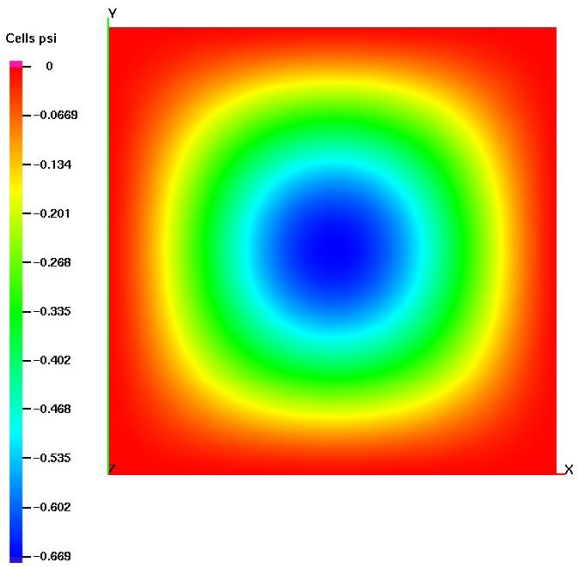

Figura 5.10: Corriente aproximada

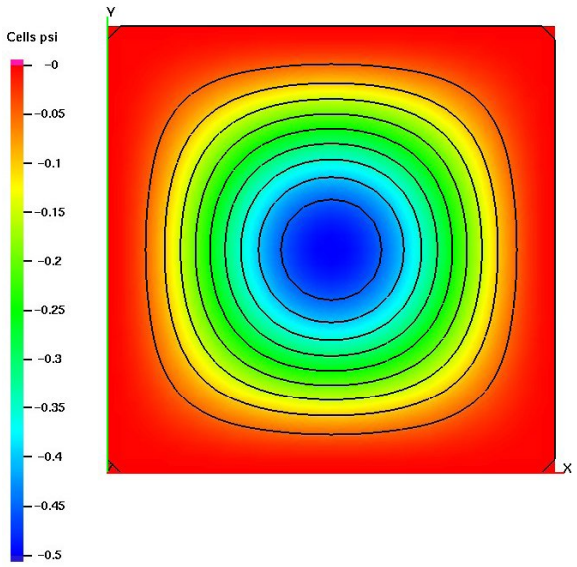

Figura 5.11: Curvas de nivel de la corriente analítica

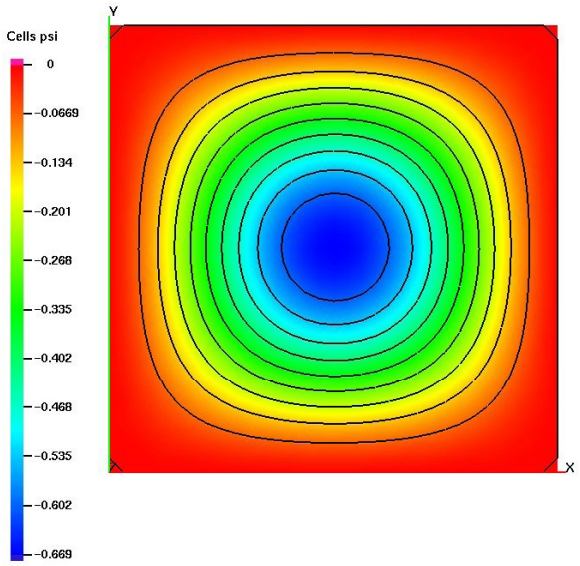

Figura 5.12: Curvas de nivel de la corriente aproximada

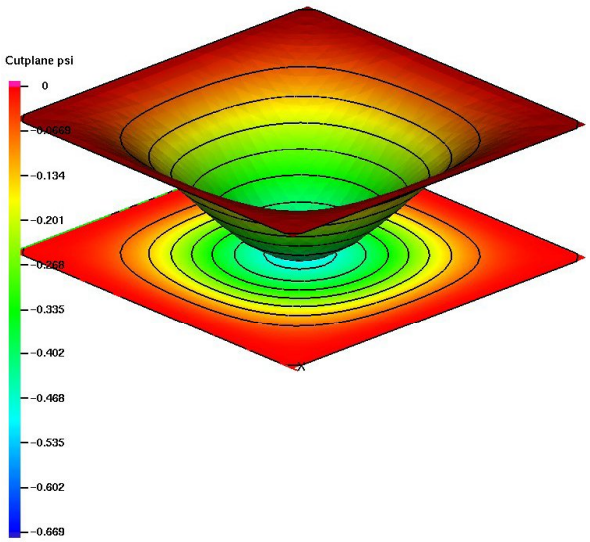

Figura 5.13: Curvas de nivel para la corriente analítica en 3D

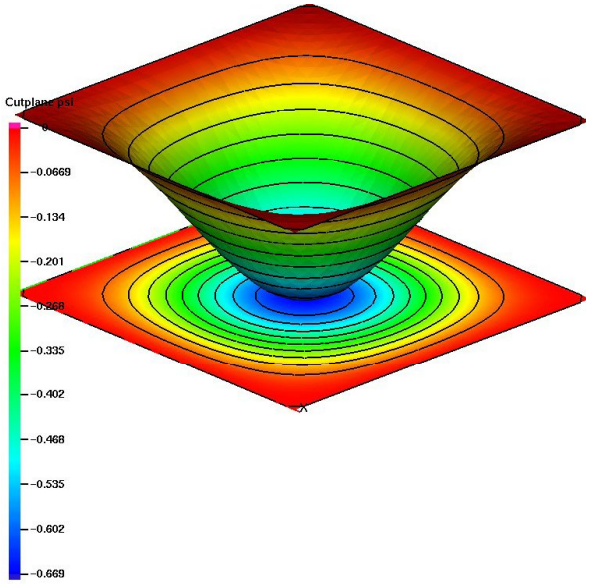

Figura 5.14: Curvas de nivel para la corriente aproximada en 3D

5.2. Test con condiciones de frontera no nula

Este ejemplo soluciona las ecuaciones de Stokes (1) en $\Omega = (0,1)^2$ para $Re = 1$, además $\mathbf{u} \neq 0$ sobre $\partial\Omega$ y la fuerza externa es nula,

$$f(x,y) = 0.$$

Los componentes de la velocidad $\mathbf{u}$ y la presión p para este test están definidos como:

$$\begin{aligned} u_1(x,y) &= 20xy^3 \\ u_2(x,y) &= 5x^4 - 5y^4 \\ p(x,y) &= 60x^2y - 20y^3 \end{aligned}$$

Con estos datos se calcula la función corriente y vorticidad analítica en forma directa y son:

$$\begin{aligned} w &= 20x^3 - 60xy^2 \\ \psi &= 5xy^4 - x^5 \end{aligned}$$

El error producido por la aproximación se visualiza en la siguiente tabla, en la primera columna se indican los grados de libertad y en la segunda y tercera el error de la aproximación en la norma $L_2(\Omega)$ además la primera fila muestra los valores obtenidos cuando se refina la malla 3 veces, la segunda fila para cuando se refina 4 veces, la tercera fila 5 veces y la cuarta cuando se refina 6 veces:

DOF	$\|w_h - w\|_{L_2(\Omega)}$	$\|\psi_h - \psi\|_{L_2(\Omega)}$
162	0.571816-01	0.181255-01
578	0.142716-01	0.455578-02
2178	0.35664-02	0.114047-02
8450	0.891507-03	0.285231-03

Las gráficas para la solución analítica y la aproximada se presentan de forma análogo a las del test de Bercovier-Engelman (5.1) y son las siguientes:

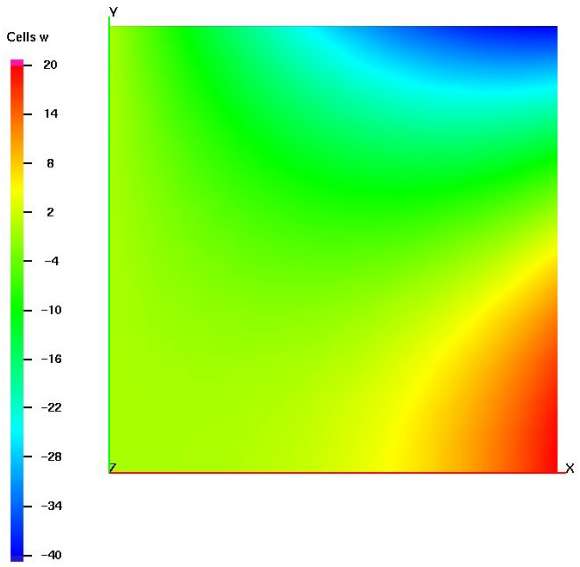

Figura 5.15: Vorticidad analítica

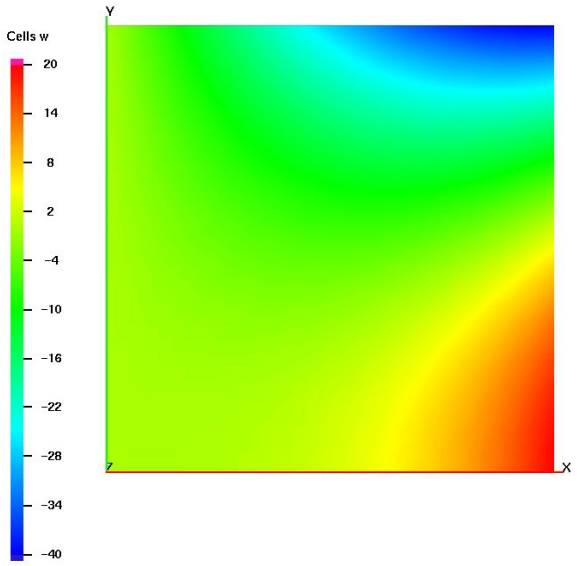

Figura 5.16: Vorticidad Aproximada

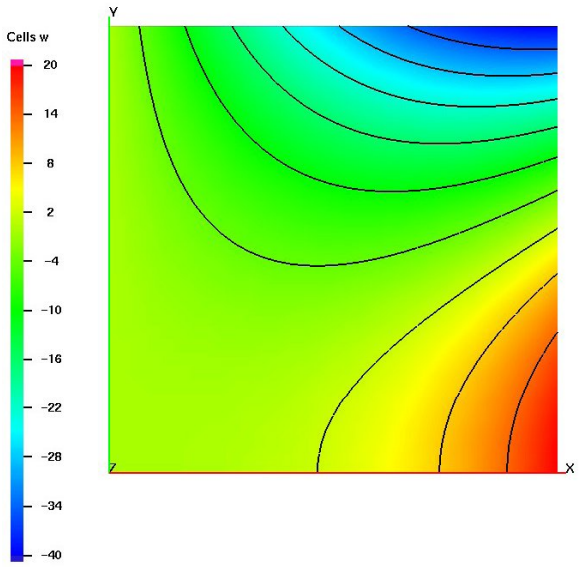

Figura 5.17: Curvas de nivel de la vorticidad analítica

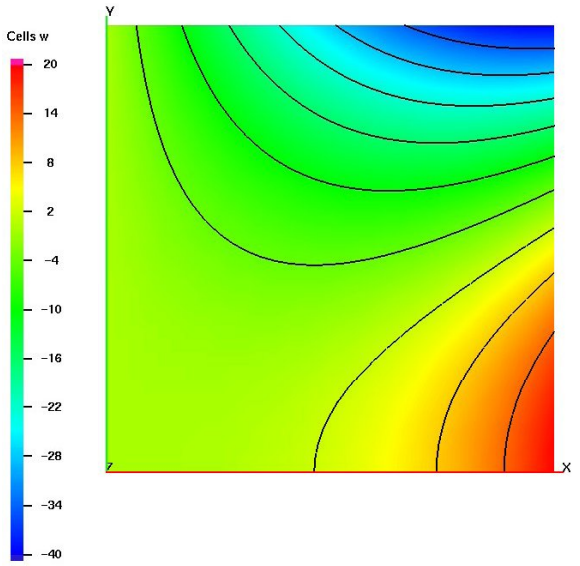

Figura 5.18: Curvas de nivel de la vorticidad aproximada

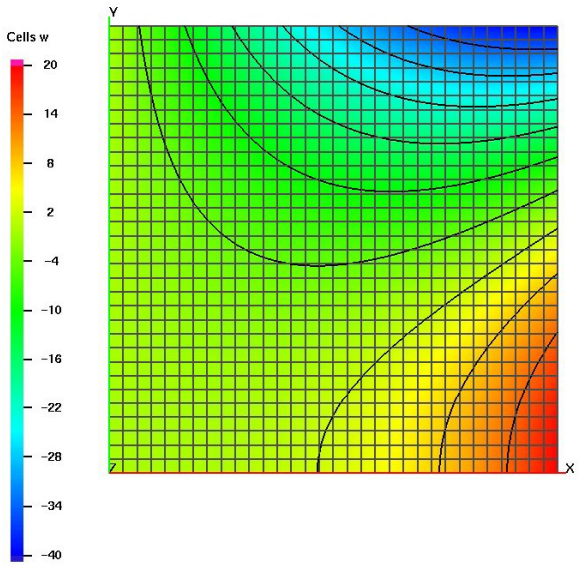

Figura 5.19: Vorticidad analítica y la malla usada en el método

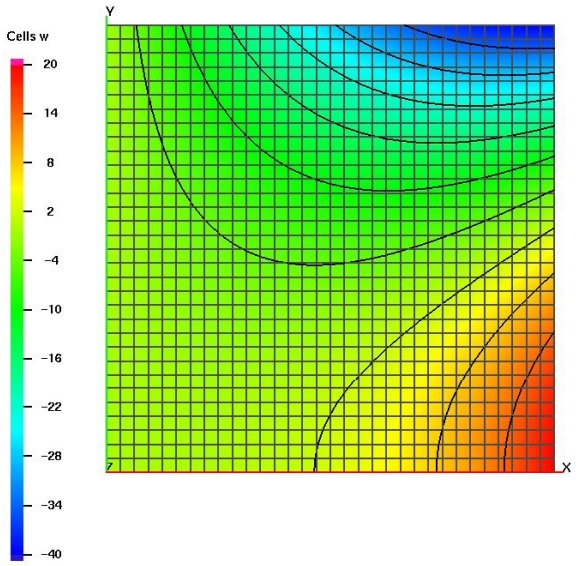

Figura 5.20: Vorticidad aproximada y la malla usada en el método

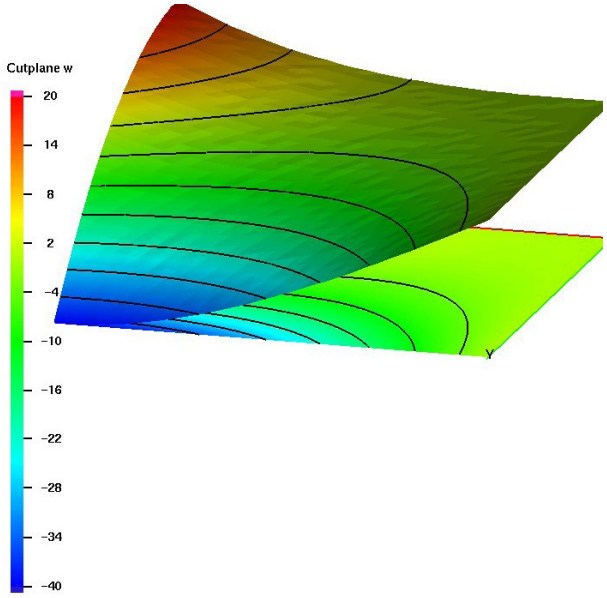

Figura 5.21: Curvas de nivel para la vorticidad analítica en 3D

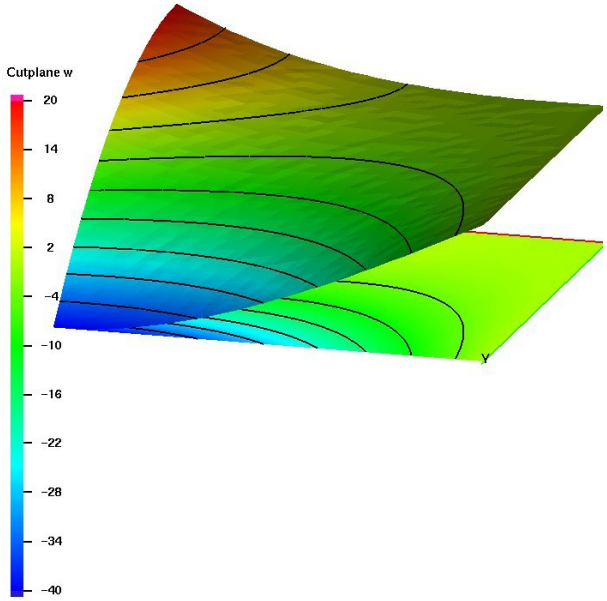

Figura 5.22: Curvas de nivel para la vorticidad aproximada en 3D

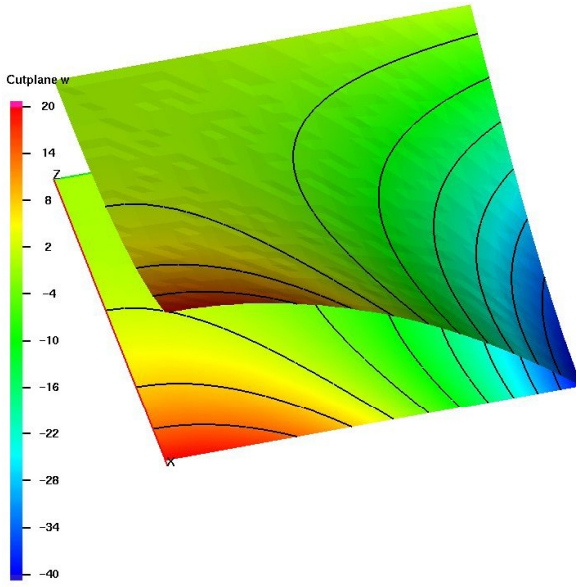

Figura 5.23: Curvas de nivel para la vorticidad analítica en 3D

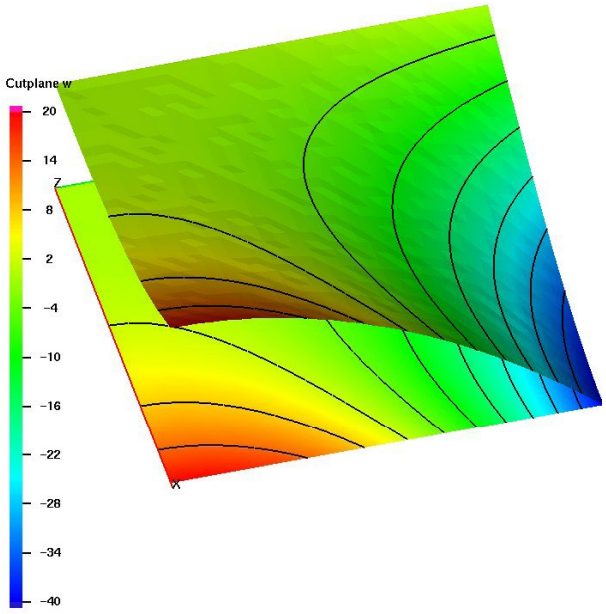

Figura 5.24: Curvas de nivel para la vorticidad aproximada en 3D

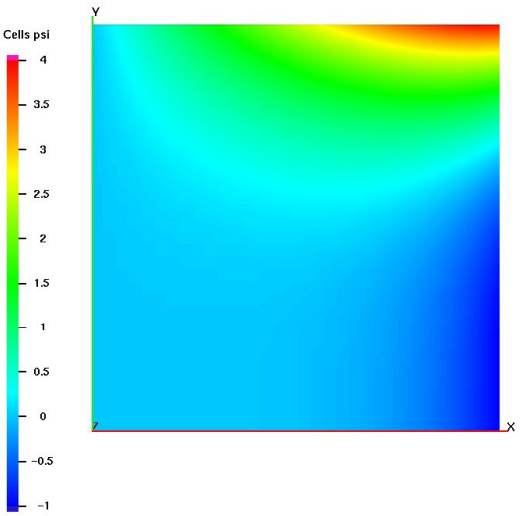

Figura 5.25: Corriente analítica

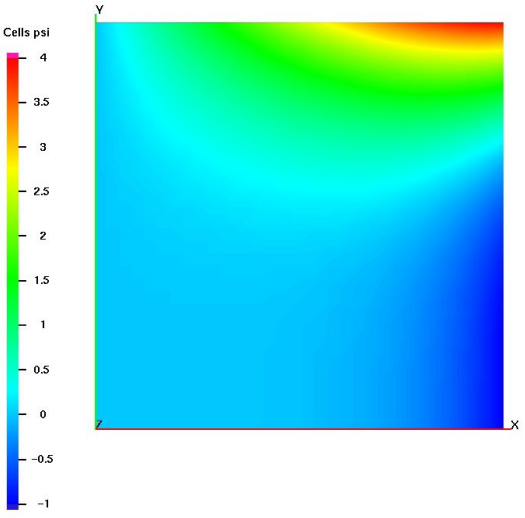

Figura 5.26: Corriente aproximada

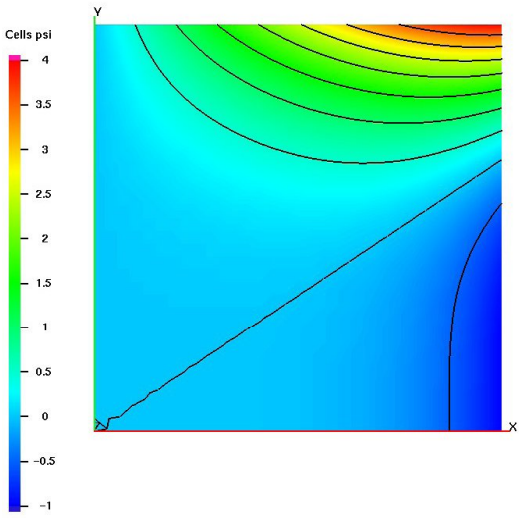

Figura 5.27: Curvas de nivel de la corriente analítica

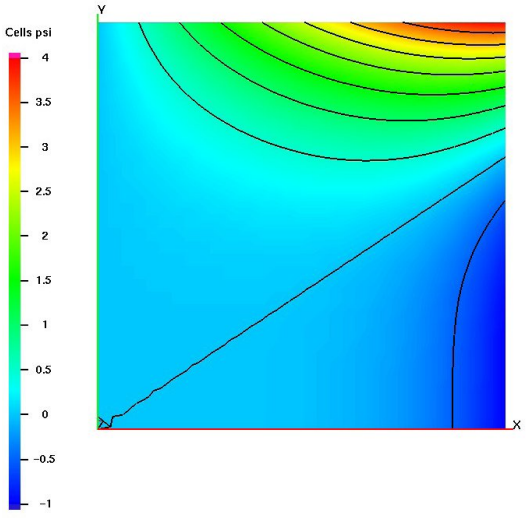

Figura 5.28: Curvas de nivel de la corriente aproximada

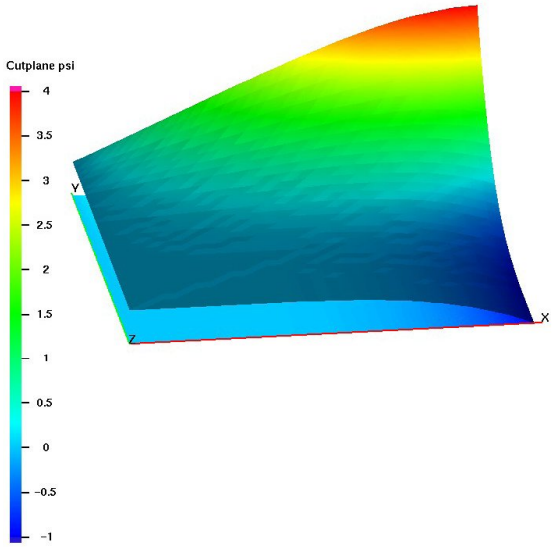

Figura 5.29: Curvas de nivel para la corriente analítica en 3D

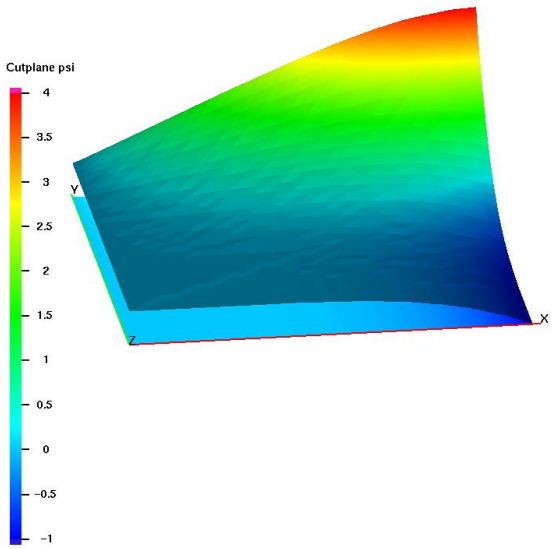

Figura 5.30: Curvas de nivel para la corriente aproximada en 3D

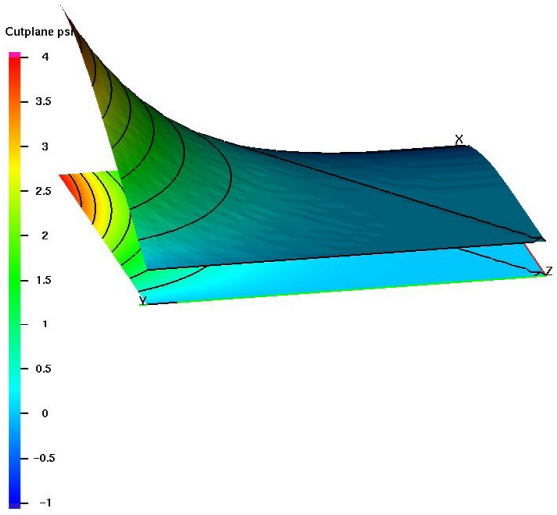

Figura 5.31: Curvas de nivel para la corriente analítica en 3D

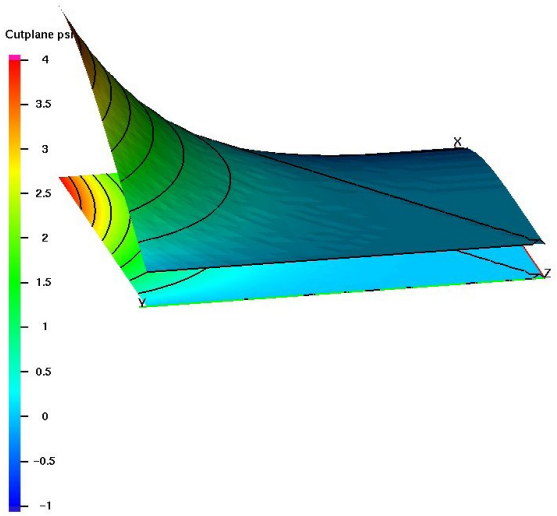

Figura 5.32: Curvas de nivel para la corriente aproximada en 3D

Capítulo 6

Conclusiones

1. En este trabajo se presenta la formulación corriente - vorticidad para el problema de Stokes; en esta formulación las curvas de nivel de la función corriente respresentan las lineas de corriente del flujo.

2. Con esta formulación no se puede aproximar la presión del fluido; para hacerlo se puede incluir en la formulación débil la ecuación de divergencia, conservando el grandiente de la presión en las ecuaciones de impulso.

3. Para calcular flujos en los cuales la vorticidad no se conoce en la frontera de la región es necesario introducir en el esquema numérico una condición de compatibilidad que permita resolver el sistema discreto.

4. Los cálculos numéricos obtenidos en la validación muestran que el proceso de aproximación mediante este método de elementos finitos es adecuado.

Bibliografía

[1] Alexander Gutiérrez;*Estabilización de un método de elementos finitos para problemas elípticos*, Tesis de grado. Universidad del Valle, Facultad de ciencias,Cali, (2003).

[2] Bruce R. Munson, Donald F. Young, TheoDore H. Okiishi; *Fundamentos de mecánica de fluidos*, Limusa Wileg,1ª edi, (2004), Mexico DF.

[3] M. B, Davis y G. F, Carey; *Interactive solution of the stream funtion-vorticity equations using a multigrid solver whit finite*, Communications in Numerical Methods in Engineering, Vol 9, No. 3, (2003).

[4] P. G. Ciarlet, *The finite element methods for elliptic problems*, North-Holland,1ª edi,(1997),Amsterdam

[5] W. Hackbusch; *Elliptic differential Equations: Theory and Numerical Treatment*, Springer-Verlag, Berlin, (1992).

[6] I G Currie; *Fundamental mechanics of fluids*, Marcel Dekker,3ª edi, (2003), Estados Unidos.

[7] Yunus A. Cengel, John M. Cimbala; *Mecánica de fluidos*, McGraw-Hill interamericana, 1ª edi, (2006), Mexico DF.

[8] Ga Hyung Jo, Hi Jun Choe; *Finite Element approximation of the vorticity-stream function equations for incompressible 2-dimensional Navier-Stokes flows*, Department of Mathematics Yonsei University, Korea.

[9] F. Dubois; *First vorticity-velocity-pressure numerical scheme for the Stokes problem* , Computer Methods In Applied. Mechanics and Engineering, (2003).

Apéndice A

Stokes2D

```
// PROGRAMA PARA CALCULAR UNA APROXIMACION A LA ECUACION DE STOKES EN LAS
VARIABLES CORRIENTE VORTICIDAD


// LA CORRIENTE ESTA REPRESENTADA POR psi Y LA VORTICIDAD POR W ES DECIR:
// CORRIENTE=psi
//VORTICIDAD=w


// ESTOS INCLUDE SON BIBLIOTECAS DE PROGRAMAS QUE USA EL Deal II PARA LOS
ELEMENTOS FINITOS

#include <base/quadrature_lib.h>
#include <base/logstream.h>
#include <base/function.h>
#include <lac/block_vector.h>
#include <lac/full_matrix.h>
#include <lac/block_sparse_matrix.h>
#include <lac/solver_cg.h>
#include <lac/solver_gmres.h>
#include <lac/precondition.h>
#include <grid/tria.h>
#include <grid/tria_boundary_lib.h>
#include <grid/grid_generator.h>
#include <grid/tria_accessor.h>
#include <grid/tria_iterator.h>
#include <dofs/dof_handler.h>
#include <dofs/dof_renumbering.h>
#include <dofs/dof_accessor.h>
#include <dofs/dof_tools.h>
#include <fe/fe_dgq.h>
#include <fe/fe_q.h>
#include <fe/fe_system.h>
#include <fe/fe_values.h>
```

```
#include <numerics/vectors.h>
#include <numerics/matrices.h>
#include <numerics/data_out.h>
#include <numerics/error_estimator.h>
#include <fstream>
#include <iostream>
#include <base/tensor_function.h>
#include <base/convergence_table.h>

using namespace dealii;
// EN ESTA PARTE DE DEFINEN ALGUNAS CLASES QUE SE VAN A USAR ASI COMO LA
ORGANIZACION QUE VA A TENER EL        PROGRAMA
template <int dim>
class MixedLaplaceProblem
{
  public:
    MixedLaplaceProblem ();
    ~MixedLaplaceProblem ();
    void run ();

  private:
    void make_grid_and_dofs ();
    void setup_system ();
    void assemble_system ();
    void solve ();
    void compute_errors () const;
    void output_results () const;


    Triangulation<dim>   triangulation;
    FESystem<dim>        fe;
    DoFHandler<dim>      dof_handler;
    ConstraintMatrix     hanging_node_constraints;


    SparsityPattern      sparsity_pattern;
    SparseMatrix<double> system_matrix;

    Vector<double>       solution;
    Vector<double>       system_rhs;
};
// EL RotRightHandSide ES EL ROTACIONAL DE LA FUNCION f ORIGINAL QUE EN
EL PROGRAMA LLAMAN RightHandSide ES DECIR EN LA ECUACION DE STOKES
REPRESENTAN LAS FUERZAS ESTERNAS
template <int dim>
class RotRightHandSide :  public Function<dim>
{
  public:
    RotRightHandSide ();

    virtual void vector_value (const Point<dim> &p,
                         Vector<double>   &values) const;
```

```
    virtual void vector_value_list (const std::vector<Point<dim> >
&points,
                                std::vector<Vector<double> >   &value_list)
const;
};
// EN ESTA CLASE SE DEFINEN LOS VALORES DE FRONTERA
template <int dim>
class BoundaryValues : public Function<dim>
{
    public:
    BoundaryValues  () : Function<dim>(dim) {}

    virtual void vector_value (const Point<dim> &p,
                           Vector<double>   &value) const;
};

//AQUI SE DEFINE EL RotRightHandSide COMO UN VECTOR
template <int dim>
void RotRightHandSide<dim>::vector_value_list (const
std::vector<Point<dim> > &points,
                                     std::vector<Vector<double> >
&value_list) const
{
  const unsigned int n_points = points.size();

  Assert (value_list.size() == n_points,
        ExcDimensionMismatch (value_list.size(), n_points));

  for (unsigned int p=0; p<n_points; ++p)
    RotRightHandSide<dim>::vector_value (points[p],
                                value_list[p]);
}
template <int dim>
RotRightHandSide<dim>::RotRightHandSide () :
            Function<dim> (dim)
{}

//POR ESTE LADO SE DEFINE EL VALOR DEL VECTOR RotRightHandSide
template <int dim>
 inline
 void RotRightHandSide<dim>::vector_value (const Point<dim> &p,
                                           Vector<double>   &values) const
{
  values(0) = 0;
  values(1) = 0;

}
//EN ESTA PARTE ESCRIBIMOS EL VALOR DE w Y psi  EN LA FRONTERA
template <int dim>
void
BoundaryValues<dim>::vector_value (const Point<dim> &p,
                           Vector<double>   &values) const
{
  Assert (values.size() == dim,
```

```
        ExcDimensionMismatch (values.size(), dim));
values(0) =20*p[0]*p[0]*p[0] - 60*p[0]*p[1]*p[1];
values(1) =5*p[0]*p[1]*p[1]*p[1]*p[1] - p[0]*p[0]*p[0]*p[0]*p[0];

}
// ESTA CLASE DEFINE EL VALOR EXACTO DE LA VARIABLE CORRIENTE psi
template <int dim>
 class ExactSolutionpsi : public Function<dim>
 {

     public:
     ExactSolutionpsi () : Function<dim>() {}

     virtual double value (const Point<dim>   &p,
                           const unsigned int  component = 0) const;

     virtual Tensor<1,dim> gradient(const Point<dim>   &p,
                                          const unsigned int  component =
0) const;

 };
// AQUI DAMOS EL VALOR EXACTO DE LA FUNCION psi
 template <int dim>
 double ExactSolutionpsi<dim>::value (const Point<dim>   &p,
                                      const unsigned int) const
 {

   return 5*p[0]*p[1]*p[1]*p[1]*p[1] - p[0]*p[0]*p[0]*p[0]*p[0];
 }



// POR ESTE LADO ESCRIBIMOS EL VALOR EXACTO DEL GRADIANTE DE psi
 template <int dim>
 Tensor<1,dim> ExactSolutionpsi<dim>::gradient (const Point<dim>   &p,
                                          const unsigned int  /*component
= 0*/) const
 {
   Tensor<1,dim> return_value;



       return_value[0] = 5*p[1]*p[1]*p[1]*p[1]-5*p[0]*p[0]*p[0]*p[0];


       return_value[1] = 20*p[0]*p[1]*p[1]*p[1];



   return return_value;
 }
```

```
//ESTA CLASE DEFINE EL VALOR EXACTO DE LA CORRIENTE Y VORTICIDAD
template <int dim>
class ExactSolution : public Function<dim>
{
  public:
    ExactSolution () : Function<dim>(dim) {}

    virtual void vector_value (const Point<dim> &p,
                               Vector<double>   &value) const;
};

template <int dim>
void
ExactSolution<dim>::vector_value (const Point<dim> &p,
                                  Vector<double>   &values) const
{
  Assert (values.size() == dim,
          ExcDimensionMismatch (values.size(), dim));


  values(0) =20*p[0]*p[0]*p[0] - 60*p[0]*p[1]*p[1];
  values(1) =5*p[0]*p[1]*p[1]*p[1]*p[1] - p[0]*p[0]*p[0]*p[0]*p[0];

}

// EN ESTA PARTE ELEGIMOS EL ELEMENTO FINITO A USAR EN ESTE CASO LINEALES
template <int dim>
MixedLaplaceProblem<dim>::MixedLaplaceProblem ()
              :
              dof_handler (triangulation),
              fe (FE_Q<dim>(1), dim)

{}
//ESTA PARTE MARCA LA TRIANGULACION Y EL REFINAMIENTO
template <int dim>
void MixedLaplaceProblem<dim>::make_grid_and_dofs ()
{
  GridGenerator::hyper_cube (triangulation, 0, 1);
  triangulation.refine_global (5);

std::cout << "   Number of active cells: "
          << triangulation.n_active_cells()
          << std::endl
          << "   Total number of cells: "
          << triangulation.n_cells()
          << std::endl;

  dof_handler.distribute_dofs (fe);

  std::cout << "   Number of degrees of freedom: "
          << dof_handler.n_dofs()
          << std::endl;
}
```

```
template <int dim>
MixedLaplaceProblem<dim>::~MixedLaplaceProblem ()
{
  dof_handler.clear ();
}

template <int dim>
void MixedLaplaceProblem<dim>::setup_system ()
{
  dof_handler.distribute_dofs (fe);
  hanging_node_constraints.clear ();
  DoFTools::make_hanging_node_constraints (dof_handler,
                                   hanging_node_constraints);
  hanging_node_constraints.close ();
  sparsity_pattern.reinit (dof_handler.n_dofs(),
                    dof_handler.n_dofs(),
                    dof_handler.max_couplings_between_dofs());
  DoFTools::make_sparsity_pattern (dof_handler, sparsity_pattern);

  hanging_node_constraints.condense (sparsity_pattern);

  sparsity_pattern.compress();

  system_matrix.reinit (sparsity_pattern);

  solution.reinit (dof_handler.n_dofs());
  system_rhs.reinit (dof_handler.n_dofs());
}

   //  AQUI SE HACE EL ENSAMBLE DEL SISTEMA ES DECIR SE DISCRETIZA EL
SISTEMA DE ECUACIONES DIFERENCIABLES
template <int dim>
void MixedLaplaceProblem<dim>::assemble_system ()
{


  QGauss<dim>  quadrature_formula(2);
  QGauss<dim-1> face_quadrature_formula(2);


  FEValues<dim> fe_values (fe, quadrature_formula,
                    update_values   | update_gradients |
                          update_quadrature_points | update_JxW_values);

 FEFaceValues<dim> fe_face_values (fe, face_quadrature_formula,
                           update_values    | update_normal_vectors |
                           update_q_points  | update_JxW_values);


  const unsigned int   dofs_per_cell = fe.dofs_per_cell;
  const unsigned int   n_q_points    = quadrature_formula.size();
  const unsigned int   n_face_q_points =
face_quadrature_formula.n_quadrature_points;
```

```
  FullMatrix<double>   cell_matrix (dofs_per_cell, dofs_per_cell);
  Vector<double>       cell_rhs (dofs_per_cell);

  std::vector<unsigned int> local_dof_indices (dofs_per_cell);

  RotRightHandSide<dim>      Rotright_hand_side;
  BoundaryValues<dim>    BoundaryValues;
  const ExactSolutionpsi<dim> exact_solutionpsi;
  std::vector<Vector<double> > rhs_values (n_q_points,
                                Vector<double>(dim));

unsigned int comp_i, comp_j;
typename DoFHandler<dim>::active_cell_iterator cell =
dof_handler.begin_active(),
                                     endc = dof_handler.end();
for (; cell!=endc; ++cell)
    {
      cell_matrix = 0;
        cell_rhs = 0;

        fe_values.reinit (cell);
        Rotright_hand_side.vector_value_list
(fe_values.get_quadrature_points(),
                             rhs_values);

  for (unsigned int q=0; q<n_q_points; ++q)
        for (unsigned int i=0; i<dofs_per_cell; ++i)
          {
          comp_i = fe.system_to_component_index(i).first;
            for (unsigned int j=0; j<dofs_per_cell; ++j)
              {
            comp_j = fe.system_to_component_index(j).first;
                  // grad_f_i * grad_f_j
            const double CD = 1;
            if ( comp_i==comp_j )
                  cell_matrix(i,j) +=  CD*fe_values.shape_grad(i,q) *
                                          fe_values.shape_grad(j,q) *
                                                fe_values.JxW(q);


                        // fi * fj
            if ( (comp_i==1) && (comp_j==0) )
                  cell_matrix(i,j) +=  -fe_values.shape_value(i,q) *
                                       fe_values.shape_value(j,q) *
                                       fe_values.JxW(q);


              }
           cell_rhs(i) += fe_values.shape_value(i,q) *
                             rhs_values[q](comp_i) * fe_values.JxW(q);
          }
```

```
//EN ESTA PARTE ESTA DEFINIDA LA DERIVADA NORMAL DE psi
  for (unsigned int face=0;
            face<GeometryInfo<dim>::faces_per_cell;
            ++face)
          if (cell->at_boundary(face))
            {
              fe_face_values.reinit (cell, face);


              for (unsigned int q=0; q<n_face_q_points; ++q)
             {
                 double neumann_value
                   = (exact_solutionpsi.gradient
(fe_face_values.quadrature_point(q)) *
                      fe_face_values.normal_vector(q));



             for (unsigned int i=0; i<dofs_per_cell; ++i) {
                    comp_i = fe.system_to_component_index(i).first;
                if (comp_i==1)
                         cell_rhs(i) += (neumann_value *
                                     fe_face_values.shape_value(i,q) *
                                     fe_face_values.JxW(q));
              }
               }

           }



       cell->get_dof_indices (local_dof_indices);
       for (unsigned int i=0; i<dofs_per_cell; ++i)
        {
          for (unsigned int j=0; j<dofs_per_cell; ++j)
            system_matrix.add (local_dof_indices[i],
                               local_dof_indices[j],
                               cell_matrix(i,j));

          system_rhs(local_dof_indices[i]) += cell_rhs(i);


    }



  hanging_node_constraints.condense (system_matrix);
  hanging_node_constraints.condense (system_rhs);


  std::map<unsigned int,double> boundary_values;
  VectorTools::interpolate_boundary_values (dof_handler,
```

```
                                   0,
                                   BoundaryValues,

boundary_values,std::vector<bool>(false,true));
  MatrixTools::apply_boundary_values (boundary_values,
                                      system_matrix,
                                      solution,
                                      system_rhs);

}

}
//AQUI SE IMPLEMENTA LA SOLUCION
template <int dim>
void MixedLaplaceProblem<dim>::solve ()
{
  SolverControl           solver_control (1000, 1e-8);
  SolverGMRES<>              gmres (solver_control);

  PreconditionJacobi<> preconditioner;
  preconditioner.initialize(system_matrix, 1.0);

  gmres.solve (system_matrix, solution, system_rhs,
          preconditioner);
  std::cout << solver_control.last_step()
              << " Solver iterations to obtain convergence."
              << std::endl;
  hanging_node_constraints.distribute (solution);
}
// ESTA PARTE CALCULA EL ERROR
template <int dim>
void MixedLaplaceProblem<dim>::compute_errors () const
{
  const ComponentSelectFunction<dim>
       w_mask (0, 2);
  const ComponentSelectFunction<dim>
    psi_mask(1, 2);
        ExactSolutionpsi<dim> exact_solutionpsi;
  ExactSolution<dim> exact_solution;
  Vector<double> cellwise_errors (triangulation.n_active_cells());


  QGauss<2>     quadrature(4);

  VectorTools::integrate_difference (dof_handler, solution,
exact_solution,
                                     cellwise_errors, quadrature,
                                      VectorTools::L2_norm,
                                     &w_mask);
  const double w_l2_error = cellwise_errors.l2_norm();

  VectorTools::integrate_difference (dof_handler, solution,
exact_solution,
                                     cellwise_errors, quadrature,
```

```
                                       VectorTools::L2_norm,
                                      &psi_mask);
  const double psi_l2_error = cellwise_errors.l2_norm();

  std::cout << "Errors: ||e_w||_L2 = " << w_l2_error
          << ",   ||e_psi||_L2 = " << psi_l2_error
          << std::endl;

}

// AQUI PEDIMOS EN QUE FORMATO QUEREMOS VER LA SOLUCION APROXIMADA
template <int dim>
void MixedLaplaceProblem<dim>::output_results () const
{
  std::vector<std::string> solution_names;
  switch (dim)
    {
      case 2:
            solution_names.push_back ("w");

            solution_names.push_back ("psi");
            break;


      default:
            Assert (false, ExcNotImplemented());
    }


  DataOut<dim> data_out;

  data_out.attach_dof_handler (dof_handler);
  data_out.add_data_vector (solution, solution_names);

  data_out.build_patches (1);

  std::ofstream output ("solution.gmv");
  data_out.write_gmv (output);
}

// AQUI SE DICE COMO DEBE CORRER EL PROGRAMA
template <int dim>
void MixedLaplaceProblem<dim>::run ()
{
  make_grid_and_dofs();
  setup_system ();
  assemble_system ();
  solve ();
  compute_errors ();
  output_results ();
}


int main ()
```

```
{
  try
    {
      deallog.depth_console (0);

      MixedLaplaceProblem<2> mixed_laplace_problem;
      mixed_laplace_problem.run ();
    }
  catch (std::exception &exc)
    {
      std::cerr << std::endl << std::endl
            << "----------------------------------------------------"
            << std::endl;
      std::cerr << "Exception on processing: " << std::endl
            << exc.what() << std::endl
            << "Aborting!" << std::endl
            << "----------------------------------------------------"
            << std::endl;

      return 1;
    }
  catch (...)
    {
      std::cerr << std::endl << std::endl
            << "----------------------------------------------------"
            << std::endl;
      std::cerr << "Unknown exception!" << std::endl
            << "Aborting!" << std::endl
            << "----------------------------------------------------"
            << std::endl;
      return 1;
    }

  return 0;
}
```