

IMPLEMENTACIÓN DEL MÉTODO DE ELEMENTOS FINITOS EN FENICS PARA PROBLEMAS DE FLUIDOS EN DOMINIOS AXISIMÉTRICOS

Brahiam Steven Zapata Pérez



UNIVERSIDAD DE CÓRDOBA
FACULTAD DE CIENCIAS BÁSICAS
DEPARTAMENTO DE MATEMÁTICAS Y ESTADÍSTICA
MONTERÍA

2022

IMPLEMENTACIÓN DEL MÉTODO DE ELEMENTOS FINITOS EN FENICS PARA PROBLEMAS DE FLUIDOS EN DOMINIOS AXISIMÉTRICOS

Brahiam Steven Zapata Pérez

Trabajo presentado como requisito parcial para optar al título de
Matemático

Asesor:

Ph D. Carlos Alberto Reales Martinez



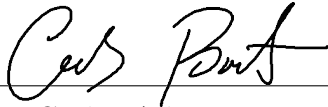
UNIVERSIDAD DE CÓRDOBA
FACULTAD DE CIENCIAS BÁSICAS
DEPARTAMENTO DE MATEMÁTICAS Y ESTADÍSTICA
MONTERÍA
2022

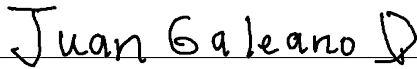
UNIVERSIDAD DE CÓRDOBA
FACULTAD DE CIENCIAS BÁSICAS
DEPARTAMENTO DE MATEMÁTICAS Y ESTADÍSTICA

Los jurados abajo firmantes certifican que han leído y que aprueban el trabajo de grado titulado: **IMPLEMENTACIÓN DEL MÉTODO DE ELEMENTOS FINITOS EN FENICS PARA PROBLEMAS DE FLUIDOS EN DOMINIOS AXISIMÉTRICOS**, el cual es presentado por el estudiante **Brahiam Steven Zapata Pérez**.

Fecha: Marzo de 2022

Asesor: 
Ph D. Carlos Alberto Reales Martinez

Jurado: 
Ph D. Carlos Alberto Banquet Brango

Jurado: 
Ph D. Juan Gabriel Galeano Delgado

*A mi madre,
Mary Angelica Pérez Oviedo*

Resumen

En este trabajo implementamos el Método de Elementos Finitos y su programación en FEniCS para problemas de fluidos en dominios axisimétricos, realizando una formulación del problema de Stokes en simetría cilíndrica. La implementación de esta formulación nos permite presentar un ejemplo con solución conocida donde usamos varios grados polinomiales en los elementos finitos. Luego presentamos un problema acoplado Stokes Darcy, donde mostramos ejemplos con solución conocida y un ejemplo de aplicación. Finalmente presentamos los códigos completos en FEnics.

Abstract

In this work we implement the Finite Element Method and its programming in FEniCS for fluid problems in axisymmetric domains, making a formulation of the Stokes problem in cylindrical symmetry. The implementation of this formulation allows us to present an example with a known solution where we use several polynomial degrees in the finite elements. We then present a coupled Stokes Darcy problem, where we show examples with known solutions and an example application. Finally we present the complete codes in FEnics.

Agradecimientos

Primero que todo, quiero darle gracias a Dios y a todas las personas que me apoyaron e hicieron posible que este trabajo se realice con éxito. En especial a mi director de tesis Dr. Carlos Reales. Sin usted y sus virtudes, su paciencia y constancia este trabajo no lo hubiese logrado tan fácil. Gracias por sus orientaciones. A los docentes del Departamento de Matemáticas, gracias por compartirme sus conocimientos. También quiero expresar mis agradecimientos al profesor Ricardo Ruiz-Baier de la Universidad de Monash en Australia por su invaluable ayuda con los códigos en FEniCs.

A toda mi familia y amigos por acompañarme en este proceso.

Montería, Colombia

Brahiam Steven Zapata Pérez

Marzo de 2022

Índice general

| | |
|---|------------------|
| Resumen | <i>iv</i> |
| Abstract | <i>v</i> |
| 1. Algunas nociones básicas de Phyton y FEniCS | 5 |
| 1.1. Elementos Finitos y Fenics | 6 |
| 1.1.1. Mallas | 6 |
| 1.1.2. Familias de Elementos Finitos | 6 |
| 1.2. Ejemplos sencillos | 8 |
| 1.2.1. El problema de Elasticidad Lineal | 8 |
| 1.2.2. Elasticidad Lineal axisimétrica | 12 |
| 2. Problema de Stokes | 14 |
| 2.1. Planteamiento del Problema | 14 |
| 2.2. Solución por Elementos Finitos | 17 |
| 3. Problema de Stokes-Darcy | 22 |
| 3.1. Planteamiento del Problema | 23 |
| 3.2. Espacios funcionales y formulación débil | 27 |
| 3.3. Aproximación de elementos finitos | 29 |
| 3.4. Ejemplos Numéricos | 31 |
| 3.4.1. Ejemplos con solución conocida | 32 |
| Test 1: Elementos Taylor Hood | 32 |
| Test 2: Mini Elementos | 32 |

| | |
|--|-----------|
| 3.4.2. Ejemplo de aplicación | 34 |
| 4. Códigos | 36 |
| 4.1. Códigos Elasticidad | 36 |
| 4.2. Códigos Stokes | 40 |
| 4.3. Códigos Stokes darcy acoplado | 44 |
| 4.4. Ejemplo de aplicación | 62 |
| Bibliografía | 69 |

Introducción

El método de los elementos finitos es un método numérico muy utilizado para calcular soluciones aproximadas de ecuaciones diferenciales parciales muy complejas. Es comunmente utilizado en diversas ramas de la ingeniería y la ciencia, como la elasticidad, la transferencia de calor, dinámica de fluidos, el electromagnetismo, la acústica, la biomecánica, etc. En este método el dominio de la solución se subdivide en elementos de diferentes formas geométricas simples, tales como triángulos, cuadrados, tetraedros, hexaedros, entre otros, dependiendo del tipo y tamaño del problema y se construye un conjunto de funciones base de tal manera que cada función de base es distinta de cero sólo sobre un número pequeño de elementos, como el número de elementos es limitado, son llamados de elementos finitos palabra que da el nombre al método .

La mayoría de los problemas físicos son formulados de manera natural como problemas de valores en la frontera en dominios espaciales tridimensionales. Pero los cálculos en estos dominios son muy costosos en términos computacionales por el elevado número de incógnitas de estos problemas. Por eso una alternativa inteligente es usar métodos numéricos que aprovechen la simetría axial reduciendo el problema a un dominio computacional bidimensional. En algunos casos, esto se puede hacer luego de asumir que la dependencia de los parámetros, los datos y la solución del problema con respecto a una variable puede ser eliminada. Muchos problemas de la física o la mecánica son axisimétricos, esto debido a la homogeneidad de las propiedades de los materiales y la propiedad de isotropía de las leyes de conservación. Este es el caso de algunos problemas en la metalurgia, donde a pesar de la naturaleza tridimensional de los fenómenos electromagnéticos, se pueden usar algunos modelos axisimétricos

(bidimensionales) que aproximan eficientemente la realidad tridimensional. Las coordenadas cilíndricas son una extensión del sistema de coordenadas polares al espacio tridimensional. Normalmente, en vez de utilizar x , y y z , se usan r , el ángulo ϕ y la variable z que es la que designa la altura máxima de la superficie.

Existen muchos libros de texto que describen los principios del método de análisis de elementos finitos y el amplio alcance de sus aplicaciones para la solución de problemas prácticos de ingeniería y científicos. Pero a pesar del creciente número de referencias estudiando este método numérico, las dedicadas a explicar como se construyen los programas de computadora mediante los cuales se producen realmente los resultados numéricos son pocas. Por lo general se supone que los lectores tienen acceso a programas escritos previamente o se recomienda el uso de paquetes comerciales para obtener los resultados numéricos. En este trabajo pretendemos explicar la programación de algunos ejemplos representativos de la mecánica de fluidos que se puede resolver con el método de elementos finitos. Esto lo haremos usando programas ya escritos para otro tipo de problemas que, aunque parecidos a los nuestros, tienen significativas diferencias.

Python es un lenguaje de programación de alto nivel interpretado, orientado a objetos y con semántica dinámica. Sus estructuras de datos integradas de alto nivel, combinadas con la tipificación dinámica y el enlace dinámico, lo hacen muy atractivo para el desarrollo rápido de aplicaciones, así como para su uso como lenguaje de secuencias de comandos o pegamento para conectar componentes existentes entre sí. La sintaxis simple y fácil de aprender de Python enfatiza la legibilidad y, por lo tanto, reduce el costo de mantenimiento del programa. Python admite módulos y paquetes, lo que fomenta la modularidad del programa y la reutilización del código. El intérprete de Python y la extensa biblioteca estándar están disponibles en formato fuente o binario sin cargo para todas las plataformas principales y se pueden distribuir gratuitamente <https://www.python.org/doc/essays/blurb/>.

Python se usa comúnmente para desarrollar sitios web y software, automatización de tareas, análisis de datos y visualización de datos. Dado que es relativamente fácil de aprender, Python ha sido adoptado por muchos no programadores, como contado-

res y científicos, para una variedad de tareas cotidianas, como organizar las finanzas. De los cientos de lenguajes de programación que existen, Python sigue siendo una opción popular entre numerosas empresas y organizaciones. Algunos nombres familiares que usan Python incluyen Google, Facebook, Venmo, Spotify, Netflix y Dropbox <https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python>.

El proyecto FEniCS es un proyecto de investigación y software destinado a crear métodos matemáticos y software para resolver ecuaciones diferenciales parciales utilizando elementos finitos. Esto incluye la creación de software intuitivo, eficiente y flexible. El proyecto se inició en 2003 y se desarrolla en colaboración entre investigadores de varias universidades e institutos de investigación de todo el mundo. Para obtener las últimas actualizaciones y más información sobre el proyecto FEniCS, visite la página web de FEniCS.

FEniCS es una plataforma informática popular de código abierto para resolver ecuaciones diferenciales parciales (PDE). FEniCS permite a los usuarios traducir rápidamente modelos científicos en código eficiente de elementos finitos. Con las interfaces Python y C++ de alto nivel para FEniCS, es fácil comenzar, pero FEniCS también ofrece capacidades potentes para programadores más experimentados. FEniCS se ejecuta en una multitud de plataformas, desde computadoras portátiles hasta clústeres de alto rendimiento.

Este trabajo está conformado por cuatro capítulos: el primero es una pequeña introducción que recoge algunos aspectos importantes del método de elementos finitos y su programación en FEniCS. En el siguiente capítulo presentamos una formulación del problema de Stokes en simetría cilíndrica. La implementación de esta formulación nos permite presentar un ejemplo con solución conocida donde usamos varios grados polinomiales en los elementos finitos. El Capítulo 3 presentamos un problema acoplado Stokes Darcy. Se presentan ejemplos con solución conocida y un ejemplo de aplicación. Finalmente presentamos un capítulo con los códigos completos en FEnics.

Capítulo 1

Algunas nociones básicas de Phyton y FEniCS

Una de las cualidades mas importante de FEniCS es la facilidad con la que se pueden crear solucionadores de Elementos Finitos describiendo la ecuación diferencial parcial utilizando formas débiles en notación casi matemática. El software FEniCS, además de una amplia documentación y ejemplos, se pueden encontrar en el sitio web del Proyecto FEniCS, <http://fenicsproject.org/>.

En particular, los comentarios (líneas que comienzan con `#`) y las funciones (palabra clave `def`, ver definido por el usuario a continuación) son útiles en la definición de un formulario. Sin embargo, suele ser una buena idea evitar el uso de funciones avanzadas de Python en la definición del formulario, para mantenerse cerca de la notación matemática.

La biblioteca `multiphenics` de Python tiene como objetivo proporcionar herramientas en FEniCS para una fácil creación de prototipos de problemas multifísicos en mallas conformes. En particular, facilita la definición de variables restringidas de subdominio/límite y permite la definición del problema por medio de una estructura de bloques. Esta biblioteca va a ser fundamental en el ultimo capítulo donde la usaremos para implementar la formulación del problema de Stokes Darcy.

1.1. Elementos Finitos y Fenics

En lo que sigue de esta sección trataremos de resaltar los aspectos más importantes de un código de elementos finitos vistos desde Fenics.

1.1.1. Mallas

Lo primero que debemos definir es el tipo de elementos geométricos que vamos a usar. Para eso debemos tener claro la geometría del elemento y la dimensión en la que vamos a trabajar. Fenics permite que usemos intervalos triangulos, cuadrilateros, tetrahedros y hexahedros. A continuación presentamos un ejemplo para crear una malla sencilla para $\Omega = [0, 1] \times [0, 1]$:

```
from dolfin import *
import matplotlib.pyplot as plt
mesh = UnitSquareMesh(10, 10)
plt.figure()
plot(mesh, linewidth=0.2, title="Cuadrado Unitario")
plt.show()
```

Si queremos un cuadrado más general usamos

```
mesh = RectangleMesh(Point(0., 0.), Point(L, H), Nx, Ny)
```

1.1.2. Familias de Elementos Finitos

Antes de definir formas bilineales, es necesario describir los espacios de elementos finitos sobre los que tiene lugar la integración. Veamos algunas familias de elementos finitos más usadas en Fenics. Antes de definir los elementos finitos que usaremos debemos definir la familia de elementos que usaremos: `family`. Veamos los valores posibles:

- "Lagrange" or "CG", representan los elementos finitos escalares estandar Lagrange (funciones continuas polinomiales a trozos);

- "Discontinuous Lagrange" or "DG", representan los elementos finitos escalares discontinuos Lagrange (funciones discontinuas polinomiales a trozos);
- "Crouzeix-Raviart" or "CR", representan los elementos escalares de Crouzeix-Raviart;
- "Brezzi-Douglas-Marini" or "BDM", representa vector-valued Brezzi-Douglas-Marini $H(\text{div})$ elements;
- "Brezzi-Douglas-Fortin-Marini" or "BDFM", representa vector-valued Brezzi-Douglas-Fortin-Marini $H(\text{div})$ elements;
- "Raviart-Thomas" or "RT", representa vector-valued Raviart-Thomas $H(\text{div})$ elements.
- "Nedelec 1st kind $H(\text{div})$ " or "N1div", representa vector-valued Nedelec $H(\text{div})$ elements (of the first kind).
- "Nedelec 2st kind $H(\text{div})$ " or "N2div", representa vector-valued Nedelec $H(\text{div})$ elements (of the second kind).
- "Nedelec 1st kind $H(\text{curl})$ " or "N1curl", representa vector-valued Nedelec $H(\text{curl})$ elements (of the first kind).
- "Nedelec 2st kind $H(\text{curl})$ " or "N2curl", representa vector-valued Nedelec $H(\text{curl})$ elements (of the second kind).
- "Bubble", representa bubble elements, useful for example to build the mini elements.

El parámetro `FiniteElement`, representa un elemento finito de alguna familia en una celda dada con un cierto grado de polinomio.

Para ver como usar las anteriores familias usamos una notación como la que sigue:

```
P2v = VectorFunctionSpace(mesh, "CG", 2)
RT0 = FunctionSpace(mesh, "RT", 1)
```


1.2. Ejemplos sencillos

1.2.1. El problema de Elasticidad Lineal

Sea $\Omega \subset \mathbb{R}^2$ un dominio conexo acotado con frontera suave. Sea \mathbf{n} el vector normal a $\partial\Omega$. Asumamos que tenemos condiciones Dirichlet en algún subconjunto Γ_D de la frontera, mientras que en un subconjunto Γ_N tenemos condiciones de frontera Neumann, donde $\partial\Omega = \bar{\Gamma}_N \cup \bar{\Gamma}_D$ y $\Gamma_N \cap \Gamma_D = \emptyset$. El problema básico de elasticidad lineal consiste en:

Problema 1.2.1 (Problema Fuerte). Hallar el tensor de estrés $\boldsymbol{\sigma} = (\sigma_{ij}) \in \mathbb{R}^{2 \times 2}$ y el desplazamiento $\mathbf{u} = (u_1, u_2)$ tal que

$$\left\{ \begin{array}{ll} -\nabla \cdot \boldsymbol{\sigma} = \mathbf{f} & \text{en } \Omega, \\ \boldsymbol{\sigma} = 2\mu\varepsilon(\mathbf{u}) + \lambda(\nabla \cdot \mathbf{u}) \mathbf{I}_2 & \text{en } \Omega, \\ \mathbf{u} = 0 & \text{en } \Gamma_D, \\ \boldsymbol{\sigma} \cdot \mathbf{n} = \mathbf{g}_N & \text{en } \Gamma_N. \end{array} \right.$$

Siendo $\mathbf{f} = (f_1, f_2) \in [L^2(\Omega)]^2$ la fuerza corporal, $\mathbf{g}_N = (g_1, g_2) \in [L(\Gamma_N)]^2$ es una función dada (carga de tracción), $\varepsilon(\mathbf{u})$ es denominado tensor de deformaciones y está dado por

$$\varepsilon(\mathbf{u}) = \frac{1}{2} \left((\nabla \mathbf{u}) + (\nabla \mathbf{u})^T \right) = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad i, j = 1, 2.$$

\mathbf{I}_2 es la matriz identidad de orden 2, los parámetros μ y λ son llamados parámetros de Lamé.

Para obtener la formulación débil del Problema 1.2.1 consideremos el espacio de Hilbert

$$\mathcal{V} = \left\{ \mathbf{v} \in [H^1(\Omega)]^2 : \mathbf{v}|_{\Gamma_D} = 0 \right\}.$$

Multiplicando $\mathbf{f} = -\nabla \cdot \boldsymbol{\sigma}$ por una función prueba $\mathbf{v} \in \mathcal{V}$ e integrando por partes

tenemos que

$$\begin{aligned}
(\mathbf{f}, \mathbf{v})_{L^2(\Omega)} &= (-\nabla \cdot \boldsymbol{\sigma}, \mathbf{v})_{L^2(\Omega)} \\
&= \sum_{i=1}^2 \sum_{j=1}^2 \left\langle -\frac{\partial \sigma_{ij}}{\partial x_j}, v_i \right\rangle_{L^2(\Omega)} \\
&= \sum_{i=1}^2 \sum_{j=1}^2 \left[-(\sigma_{ij}, n_j v_i)_{L^2(\partial\Omega)} + \left\langle \sigma_{ij}, \frac{\partial v_i}{\partial x_j} \right\rangle_{L^2(\Omega)} \right].
\end{aligned}$$

Así,

$$-(\boldsymbol{\sigma} \cdot \mathbf{n}, \mathbf{v})_{[L^2(\partial\Omega)]^2} + (\boldsymbol{\sigma} : \nabla \mathbf{v})_{L^2(\Omega)} = (\mathbf{f}, \mathbf{v})_{[L^2(\Omega)]^2}, \quad (1.1)$$

donde

$$\begin{aligned}
(\boldsymbol{\xi}, \boldsymbol{\zeta})_{L^2(\Omega)} &= \int_{\Omega} \boldsymbol{\xi} \cdot \boldsymbol{\zeta} \, dx, \quad \text{para todo } \boldsymbol{\xi}, \boldsymbol{\zeta} \in [L^2(\Omega)]^2, \\
\mathbf{A} : \mathbf{B} &= \sum_{i=1}^2 \sum_{j=1}^2 a_{ij} b_{ij} \quad \text{y} \quad (\mathbf{A} : \mathbf{B})_{L^2(\Omega)} = \int_{\Omega} \mathbf{A} : \mathbf{B} \, dx.
\end{aligned}$$

para todo par de matrices $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{2 \times 2}$ y $\mathbf{B} = (b_{ij}) \in \mathbb{R}^{2 \times 2}$. Usando las condiciones de frontera $\boldsymbol{\sigma} \cdot \mathbf{n} = \mathbf{g}_N$ en Γ_N y $\mathbf{v} = 0$ en Γ_D , la ecuación (1.1) se transforma en

$$(\boldsymbol{\sigma} : \nabla \mathbf{v})_{L^2(\Omega)} = (\mathbf{f}, \mathbf{v})_{L^2(\Omega)} + (\mathbf{g}_N, \mathbf{v})_{L^2(\Gamma_N)}. \quad (1.2)$$

Ahora bien, recordemos que cualquier matriz se puede descomponer en la suma de su parte simétrica y antisimétrica, esto es, si $\mathbf{A} \in \mathbb{R}^{2 \times 2}$, entonces

$$\mathbf{A} = (\mathbf{A} + \mathbf{A}^T)/2 + (\mathbf{A} - \mathbf{A}^T)/2,$$

de modo que

$$\boldsymbol{\sigma} : \nabla \mathbf{v} = \boldsymbol{\sigma} : \frac{1}{2}(\nabla \mathbf{v} + (\nabla \mathbf{v})^T) + \boldsymbol{\sigma} : \frac{1}{2}(\nabla \mathbf{v} - (\nabla \mathbf{v})^T) = \boldsymbol{\sigma} : \varepsilon(\mathbf{v}) + 0 = \boldsymbol{\sigma} : \varepsilon(\mathbf{v}).$$

Entonces, al reemplazar $\nabla \mathbf{v}$ por $\varepsilon(\mathbf{v})$ en la ecuación (1.2) obtenemos

$$(\boldsymbol{\sigma}(\mathbf{u}) : \varepsilon(\mathbf{v}))_{L^2(\Omega)} = (\mathbf{f}, \mathbf{v})_{L^2(\Omega)} + (\mathbf{g}_N, \mathbf{v})_{L^2(\Gamma_N)} \quad \text{para todo } \mathbf{v} \in \mathcal{V}.$$

Insertando la Ley de Hooke y usando el hecho de que $\mathbf{I}_2 : \varepsilon(\mathbf{v}) = \nabla \cdot \mathbf{v}$ se sigue que

$$2\mu(\varepsilon(\mathbf{u}) : \varepsilon(\mathbf{v}))_{L^2(\Omega)} + \lambda(\nabla \cdot \mathbf{u}, \nabla \cdot \mathbf{v})_{L^2(\Omega)} = (\mathbf{f}, \mathbf{v})_{L^2(\Omega)} + (\mathbf{g}_N, \mathbf{v})_{L^2(\Gamma_N)}, \quad \text{para todo } \mathbf{v} \in \mathcal{V}.$$

Por lo tanto, la formulación débil del Problema 1.2.1 es la siguiente:

Problema 1.2.2 (Problema Débil). Hallar $\mathbf{u} \in \mathcal{V}$ tal que $a(\mathbf{u}, \mathbf{v}) = \ell(\mathbf{v})$, para todo $\mathbf{v} \in \mathcal{V}$. donde $a(\mathbf{u}, \mathbf{v}) = 2\mu(\varepsilon(\mathbf{u}) : \varepsilon(\mathbf{v}))_{L^2(\Omega)} + \lambda(\nabla \cdot \mathbf{u}, \nabla \cdot \mathbf{v})_{L^2(\Omega)}$ y $\ell(\mathbf{v}) = (\mathbf{f}, \mathbf{v})_{L^2(\Omega)} + (\mathbf{g}_N, \mathbf{v})_{L^2(\Gamma_N)}$.

```

1  from dolfin import *
2  import matplotlib.pyplot as plt
3  # Programa para el problema de Elasticidad Lineal 2D
4  # Parametros Fisicos
5  E=10.0;nu=0.3;rho_g = 1e-3
6  mu=E/(2.0*(1.0 + nu))
7  mylambda= E*nu/((1.0 + nu)*(1.0 - 2.0*nu))
8  # Definicion de los operadores diferenciales
9  def eps (v) :
10     return sym ( grad (v) )
11  def sigma (v) :
12     dim = v. geometric_dimension ()
13     return 2.0* mu* eps (v) + mylambda *tr(eps (v) ) * Identity (dim )
14  # Creacion d ela malla
15  mesh = RectangleMesh ( Point (0.,0.) ,Point (10.,1.) ,100 , 10)
16  # Vector de carga(mismo peso)
17  f = Constant ((0. , -rho_g) )
18  # Definicion de los espacios funcionales
19  V = VectorFunctionSpace (mesh , 'CG', degree =2)
20  # Definicion de la frontera y d elas condiciones de frontera
21  def left (x, on_boundary ) :
22     return near (x[0] ,0.) and on_boundary
23  bc = DirichletBC (V, Constant ((0. ,0.) ) , left )
24  # Definicion Problema Variacional
25  u = TrialFunction (V)
26  v = TestFunction (V)
27  a = inner(sigma(u) , eps(v)) *dx

```

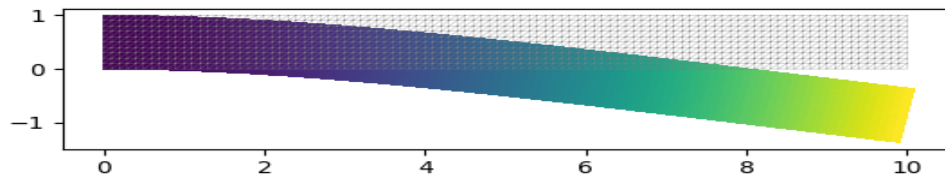


Figura 1.1: Desplazamiento de una barra.

```

28 L = dot (f, v) *dx
29 u = Function (V)
30 solve (a == L, u, bc)
31 # Visualizacion
32 plt.figure()
33 plot(mesh, linewidth=0.2)
34 plot(u, mode="displacement")
35 plt.show()

```

Resaltemos las similitudes que se pueden observar entre las expresiones matemáticas

$$\varepsilon(\mathbf{u}) = \frac{1}{2} \left((\nabla \mathbf{u}) + (\nabla \mathbf{u})^T \right), \quad \boldsymbol{\sigma} = 2\mu \varepsilon(\mathbf{u}) + \lambda (\nabla \cdot \mathbf{u}) \mathbf{I}_2$$

$$\int_{\Omega} \boldsymbol{\sigma}(\mathbf{u}) : \varepsilon(\mathbf{v}) d\Omega = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} d\Omega$$

y las expresiones del código

```

def eps (v) :
    return sym ( grad (v) )
def sigma (v) :
    dim = v. geometric_dimension ()
    return 2.0* mu* eps (v) + mylambda *tr(eps (v) ) * Identity (dim )
a = inner(sigma(u) , eps(v)) *dx

```

Otro aspecto a resaltar es la facilidad con que se define la frontera y las condiciones de frontera. En este caso consideramos que la barra esta empotrada en la parte izquierda por lo que el desplazamiento debe ser cero en ambas componentes:

```
def left (x, on_boundary ) :
    return near (x[0] ,0.) and on_boundary
bc = DirichletBC (V, Constant ((0. ,0.) ) , left )
```

1.2.2. Elasticidad Lineal axisimétrica

Para el caso axisimétrico, el desplazamiento tiene la forma:

$$\mathbf{u} = u_r(r, z)\mathbf{e}_r + u_z(r, z)\mathbf{e}_z \quad (1.3)$$

Podemos seguir trabajando con un espacio funcional similar al caso anterior, (VectorFunctionSpace of dimension 2.) pero las componentes del tensor de stress toma la forma:

$$\boldsymbol{\varepsilon}(\mathbf{u}) = \begin{bmatrix} \partial_r u_r & 0 & (\partial_z u_r + \partial_r u_z)/2 \\ 0 & u_r/r & 0 \\ (\partial_z u_r + \partial_r u_z)/2 & 0 & \partial_z u_z \end{bmatrix}_{(\mathbf{e}_r, \mathbf{e}_\theta, \mathbf{e}_z)} \quad (1.4)$$

El resto de la formulación es similar al caso elástico 2-D con una pequeña diferencia en la medida de integración. De hecho, el principio del trabajo virtual se lee como:

$$\text{Encontrar } \mathbf{u} \in V \text{ s.t. } \int_{\Omega} \boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{v}) d\Omega = \int_{\partial\Omega_T} \mathbf{T} \cdot \mathbf{v} dS \quad \forall \mathbf{v} \in V \quad (1.5)$$

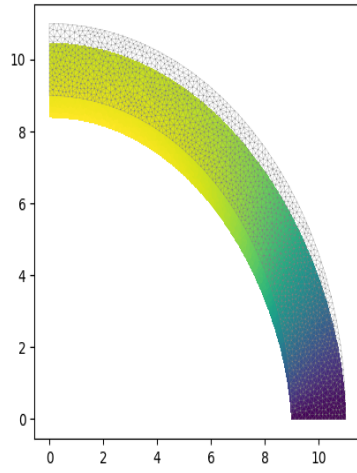
donde \mathbf{T} es la imposición de la tracción en alguna parte $\partial\Omega_T$.

Encontrar $\mathbf{u} \in V$ s.t.

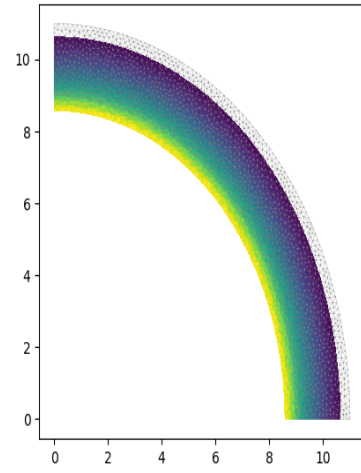
$$\int_{\omega} \boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{v}) r d\omega = \int_{\partial\omega_T} \mathbf{T} \cdot \mathbf{v} r ds \quad \forall \mathbf{v} \in V \quad (1.6)$$

En este caso se deben hacer algunos cambios entre los cuales se encuentra la nueva definición de $\boldsymbol{\varepsilon}(\mathbf{u})$ que en este caso se escribirá como

```
def eps(v):
    return sym(as_tensor([[v[0].dx(0), 0, v[0].dx(1)],
```



(a) Placa empotrada en el extremo derecho.



(b) Placa apoyada en sus dos extremos.

```
[0, v[0]/x[0], 0],
[v[1].dx(0), 0, v[1].dx(1)]]))
```

El código completo se puede encontrar en el Capítulo 4.

Capítulo 2

Problema de Stokes

2.1. Planteamiento del Problema

Estamos interesados en modelar un flujo a través de un dominio $\hat{\Omega}$ simétrico con respecto al eje z . Usaremos coordenadas cilíndricas (r, θ, z) y notamos Ω la mitad de la sección $(r, 0, z)$. En la frontera $\hat{\Gamma}$ del dominio físico $\hat{\Omega}$ imponemos una condición de frontera de Dirichlet. Sea Γ la sección media de $\hat{\Gamma}$ y Γ_0 la intersección de $\hat{\Omega}$ con los ejes, tal que $\partial\Omega$ es la unión de Γ y Γ_0 . Todos los campos vectoriales en $\hat{\Omega}$ son expresados en coordenadas cilíndricas. El flujo está modelado por las ecuaciones de Stokes en el dominio $\hat{\Omega}$ y suponemos que la condición de frontera y las fuerzas externas son axisimétricas y que su componente angular es cero. Una función axisimétrica \hat{p} en $\hat{\Omega}$ depende solo del radio y las coordenadas axiales, por tanto asociamos una función p en Ω tal que $p(r, z) = \hat{p}(r, 0, z)$. Un campo vectorial axisimétrico \hat{u} depende de (r, z) . Para cada campo vectorial \hat{u} nosotros denotamos por $\hat{u}_r, \hat{u}_\theta, \hat{u}_z$ su radio, ángulo y componente axial respectivamente. Si tiene un componente angular cero ($\hat{u}_\theta = 0$) asociamos un campo vectorial $u = (u_r, u_z)$ en Ω tal que $u_r = \hat{u}_r$ y $u_z = \hat{u}_z$.

Suponga que el dominio axisimétrico $\hat{\Omega}$ está acotado, tiene continuidad de Lipschitz, Γ_0 es una unión finita de segmentos de longitud positiva y los datos son axisimétricos con componente angular cero. El problema de Stokes tridimensional homogéneo estacionario está dado por

$$\begin{cases} -v\Delta\hat{u} + \nabla\hat{p} &= \hat{f} \text{ en } \hat{\Omega} \\ \operatorname{div}\hat{u} &= 0 \text{ en } \hat{\Omega} \\ \hat{u} &= 0 \text{ en } \widehat{\partial\Omega} \end{cases} \quad (2.1)$$

Por simplicidad escogemos datos con frontera cero, sin embargo, el análisis posterior se extiende sin dificultad a los datos de frontera axisimétrico \hat{g} con componente angular cero y flujo cero en $\partial\hat{\Omega}$. La ecuación diferencial (2,1) se escribe en forma débil como:
Encontrar (\hat{u}, \hat{p}) en $H_0^1(\hat{\Omega})^3 \times L_0^2(\hat{\Omega})$ tal que para cada (\hat{v}, \hat{q}) en $H_0^1(\hat{\Omega})^3 \times L_0^2(\hat{\Omega})$

$$\begin{cases} \hat{a}(\hat{u}, \hat{v}) + \hat{b}(\hat{v}, \hat{p}) &= \int_{\Omega} \hat{f} \cdot \hat{v} d\hat{x} \\ \hat{b}(\hat{u}, \hat{q}) &= 0 \end{cases} \quad (2.2)$$

donde la forma bilineal \hat{a} y \hat{b} están definidas como

$$\hat{a}(\hat{u}, \hat{v}) = v \int_{\Omega} (\nabla\hat{u} : \nabla\hat{v}) d\hat{x}$$

y

$$\hat{b}(\hat{u}, \hat{q}) = - \int_{\hat{\Omega}} (\operatorname{div}\hat{u}) \hat{q} d\hat{x},$$

$H_0^1(\hat{\Omega})$ representa el espacio de funciones en $H^1(\hat{\Omega})$ con trazo cero y $L_0^2(\hat{\Omega})$ el espacio de funciones en $L^2(\hat{\Omega})$ con integral igual a cero.

Se puede verificar por el Lema de Lax-Milgram que este problema tiene una única solución axisimétrica y se puede dividir en dos problemas separados en Ω , una por el componente angular \hat{u}_θ y el otro por $(\hat{u}_r, \hat{u}_z, p)$. Si los datos no tienen rotación como se supone, es decir, el componente angular \hat{f}_θ es igual a cero, entonces \hat{u}_θ es igual a cero. Por esta razón solo nos ocuparemos del siguiente problema equivalente a (2.2):

Denotemos por $L_r^2(\Omega)$ el espacio de Lebesgue ponderado de todas las funciones medibles u definidas en Ω para las cuales

$$\|u\|_{L_r^2(\Omega)} := \int_{\Omega} |u|^2 r dr dz < \infty.$$

El espacio de Sobolev ponderado $H_r^k(\Omega)$ consiste en todas las funciones en $L_r^2(\Omega)$ cuyas derivadas hasta el orden k están en $L_r^2(\Omega)$. También definimos las normas y las

seminormas en la forma usual; en particular,

$$|u|_{\mathbf{H}_r^1(\Omega)}^2 := \int_{\Omega} \left(|\partial_r u|^2 + |\partial_z u|^2 \right) r \, dr dz,$$

Sea $\tilde{\mathbf{H}}_r^1(\Omega) := \mathbf{H}_r^1(\Omega) \cap \mathbf{L}_{1/r}^2(\Omega)$, donde $\mathbf{L}_{1/r}^2(\Omega)$ denota el conjunto de todas las funciones medibles u definidas en Ω para las cuales

$$\|u\|_{\mathbf{L}_{1/r}^2(\Omega)}^2 := \int_{\Omega} \frac{|u|^2}{r} \, dr dz < \infty.$$

$\tilde{\mathbf{H}}_r^1(\Omega)$ es un espacio de Hilbert con la norma

$$\|u\|_{\tilde{\mathbf{H}}_r^1(\Omega)} := \left(\|u\|_{\mathbf{H}_r^1(\Omega)}^2 + \|u\|_{\mathbf{L}_{1/r}^2(\Omega)}^2 \right)^{1/2}.$$

A partir de estos espacios funcionales definamos \mathcal{V} y \mathcal{Q} de la siguiente forma

$$\mathcal{V} = \{ \mathbf{v} = (v_r, v_z) \in \tilde{\mathbf{H}}_r^1(\Omega) \times \mathbf{H}_r^1(\Omega), u = 0, \quad v = 0 \quad \text{en } \partial\Omega \}$$

$$\mathcal{Q} = \left\{ q \in \mathbf{L}_r^2(\Omega) : \int_{\Omega} q r \, dr dz = 0 \right\}$$

En esta primera sección estamos interesados en un problema de Stokes axisimétrico abordado en [1]. El problema consiste en encontrar (u, p) en $\mathcal{V} \times \mathcal{Q}$ tal que, para cada (v, q) en $\mathcal{V} \times \mathcal{Q}$

$$\begin{cases} a(u, v) + b(v, p) = \int_{\Omega} f \cdot v r dx, \\ b(u, q) = 0. \end{cases} \quad (2.3)$$

donde las formas a y b están definidas por:

$$a(u, v) = v \int_{\Omega} (\nabla u : \nabla v) r dx + v \int_{\Omega} u_r v_r \frac{1}{r} dx$$

y

$$b(u, q) = - \int_{\Omega} (\operatorname{div}_a u) q r dx - \int_{\Omega} u_r q dx,$$

tal que

$$\nabla u = \begin{pmatrix} \partial_r u_r & \partial_r u_z \\ \partial_z u_r & \partial_z u_z \end{pmatrix}$$

y $\text{div } u = \partial_r u_r + \partial_z u_z$.

Para tener una notación mas compacta podemos redefinir los operadores de la siguiente forma:

$$\nabla_{\text{axi}} \mathbf{u} = \begin{bmatrix} \partial_r u_r & 0 & \partial_r u_z \\ 0 & u_r/r & 0 \\ \partial_z u_r & 0 & \partial_z u_z \end{bmatrix}$$

y $\text{div}_a \mathbf{u} = \partial_r u_r + u_r/r \partial_z u_z$, Encontrar (u, p) en $\mathcal{V} \times \mathcal{Q}$ tal que, para cada (v, q) en $\mathcal{V} \times \mathcal{Q}$

$$\begin{cases} a(\mathbf{u}, \mathbf{v}) + b(\mathbf{v}, p) = \int_{\Omega} \mathbf{v} \mathbf{r} \, d\mathbf{r} dz, \\ b(\mathbf{u}, q) = 0. \end{cases} \quad (2.4)$$

Donde las formas a y b están definidas por:

$$a(\mathbf{u}, \mathbf{v}) = \int_{\Omega} (\nabla_a \mathbf{u} : \nabla_a \mathbf{v}) r \, dr dz$$

y

$$b(\mathbf{u}, q) = - \int_{\Omega} \text{div}_a \mathbf{v} q r \, dr dz.$$

2.2. Solución por Elementos Finitos

Sea $\{\mathcal{T}_h\}_{h>0}$ una familia regular de triangulaciones Ω con h como tamaño de malla. Sea $\mathbb{P}_k(T)$ el conjunto de restricciones a T de polinomios de grado menor o igual que k .

En la referencia [1] usan elementos \mathbb{P}_1 en la triangulación $\{\mathcal{T}_h\}$ para aproximar la presión. Para aproximar la velocidad usan tambien elementos \mathbb{P}_1 pero en una nueva triangulación: $\{\mathcal{T}_{h/2}\}$ obtenida a partir de $\{\mathcal{T}_h\}$ dividiendo cada triángulo en cuatro triángulos iguales uniendo los puntos medios de las aristas. Sea

$$\mathcal{V}_{h/2} := \left\{ \mathbf{v}_h \in \mathcal{C}^0(\Omega)^2 : \mathbf{v}_h|_{\partial\Omega} = \mathbf{0} \, v_{r,h}|_{r=0} = 0. \, \mathbf{u}_h|_T \in [\mathbb{P}_1]^2 \, \forall T \in \mathcal{T}_{h/2} \right\},$$

$$\mathcal{Q}_h := \left\{ q_h \in \mathcal{C}^0(\Omega) : \int_{\Omega} q_h r = 0 \, q_h|_T \in \mathbb{P}_1 \, \forall T \in \mathcal{T}_h \right\},$$

En este trabajo no usaremos ese enfoque y en su lugar usaremos algo equivalente. Usaremos \mathbb{P}_2 para aproximar cada una de las componentes de la velocidad en la misma triangulación $\{\mathcal{T}_h\}$. Esto es, usaremos los siguientes espacios:

$$\mathcal{V}_h := \left\{ \mathbf{v}_h \in \mathcal{C}^0(\Omega)^2 : \mathbf{v}_h|_{\partial\Omega} = \mathbf{0} \ v_{r,h}|_{r=0} = 0. \ \mathbf{u}_h|_T \in [\mathbb{P}_2]^2 \ \forall T \in \mathcal{T}_{h/2} \right\},$$

$$\mathcal{Q}_h := \left\{ q_h \in \mathcal{C}^0(\Omega) : \int_{\Omega} q_h r = 0 \ q_h|_T \in \mathbb{P}_1 \ \forall T \in \mathcal{T}_h \right\},$$

El problema discreto se lee: Encontrar (\mathbf{u}_h, p_h) en $\mathcal{V}_h \times \mathcal{Q}_h$ tal que, para cada (\mathbf{v}_h, q_h) en $\mathcal{V}_h \times \mathcal{Q}_h$

$$\begin{cases} a(\mathbf{u}_h, \mathbf{v}_h) + b(\mathbf{v}_h, p_h) &= \int_{\Omega} \mathbf{v}_h \mathbf{r} \, d\mathbf{r} dz, \\ b(\mathbf{u}_h, q_h) &= 0. \end{cases} \quad (2.5)$$

La condición

$$\int_{\Omega} q_h r \, d\mathbf{r} dz = 0$$

la impondremos de manera débil en la formulación a través de un multiplicador de Lagrange: Encontrar (\mathbf{u}, p, λ) en $\mathcal{V} \times \mathcal{Q} \times \mathbb{R}$ tal que, para cada (\mathbf{v}, q, μ) en $\mathcal{V} \times \mathcal{Q}$

$$\begin{cases} a(\mathbf{u}, \mathbf{v}) + b(\mathbf{v}, p) &= \int_{\Omega} f \cdot \mathbf{v} r \, d\mathbf{r} dz, \\ b(\mathbf{u}, q) + \int_{\Omega} \lambda q r \, d\mathbf{r} dz &= 0. \\ \int_{\Omega} \mu p r \, d\mathbf{r} dz &= 0 \end{cases} \quad (2.6)$$

Para comprobar que nuestro código está correcto, consideramos un problema con lado derecho construido a partir de una solución conocida:

$$\mathbf{u}(r, z) = (u_r(r, z), u_z(r, z)) = (r^3(r-1)z(3z-4), -r^2(5r-4)(z^2(z-2)))$$

$$p(r, z) = r^2 + z^2 - \frac{10}{12}$$

Cuadro 2.1: Test 1 Errores experimentales y tasas de convergencia usando $[\mathbb{P}_2]^2 - \mathbb{P}_1$

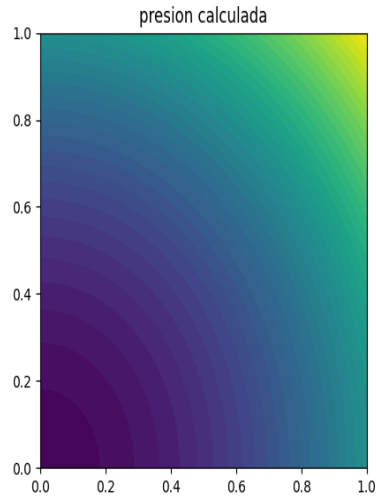
| DoFs | h | $\ u - u_h\ _{\tilde{H}_r^1(\Omega) \times \tilde{H}_r^1(\Omega)}$ | r_1 | $\ p - p_h\ _{L_1^2(\Omega)}$ | r_1 |
|-------|--------|--|-------|-------------------------------|-------|
| 106 | 0.5590 | 0.2235 | 0.000 | 0.05063 | 0.000 |
| 352 | 0.2795 | 0.05944 | 1.911 | 0.00811 | 2.642 |
| 1276 | 0.1398 | 0.0151 | 1.977 | 0.001338 | 2.600 |
| 4852 | 0.0699 | 0.003791 | 1.994 | 0.0002522 | 2.407 |
| 18916 | 0.0349 | 0.0009487 | 1.998 | 5.585e-05 | 2.175 |

Cuadro 2.2: Test 1 Errores experimentales y tasas de convergencia $[\mathbb{P}_3]^2 - \mathbb{P}_2$.

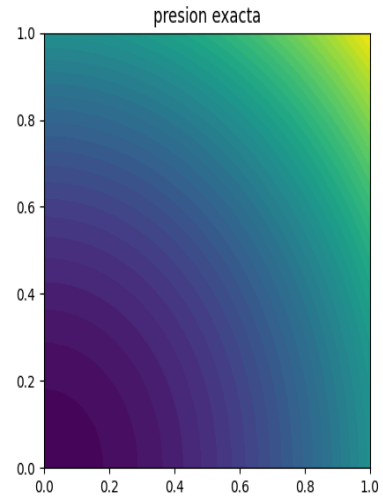
| DoFs | h | $\ u - u_h\ _{\tilde{H}_r^1(\Omega) \times \tilde{H}_r^1(\Omega)}$ | r_1 | $\ p - p_h\ _{L_1^2(\Omega)}$ | r_1 |
|-------|--------|--|-------|-------------------------------|-------|
| 228 | 0.5590 | 0.02187 | 0.000 | 0.01447 | 0.000 |
| 804 | 0.2795 | 0.00275 | 2.991 | 0.001185 | 3.610 |
| 3012 | 0.1398 | 0.0003407 | 3.013 | 9.777e-05 | 3.599 |
| 11652 | 0.0699 | 4.227e-05 | 3.011 | 8.217e-06 | 3.573 |
| 45828 | 0.0349 | 5.26e-06 | 3.007 | 7.04e-07 | 3.545 |

Cuadro 2.3: Test 1 Errores experimentales y tasas de convergencia $[\mathbb{P}_4]^2 - \mathbb{P}_3$.

| DoFs | h | $\ u - u_h\ _{\tilde{H}_r^1(\Omega) \times \tilde{H}_r^1(\Omega)}$ | r_1 | $\ p - p_h\ _{L_1^2(\Omega)}$ | r_1 |
|-------|--------|--|-------|-------------------------------|-------|
| 398 | 0.5590 | 0.001422 | 0.000 | 0.001503 | 0.000 |
| 1448 | 0.2795 | 9.139e-05 | 3.960 | 6.954e-05 | 4.434 |
| 5516 | 0.1398 | 5.684e-06 | 4.007 | 3.284e-06 | 4.404 |
| 21524 | 0.0699 | 3.523e-07 | 4.012 | 1.651e-07 | 4.314 |
| 85028 | 0.0349 | 2.189e-08 | 4.009 | 8.899e-09 | 4.213 |

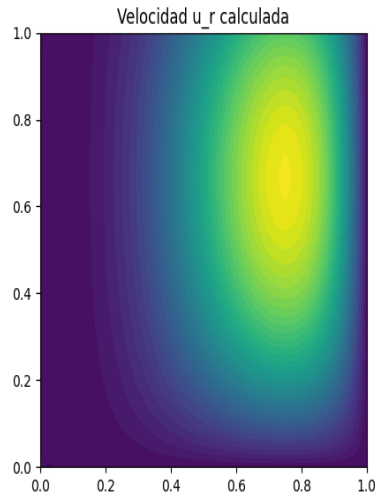


(a) Presión Calculada.

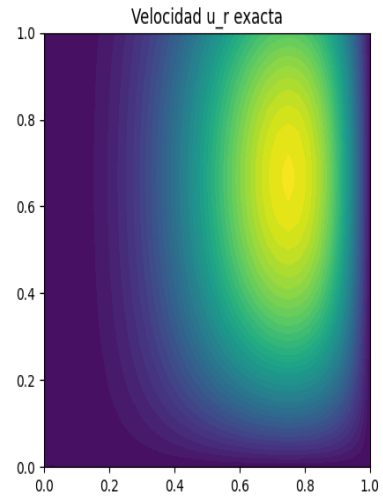


(b) Presión Exacta.

Figura 2.1: Comparación de las presiones.

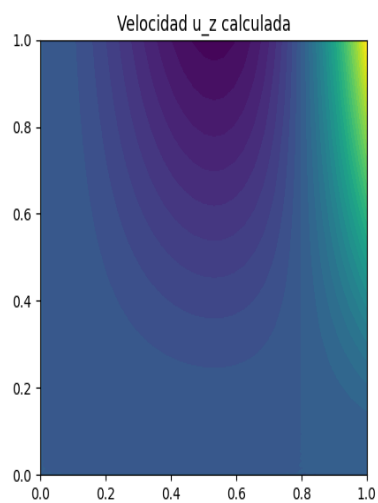


(a) u_r calculada.

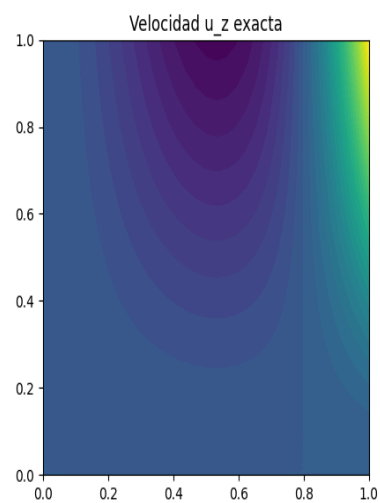


(b) u_r exacta.

Figura 2.2: Comparación de las velocidades en r .



(a) u_z calculada.



(b) u_z exacta.

Figura 2.3: Comparación de las velocidades en z .

Capítulo 3

Problema de Stokes-Darcy

En este capítulo estudiaremos la aproximación numérica de ecuaciones acopladas de flujo de fluidos de Stokes y Darcy en un dominio axisimétrico. Se supone que el flujo es simétrico en el eje y reescribir el problema en coordenadas cilíndricas reduce el problema 3-D a un problema en 2-D. Para la implementación en el Feenics, modificaremos la implementación de un problema equivalente 3-D cuyo análisis es realizado en [9]. Con nuestra propia implementación reproduciremos los ejemplos de [8] y otros de interés.

Antes de escribir el modelo a estudiar recordemos que las contrapartes axisimétricas de los operadores diferenciales habituales que actúan sobre vectores y escalares son

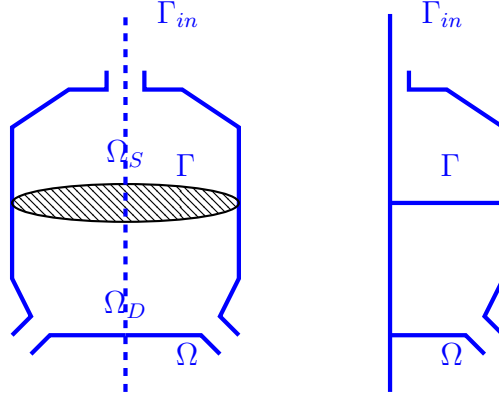
$$\begin{aligned}\nabla_a \cdot \mathbf{v} &= \text{div}_a \mathbf{v} := \partial_z v_z + \frac{1}{r} \partial_r (r v_r), & \Delta \mathbf{v} &= (\Delta v_r - r^{-2} v_r, \Delta v_z) \\ \nabla_a \varphi &:= (\partial_r \varphi, \partial_z \varphi)^T, & \Delta \varphi &= r^{-1} \partial_r (r \partial_r \varphi) + \partial_{zz} \varphi \\ \nabla_a \mathbf{u} &= \begin{bmatrix} \partial_r u_r & 0 & \partial_r u_z \\ 0 & u_r/r & 0 \\ \partial_z u_r & 0 & \partial_z u_z \end{bmatrix} & \mathbf{d}_a(\mathbf{u}) &= \frac{1}{2} (\nabla_a \mathbf{u} + (\nabla_a \mathbf{u})^t).\end{aligned}$$

En las fórmulas anteriores estamos considerando una función escalar $\varphi(r, z)$ y una función vectorial $\mathbf{u}(r, z) = (u_r(r, z), 0, u_z(r, z))$. Finalmente, si $S(r, z)$ es un tensor

entonces

$$\operatorname{div}_a S = \operatorname{div}_a \begin{bmatrix} S_{rr} & S_{r\theta} & S_{rz} \\ S_{\theta r} & S_{\theta\theta} & S_{\theta z} \\ S_{zr} & S_{z\theta} & S_{zz} \end{bmatrix} = \begin{bmatrix} \frac{1}{r} \left[\partial_r(rS_{rr}) + r\partial_z S_{rz} - \frac{1}{r}S_{\theta\theta} \right] \\ \partial_r S_{\theta r} + \partial_z S_{\theta z} + \frac{1}{r}S_{r\theta} + \frac{1}{r}S_{\theta r} \\ \frac{1}{r} [\partial_r(rS_{zr}) + r\partial_z S_{zz}] \end{bmatrix} \quad (3.1)$$

3.1. Planteamiento del Problema



Sea $\Omega \subset \mathbb{R}^2$, el dominio de flujo de interés. Además, sean Ω_S y Ω_D dominios acotados, poligonales, convexos para el flujo de Stokes y el flujo de Darcy, respectivamente. El límite de la interfaz entre los dominios se denota por $\Gamma = \partial\Omega_S \cap \partial\Omega_D$. Note que $\Omega = \Omega_S \cup \Omega_D \cup \Gamma$. Los vectores normales unitarios que apuntan hacia afuera a Ω_S y Ω_D se denotan n_S y n_D , respectivamente. En Γ sean t_1, t_2 los vectores tangentes unitarios linealmente independientes. Suponemos que hay un límite de flujo de entrada Γ_{in} , un subconjunto de $\partial\Omega_S \setminus \Gamma$ que está separado de Γ y un límite de flujo de salida Γ_{out} , un subconjunto de $\partial\Omega_D \setminus \Gamma$ que también está separado de Γ . Definamos $\Gamma_S := \partial\Omega_S \setminus (\Gamma \cup \Gamma_{in})$, y $\Gamma_D := \partial\Omega_D \setminus (\Gamma \cup \Gamma_{out})$. Suponemos que el flujo en el dominio poroso Ω_D está asociado con la ecuación de Darcy sujeta a la incompresibilidad del fluido.

Para el flujo de Stokes tenemos

$$-\nabla_a \cdot (2\nu \mathbf{d}_a(\mathbf{u}_S) - p_S I) = f_S \quad \text{en } \Omega_S \quad (3.2)$$

$$\nabla_a \cdot \mathbf{u}_S = 0 \quad \text{en } \Omega_S \quad (3.3)$$

$$\mathbf{u}_S = 0 \quad \text{en } \Gamma_S \quad (3.4)$$

donde $\mathbf{u}_S = \mathbf{u}_r e_r + \mathbf{u}_z e_z$, para los vectores unitarios e_r, e_z en las direcciones r y z respectivamente y $\mathbf{d}_a(\mathbf{u}) = \frac{1}{2}(\nabla_a \mathbf{u} + (\nabla_a \mathbf{u})^t)$ representa el tensor de deformación, \mathbf{u}_S denota la velocidad del fluido, p_S la presión, f_S una función de fuerza externa, ν la viscosidad cinemática del fluido.

Para el medio poroso

$$\nu_{eff} K^{-1} \mathbf{u}_D + \nabla_a p_D = f_D \quad \text{en } \Omega_D \quad (3.5)$$

$$\nabla_a \cdot \mathbf{u}_D = 0 \quad \text{en } \Omega_D \quad (3.6)$$

$$\mathbf{u}_D \cdot \mathbf{n}_D = 0 \quad \text{en } \Gamma_D \quad (3.7)$$

En esta ecuación \mathbf{u}_D , p_D , f_D , denotan la velocidad del fluido, presión y función de fuerza externa, respectivamente. Además, ν_{eff} representa una viscosidad cinemática efectiva del fluido, y K la permeabilidad (simétrico, definido positivo) del dominio. Por simplicidad, asumimos que existe una función escalar κ tal que $\kappa I = \nu_{eff} K^{-1}$. A través de la interfaz Γ los flujos se acoplan mediante la conservación de la masa y el equilibrio de fuerzas normales. Además, la condición Beavers-Joseph-Saffman (BJS) se utiliza para la fuerza tangencial la condición de contorno en Γ .

$$\mathbf{u}_S \cdot \mathbf{n}_S + \mathbf{u}_D \cdot \mathbf{n}_D = 0, \quad p_S - (2\nu \mathbf{d}_a(\mathbf{u}_S) \cdot \mathbf{n}_S) \cdot \mathbf{n}_S = p_D \quad \text{en } \Gamma \quad (3.8)$$

$$\mathbf{u}_S \cdot \mathbf{t}_l = -\alpha_l (2\nu \mathbf{d}_a(\mathbf{u}_S) \cdot \mathbf{n}_S) \cdot \mathbf{t}_l \quad \text{en } \Gamma, \quad l = 1, 2, \quad (3.9)$$

donde α_1, α_2 denotan constantes de fricción.

Antes de hacer la respectiva integración por partes y obtener la formulación variacional veamos que forma tiene el término $\nabla_a \cdot (\mathbf{d}_a(\mathbf{u}_S))$:

$$\begin{aligned}
\nabla_a \mathbf{d}_a(\mathbf{u}_S) &= \text{div}_a \begin{bmatrix} \partial_r u_r & 0 & (\partial_r u_z + \partial_z u_r)/2 \\ 0 & u_r/r & 0 \\ (\partial_z u_r + \partial_r u_z)/2 & 0 & \partial_z u_z \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{r} \left[\partial_r(r \partial_r u_r) + \partial_z(r(\partial_r u_z + \partial_z u_r)/2) - \frac{1}{r^2} u_r \right] \\ 0 \\ \frac{1}{r} [\partial_r(r(\partial_r u_z + \partial_z u_r)/2) + r \partial_{zz} u_z] \end{bmatrix}
\end{aligned}$$

Entonces cuando multiplicamos ese termino por una función test $\mathbf{v}(r, z) = (v_r, 0, v_z)$ e integramos obtenemos:

$$\begin{aligned}
\int_{\Omega} \text{div}_a(\mathbf{d}_a(\mathbf{u}_S)) \mathbf{v}_S r \, dr dz &= \int_{\Omega} [\partial_r(r \partial_r u_r) + \partial_z(r(\partial_r u_z + \partial_z u_r)/2)] v_r \, dr dz - \int_{\Omega} \frac{u_r}{r} \frac{v_r}{r} \, dr dz \\
&+ \int_{\Omega} [\partial_r(r(\partial_r u_z + \partial_z u_r)/2) + r \partial_{zz} u_z] v_z \, dr dz \\
&= - \int_{\Omega} \partial_r u_r \partial_r v_r r \, dr dz - \int_{\Omega} (\partial_r u_z + \partial_z u_r)/2 \partial_z v_r r \, dr dz \\
&- \int_{\Omega} (\partial_r u_z + \partial_z u_r)/2 \partial_r v_z r \, dr dz - \int_{\Omega} \partial_z u_z \partial_z v_z r \, dr dz \\
&+ \int_{\Gamma} \mathbf{d}(\mathbf{u}_S) \cdot \mathbf{n} \mathbf{v} \, ds - \int_{\Omega} \frac{u_r}{r} \frac{v_r}{r} \, dr dz \\
&= - \int_{\Omega} \mathbf{d}(\mathbf{u}_S) : \nabla \mathbf{v}_S r \, dr dz + \int_{\Gamma} \mathbf{d}(\mathbf{u}_S) \cdot \mathbf{n} \mathbf{v} \, ds - \int_{\Omega} \frac{u_r}{r} \frac{v_r}{r} \, dr dz \\
&= - \int_{\Omega} \mathbf{d}(\mathbf{u}_S) : \mathbf{d}(\mathbf{v}_S) r \, dr dz + \int_{\Gamma} \mathbf{d}(\mathbf{u}_S) \cdot \mathbf{n} \mathbf{v}_S \, ds - \int_{\Omega} \frac{u_r}{r} \frac{v_r}{r} \, dr dz
\end{aligned}$$

Debemos resaltar que el operador $\mathbf{d}(\cdot)$ corresponde al del caso cartesiano porque aislamos los términos divididos por r en un término independiente.

Por otro lado

$$\int_{\Omega} \mathbf{v}_S \cdot \nabla p_S r \, dr dz = - \int_{\Omega} \text{div}_a \mathbf{v}_S p_S r \, dr dz + \int_{\Gamma} \mathbf{v} \cdot \mathbf{n} p_S r \, ds. \quad (3.10)$$

Entonces

$$\begin{aligned}
\int_{\Omega_S} f_S \mathbf{v}_S r \, dr dz &= - \int_{\Omega_S} \nabla_a \cdot (2\nu \mathbf{d}_a(\mathbf{u}_S) - p_S I) \mathbf{v}_S r \, dr dz \\
&= - \int_{\Omega_S} \nabla_a \cdot (2\nu \mathbf{d}_a(\mathbf{u}_S)) r \, dr dz + \int_{\Omega_S} \nabla_a p_S \mathbf{v}_S r \, dr dz \\
&= \int_{\Omega_S} 2\nu \mathbf{d}(\mathbf{u}_S) : \mathbf{d}(\mathbf{v}_S) r \, dr dz - \int_{\Gamma} 2\nu \mathbf{d}(\mathbf{u}_S) \cdot \mathbf{n} \mathbf{v} \, ds + \int_{\Omega_S} 2\nu \frac{u_r}{r} \frac{v_r}{r} \, dr dz \\
&\quad - \int_{\Omega_S} p_S \operatorname{div}_a \mathbf{v}_S r \, dr dz + \int_{\Gamma} \mathbf{v}_S \cdot \mathbf{n} p_S \, ds
\end{aligned}$$

Por otro lado

$$\begin{aligned}
\int_{\Omega} f_D \mathbf{v}_D r \, dr dz &= \int_{\Omega_D} (\nu_{eff} K^{-1} \mathbf{u}_D + \nabla_a p_D) \mathbf{v}_D r \, dr dz \\
&= \int_{\Omega_D} \nu_{eff} K^{-1} \mathbf{u}_D \mathbf{v}_D r \, dr dz + \int_{\Omega_D} \nabla_a p_D \mathbf{v}_D r \, dr dz \\
&= \int_{\Omega_D} \nu_{eff} K^{-1} \mathbf{u}_D \mathbf{v}_D r \, dr dz - \int_{\Omega} p_D \operatorname{div}_a \mathbf{v}_D r \, dr dz + \int_{\Gamma} \mathbf{v}_D \cdot \mathbf{n} p_D \, ds
\end{aligned}$$

Los términos de frontera que quedan son

$$\begin{aligned}
& - \int_{\Gamma} 2\nu \mathbf{d}(\mathbf{u}_S) \cdot \mathbf{n} \mathbf{v} r \, ds + \int_{\Gamma} \mathbf{v}_S \cdot \mathbf{n} p_S r \, ds + \int_{\Gamma} \mathbf{v}_D \cdot \mathbf{n} p_D r \, ds \\
& - \int_{\Gamma} 2\nu \mathbf{d}(\mathbf{u}_S) \cdot \mathbf{n} \mathbf{v}_S r \, ds + \int_{\Gamma} \mathbf{v}_S \cdot \mathbf{n} p_S r \, ds + \int_{\Gamma} \mathbf{v}_D \cdot \mathbf{n} p_D r \, ds \\
& = - \int_{\Gamma} (2\nu \mathbf{d}(\mathbf{u}_S) \cdot \mathbf{n}) \cdot \mathbf{n} (\mathbf{v}_S \cdot \mathbf{n}) r \, ds - \int_{\Gamma} (2\nu \mathbf{d}(\mathbf{u}_S) \cdot \mathbf{n}) \cdot \mathbf{t} (\mathbf{v}_S \cdot \mathbf{t}) r \, ds \\
& \quad + \int_{\Gamma} \mathbf{v}_S \cdot \mathbf{n} p_S r \, ds + \int_{\Gamma} \mathbf{v}_D \cdot \mathbf{n} p_D r \, ds \\
& = \int_{\Gamma} (p_D - p_S) \mathbf{v}_S \cdot \mathbf{n} r \, ds + \int_{\Gamma} \alpha_{as}^{-1}(\mathbf{u}_S \cdot \mathbf{t}) (\mathbf{v}_S \cdot \mathbf{t}) r \, ds \\
& \quad + \int_{\Gamma} \mathbf{v}_S \cdot \mathbf{n} p_S r \, ds + \int_{\Gamma} \mathbf{v}_D \cdot \mathbf{n} p_D r \, ds \\
& = \int_{\Gamma} p_D (\mathbf{v}_S \cdot \mathbf{n}_S + \mathbf{v}_D \cdot \mathbf{n}_D) r \, ds + \int_{\Gamma} \alpha_{as}^{-1}(\mathbf{u}_S \cdot \mathbf{t}) (\mathbf{v}_S \cdot \mathbf{t}) r \, ds \\
& = \int_{\Gamma} \alpha_{as}^{-1}(\mathbf{u}_S \cdot \mathbf{t}) (\mathbf{v}_S \cdot \mathbf{t}) r \, ds.
\end{aligned}$$

donde hemos usado (3.8) y (3.9). Para incluir la condición de la primera ecuación de (3.8) incluimos un multiplicador de Lagrange:

$$\int_{\Gamma} \mathbf{v}_S \cdot \mathbf{n}_S \zeta r \, ds + \langle \mathbf{v}_D \cdot \mathbf{n}_D, \zeta \rangle_{\Gamma},$$

donde se el término relacionado con el dominio Ω_D esta expresado, inicialmente, como producto de dualidad porque no \mathbf{v}_D no tiene la suficiente regularidad.

3.2. Espacios funcionales y formulación débil

Denotemos por $L_r^2(\Omega)$ el espacio de Lebesgue con pesos de todas la funciones medibles u definidas en Ω para las cuales

$$\|u\|_{L_r^2(\Omega)} := \int_{\Omega} |u|^2 r \, dr dz < \infty.$$

El espacio de Sobolev con pesos $H_r^k(\Omega)$ consiste en todas las funciones en $L_r^2(\Omega)$ cuyas derivadas hasta el orden k estan en $L_r^2(\Omega)$. También definimos las normas y las seminormas en la forma usual; en particular,

$$|u|_{H_r^1(\Omega)}^2 := \int_{\Omega} (|\partial_r u|^2 + |\partial_z u|^2) r \, dr dz,$$

Sea $\tilde{H}_r^1(\Omega) := H_r^1(\Omega) \cap L_{1/r}^2(\Omega)$, donde $L_{1/r}^2(\Omega)$ denota el conjunto de todas las funciones medibles u definidas en Ω para las cuales

$$\|u\|_{L_{1/r}^2(\Omega)}^2 := \int_{\Omega} \frac{|u|^2}{r} \, dr dz < \infty.$$

$\tilde{H}_r^1(\Omega)$ es un espacio de Hilbert con la norma

$$\|u\|_{\tilde{H}_r^1(\Omega)} := \left(\|u\|_{H_r^1(\Omega)}^2 + \|u\|_{L_{1/r}^2(\Omega)}^2 \right)^{1/2}.$$

A partir de estos espacions funcionales definamos X_S y M_S de la siguiente forma

$$X_S = \{\mathbf{v} = (v_r, v_z) \in \tilde{H}_r^1(\Omega) \times H_r^1(\Omega), v_r = 0, \quad v_z = 0 \quad \text{en } \partial\Omega_S\}$$

$$M_S = \left\{ q \in L_r^2(\Omega_S) : \int_{\Omega_S} q r \, dr dz = 0 \right\}$$

Para la descripción del flujo de fluido en Ω_D , sea

$$X_D := \{w \in H(\text{div}_a, \Omega_D) : w \cdot \mathbf{n}|_{\Gamma_D} = 0\}$$

y

$$M_D = \left\{ q \in L_r^2(\Omega_D) : \int_{\Omega_D} q r \, dr dz = 0 \right\}$$

$$\|w\|_{X_D} := \left(\|div_{axi}(w)\|_{1L^2(\Omega_D)}^2 + \|w\|_{1L^2(\Omega_D)}^2 \right)^{1/2}, \quad \|\cdot\|_{M_D} = \|\cdot\|_{1L^2(\Omega_D)},$$

Sea

$$X := X_S \times X_D, \text{ y } M := \left\{ q \in M_S \times M_D : \int_{\Omega} q r \, dx = 0 \right\},$$

y llamemos al espacio dual de X por X^* .

La formulación axisimétrica débil para las ecuaciones puede expresarse como:

Dado $f \in X^*$, determinar $(\mathbf{u}, p, \lambda) \in X \times M \times H^{1/2}(\Gamma)$ tal que, para todo $\mathbf{v} \in X$ y $(q, \zeta) \in M \times H^{1/2}(\Gamma)$,

$$a(\mathbf{u}, \mathbf{v}) - b(\mathbf{v}, p) + b_I(\mathbf{v}, \lambda) = (f, \mathbf{v}),$$

$$b(\mathbf{u}, q) = 0$$

$$b_I(\mathbf{u}, \zeta) = 0$$

donde

$$a(\mathbf{u}, \mathbf{v}) := a_S(\mathbf{u}_S, \mathbf{v}_S) + a_D(\mathbf{u}_D, \mathbf{v}_D), \quad b(\mathbf{v}, q) := b_S(\mathbf{v}_S, q_S) + b_D(\mathbf{v}_D, q_D)$$

$$b_I(\mathbf{v}, \zeta) = \int_{\Gamma} \mathbf{v}_S \cdot \mathbf{n}_S \zeta r ds + \langle \mathbf{v}_D \cdot \mathbf{n}_D, \zeta \rangle_{\Gamma},$$

y

$$a_S(\mathbf{u}_S, \mathbf{v}_S) := \int_{\Omega_S} 2\nu \mathbf{d}(\mathbf{u}_S) : \mathbf{d}(\mathbf{v}_S) r \, dr dz + \int_{\Omega} 2\nu \frac{u_r}{r} \frac{v_r}{r} r \, dr dz + \int_{\Gamma} \alpha_{as}^{-1}(\mathbf{u}_S \cdot \mathbf{t})(\mathbf{v}_S \cdot \mathbf{t}) r ds,$$

$$a_D(\mathbf{u}_D, \mathbf{v}_D) := \int_{\Omega_D} \kappa \mathbf{u}_D \cdot \mathbf{v}_D r \, dr dz,$$

$$b_S(\mathbf{v}, q) := \int_{\Omega_S} q \nabla_a \cdot \mathbf{v} r \, dr dz, \quad b_D(\mathbf{v}, q) := \int_{\Omega_D} q \nabla_a \cdot \mathbf{v} r \, dr dz,$$

donde α_{as} es la constante de fricción de la condición BJS.

3.3. Aproximación de elementos finitos

En esta sección analizaremos la aproximación de elementos finitos al sistema acoplado axisimétrico de Stokes-Darcy. Centramos nuestra atención en conformar espacios aproximados $X_{S,h} \subset X_S$, $M_{S,h} \subset M_S$, $X_{D,h} \subset X_D$, $M_{D,h} \subset M_D$, $L_h \subset_1 H^{1/2}(\Gamma)$, donde $X_{S,h}, M_{S,h}$ representen espacios de velocidad y presión utilizados típicamente para aproximaciones de flujo de fluido, y $X_{D,h}, M_{D,h}$ espacios de velocidad y presión que se utilizan normalmente para (formulación mixta) aproximaciones de flujo Darcy.

Comenzamos describiendo el marco de aproximación de elementos finitos utilizado en el análisis. Sea $\Omega_j \subset \mathbb{R}^2$, $j = f, p$, un dominio poligonal y sea $\mathcal{T}_{j,h}$ una triangulación de $\overline{\Omega_j}$. Por tanto, el dominio computacional está definido por $\overline{\Omega} = \cup K$; $K \in \tau_{S,h} \cup \tau_{D,h}$. Suponemos que la triangulación es de forma regular y cuasi-uniforme, es decir, que existen constantes c_1, c_2 tal que $c_1 h \leq h_K \leq c_2 \rho_K$, donde h_K es el diámetro del triángulo K , ρ_K es el diámetro de la bola más grande incluida en K , y $h = \max_{K \in \tau_{S,h} \cup \tau_{D,h}} h_K$. También asumimos que la triangulación en $\overline{\Omega_D}$ induce la partición en Γ , que denotamos $\tau_{\Gamma,h}$.

Sea $P_k(K)$ el espacio de polinomios en K de grado no mayor que k , y $RT_k(K) := (P_k(K))^2 + x P_k(K)$ denotan los elementos de Raviart-Thomas (R-T) de k -ésimo orden. Luego definimos los espacios de elementos finitos de la siguiente manera

$$\begin{aligned} X_{S,h} &:= \{v \in X_S \cap C(\overline{\Omega_S})^2 : v|_K \in P_m(K), \forall K \in \tau_{S,h}\}, \\ M_{S,h} &:= \{q \in M_S \cap C(\overline{\Omega_S}) : q|_K \in P_{m-1}(K), \forall K \in \tau_{S,h}\}, \\ X_{D,h} &:= \{v \in RT_k(K), \forall K \in \tau_{D,h}\} \\ M_{D,h} &:= \{q \in M_S : q|_K \in P_k(K), \forall K \in \tau_{D,h}\} \\ L_h &:= \{\zeta \in C(\Gamma) : \zeta|_K \in P_l(K), \forall K \in \tau_{\Gamma,h}\}. \end{aligned}$$

Los espacios $(X_{S,h}, M_{f,h})$ representan el par de espacios de aproximación de Taylor-Hood. El análisis presentado en [8] también es válido para $(X_{D,h}, M_{p,h})$ correspondiente a la aproximación de espacios de elementos finitos de Brezzi-Douglas-Marini (BDM). De manera análoga a la formulación continua, llamemos $X_h := X_{f,h} \times X_{p,h}$,

y $M_h := \{q \in M_{f,h} \times M_{p,h} : \int_{\Omega} q r dx = 0\}$.

Observación: En el ajuste axisimétrico, para la construcción de las integrales ponderadas interpolantes R-T, es decir, $\int_{\partial K} \cdots r ds$, $\int_K \cdots r dx$ son usadas.

A continuación, suponemos que $m \geq 2$, $k \geq 1$ y $l \leq k$. Note que la presión interfacial que se aproxima al espacio L_h está contenida en el espacio de la traza de la componente normal de las velocidades de $X_{D,h}$, es decir, $L_h \subset \{v \cdot n_D|_{\Gamma} : v \in X_{p,h}\}$.

También se utiliza en el análisis el espacio funcional discreto:

$$V_h := \{v \in X_h : b_I(v_h, \zeta) = 0, \text{ para todo } \zeta \in L_h\},$$

$$Z_h := \{v \in V_h : b(v, q) = 0, \text{ para todo } q \in M_h \times \mathbb{R}^2\}$$

Sea

$$X_{S,h}^0 := \{v \in X_{S,h} : v|_{\partial\Omega_S \setminus \Gamma} = 0\}$$

Lema 3.3.1. Existe $C_{S,h} > 0$ tal que

$$\inf_{0 \neq q_h \in M_{S,h}} \sup_{v_h \in X_{S,h}^0} \frac{\int_{\Omega_S} q_h \operatorname{div}_{axi}(v_h) r dx}{\|q_h\|_{M_S} \|v_h\|_{X_S}}$$

Para $(X_{D,h}, M_{D,h})$ espacios de aproximación de Raviart-Thomas para la velocidad y la presión, a diferencia del ajuste cartesiano, $a_D(\cdot, \cdot)$ no es coercitivo, con respecto a la norma $H(\operatorname{div}, \Omega_D)$, en

$$Z_{D,h} := \{v \in X_{D,h} : \int_{\Omega_D} q \operatorname{div}_{axi}(v) r dx = 0, \forall q \in M_{D,h}\}.$$

Para compensar esto agregamos el término

$$\gamma \int_{\Omega_D} \operatorname{div}_{axi}(v) r dx$$

a $a_D(u, v)$ donde $\gamma > 0$ es una constante fija, y definimos

$$a_{p,\gamma}(u, v) := a_D(v) + \gamma \int_{\Omega_D} \operatorname{div}_{axi}(v) r dx$$

En caso de que un término fuente axisimétrico, g , se modele en el dominio poroso $\check{\Omega}_D$ tenemos la ecuación $\nabla \cdot \mathbf{u}_D = g$ en Ω_D , junto con la suma $a_D(\mathbf{u}, \mathbf{v})$ el término $\gamma \int_{\Omega_D} \operatorname{div}_{axi}(\mathbf{v}) r dr dz$ se agregaría al lado derecho. La condición de integral nula para

p_S y p_D en Ω_S y Ω_D respectivamente, es impuesta a través de un multiplicador de Lagrange en las siguientes formas bilineales

$$c(\rho, q) = \int_{\Omega_S} \rho q_S r \, dr dz + \int_{\Omega_D} \rho q_D r \, dr dz$$

Problema de aproximación discreta: Dado $f \in X^*$, determinar $(\mathbf{u}_h, p_h, \lambda_h, \rho_h) \in (X_h \times M_h \times L_h \times \mathbb{R})$ tal que, para cada $\mathbf{v} \in X_h$ y $(q, \zeta, \beta) \in M_h \times L_h \times \mathbb{R}$,

$$a_\gamma(\mathbf{u}_h, v) - b(\mathbf{v}, p_h) + b_I(\mathbf{v}, \lambda_h) = (f, v),$$

$$b(\mathbf{u}_h, q) + c(\beta, q) = (g, q)$$

$$b_I(\mathbf{u}_h, \zeta) = 0$$

$$c(\rho, p_h) = (G, \rho)$$

donde

$$a_\gamma(\mathbf{u}, \mathbf{v}) := a_S(\mathbf{u}_S, v_S) + a_{p,\gamma}(\mathbf{u}_D, v_D), \quad b(\mathbf{v}, q) := b_S(\mathbf{v}_S, q_S) + b_D(\mathbf{v}_D, q_D)$$

$$b_I(\mathbf{v}, \zeta) = \int_\Gamma \mathbf{v}_S \cdot \mathbf{n}_S \zeta r ds + \langle \mathbf{v}_D \cdot \mathbf{n}_D, \zeta \rangle_\Gamma,$$

y

$$\begin{aligned} a_S(\mathbf{u}, \mathbf{v}) &:= \int_{\Omega_S} 2\nu \mathbf{d}(\mathbf{u}) : \mathbf{d}(\mathbf{v}) r \, dr dz + \int_{\Omega_S} 2\nu \frac{u_r}{r} \frac{v_r}{r} r \, dr dz + \int_\Gamma \alpha_{as}^{-1} (u \cdot \mathbf{t})(v \cdot \mathbf{t}) r ds, \\ a_{p,\gamma}(\mathbf{u}, \mathbf{v}) &:= \int_{\Omega_D} \kappa \mathbf{u} \cdot \mathbf{v} r dx + \gamma \int_{\Omega_D} \text{div}_{axi}(\mathbf{v}) r \, dr dz \\ b_S(\mathbf{v}, q) &:= \int_{\Omega_S} q \nabla_a \cdot \mathbf{v} r \, dr dz, \quad b_D(\mathbf{v}, q) := \int_{\Omega_D} q \nabla_a \cdot \mathbf{v} r \, dr dz, \\ b_I(\mathbf{v}, \zeta) &= \int_\Gamma \mathbf{v}_S \cdot \mathbf{n}_S \zeta r ds + \langle \mathbf{v}_D \cdot \mathbf{n}_D, \zeta \rangle_\Gamma, \end{aligned}$$

3.4. Ejemplos Numéricos

En esta sección estudiaremos numéricamente la aproximación de dos problemas de flujo acoplados de Stokes-Darcy, cilíndricamente simétricos. El primer experimento se realiza en un ejemplo con una solución conocida. Las tasas de convergencia de la aproximación a las soluciones conocidas se calculan para dos familias diferentes de

elementos finitos. El segundo ejemplo que investigamos es el del flujo de fluidos a través del ojo.

3.4.1. Ejemplos con solución conocida

Para este ejemplo tomamos $\Omega = [0, 1] \times [-1, 1]$, $\Omega_S = [0, 1] \times [0, 1]$, $\Omega_P = [0, 1] \times [-1, 0]$

$$\mathbf{u}_S(r, z) = \mathbf{u}_D(r, z) = [-r \cos(\pi r) \sin(\pi z), -\frac{2}{\pi} \cos(\pi r) \cos(\pi z) + r \sin(\pi r) \cos(\pi z)]$$

$$p_S(r, z) = p_D(r, z) = \sin(\pi z)(-\cos(\pi r) + 2\pi r \sin(\pi r)) + 4re^{-4r} \cos(\pi z) - \frac{2}{\pi}(1 - 5e^{-2}).$$

Test 1: Elementos Taylor Hood

Para aproximar la velocidad y la presión en el medio Ω_S , (\mathbf{u}_S, p_S) usaremos elementos finitos tipo Taylor Hood es decir $P_2 - P_1$ y elementos finitos $RT_1 - \text{disc}P_1$ para aproximar (\mathbf{u}_D, p_D) . Para aproximar la presión en la interface, λ usaremos P_1 .

Cuadro 3.1: Test 1 Errores experimentales y tasas de convergencia en Ω_S .

| N | h | $\ u_S - u_{Sh}\ _{X_S}$ | r_1 | $\ p_S - p_{Sh}\ _{M_S}$ | r_1 |
|-------|--------|--------------------------|-------|--------------------------|-------|
| 143 | 0.5000 | 0.4085 | 0.000 | 0.4577 | 0.000 |
| 505 | 0.2500 | 0.1807 | 1.177 | 0.176 | 1.379 |
| 1901 | 0.1250 | 0.06191 | 1.545 | 0.05932 | 1.569 |
| 7381 | 0.0625 | 0.0214 | 1.533 | 0.02049 | 1.534 |
| 29093 | 0.0312 | 0.007468 | 1.519 | 0.007158 | 1.517 |

Test 2: Mini Elementos

Repetimos este experimento con otra familia de elementos finitos. En este caso aproximaremos la velocidad y la presión en el medio Ω_S , (\mathbf{u}_S, p_S) con elementos finitos tipo Mini element es decir $(P_1 + \text{bubble} - P_1)$ y elementos finitos $RT_1 - \text{disc}P_1$ para

Cuadro 3.2: Test 1 Errores experimentales y tasas de convergencia en Ω_D .

| N | h | $\ u_D - u_{Dh}\ _{X_D}$ | r_1 | $\ p_D - p_{Dh}\ _{M_D}$ | r_1 | $\ \lambda - \lambda_h\ _{L^2_1(\Omega)}$ | r_1 |
|-------|--------|--------------------------|-------|--------------------------|-------|---|-------|
| 143 | 0.5000 | 0.2406 | 0.000 | 0.5642 | 0.000 | 1.511 | 0.000 |
| 505 | 0.2500 | 0.1385 | 0.797 | 0.2969 | 0.926 | 0.9775 | 0.629 |
| 1901 | 0.1250 | 0.07428 | 0.898 | 0.1498 | 0.987 | 0.4658 | 1.069 |
| 7381 | 0.0625 | 0.03855 | 0.946 | 0.07519 | 0.995 | 0.2264 | 1.041 |
| 29093 | 0.0312 | 0.01983 | 0.959 | 0.03763 | 0.998 | 0.1115 | 1.022 |

aproximar (\mathbf{u}_D, p_D) . Para aproximar la presión en la interface, λ usaremos P_1 . En este caso las

$$X_{S,h} := \{v \in X_S \cap C(\overline{\Omega_S})^2 : v|_K \in P_{1+b}(K), \forall K \in \tau_{S,h}\},$$

es decir

$$v|_K = a + br + cz + d\phi(r, z) \cdot \phi_2(r, z) \cdot \phi_3(r, z)$$

donde las funciones ϕ_1, ϕ_2 y ϕ_3 son las funciones base nodales lineales (coordenadas baricéntricas) del propio triángulo.

Cuadro 3.3: Test 2 Errores experimentales y tasas de convergencia en Ω_S .

| N | h | $\ u_S - u_{Sh}\ _{X_S}$ | r_1 | $\ p_S - p_{Sh}\ _{M_S}$ | r_1 |
|-------|--------|--------------------------|-------|--------------------------|-------|
| 119 | 0.5000 | 0.9912 | 0.000 | 0.97 | 0.000 |
| 425 | 0.2500 | 0.4857 | 1.029 | 0.2517 | 1.946 |
| 1613 | 0.1250 | 0.2314 | 1.070 | 0.0734 | 1.778 |
| 6293 | 0.0625 | 0.1128 | 1.037 | 0.02329 | 1.656 |
| 24869 | 0.0312 | 0.05569 | 1.018 | 0.007746 | 1.588 |

Cuadro 3.4: Test 2 Errores experimentales y tasas de convergencia en Ω_D .

| N | h | $\ u_D - u_{Dh}\ _{X_D}$ | r_1 | $\ p_D - p_{Dh}\ _{M_D}$ | r_1 | $\ \lambda - \lambda_h\ _{L^2_1(\Omega)}$ | r_1 |
|-------|--------|--------------------------|-------|--------------------------|-------|---|-------|
| 119 | 0.5000 | 0.2619 | 0.000 | 0.69 | 0.000 | 2.551 | 0.000 |
| 425 | 0.2500 | 0.1382 | 0.923 | 0.3053 | 1.176 | 1.315 | 0.956 |
| 1613 | 0.1250 | 0.07416 | 0.898 | 0.1503 | 1.022 | 0.5908 | 1.155 |
| 6293 | 0.0625 | 0.03854 | 0.944 | 0.07523 | 0.999 | 0.2803 | 1.076 |
| 24869 | 0.0312 | 0.01983 | 0.958 | 0.03764 | 0.999 | 0.1366 | 1.037 |

3.4.2. Ejemplo de aplicación

Finalizamos este capítulo mostrando un ejemplo aplicado. Usaremos el método numérico estudiado para modelar un problema de interacción entre el humor acuoso de la cámara anterior y la malla trabecular del ojo humano. Este ejemplo se puede encontrar, por ejemplo, en [8] y [10] donde es estudiado con modelos más realistas y complejos. Para nuestro ejemplo usaremos una de las mallas usadas en [10].

En este caso estamos ante un problema evolutivo por lo que debemos incluir en nuestro conjunto de ecuaciones un nuevo término con la derivada de la velocidad en Ω_S ;

$$\frac{\partial \mathbf{u}_S}{\partial t} - \nabla_a \cdot (2\nu \mathbf{d}_a(\mathbf{u}_S) - p_S I) = f_S \quad \text{en } \Omega_S \quad (3.11)$$

$$\nabla_a \cdot \mathbf{u}_S = 0 \quad \text{en } \Omega_S \quad (3.12)$$

$$\mathbf{u}_S = 0 \quad \text{en } \Gamma_S \quad (3.13)$$

$$\nu_{eff} K^{-1} \mathbf{u}_D + \nabla_a p_D = f_D \quad \text{en } \Omega_D \quad (3.14)$$

$$\nabla_a \cdot \mathbf{u}_D = 0 \quad \text{en } \Omega_D \quad (3.15)$$

$$\mathbf{u}_D \cdot \mathbf{n}_D = 0 \quad \text{en } \Gamma_D \quad (3.16)$$

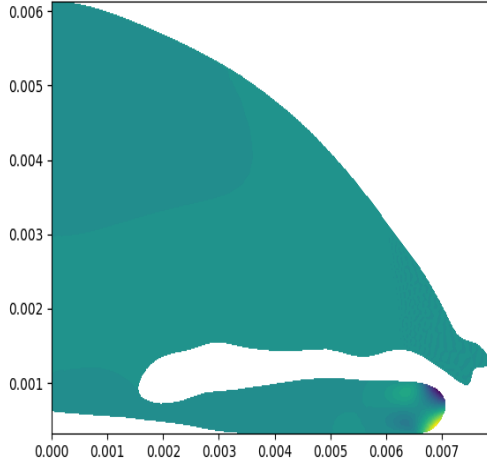
Usaremos una aproximación tipo Euler para aproximar esa derivada temporal:

$$\frac{\partial \mathbf{u}_S}{\partial t} \approx \frac{\mathbf{u}_S - \mathbf{u}_{S0}}{\Delta t}$$

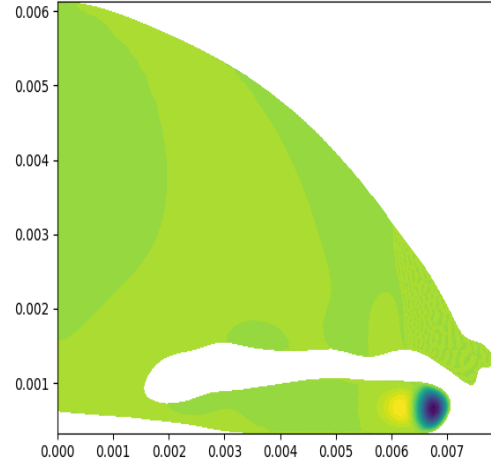
y la forma bilineal $a_S(\mathbf{u}\mathbf{v})$ ahora se define como

$$a_S(\mathbf{u}, \mathbf{v}) := \frac{1}{\Delta t} \int_{\Omega_S} \mathbf{u}_S \mathbf{v}_S r \, dr dz + \int_{\Omega_S} 2\nu \mathbf{d}(\mathbf{u}) : \mathbf{d}(\mathbf{v}) r \, dr dz + \int_{\Omega_S} 2\nu \frac{u_r}{r} \frac{v_r}{r} r \, dr dz + \int_{\Gamma} \alpha_{as}^{-1} (u \cdot \mathbf{t})(v \cdot \mathbf{t}) r \, ds,$$

y el término restante se suma al lado derecho.



(a) Primera componente de la velocidad.



(b) Segunda componente de la velocidad.

Capítulo 4

Códigos

4.1. Códigos Elasticidad

Elasticidad Lineal 2D

```
1  from dolfin import *
2  import matplotlib.pyplot as plt
3  # Programa para el problema de Elasticidad Lineal 2D
4  # Parametros Fisicos
5  E=10.0;nu=0.3;rho_g = 1e-3
6  mu=E/(2.0*(1.0 + nu))
7  mylambda= E*nu/((1.0 + nu)*(1.0 - 2.0*nu))
8  # Definicion de los operadores diferenciales
9  def eps (v) :
10     return sym ( grad (v) )
11  def sigma (v) :
12     dim = v. geometric_dimension ()
13     return 2.0* mu* eps (v) + mylambda *tr(eps (v) ) * Identity (dim )
14  # Creacion d ela malla
15  mesh = RectangleMesh ( Point (0.,0.) ,Point (10.,1.) ,100 , 10)
16  # Vector de carga(mismo peso)
```

```

17 f = Constant ((0. , -rho_g) )
18 # Definicion de los espacios funcionales
19 V = VectorFunctionSpace (mesh , 'CG', degree =2)
20 # Definicion de la frontera y d elas condiciones de frontera
21 def left (x, on_boundary ) :
22     return near (x[0] ,0.) and on_boundary
23 bc = DirichletBC (V, Constant ((0. ,0.) ) , left )
24 # Definicion Problema Variacional
25 u = TrialFunction (V)
26 v = TestFunction (V)
27 a = inner(sigma(u) , eps(v)) *dx
28 L = dot (f, v) *dx
29 u = Function (V)
30 solve (a == L, u, bc)
31 # Visualizacion
32 plt.figure()
33 plot(mesh, linewidth=0.2)
34 plot(u, mode="displacement")
35 plt.show()

```

Elasticidad Lineal Axisimetrica

```

1 # Programa para elasticidad lineal axisimetrica
2 from __future__ import print_function
3 from dolfin import *
4 from mshr import *
5 import matplotlib.pyplot as plt
6
7 Re = 11.;Ri = 9.
8 rect = Rectangle(Point(0., 0.), Point(Re, Re))
9 domain = Circle(Point(0., 0.), Re, 100) - Circle(Point(0., 0.), Ri, 100)

```

```

10 domain = domain - Rectangle(Point(0., -Re), Point(-Re, Re))-
11 Rectangle(Point(0., 0.), Point(Re, -Re))
12
13 mesh = generate_mesh(domain, 40)
14 plot(mesh)
15 # Definicion de la frontera
16 class Bottom(SubDomain):
17     def inside(self, x, on_boundary):
18         return near(x[1], 0) and on_boundary
19 class Left(SubDomain):
20     def inside(self, x, on_boundary):
21         return near(x[0], 0) and on_boundary
22 class Outer(SubDomain):
23     def inside(self, x, on_boundary):
24         return near(sqrt(x[0]**2+x[1]**2), Re, 1e-1) and on_boundary
25 facets = MeshFunction("size_t", mesh, 1)
26 facets.set_all(0)
27 Bottom().mark(facets, 1)
28 Left().mark(facets, 2)
29 Outer().mark(facets, 3)
30 ds = Measure("ds", subdomain_data=facets)
31 x = SpatialCoordinate(mesh)
32 # Definicion de los operadores diferenciales
33 def eps(v):
34     return sym(as_tensor([[v[0].dx(0), 0, v[0].dx(1)],
35                             [0, v[0]/x[0], 0],
36                             [v[1].dx(0), 0, v[1].dx(1)]]))
37 E = Constant(1e5)
38 nu = Constant(0.3)
39 mu = E/2/(1+nu)

```

```

40  lambda = E*nu/(1+nu)/(1-2*nu)
41  def sigma(v):
42      return lambda*tr(eps(v))*Identity(3) + 2.0*mu*eps(v)
43  n = FacetNormal(mesh)
44  p = Constant(10.)
45  # Definicion de los espacios funcionales
46  V = VectorFunctionSpace(mesh, 'CG', degree=2)
47  u = TrialFunction(V)
48  v = TestFunction(V)
49  a = inner(sigma(u), eps(v))*x[0]*dx
50  l = inner(-p*n, v)*x[0]*ds(3)
51
52  u = Function(V, name="Displacement")
53  #Condiciones de frontera apoyadas
54  bcs = [DirichletBC(V.sub(1), Constant(0), facets, 1),
55         DirichletBC(V.sub(0), Constant(0), facets, 2)]
56  solve(a == l, u, bcs)
57  # Visualizacion
58  plt.figure()
59  plot(mesh, linewidth=0.2)
60  plot(200*u, mode="displacement")
61  plt.show()
62  #Condiciones de frontera empotradas
63  bcs = [DirichletBC(V, Constant((0., 0.)), facets, 1),
64         DirichletBC(V.sub(0), Constant(0), facets, 2)]
65  solve(a == l, u, bcs)
66  plot(mesh, linewidth=0.2)
67  plot(200*u, mode="displacement")
68  plt.show()

```


4.2. Códigos Stokes

Este código está diseñado para realizar el análisis del error pero se puede modificar de manera directa para resolver un problema sin solución conocida. En el programa se escriben las soluciones exactas y el mismo código calcula el lado derecho del problema.

```
1  # Programa para estimar los errores y las tasas de convergencia
2  # Problema de Stokes Azisimetrico
3  from dolfin import *
4  import sympy2fenics as sf
5  from helpers import *
6  from multiphenics import *
7  import matplotlib.pyplot as plt
8  parameters["ghost_mode"] = "shared_facet" # required by dS
9  # Definicion de los operadores diferenciales
10 epsilon = lambda vec: sym(grad(vec))
11 diva = lambda vec: div(vec) + vec[0]/r
12 def mydiv(v):
13     return div(v)+v[0]/x[0]
14 def mygrad(v):
15     return as_matrix([[v[0].dx(0),0,v[1].dx(0)], [0,v[0]/x[0],0], [v[0].dx(1),0,v[1].dx(1)]]
16 def tdiva(u):
17     """ 2D axisymetric tensor div: diva of rows """
18     nrows = u.ufl_shape[0]
19     return as_vector([diva(u[i,:]) for i in range(0,nrows)])
20 str2exp = lambda s: sf.sympy2exp(sf.str2sympy(s))
21 u_str = '(pow(x,3)*(x-1)*y*(3*y-4)),(-pow(x,2)*(5*x-4)*(pow(y,2))*(y-2))'
22 p_str = 'pow(x,2)+pow(y,2)-10.0/12.0'
23 nkmax = 5
24 hh = []; nn = []; eu = []; ru = [];
25 euD = []; ruD = []; ep = []; rp = []
```

```

26 epD = []; rpD = []; elam = []; rlam = []
27 ru.append(0.0); rp.append(0.0);
28 # ***** Analisis del error ***** #
29 # Se crean las mallas, se resuelve el problema y se vuelve a refinar.
30 for nk in range(nkmax):
31     print("..... Refinement level : nk = ", nk)
32     nps = pow(2,nk+1)
33     mesh = RectangleMesh(Point(0.0,0.0),Point(1.0,1.0),nps,2*nps)
34     x = SpatialCoordinate(mesh); r = x[0]
35     subdomains = MeshFunction("size_t", mesh, 2)
36     subdomains.set_all(0)
37     boundaries = MeshFunction("size_t", mesh, 1)
38     boundaries.set_all(0)
39     outlet = 14
40     inlet = 15;
41     wallR = 18;
42     axis = 19;
43     u_ex = Expression(str2exp(u_str), degree=6, domain=mesh)
44     p_ex = Expression(str2exp(p_str), degree=6, domain=mesh)
45     fS = -tdiva(grad(u_ex)) + grad(p_ex) \
46         + as_vector([u_ex[0]/r**2,0.0])
47     def boundary(x, on_boundary):
48         return on_boundary
49     hh.append(mesh.hmax())
50 # Espacios funcionales
51 P2v = VectorFunctionSpace(mesh, "CG", 4)
52 P1 = FunctionSpace(mesh, "CG", 3)
53 R0 = FunctionSpace(mesh, "Real", 0)
54 #
55 Hh = BlockFunctionSpace([P2v, P1, R0])

```

```

56     trial = BlockTrialFunction(Hh)
57     u, p, la = block_split(trial)
58     test = BlockTestFunction(Hh)
59     v, q, mu = block_split(test)
60
61     print("DoFs = ", Hh.dim())
62     nn.append(Hh.dim())
63     bc = DirichletBC(Hh.sub(0),u_ex,boundary)
64     bcs = BlockDirichletBC([bc])
65     A=inner(mygrad(u),mygrad(v)*r)*dx
66     Bt=-p*mydiv(v)*r*dx
67     B= - q * mydiv(u) * r * dx
68     Ct= -la*q*r*dx
69     C= -mu*p*r*dx
70     #Ensamble sistema lineal
71     fv=(inner(fS, v*r))*dx
72     gv=(inner(p_ex,mu*r))*dx
73     rhs = [fv, 0, gv]
74
75     # Se ordenan las incognitas
76     #           u       p       la
77     lhs = [[ A,    Bt,    0], #v
78            [ B,    0,    Ct], #q
79            [ 0,    C,    0]] #mu
80
81     AA = block_assemble(lhs)
82     FF = block_assemble(rhs)
83     bcs.apply(AA)
84     bcs.apply(FF)
85     sol = BlockFunction(Hh)

```

```

86     block_solve(AA, sol.block_vector(), FF, "mumps")
87     u_h, p_h, la_h = block_split(sol)
88     # Calculo del error
89     eu.append(pow(assemble((u_h-u_ex)**2*r*dx \
90                     +(grad(u_h)-grad(u_ex))**2*r*dx \
91                     +(u_ex[0]-u_h[0])**2/r*dx),0.5))
92
93     ep.append(pow(assemble((p_h - p_ex)**2*r*dx),0.5))
94     if(nk>0):
95         ru.append(ln(eu[nk]/eu[nk-1])/ln(hh[nk]/hh[nk-1]))
96         rp.append(ln(ep[nk]/ep[nk-1])/ln(hh[nk]/hh[nk-1]))
97
98     # ***** Generacion de la tabla de errores **** #
99     print('=====')
100    print('\nn      &   hh      &   e(u)  & r(u) &   e(p)  & r(p) ')
101    print('=====')
102
103    for nk in range(nkmax):
104        print('{:5d} & {:.4f} & {:.8.4g} & {:.3f} & {:.8.4g} & {:.3f} \
105            '.format(nn[nk], hh[nk], eu[nk], ru[nk], ep[nk], rp[nk]))
106    print('=====')
107
108    # ***** #
109    # Aprovechamos y graficamos con la ultima malla
110    P2h = VectorFunctionSpace(mesh,'CG',4)
111    P1h = FunctionSpace(mesh,'CG',3)
112    uSh = project(u_h,P2h);
113    uSexa = project(u_ex,P2h);
114    pSh = project(p_h,P1h);
115    pSexa = project(p_ex,P1h);

```

```

116 plt.figure()
117 plot(uSexa[0], title="Velocidad u_r exacta")
118 plt.show()
119 plt.figure()
120 plot(uSh[0], title="Velocidad u_r calculada")
121 plt.show()
122 plt.figure()
123 plot(uSexa[1], title="Velocidad u_z exacta")
124 plt.show()
125 plt.figure()
126 plot(uSh[1], title="Velocidad u_z calculada")
127 plt.show()
128 plt.figure()
129 plot(pSexa, title="presion exacta")
130 plt.show()
131 plt.figure()
132 plot(pSh, title="presion calculada")
133 plt.show()

```

4.3. Códigos Stokes darcy acoplado

Código Stokes Darcy con elementos Taylor Hood

En este código se usa para realizar el analisis del error pero se puede modificar de manera directa para resolver un problema sin solución conocida. En el programa se escriben las soluciones exactas y el mismo código calcula el lado derecho del problema. Ademas se deben incluir términos de corrección para que la solución manufacturada cumpla las condiciones del problema.

```

1 # Programa para analizar errores formulacion
2 # StokesDarcy con elementos Taylor Hood

```

```

3  from dolfin import *
4  from multiphenics import *
5  from helpers import *
6  import sympy2fenics as sf
7  parameters["ghost_mode"] = "shared_facet" # required by dS
8  darcy = 10; stokes = 13; outlet = 14
9  inlet = 15; interf = 16; wallS = 17
10 wallD = 18; axis = 19; axiD = 20
11
12 # Definicion de los operadores diferenciales
13 epsilon = lambda vec: sym(grad(vec))
14 diva = lambda vec: div(vec) + vec[0]/r
15 str2exp = lambda s: sf.sympy2exp(sf.str2sympy(s))
16
17 def tdiva(u):
18     """ 2D axisymmetric tensor div: diva of rows """
19     nrows = u.ufl_shape[0]
20     return as_vector([diva(u[i,:]) for i in range(0,nrows)])
21
22 nu = Constant(1.)
23 alpha = Constant(1.)
24 kappa = Constant(1.)
25 gamma = Constant(1.)
26
27 # Solucion Exacta definida en Ervin 2013
28 u_str = '(-x*cos(pi*x)*sin(pi*y),-2/pi*cos(pi*x)*cos(pi*y)\
29 +x*sin(pi*x)*cos(pi*y))'
30 p_str = 'sin(pi*y)*(-cos(pi*x)+2*pi*x*sin(pi*x))\
31 +4*x*exp(-4*x)*cos(pi*y)-2/pi*(1-5*exp(-2))'
32

```

```

33 nkmax = 5
34
35 hh = []; nn = []; euS = []; ruS = [];
36 euD = []; ruD = []; epS = []; rpS = []
37 epD = []; rpD = []; elam = []; rlam = []
38
39 ruS.append(0.0); rpS.append(0.0); ruD.append(0.0);
40 rpD.append(0.0); rlam.append(0.0)
41
42 # ***** Analisis de error***** #
43
44 for nk in range(nkmax):
45     print("..... Refinement level : nk = ", nk)
46     nps = pow(2,nk+1)
47
48     mesh = RectangleMesh(Point(0,-1.0),Point(1.0,1.0),nps,2*nps, 'crossed')
49     x = SpatialCoordinate(mesh); r = x[0]
50     subdomains = MeshFunction("size_t", mesh, 2)
51     subdomains.set_all(0)
52     boundaries = MeshFunction("size_t", mesh, 1)
53     boundaries.set_all(0)
54
55     class Top(SubDomain):
56         def inside(self, x, on_boundary):
57             return (near(x[1], 1.0) and on_boundary)
58
59     class SRight(SubDomain):
60         def inside(self, x, on_boundary):
61             return (near(x[0], 1.0) and between(x[1], (0.0, 1.0)) and on_boundary)
62

```

```

63     class SLeft(SubDomain):
64         def inside(self, x, on_boundary):
65             return (near(x[0], 0.0) and between(x[1], (0.0, 1.0)) and on_boundary)
66
67     class DRight(SubDomain):
68         def inside(self, x, on_boundary):
69             return (near(x[0], 1.0) and between(x[1], (-1.0, 0.0)) and on_boundary)
70
71     class DLeft(SubDomain):
72         def inside(self, x, on_boundary):
73             return (near(x[0], 0.0) and between(x[1], (-1.0, 0.0)) and on_boundary)
74
75     class Bot(SubDomain):
76         def inside(self, x, on_boundary):
77             return (near(x[1], -1.0) and on_boundary)
78
79     class MStokes(SubDomain):
80         def inside(self, x, on_boundary):
81             return x[1]>=0.
82
83     class MDarcy(SubDomain):
84         def inside(self, x, on_boundary):
85             return x[1]<=0.
86
87     class Interface(SubDomain):
88         def inside(self, x, on_boundary):
89             return near(x[1], 0.0)
90
91     MDarcy().mark(subdomains, darcy)
92     MStokes().mark(subdomains, stokes)

```



```

93     Interface().mark(boundaries, interf)
94
95     Top().mark(boundaries, inlet)
96     SRight().mark(boundaries, wallS)
97     SLeft().mark(boundaries, axis)
98     DRight().mark(boundaries, wallD)
99     DLeft().mark(boundaries, axisD)
100    Bot().mark(boundaries, outlet)
101
102    n = FacetNormal(mesh); t = as_vector((-n[1], n[0]))
103    hh.append(mesh.hmax())
104
105    # ***** Creacion Subdominios, fronteras e interface ***** #
106
107    OmS = MeshRestriction(mesh, MStokes())
108    OmD = MeshRestriction(mesh, MDarcy())
109    Sig = MeshRestriction(mesh, Interface())
110    InL = MeshRestriction(mesh, Top())
111    OutL = MeshRestriction(mesh, Bot())
112    dx = Measure("dx", domain=mesh, subdomain_data=subdomains)
113    ds = Measure("ds", domain=mesh, subdomain_data=boundaries)
114    dS = Measure("dS", domain=mesh, subdomain_data=boundaries)
115
116    # ***** Soluciones exactas y creacion del lado derecho ***** #
117
118    u_ex = Expression(str2exp(u_str), degree=5, domain=mesh)
119    p_ex = Expression(str2exp(p_str), degree=5, domain=mesh)
120
121    gS = diva(u_ex)
122    gD = diva(u_ex)

```

```

123
124     fS = -tdiva(2.*nu*epsilon(u_ex)) + grad(p_ex) \
125           + 2*nu*as_vector([u_ex[0]/r**2,0.0])
126     fD = nu/kappa*u_ex + grad(p_ex)
127
128     #Estos terminos deben ser agregados para que la solucion exacta
129     # se ajuste a las condiciones del problema
130
131     la_ex = -p_ex('-')
132
133     sigma_ex = 2.*nu*epsilon(u_ex)-p_ex*Identity(2)
134     h1n_ex = dot(sigma_ex('+')*n('+'),n('+'))+p_ex('-')
135
136     h2t_ex = alpha*nu/sqrt(kappa)*dot(u_ex('+'),t('+')) \
137           + dot(sigma_ex('+')*n('+'),t('+'))
138
139     h1_ex = dot(2.*nu*epsilon(u_ex)*n,n)-p_ex
140
141     # Espacios Funcionales Taylor Hood
142     P2v = VectorFunctionSpace(mesh, "CG", 2)
143     P1  = FunctionSpace(mesh, "CG", 1)
144     RT0 = FunctionSpace(mesh, "RT", 1)
145     P0  = FunctionSpace(mesh, "DG", 0)
146     R0  = FunctionSpace(mesh, "Real", 0)
147
148     #                               uS, uD, pS, pD, la, beta0
149     Hh = BlockFunctionSpace([P2v, RT0, P1, P0, P1, R0],
150                             restrict=[OmS, OmD, OmS, OmD, Sig, []])
151
152     trial = BlockTrialFunction(Hh)

```

```

153     uS, uD, pS, pD, la, beta0 = block_split(trial)
154     test = BlockTestFunction(Hh)
155     vS, vD, qS, qD, ze, rho0 = block_split(test)
156
157     print("DoFs = ", Hh.dim())
158     nn.append(Hh.dim())
159
160     #Condiciones de borde #
161
162     uS_0 = project(u_ex, P2v)
163
164     bcUin  = DirichletBC(Hh.sub(0), uS_0, boundaries, inlet)
165     bcUS   = DirichletBC(Hh.sub(0), uS_0, boundaries, wallS)
166     bcUaS  = DirichletBC(Hh.sub(0).sub(0),project(Constant(0)\
167     ,Hh.sub(0).sub(0).collapse()), boundaries, axis)
168     bcUD   = DirichletBC(Hh.sub(1), u_ex, boundaries, wallD)
169     bcUout = DirichletBC(Hh.sub(1), u_ex, boundaries, outlet)
170     bcUaD  = DirichletBC(Hh.sub(1), Constant((0,0)), boundaries, axiD)
171
172     bcs = BlockDirichletBC([bcUin, bcUS, bcUD, bcUout])
173
174
175     # Formas bilineales #
176
177     af = 2.0 * nu * inner(epsilon(uS), epsilon(vS))*r * dx(stokes) \
178         + 2.0 * nu * uS[0]*vS[0]/r*dx(stokes) \
179         + alpha * nu/sqrt(kappa) * dot(uS(' '),t(' '))*dot(vS(' '),t(' '))*r(' ')*d
180     ap = nu/kappa * dot(uD, vD) * r * dx(darcy) \
181         + gamma * diva(uD) * diva(vD) * r * dx(darcy)
182

```

```

183     bf1t = - pS * diva(vS) * r * dx(stokes)
184     bf1  = - qS * diva(uS) * r * dx(stokes)
185     bp1t = - pD * diva(vD) * r * dx(darcy)
186     bp1  = - qD * diva(uD) * r * dx(darcy)
187
188     bI1t = avg(la) * dot(vS('++'), n('++')) * r('++') * dS(interf)
189     bI1  = avg(ze) * dot(uS('++'), n('++')) * r('++') * dS(interf)
190     bI2t = avg(la) * dot(vD('--'), n('--')) * r('--') * dS(interf)
191     bI2  = avg(ze) * dot(uD('--'), n('--')) * r('--') * dS(interf)
192
193     AuxD = pD*rho0*r*dx(darcy)
194     AuxS = pS*rho0*r*dx(stokes)
195     AuxSt= qS*beta0*r*dx(stokes)
196     AuxDt= qD*beta0*r*dx(darcy)
197     Grho0= p_ex*rho0*r*dx
198
199     FvS = dot(fS, vS) * r * dx(stokes) \
200           + h2t_ex*dot(vS('++'),t('++'))* r('++') *dS(interf) \
201           + h1n_ex*dot(vS('++'),n('++'))* r('++') *dS(interf)
202     FvD = dot(fD, vD) * r * dx(darcy) \
203           + gamma * gD * diva(vD) * r * dx(darcy)
204
205
206     GqS = -gS*qS *r* dx(stokes)
207     GqD = -gD*qD *r* dx(darcy)
208
209     # ***** Ensamble del sistema lineal ***** #
210
211     rhs = [FvS, FvD, GqS, GqD, 0, Grho0]
212

```

```

213     # Ordenando las incognitas
214     #          uS   uD   pS   pD   la   beta0
215     lhs = [[ af,    0, bf1t,    0, bI1t,    0], #vS
216            [ 0,   ap,    0, bp1t, bI2t,    0], #vD
217            [bf1,    0,    0,    0,    0,AuxSt], #qS
218            [ 0, bp1,    0,    0,    0,AuxDt], #qD
219            [bI1, bI2,    0,    0,    0,    0], #ze
220            [ 0,    0, AuxS, AuxD,    0,    0]] #rho0
221
222
223     AA = block_assemble(lhs)
224     FF = block_assemble(rhs)
225     bcs.apply(AA)
226     bcs.apply(FF)
227
228     sol = BlockFunction(Hh)
229     block_solve(AA, sol.block_vector(), FF, "mumps")
230     uS_h, uD_h, pS_h, pD_h, la_h, beta0_h = block_split(sol)
231
232     meshS = SubMesh(mesh, subdomains, stokes)
233     meshD = SubMesh(mesh, subdomains, darcy)
234     VhS = VectorFunctionSpace(meshS, 'CG', 2)
235     VhD = FunctionSpace(meshD, 'RT', 1)
236     QhS = FunctionSpace(meshS, 'CG', 1)
237     QhD = FunctionSpace(meshD, 'DG', 0)
238
239     uSh = project(uS_h, VhS); uDh = interpolate(uD_h, VhD)
240     pSh = project(pS_h, QhS); pDh = project(pD_h, QhD)
241
242     # Calculo del error

```

```

243
244     euS.append(pow(assemble((uS_h-u_ex)**2*r*dx(stokes) \
245                               +(grad(uS_h)-grad(u_ex))**2*r*dx(stokes) \
246                               +(u_ex[0]-uS_h[0])**2/r*dx(stokes)),0.5))
247
248     euD.append(pow(assemble((uD_h-u_ex)**2*r*dx(darcy) \
249                               +(diva(uD_h)-diva(u_ex))**2*r*dx(darcy)),0.5))
250     epS.append(pow(assemble((pS_h - p_ex)**2*r*dx(stokes)),0.5))
251     epD.append(pow(assemble((pD_h - p_ex)**2*r*dx(darcy)),0.5))
252     elam.append(pow(assemble((la_h - la_ex)**2*r*dS(interf)),0.5))
253
254     if(nk>0):
255         ruS.append(ln(euS[nk]/euS[nk-1])/ln(hh[nk]/hh[nk-1]))
256         rpS.append(ln(epS[nk]/epS[nk-1])/ln(hh[nk]/hh[nk-1]))
257         ruD.append(ln(euD[nk]/euD[nk-1])/ln(hh[nk]/hh[nk-1]))
258         rpD.append(ln(epD[nk]/epD[nk-1])/ln(hh[nk]/hh[nk-1]))
259         rlam.append(ln(elam[nk]/elam[nk-1])/ln(hh[nk]/hh[nk-1]))
260
261     # ***** Tablas de convergencia ***** #
262     print('=====')
263     print('nn      &   hh      &   e(uS)   & r(uS) &   e(pS)   & r(pS) &   e(uD)   & r(uD) &   e(lam)')
264     print('=====')
265
266     for nk in range(nkmax):
267         print('{:5d} & {:.4f} & {:.8.4g} & {:.3f} & {:.8.4g} & {:.3f} & {:.8.4g} & {:.3f} & {:.8.4g} & {:.3f}')
268     print('=====')

```

Código Stokes Darcy con elementos Mini

```

1  # Programa para analizar errores formulacion
2  # StokesDarcy con elementos Taylor Hood

```

```

3  from dolfin import *
4  from multiphenics import *
5  from helpers import *
6  import sympy2fenics as sf
7  parameters["ghost_mode"] = "shared_facet" # required by dS
8
9  darcy = 10; stokes = 13; outlet = 14
10 inlet = 15; interf = 16; wallS = 17
11 wallD = 18; axis = 19; axiD = 20
12 # Definicion de los operadores diferenciales
13 epsilon = lambda vec: sym(grad(vec))
14 diva = lambda vec: div(vec) + vec[0]/r
15 str2exp = lambda s: sf.sympy2exp(sf.str2sympy(s))
16
17 def tdiva(u):
18     """ 2D axisymmetric tensor div: diva of rows """
19     nrows = u.ufl_shape[0]
20     return as_vector([diva(u[i,:]) for i in range(0,nrows)])
21
22 nu = Constant(1.)
23 alpha = Constant(1.)
24 kappa = Constant(1.)
25 gamma = Constant(1.)
26 # Solucion Exacta definida en Ervin 2013
27
28 u_str = '(-x*cos(pi*x)*sin(pi*y),-2/pi*cos(pi*x)*cos(pi*y)+x*sin(pi*x)*cos(pi*y))'
29 p_str = 'sin(pi*y)*(-cos(pi*x)+2*pi*x*sin(pi*x))+4*x*exp(-4*x)*cos(pi*y)-2/pi*(1-5*'
30
31 nkmax = 5
32

```

```

33 hh = []; nn = []; euS = []; ruS = [];
34 euD = []; ruD = []; epS = []; rpS = []
35 epD = []; rpD = []; elam = []; rlam = []
36
37 ruS.append(0.0); rpS.append(0.0); ruD.append(0.0);
38 rpD.append(0.0); rlam.append(0.0)
39
40 # ***** Error analysis ***** #
41
42 for nk in range(nkmax):
43     print("..... Refinement level : nk = ", nk)
44     nps = pow(2,nk+1)
45
46     mesh = RectangleMesh(Point(0,-1.0),Point(1.0,1.0),nps,2*nps, 'crossed')
47     x = SpatialCoordinate(mesh); r = x[0]
48     subdomains = MeshFunction("size_t", mesh, 2)
49     subdomains.set_all(0)
50     boundaries = MeshFunction("size_t", mesh, 1)
51     boundaries.set_all(0)
52
53     class Top(SubDomain):
54         def inside(self, x, on_boundary):
55             return (near(x[1], 1.0) and on_boundary)
56
57     class SRight(SubDomain):
58         def inside(self, x, on_boundary):
59             return (near(x[0], 1.0) and between(x[1], (0.0, 1.0)) and on_boundary)
60
61     class SLeft(SubDomain):
62         def inside(self, x, on_boundary):

```



```

63         return (near(x[0], 0.0) and between(x[1], (0.0, 1.0)) and on_boundary)
64
65     class DRight(SubDomain):
66         def inside(self, x, on_boundary):
67             return (near(x[0], 1.0) and between(x[1], (-1.0, 0.0)) and on_boundary)
68
69     class DLeft(SubDomain):
70         def inside(self, x, on_boundary):
71             return (near(x[0], 0.0) and between(x[1], (-1.0, 0.0)) and on_boundary)
72
73     class Bot(SubDomain):
74         def inside(self, x, on_boundary):
75             return (near(x[1], -1.0) and on_boundary)
76
77     class MStokes(SubDomain):
78         def inside(self, x, on_boundary):
79             return x[1]>=0.
80
81     class MDarcy(SubDomain):
82         def inside(self, x, on_boundary):
83             return x[1]<=0.
84
85     class Interface(SubDomain):
86         def inside(self, x, on_boundary):
87             return near(x[1], 0.0)
88
89     MDarcy().mark(subdomains, darcy)
90     MStokes().mark(subdomains, stokes)
91     Interface().mark(boundaries, interf)
92

```

```

93     Top().mark(boundaries, inlet)
94     SRight().mark(boundaries, wallS)
95     SLeft().mark(boundaries, axis)
96     DRight().mark(boundaries, wallD)
97     DLeft().mark(boundaries, axisD)
98     Bot().mark(boundaries, outlet)
99
100     n = FacetNormal(mesh); t = as_vector((-n[1], n[0]))
101     hh.append(mesh.hmax())
102
103     # ***** Creacion Subdominios, fronteras e interface ***** #
104     OmS = MeshRestriction(mesh, MStokes())
105     OmD = MeshRestriction(mesh, MDarcy())
106     Sig = MeshRestriction(mesh, Interface())
107     InL = MeshRestriction(mesh, Top())
108     OutL = MeshRestriction(mesh, Bot())
109     dx = Measure("dx", domain=mesh, subdomain_data=subdomains)
110     ds = Measure("ds", domain=mesh, subdomain_data=boundaries)
111     dS = Measure("dS", domain=mesh, subdomain_data=boundaries)
112
113     # ***** Soluciones exactas y creacion del lado derecho ***** #
114     u_ex = Expression(str2exp(u_str), degree=5, domain=mesh)
115     p_ex = Expression(str2exp(p_str), degree=5, domain=mesh)
116
117     gS = diva(u_ex)
118     gD = diva(u_ex)
119
120     fS = -tdiva(2.*nu*epsilon(u_ex)) + grad(p_ex) \
121         + 2*nu*as_vector([u_ex[0]/r**2, 0.0])
122     fD = nu/kappa*u_ex + grad(p_ex)

```

```

123
124     #Estos terminos deben ser agregados para que la solucion exacta
125     # se ajuste a las condiciones del problema
126     la_ex = -p_ex('-')
127     sigma_ex = 2.*nu*epsilon(u_ex)-p_ex*Identity(2)
128     h1n_ex = dot(sigma_ex('+')*n('+'),n('+'))+p_ex('-')
129
130     h2t_ex = alpha*nu/sqrt(kappa)*dot(u_ex('+'),t('+')) \
131             + dot(sigma_ex('+')*n('+'),t('+'))
132
133     # Espacios Funcionales
134     P2v = VectorFunctionSpace(mesh, "CG", 2)
135     P1  = FunctionSpace(mesh, "CG", 1)
136     RT0 = FunctionSpace(mesh, "RT", 1)
137     P0  = FunctionSpace(mesh, "DG", 0)
138     R0  = FunctionSpace(mesh, "Real", 0)
139
140     aP1 = FiniteElement("CG", mesh.ufl_cell(), 1)
141     aBub = FiniteElement("Bubble", mesh.ufl_cell(), 3)
142     # Espacios Mini element
143     P1b = FunctionSpace(mesh, VectorElement(aP1 + aBub))
144
145
146     #                                uS, uD, pS, pD, la, beta0
147     Hh = BlockFunctionSpace([P1b, RT0, P1, P0, P1, R0],
148                             restrict=[OmS, OmD, OmS, OmD, Sig, []])
149
150     trial = BlockTrialFunction(Hh)
151     uS, uD, pS, pD, la, beta0 = block_split(trial)
152     test = BlockTestFunction(Hh)

```

```

153     vS, vD, qS, qD, ze, rho0 = block_split(test)
154
155     print("DoFs = ", Hh.dim())
156     nn.append(Hh.dim())
157
158     # Condiciones de frontera #
159     uS_0 = project(u_ex, P1b)
160
161     bcUin = DirichletBC(Hh.sub(0), uS_0, boundaries, inlet)
162     bcUS = DirichletBC(Hh.sub(0), uS_0, boundaries, wallS)
163     bcUaS = DirichletBC(Hh.sub(0).sub(0), project(Constant(0), Hh.sub(0).sub(0).col
164     bcUD = DirichletBC(Hh.sub(1), u_ex, boundaries, wallD)
165     bcUout = DirichletBC(Hh.sub(1), u_ex, boundaries, outlet)
166     bcUaD = DirichletBC(Hh.sub(1), Constant((0,0)), boundaries, axiD)
167
168     bcs = BlockDirichletBC([bcUin, bcUS, bcUD, bcUout])
169
170     # Formas bilineales #
171
172     af = 2.0 * nu * inner(epsilon(uS), epsilon(vS))*r * dx(stokes) \
173         + 2.0 * nu * uS[0]*vS[0]/r*dx(stokes) \
174         + alpha * nu/sqrt(kappa) * dot(uS('+'),t('+'))*dot(vS('+'),t('+'))*r('+')*d
175     ap = nu/kappa * dot(uD, vD) * r * dx(darcy) \
176         + gamma * diva(uD) * diva(vD) * r * dx(darcy)
177
178     bf1t = - pS * diva(vS) * r * dx(stokes)
179     bf1 = - qS * diva(uS) * r * dx(stokes)
180     bp1t = - pD * diva(vD) * r * dx(darcy)
181     bp1 = - qD * diva(uD) * r * dx(darcy)
182

```

```

183     bI1t = avg(la) * dot(vS('+'), n('+')) * r('+') * dS(interf)
184     bI1  = avg(ze) * dot(uS('+'), n('+')) * r('+') * dS(interf)
185     bI2t = avg(la) * dot(vD('-'), n('-')) * r('-') * dS(interf)
186     bI2  = avg(ze) * dot(uD('-'), n('-')) * r('-') * dS(interf)
187
188     AuxD = pD*rho0*r*dx(darcy)
189     AuxS = pS*rho0*r*dx(stokes)
190     AuxSt= qS*beta0*r*dx(stokes)
191     AuxDt= qD*beta0*r*dx(darcy)
192     Grho0= p_ex*rho0*r*dx
193
194     FvS = dot(fS, vS) * r * dx(stokes) \
195           + h2t_ex*dot(vS('+'),t('+'))* r('+') *dS(interf) \
196           + h1n_ex*dot(vS('+'),n('+'))* r('+') *dS(interf)
197     FvD = dot(fD, vD) * r * dx(darcy) \
198           + gamma * gD * diva(vD) * r * dx(darcy)
199     GqS = -gS*qS *r* dx(stokes)
200     GqD = -gD*qD *r* dx(darcy)
201     # ***** Ensamble del sistema lineal ***** #
202     rhs = [FvS, FvD, GqS, GqD, 0, Grho0]
203     # Incognitas ordenadas
204     #          uS    uD    pS    pD    la    beta0
205     lhs = [[ af,    0, bf1t,    0, bI1t,    0], #vS
206            [ 0,   ap,    0, bp1t, bI2t,    0], #vD
207            [bf1,    0,    0,    0,    0,AuxSt], #qS
208            [ 0,   bp1,    0,    0,    0,AuxDt], #qD
209            [bI1, bI2,    0,    0,    0,    0], #ze
210            [ 0,    0, AuxS, AuxD,    0,    0]] #rho0
211
212     AA = block_assemble(lhs)

```

```

213     FF = block_assemble(rhs)
214     bcs.apply(AA)
215     bcs.apply(FF)
216     sol = BlockFunction(Hh)
217     block_solve(AA, sol.block_vector(), FF, "mumps")
218     uS_h, uD_h, pS_h, pD_h, la_h, beta0_h = block_split(sol)
219
220     # Calculo del error
221
222     euS.append(pow(assemble((uS_h-u_ex)**2*r*dx(stokes) \
223                          +(grad(uS_h)-grad(u_ex))**2*r*dx(stokes) \
224                          +(u_ex[0]-uS_h[0])**2/r*dx(stokes))),0.5))
225
226     euD.append(pow(assemble((uD_h-u_ex)**2*r*dx(darcy) \
227                          +(diva(uD_h)-diva(u_ex))**2*r*dx(darcy))),0.5))
228     epS.append(pow(assemble((pS_h - p_ex)**2*r*dx(stokes)),0.5))
229     epD.append(pow(assemble((pD_h - p_ex)**2*r*dx(darcy)),0.5))
230     # elam.append(pow(assemble((la_h - h1_ex('+'))**2*r*dS(interf)),0.5))
231     elam.append(pow(assemble((la_h - la_ex)**2*r*dS(interf)),0.5))
232
233     if(nk>0):
234         ruS.append(ln(euS[nk]/euS[nk-1])/ln(hh[nk]/hh[nk-1]))
235         rpS.append(ln(epS[nk]/epS[nk-1])/ln(hh[nk]/hh[nk-1]))
236         ruD.append(ln(euD[nk]/euD[nk-1])/ln(hh[nk]/hh[nk-1]))
237         rpD.append(ln(epD[nk]/epD[nk-1])/ln(hh[nk]/hh[nk-1]))
238         rlam.append(ln(elam[nk]/elam[nk-1])/ln(hh[nk]/hh[nk-1]))
239
240     # ***** Tablas de error ***** #
241     print('=====')
242     print('\nn      &   hh      &   e(uS)      & r(uS) &   e(pS)      & r(pS) &   e(uD)      & r(uD) &

```

```

243 print('=====')
244
245 for nk in range(nkmax):
246     print('{:5d} & {:.4f} & {:8.4g} & {:.3f} & {:8.4g} & {:.3f} & {:8.4g} & {:.3f}')
247 print('=====')

```

4.4. Ejemplo de aplicación

```

1 from __future__ import print_function
2 from dolfin import *
3 from multiphenics import *
4 #from mshr import *
5 import numpy as np
6 from helpers import *
7 import matplotlib.pyplot as plt
8
9 parameters["form_compiler"]["representation"] = "uflacs"
10 parameters["form_compiler"]["cpp_optimize"] = True
11 parameters["allow_extrapolation"] = True
12 parameters["ghost_mode"] = "shared_facet" # requerida para ds
13
14 mesh = Mesh("axiSymmEyeCoarse.xml")
15
16 # ***** Constantes del modelo ***** #
17
18 x = SpatialCoordinate(mesh); r = x[0]
19
20 epsilon = lambda vec: sym(grad(vec))
21 diva = lambda vec: div(vec) + vec[0]/r
22

```

```

23 kappa = Constant(2.0e-6)
24 nu= Constant(0.66)
25 alpha = Constant(1.0)
26
27 gamma = Constant(1.0)
28 t = 0.0; dt = 0.1; tfinal = 2;
29 #pIn = Constant(933.254)      # Pa
30 pOut = Constant(0.)
31 v0 = Constant(4.89e-3)      # m/s
32 fS = Constant((0.,0.))
33 fD = fS
34 gS = Constant(0.)
35 gD = gS
36
37 darcy = 4001; stokes = 4002; outlet = 1922;
38 interf = 1923; wallS = 1925; wallD = 1924;
39 axis = 1927; inlet = 1921
40
41 # ***** Subdominios, fronteras e interface ***** #
42
43 subdomains = MeshFunction("size_t", mesh, "axiSymmEyeCoarse_physical_region.xml")
44 boundaries = MeshFunction("size_t", mesh, "axiSymmEyeCoarse_facet_region.xml")
45
46 OmD = generate_subdomain_restriction(mesh, subdomains, darcy)
47 OmS = generate_subdomain_restriction(mesh, subdomains, stokes)
48 Sig = generate_interface_restriction(mesh, subdomains, {darcy, stokes})
49
50 meshF = SubMesh(mesh, subdomains, stokes)
51 meshP = SubMesh(mesh, subdomains, darcy)
52

```



```

53  # Separamos los subdominios para viasualizarlos mejor
54  shift = Expression(["0.0001", "-0.00002"], degree = 0)
55  ALE.move(meshP,shift)
56
57  n = FacetNormal(mesh);tt = as_vector((-n[1], n[0]))
58
59  dx = Measure("dx", domain=mesh, subdomain_data=subdomains)
60  ds = Measure("ds", domain=mesh, subdomain_data=boundaries)
61
62  # Elementos finitos con sus restricciones** #
63  P2v = VectorFunctionSpace(mesh, "CG", 2)
64  P1  = FunctionSpace(mesh, "CG", 1)
65  RT0 = FunctionSpace(mesh,"RT",1)
66  P0  = FunctionSpace(mesh, "DG", 0)
67  R0  = FunctionSpace(mesh,"Real", 0)
68
69  # uS, uD, pS, pD, la, beta0
70  Hh = BlockFunctionSpace([P2v, RT0, P1, P0, P1],
71                          restrict=[0mS, 0mD, 0mS, 0mD, Sig])
72
73  trial = BlockTrialFunction(Hh)
74  uS, uD, pS, pD, la = block_split(trial)
75  test = BlockTestFunction(Hh)
76  vS, vD, qS, qD, ze = block_split(test)
77
78  print("DoFs = ", Hh.dim())
79
80  # datos iniciales
81  uSold = project(Constant((0,0)),Hh.sub(0))
82

```

```

83  af = 1/dt * dot(uS, vS) * r * dx(stokes) \
84      + 2.0 * nu * inner(epsilon(uS), epsilon(vS))*r * dx(stokes) \
85      + 2.0 * nu * uS[0]*vS[0]/r*dx(stokes) \
86      + alpha * nu/sqrt(kappa) * dot(uS('+'),tt('+'))*dot(vS('+'),tt('+'))*r('+')
87
88  ap = nu/kappa * dot(uD, vD) * r * dx(darcy) \
89      + gamma * diva(uD) * diva(vD) * r * dx(darcy)
90
91  bf1t = - pS * diva(vS) * r * dx(stokes)
92  bf1  = - qS * diva(uS) * r * dx(stokes)
93  bp1t = - pD * diva(vD) * r * dx(darcy)
94  bp1  = - qD * diva(uD) * r * dx(darcy)
95
96  bI1t = avg(la) * dot(vS('+'), n('+')) * r('+') * dS(interf)
97  bI1  = avg(ze) * dot(uS('+'), n('+')) * r('+') * dS(interf)
98  bI2t = avg(la) * dot(vD('-'), n('-')) * r('-') * dS(interf)
99  bI2  = avg(ze) * dot(uD('-'), n('-')) * r('-') * dS(interf)
100
101  FvS = 1/dt * dot(uSold, vS) * r * dx(stokes)\
102      +dot(fS, vS) * r * dx(stokes) #\
103
104
105  FvD = dot(fD, vD) * r * dx(darcy) \
106      + gamma * gD * diva(vD) * r * dx(darcy)
107
108  GqS = -gS*qS *r* dx(stokes)
109  GqD = -gD*qD *r* dx(darcy)
110
111      # ***** Ensamble del sistema lineal ***** #
112

```

```

113 rhs = [FvS, FvD, GqS, GqD, 0]
114
115
116      #      uS   uD   pS      pD   la   beta0
117 lhs = [[ af,    0, bf1t,    0, bI1t], #vS
118        [ 0, ap,    0, bp1t, bI2t], #vD
119        [bf1,    0,    0,    0,    0], #qS
120        [ 0, bp1,    0,    0,    0], #qD
121        [bI1, bI2,    0,    0,    0]] #ze
122
123 plt.show()
124 plt.figure()
125 plot(uSold[0], title="Velocidad inicial")
126 plt.show()
127
128 # ***** Ciclo Temporal ***** #
129 while(t < tfinal):
130     t += dt; print("t=%.3f" % t)
131     mvin=-v0*pow(sin(pi*t),2)
132     vin = Expression('-v0*pow(sin(pi*t),2)', v0=v0, t=t, degree=2)
133     #     print("vin=%.3f" % vin)
134     vin.t = t
135
136     # ***** Condiciones de borde esenciales ***** #
137     zerov = Constant((0.,0.))
138
139     bcUin = DirichletBC(Hh.sub(0).sub(1),vin , boundaries, inlet)
140     bcUS = DirichletBC(Hh.sub(0), zerov, boundaries, wallS)
141     bcUA = DirichletBC(Hh.sub(0).sub(0), Constant(0), boundaries, axis)
142     #bcUD = DirichletBC(Hh.sub(1), u_ex, boundaries, wallD)

```

```

143     #bcUout = DirichletBC(Hh.sub(1), u_ex, boundaries, outlet)
144     bcpD = DirichletBC(Hh.sub(3), pOut, boundaries, outlet)
145     bcs = BlockDirichletBC([bcUin,bcUS, bcUA,bcpD])
146
147     AA = block_assemble(lhs)
148     FF = block_assemble(rhs)
149
150     bcs.apply(AA)
151     bcs.apply(FF)
152     sol = BlockFunction(Hh)
153     block_solve(AA, sol.block_vector(), FF)
154     uS_h, uD_h, pS_h, pD_h, la_h= block_split(sol)
155
156     assign(uSold,uS_h);
157
158     # Si queremos visualizar en las mallas por aparte:
159
160     meshS = SubMesh(mesh, subdomains, stokes)
161     meshD = SubMesh(mesh, subdomains, darcy)
162     VhS = VectorFunctionSpace(meshS, 'CG',1)
163     VhD = FunctionSpace(meshD, 'RT',1)
164     QhS = FunctionSpace(meshS, 'CG',1)
165     QhD = FunctionSpace(meshD, 'DG',0)
166
167     uSh = project(uS_h,VhS)
168     uDh = interpolate(uD_h,VhD)
169     pSh = project(pS_h,QhS)
170     pDh = project(pD_h,QhD)
171
172     plt.show()

```

```
173 plt.figure()
174 plot(uS_h[0])
175 plt.show()
176
177 plt.show()
178 plt.figure()
179 plot(uS_h[1])
180 plt.show()
```

Bibliografía

- [1] Z. Belhachmi, C. Bernardi, & S. Deparis. *Weighted Clément operator and application to the finite element discretization of the axisymmetric Stokes problem*, *Numer. Math.* **105**,(2002) 217–247.
- [2] B. Mercier,C. & Raugel, G.*Resolution d'un problème aux limites dans un ouvert axisymétrique par éléments finis en r , z et séries de Fourier en θ* , *RAIRO, Anal. Numér.* **16**, 405–461.
- [3] Babuska, I. *Error bounds for finite element method*.Numer. Math, **16** (1971) 322–333.
- [4] Brezzi, F. *On the existence, uniqueness and aproximation of saddle-point problems arising from Lagrange multipliers*. Rev. Française Automat. Informat. Recherche Opérationnelle Sér. Rouge, **8** (1974) 129–151.
- [5] A. Ern, J.L. Guermond. *Theory and Practice of Finite Elements* Springer Series in Applied Mathematical Sciences , Vol. 159 (2004) 530 p., Springer-Verlag, New York.
- [6] Susanne C. Brenner, L. Ridway Scott *The Mathematical Theory of Finite Element Methods*. Department of Mathematics and Center for Computation and Technology Louisiana State University. Baton Rouge, USA and University of Chicago Chicago, USA.
- [7] J. Albery, Keil, C. Carstensen, Viena, S. A. Funken, Keil and R. Klose, Keil. *Matlab Implementation of the Finite Element Methods in Elasticity*.

- [8] V.Ervin.*Approximation of coupled Stokes-Darcy flow in an axisymmetric domain. Comput. Methods Appl. Mech. Engrg.***258** (2013), 96–108.
- [9] W. Layton, F. Schieweck, I. Yotov. *Coupling fluid flow with porous media flow.**SIAM J. Numer. Anal.***40** (2002), 2195–2218.
- [10] R. Ruiz-Baier, M. Taffetani,H. Westermeyer, I. Yotov.*The Biot-Stokes coupling using total pressure: formulation, analysis and application to interfacial flow in the eye.**Comput. Methods Appl. Mech. Engrg.***389** (2022).