Proyecto FEniCS con Python

Martín A. Díaz Viera[1], Sinai Morales Chávez[2]

Insituto Mexicano del Petróleo
mdiazv@imp.mx[1]
geomorales91@gmail.com[2]

Posgrado del Instituto Mexicano del Petróleo

21 de Abril, 2023

Contenido I

Proyecto FEniCS

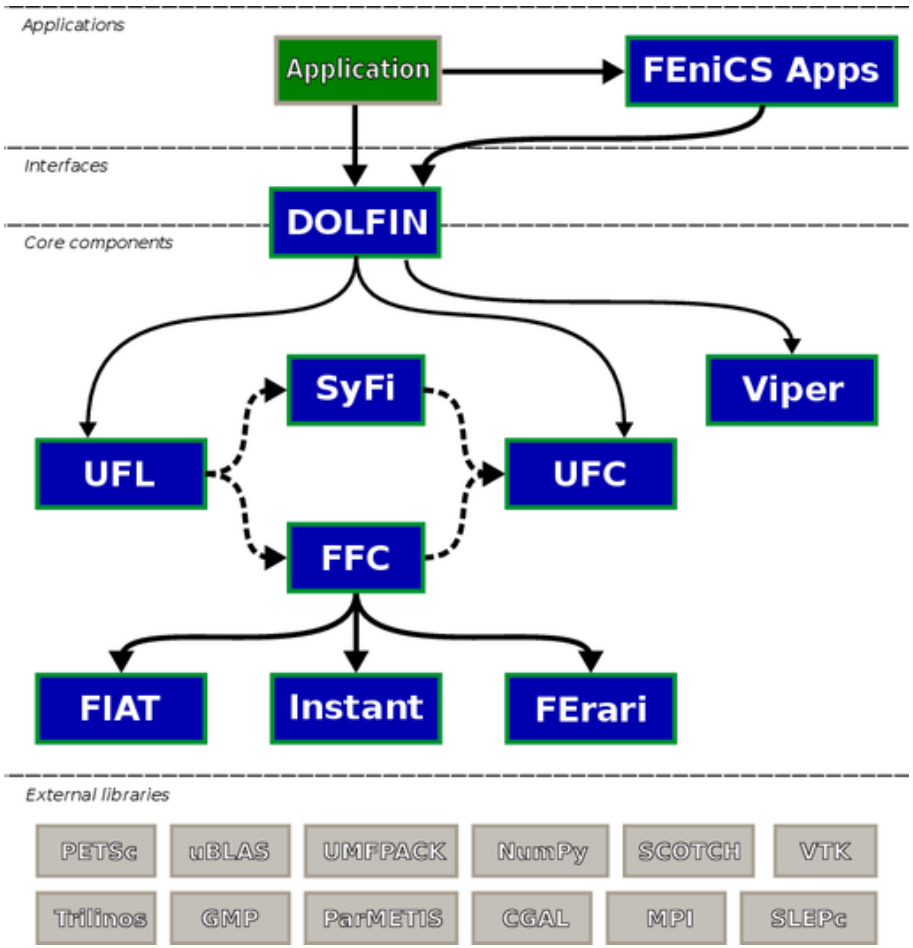
FEniCS es un proyecto de código abierto apoyado fiscalmente por NumFOCUS. Es desarrollado y mantenido por una comunidad global de científicos y desarrolladores de software.

`https://fenicsproject.org`

Proyecto FEniCS

- Es una plataforma de cómputo de código abierto para automatizar la solución de ecuaciones diferenciales parciales (EDP).
- Disponible para diferentes sistemas operativos y funciona en multitud de plataformas.
- Su elevado nivel de abstracción permite traducir los modelos matemáticos y numéricos en código eficiente de elemento finito.
- Fácil implementación utilizando el lenguaje de Python, además ofrece potentes funciones para programadores experimentados.
- Dos versiones: FEniCSx y **FEniCS legacy 2019**.

Proyecto FEniCS



- **DOLFIN** - Entorno de resolución de problemas.
- **FFC** - Compilador de las formas variacionales de elementos finitos.
- **FIAT** - Generación automática de funciones base de elementos finitos.
- **UFC** - Entorno unificado para el ensamble de elementos finitos.
- **UFL** - Interfaces flexibles para la elección de espacios de elementos finitos y notación cercana a la matemática.

Lenguaje de programación Python

Python fue creado por Guido van Rossum. Su primer lanzamiento fue en 1991. Versión actual: 3.11.2.

`https://www.python.org/`

Lenguaje de programación Python

- Lenguaje de alto nivel de programación interpretado, interactivo y orientado a objetos.
- Tipo de sistema dinámico y gestión automático de la memoria.
- Paradigmas múltiples de programación: orientado a objetos, imperativo, funcional y procedimental.
- Cuenta con una gran lista de bibliotecas fáciles de implementar.

Lenguaje de programación Python

- Combina una notable potencia con una sintaxis clara y sencilla.
- Corre en diferentes variantes de Unix como Linux y macOS. También en Windows.
- CPython, la implementación referencia de Python, es un software de código abierto.
- Modelo de desarrollo basado en la comunidad.
- Python y CPython son gestionados por la Fundación sin fines de lucro de Python.

IDEs y Notebooks

- Integrated Development Enviroment (IDE) o Entorno de Desarrollo Integrado (EDI).
- Un conjunto de herramientas para facilitar la edición de código fuente, construcciones automáticas y depurado.
- Notebook platforms o cuadernos de trabajo son similares a los IDEs pero presentado en un formato diferente.
- Opciones: **Jupyter**, Spyder, Visual Studio Code, PyCharm.

JupyterLab, Jupyter Notebook y Google Colab

- JupyterLab es el más reciente IDE de Jupyter basado en web (ejecuta código a través del navegador) para crear cuadernos de trabajo.
- Jupyter Notebook es el IDE original para crear cuadernos de trabajo.

- Google Colab son cuadernos de trabajo de Jupyter Notebook basados en la nube de Google.

Instalación del Software

- Para conocer las diferentes opciones de instalación y sus instrucciones, es necesario consultar el archivo **Instrucciones FEniCS.pdf**. El cual se encuentra disponible en la carpeta *Software* del curso.

Ejemplo de código en Python

- Se creó un cuaderno de trabajo utilizando el IDE Jupyter en donde se muestra una breve introducción a la programación empleando el lenguaje de Python, el archivo se llama **Introducción a Jupyter y Python.ipynb** y se encuentra disponible en la carpeta *Software* $\rightarrow$ *Notebooks* del curso.

Problema de Poisson

Uno de los ejemplos más sencillos para programar en FEniCS es la ecuación de Poisson en $\mathbb{R}^2$,

$$-\nabla \cdot (\nabla u(\underline{x})) = f(\underline{x}), \quad \forall \underline{x} \in \Omega \tag{1}$$

$$u(\underline{x}) = u_{\partial}(\underline{x}), \quad \forall \underline{x} \in \partial\Omega \tag{2}$$

donde:

- $u(\underline{x})$ es la función desconocida,
- $f(\underline{x}) = 2\pi^2 sen(\pi x) sen(\pi y)$ es una función prescrita,
- $\Omega = [0,1] \times [0,1]$ es el dominio espacial,
- $u_{\partial}(\underline{x}) = 0$ es una fución prescrita,
- $\partial\Omega$ es la frontera de Ω.

Formulación Variacional

Para reformular la ecuación de Poisson como un problema variacional, primero multiplicamos la ecuación por una función de peso v e integramos:

$$-\int_{\Omega}(\nabla\cdot(\nabla u))v\ d\underline{x}=\int_{\Omega}f\ v\ d\underline{x} \tag{3}$$

donde:

- $u\in V=\{u\in H^1(\Omega):u(\underline{x})=u_\partial(\underline{x}),\forall\underline{x}\in\partial\Omega_D\}$
- $v\in\hat{V}=\{v\in H^1(\Omega):v(\underline{x})=0,\forall\underline{x}\in\partial\Omega_D\}$

$H^1(\Omega)$ es el espacio de Sobolev que contiene a la función v tal que v^2 y $||\nabla v||^2$ tienen integrales finitas sobre Ω.

Formulación Variacional

Aplicando la integración por partes a las integrales con derivadas de segundo orden:

$$\int_{\Omega} \nabla u \cdot \nabla v \; d\underline{x} - \int_{\partial\Omega_D} v(\nabla u) \cdot \underline{n} \; d\underline{x} = \int_{\Omega} f \; v \; d\underline{x}, \quad \forall v \in \hat{V} \qquad (4)$$

La función de peso $v \in \hat{V}$, por lo que $v = 0$ en la frontera $\partial\Omega_D$, de tal manera que el segundo término del lado derecho de (4) desaparece.

$$\int_{\Omega} \nabla u \cdot \nabla v \; d\underline{x} = \int_{\Omega} f \; v \; d\underline{x} \qquad \forall v \in \hat{V} \qquad (5)$$

La ecuación (5) es la forma débil del problema con valores en la frontera (1)-(2).

Discretización por Elementos Finitos

Para resolver la ecuación de Poisson numéricamente, es necesario transformar el problema variacional continuo (5) en un problema variacional discreto. Esto se realiza introduciendo espacios dimensionales finitos de base y de peso, comúnmente denotados como $\hat{V}_h \subset \hat{V}$ y $V_h \subset V$. El problema variacional discreto es: encontrar $u_h \in V_h \subset V$ tal que

$$a(u_h, v_h) = L(v_h), \quad \forall v_h \in \hat{V}_h \subset \hat{V} \tag{6}$$

donde

$$a(u_h, v_h) = \int_\Omega \nabla u_h \cdot \nabla v_h \; d\underline{x} \qquad y \qquad L(v_h) = \int_\Omega f \; v_h \; d\underline{x} \tag{7}$$

$a(u_h, v_h)$ es la forma bilineal y $L(v_h)$ es la forma lineal.

Discretización del Sistema de Ecuaciones

Escogiendo una base para el espacio discreto de funciones:

$$V_h = \{\phi\}_{j=1}^{N} \tag{8}$$

Esto es, vamos de un problema abstracto que se aplica a cualquier base a un sistema lineal concreto en una base dada. Entonces, hacemos una estimación para la solución discreta:

$$u_h = \sum_{j=i}^{N} U_j \phi_j \tag{9}$$

Discretización del Sistema de Ecuaciones

Probando con las funciones de base:

$$\int_\Omega \nabla \underbrace{(\sum_{j=i}^{N} U_j \phi_j)}_{u_h} \cdot \nabla \phi_i \; d\underline{x} = \int_\Omega f \phi_i \; d\underline{x} \tag{10}$$

Reordenando la ecuación:

$$\sum_{j=i}^{N} U_j \underbrace{\int_\Omega \nabla \phi_j \cdot \nabla \phi_i \; d\underline{x}}_{A_{ij}} = \underbrace{\int_\Omega f \phi_i \; d\underline{x}}_{b_i} \tag{11}$$

Discretización del Sistema de Ecuaciones

Resultando en el sistema de ecuaciones:

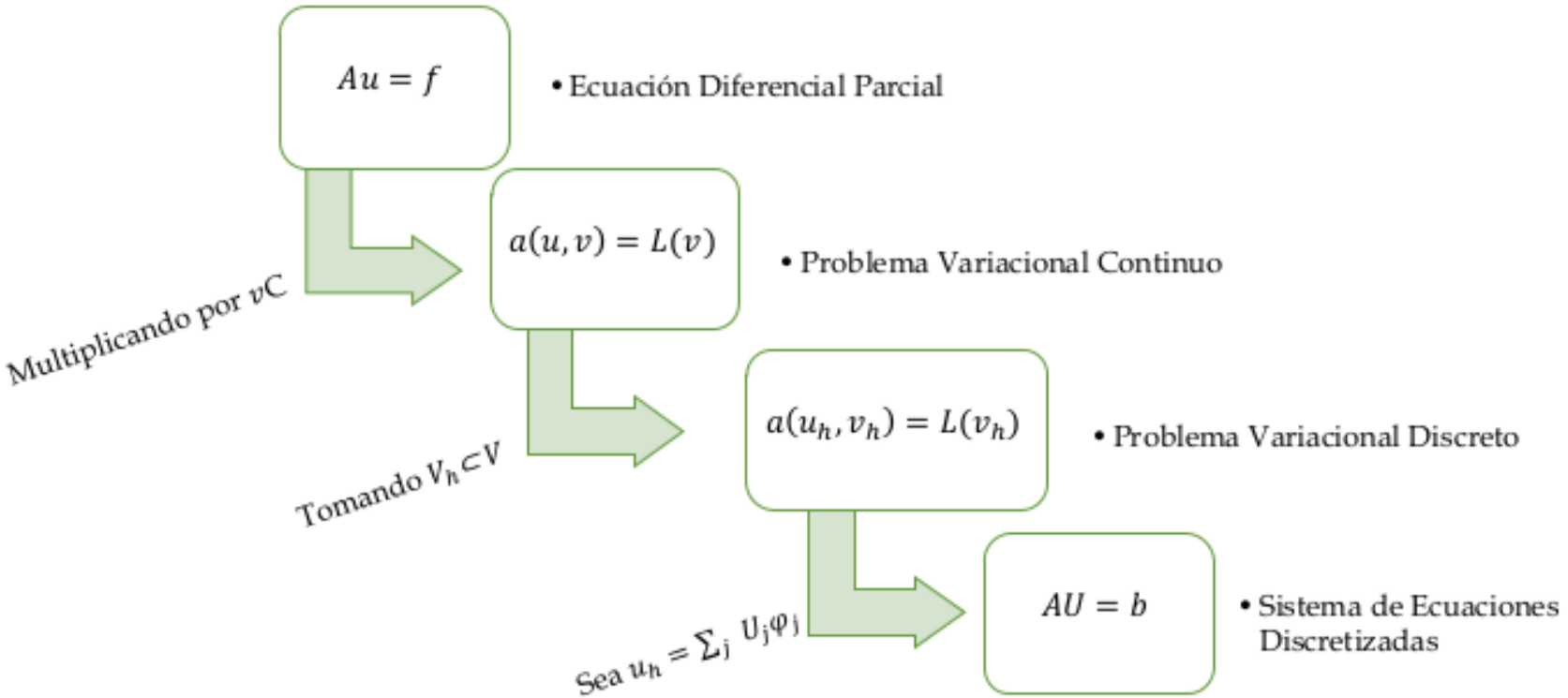
$$AU = b \tag{12}$$

donde:

$$A_{ij} = \int_{\Omega} \nabla \phi_j \cdot \nabla \phi_i \; d\underline{x} \tag{13}$$

$$b_i = \int_{\Omega} f \phi_i \; d\underline{x} \tag{14}$$

En resumen, para aproximar la solución de un problema empleando MEF en FEniCS:



Código de FEniCS en Python

```
# Importar la biblioteca de FEniCS:
from fenics import *
# Crea la malla y define el espacio de funciones
mesh = UnitSquareMesh(8, 8)
V = FunctionSpace(mesh, 'CG', 1)
# Definir las funciones de base y de peso
u = TrialFunction(V)
v = TestFunction(V)
# Definir las fronteras
def boundary(x, on_boundary):
    return on_boundary
# Define la condicion de frontera
u_d = Constant(0.0)
bc = DirichletBC(V, u_d, boundary)
```

Código de FEniCS en Python

```
1  # Define la variable f usando la clase Expression
2  f = 2*pi**2*Expression('sin(pi*x[0])*sin(pi*x[1])',
       degree=4)
3
4  # Define el problema variacional(usando operadores UFL
       )
5  a = inner(grad(u),grad(v))*dx
6  L = f*v*dx
7
8  # Calcula la solucion
9  u_h = Function(V)
10 solve(a == L, u_h, bc)
11
12 # Grafica la solucion
13 plot(u_h)
```

Código de FEniCS en Python

La primera línea del código,

```
1 from fenics import *
```

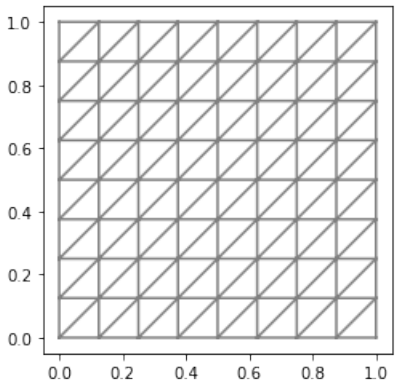
importa las clases necesarias de la biblioteca **DOLFIN**, la cual contiene clases eficientes y convenientes en el lenguaje de C++ para el cómputo de elemento finito, **fenics** en un paquete de Python que brinda acceso a esta biblioteca en C++ desde programas de Python.

Código de FEniCS en Python

Como dominio estándar se utilizará un cuadrado unitario empleando la clase **UnitSquareMesh**.

```
1 mesh = UnitSquareMesh(8, 8)
```

Los parámetros (8, 8) indican que cada lado del cuadrado será dividido en rectángulos de 8×8, y a su vez cada uno estará dividido en dos triángulos.



La malla está formada por celdas, que son triángulos de lados rectos.

Código de FEniCS en Python

Teniendo la malla, podemos definir el espacio de funciones V sobre la malla:

```
1 V = FunctionSpace(mesh, 'CG', 1)
```

El segundo argumento especifica el tipo de elemento ('CG' o 'Lagrange'), mientras que el tercer argumento especifica el grado del polinomio de las funciones base en el elemento.

Por lo que, para este caso, nuestro espacio V consiste en funciones de elementos finitos de Lagrange continuos de primer orden (o, en otras palabras, polinomios lineales continuos a trozos), los cuales son triángulos con nodos en los tres vértices.

Otras familias de elementos finitos se pueden consultar en `http://femtable.org/`

Código de FEniCS en Python

En matemáticas se distingue entre los espacios de base y de peso V y $\hat{V}$. La única diferencia en el problema presente son las condiciones de frontera.

$$V = \{u \in H^1(\Omega) : u(\underline{x}) = u_\partial(\underline{x}),\ \forall \underline{x} \in \partial\Omega_D\}$$

$$\hat{V} = \{v \in H^1(\Omega) : v(\underline{x}) = 0,\ \forall \underline{x} \in \partial\Omega_D\}$$

En **FEniCS** no se especifican las condiciones de frontera como parte del espacio de funciones, por lo que es suficiente trabajar con un espacio común V para las funciones base y de peso:

```
1 u = TrialFunction(V) #Funcion base
2 v = TestFunction(V)  #Funcion de peso
```

Código de FEniCS en Python

La información acerca de la frontera $\partial\Omega$ se escribe como una función:

```
1 def boundary(x, on_boundary):
2     return on_boundary
```

La función **boundary** que marca la frontera, regresa un valor **booleano**: **Verdadero** si el punto **x** dado se encuentra en la frontera y **Falso** en caso contrario.

El argumento **on_boundary** está dado por DOLFIN y es **Verdadero** si **x** está en la frontera física de la malla.

La función **boundary** será llamada para cada punto discreto en la malla,la cual nos permite tener fronteras donde u se conoce, incluso dentro del dominio, si se desea.

Código de FEniCS en Python

$u_\partial = 0$ se puede representar más eficiente si se representa como un objeto constante:

```
1 u_d = Constant(0.0)
```

El siguiente paso es especificar la condición de frontera: $u = u_\partial$ en $\partial\Omega$.

```
1 bc = DirichletBC(V, u_d, boundary)
```

DirichletBC es la clase de DOLFIN que contiene la información acerca de las condiciones frontera, en donde **u_d** es una instancia que contiene los valores de u_∂, y **boundary** es una función (u objeto) que describe si un punto se encuentra en la frontera donde se especifica u.

Código de FEniCS en Python

La variable f se refiere a un objeto **Expression**, la cual se emplea para representar una función matemática. La construcción típica es:

```
1 f = Expression('formula', degree=4)
```

donde 'formula' es una línea que contiene la expresión matemática.

Esta fórmula es escrita con la sintaxis de C++ (la expresión es convertida automáticamente en una función eficiente compilada en C++).

Código de FEniCS en Python

Las variables independientes en la función **Expression** se suponen disponibles como un vector punto $\underline{x}$, donde:

x[0] corresponde a la coordenada **x**,
x[1] a la coordenada **y**,
x[2] a la coordenada **z** (en un problema en tres dimensiones).

Con nuestra elección de $f(x, y) = 2\pi^2 sin(\pi x) sin(\pi y)$, la línea se escribe como:

```
1 f = 2*pi**2*Expression('sin(pi*x[0])*sin(pi*x[1])',
      degree=4)
```

Código de FEniCS en Python

Ahora que se tienen todos los objetos que se necesitan para especificar $a(u, v)$ y $L(v)$ de este problema:

```
1 a = inner(grad(u),grad(v))*dx
2 L = f*v*dx
```

En escencia, estas dos lineas especifican la EDP que hay que resolver. Es de notar la estrecha correspondencia entre la sintaxis de Python y las fórmulas matemáticas $\int \nabla u \cdot \nabla\ v\ d\underline{x}$ y $\int f\ v\ d\underline{x}$.

El lenguaje empleado para expresar las formas débiles es llamado UFL (Unified Form Language) y es una parte integral de FEniCS.

La función **grad** es consistente con el tensol algebraico comunmente empleado para derivar vectores y tensores de EDP, donde ∇ actúa como un operador vectorial.

Código de FEniCS en Python

Teniendo definidos a **a** y **L**, y la información acerca de las condiciones de frontera esenciales (Dirichlet) en **bc**, se puede calcular la solución, una función de elemento finito u:

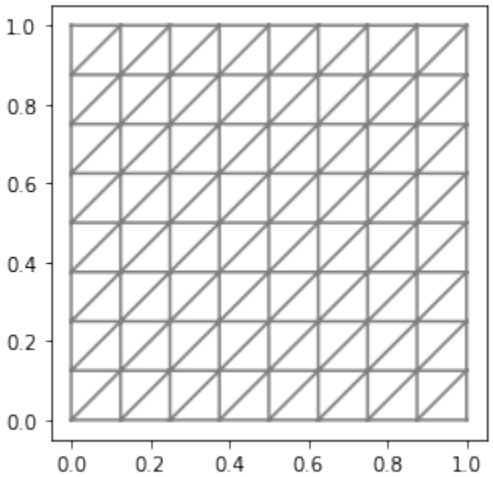
```
1 u_h = Function(V)
2 solve(a == L, u_h, bc)
```

u_h es un objeto **Function** que representa a la solución; esto es, la función de elemento finito calculada u.

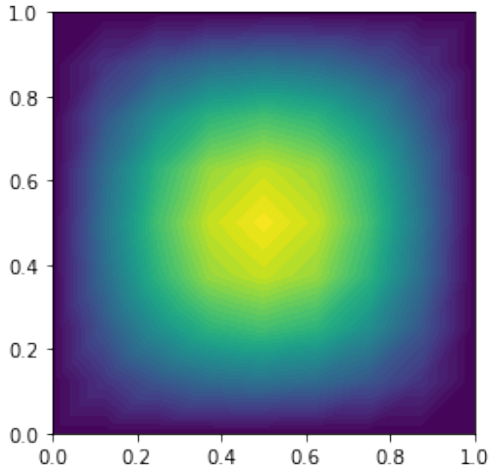
La forma más sencilla de graficar rápidamente u_h y **mesh** es con las siguientes líneas:

```
1 plot(u_h)
2 plot(mesh)
```

Código de FEniCS en Python



(a) Gráfica de la malla del ejemplo en FEniCS.



(b) Gráfica de la solución en el ejemplo en FEniCS.

La función **plot** es útil para la depuración y como inicio de investigaciones científicas.

ParaView

ParaView es una aplicación de código abierto multiplataforma para el análisis de datos y visualización.

www.paraview.org

ParaView

- Se pueden crear rápidamente visualizaciones para analizar los datos mediante técnicas cualitativas y cuantitativas.
- La exploración de datos puede realizarse de forma interactiva en 3D o mediante programación utilizando las funciones de procesamiento.
- Se desarrolló para analizar conjuntos de datos extremadamente grandes utilizando recursos informáticos de memoria distribuida.
- Puede correrse desde supercomputadoras hasta laptops.
- Se ha convertido en una herramienta integral en muchos laboratorios nacionales, universidades e industria.

Graficando la solución en ParaView

Visualizaciones más avanzadas se crean mejor mediante la exportación de la solución a un archivo y el uso de una herramienta de visualización avanzada como lo es **ParaView**,el cual es una poderosa herramienta para la visualización de campos escalares y vectoriales, incluidos los calculados por FEniCS.

El primer paso consiste en exportar la solución en formato VTK:

```
1 vtkfile = File('Resultados/solucion.pvd')
2 vtkfile << u_h
```

Graficando la solución en ParaView

Lla siguiente secuencia de pasos sirven para crear un gráfico de la solución de nuestro problema de Poisson en ParaView.

- Inicia la aplicación ParaView dependiendo del sistema operativo (consultar la parte final del documento **Instrucciones FEniCS.pdf**).
- Haga clic en **File** $\rightarrow$ **Open**... en el menú superior y navegue hasta el directorio que contiene la solución exportada.Esto debe estar dentro de un subdirectorio llamado **Resultados** en la carpeta en donde se encuentre guardado el código de FEniCS.
- Seleccione el archivo nombrado **solucion.pvd** y entonces presione **Ok**.

Graficando la solución en ParaView

- Haga clic en **Apply** en el panel de propiedades de la izquierda. Esto mostrará un gráfico de la solución.

- Para hacer un gráfico 3D de la solución, utilizaremos uno de filtros de ParaView.

  Haga clic en **Filters** $\rightarrow$ **Alphabetical** $\rightarrow$ **Warp By Scalar** en el menú superior y luego en **Apply** en el panel de propiedades ubicado a la izquierda. Esto creará una superficie elevada con la altura determinada por el valor de la solución.

- Para mostrar el gráfico original debajo de la superficie, haga clic en el ícono en forma de ojo del lado izquierdo de **solucion.pvd** en el panel de navegación.

Graficando la solución en ParaView

- Haga clic en el botón 2D que se encuentra en la parte superior de la ventana de visualización para cambiar la visualización a 3D.

  Esto permitirá interactuar con la gráfica, para rotar presione el botón izquierdo del ratón y para hacer un acercamiento presione Ctrl + botón izquierdo del ratón.

- Si se quiere mostrar la malla de elemento finito, haga clic en **solucion.pvd** en el panel de navegación, después en el panel de propiedades seleccionar **Representation**, y seleccione **Surface With Edges**.

Graficando la solución en ParaView

- Para cambiar el radio de aspecto del gráfico 3D, haga clic en **WarpByScalar1** en el panel de navegación y **Scale Factor** en el panel de propiedades.
- Cambie el valor a 0.2 y presione clic en **Apply**.

  Esto cambiará la escala del gráfico.

- En caso de ser necesario, también se pueden remover los ejes 3D desmarcando la casilla **Orientation Axis Visibility** al fondo del panel de propiedades.

- Para exportar la visualización a un archivo, seleccione en el menú superior **File** $\rightarrow$ **Save Screenshot**... y elija un nombre apropiado como **poisson.png**.

Graficando la solución en ParaView

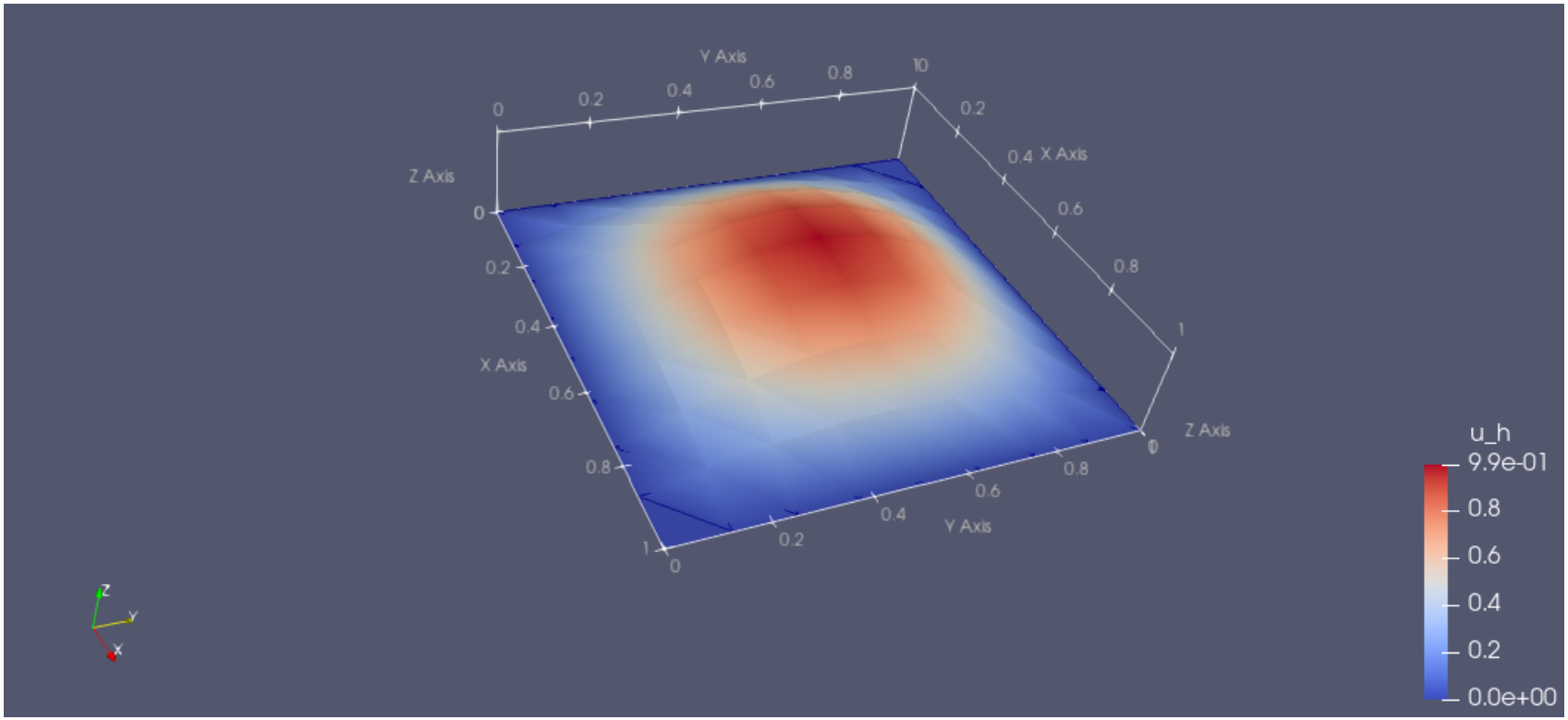
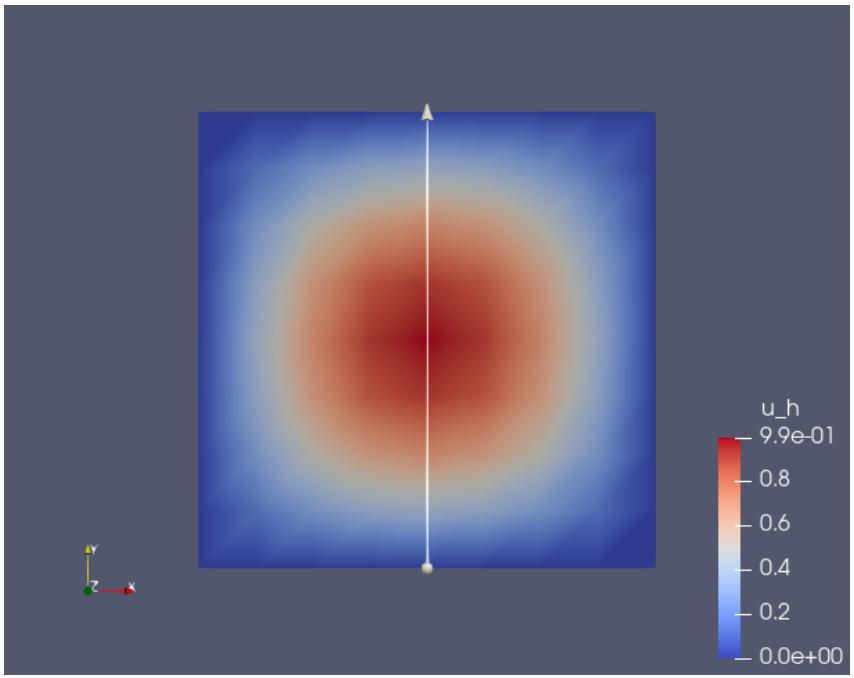


Figure 1: Gráfica de la solución de Poisson en ParaView.

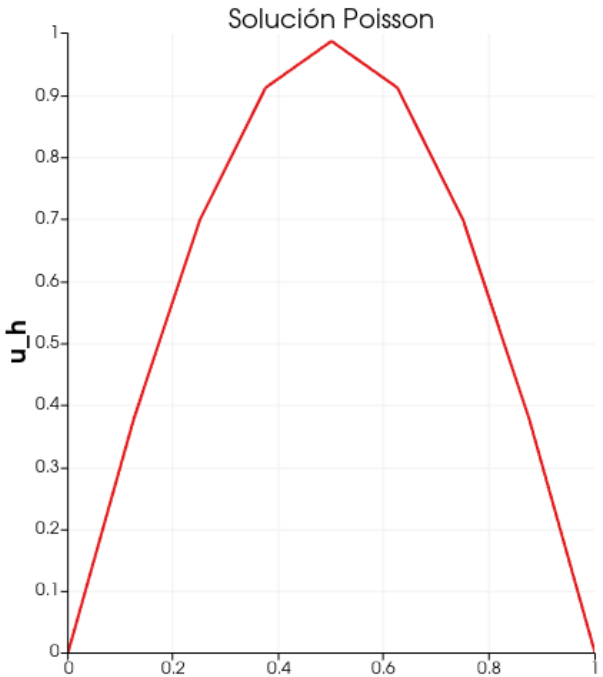
Graficando la solución en ParaView

Con ParaView se pueden graficar perfiles de la solución con la siguiente secuencia de pasos:

- Primero, haga clic en **solucion.pvd** en el panel de navegación.
- Ahora, en el menú superior haga clic en **Filters** $\rightarrow$ **Data Analysis** $\rightarrow$ **Plot Overline**.
- En este ejemplo crearemos un perfil a lo largo de la recta que inicia en la coordenada (0.5,0) y finaliza en el punto (0.5,1). Para colocar los puntos, vaya al menú de propiedades y baje a la opción **Line Parameters**. Después, coloque las coordenadas en **Point1** y en **Point2**, y luego en **Apply**.
- Al lado izquierdo de la gráfica de la solución, se creará una gráfica del perfil a lo largo de la recta.



(a) Gráfica de la solución.



(b) Gráfica del perfil de la solución.

Calculando el error

Por último, se calcula el error para revisar la precisión de la solución.

Esto se realiza comparando la solución por elementos finitos u_h con la solución exacta $u_e = sin(\pi x)sin(\pi y)$.

```
1 u_e = Expression('sin(pi*x[0])*sin(pi*x[1])', degree
      =4)
```

Puede conseguirse un mejor valor interpolando la solución exacta en un espacio de funciones de orden superior, lo que puede hacerse simplemente aumentando el grado.

Calculando el error

Primero, calculamos la norma del error L^2, definido por:

$$||u_e - u_h||_{L^2} = \left(\int_\Omega (u_e - u_h)^2 d\underline{x} \right)^{1/2} \tag{15}$$

Para calcularlo en FEniCS, escribimos:

```
1 error_L2 = errornorm(u_e, u_h, 'L2')
```

La función **errornorm** también puede calcular otras normas de error.

Calculando el error

También, se calcula el máximo valor del error en los vértices (nodos o grados de libertad) de la malla de elementos finitos.

Primero se indica a FEniCS el cálculo de ambos valores u_e y u_h en todos los vértices, luego se sustraen los resultados:

```
1 vertex_u_e = u_e.compute_vertex_values(mesh)
2 vertex_u_h = u_h.compute_vertex_values(mesh)
3 import numpy as np
4 error_max = np.max(np.abs(vertex_u_e - vertex_u_h))
```

Aquí usamos las funciones de **numpy** de valor máximo y valor absoluto dado que son más eficientes en arreglos grandes que las funciones **max** y **abs** de Python.

Cálculo del orden de convergencia

Una pregunta central para cualquier método numérico es el orden de convergencia:

¿Qué tan rápido el error se aproxima a cero cuando la resolución incrementa?

Para los métodos de elementos finitos, esto suele corresponder a demostrar, teórica o empíricamente, que el error: $e = u_e - u_h$ esta delimitado por el tamano de la malla h a cierta potencia m; es decir,

$$||e|| \leq Ch^m \tag{16}$$

para alguna constante C.

El número m es llamado el *orden de convergencia* del método.

Cálculo del orden de convergencia

Ahora, examinaremos como calcular el orden de convergencia en FEniCS.

La función **solver** permite fácilmente calcular las soluciones para mallas cada vez más finas y permite estudiar el orden de convergencia.

Definiendo el tamano del elemento $h = 1/n$, donde n es el número de divisiones de celda en las direcciones x y y ($n = nx = ny$ en el código).

```
1 nx = 16
2 ny = nx
3 mesh = UnitSquareMesh(nx, ny)
```

Nosotros probaremos con $h_0 > h_1 > h_2 > ...$ y calcularemos los errorese_0, e_1, e_2, etc.

Cálculo del orden de convergencia

Asumiendo $e_i = Ch_i^m$ para las constantes desconocidas C y m, podemos comparar dos experimentos consecutivos, $e_{i-1} = Ch_{i-1}^m$ y $e_i = Ch_i^m$, y resolviendo para m:

$$m = \frac{ln(e_i/e_{i-1})}{ln(h_i/h_{i-1})} \tag{17}$$

Demostrar: Dado que $e_{i-1} = Ch_{i-1}^m$ y $e_i = Ch_i^m$

$$\Rightarrow \frac{e_{i-1}}{h_{i-1}^m} = C = \frac{e_i}{h_i^m} \quad \Rightarrow ln\left(\frac{e_{i-1}}{h_{i-1}^m}\right) = ln\left(\frac{e_i}{h_i^m}\right)$$

$$\Rightarrow ln(e_{i-1}) - m \cdot ln(h_{i-1}) = ln(e_i) - m \cdot ln(h_i)$$

$$\Rightarrow m(ln(h_i) - ln(h_{i-1})) = ln(e_i) - ln(e_{i-1}) \quad \therefore \quad m = \frac{ln(e_i/e_{i-1})}{ln(h_i/h_{i-1})}$$

Los valores m deberían de aproximarse al orden de convergencia esperado (típicamente el polinomio de *grado+1* para el error L^2) a medida que i aumenta.

Cálculo del orden de convergencia

Para demostrar el cálculo del orden de convergencia, tomamos la solución exacta:

$$u_e(x, y) = sin(\pi\ x)sin(\pi\ y) \tag{18}$$

Esta elección implica que $f(x, y) = 2\pi^2 sin(\pi\ x)sin(\pi\ y)$.

De esta manera, escribiendo la solución exacta en el código:

```
1 u_e = Expression('sin(pi*x[0])*sin(pi*x[1])', degree
      =6)
```

Este es un ejemplo donde es importante interpolar u_e a un espacio de orden mayor (polinomios de grado 3 son suficientes en este ejemplo).

Cálculo del orden de convergencia

El procedimiento anterior puede convertirse en código de Python.

Seleccionando una lista de grados de elementos ($P1$, $P2$ y $P3$), se realizan pruebas en diferentes mallas refinadas.

Utilizando la norma calculada del error L^2 con la función **errornorm**, se muestra el orden esperado h^{d+1} para u:

elemento	n = 8	n = 16	n = 32	n = 64
P_1	1.97	1.99	2.00	2.00
P_2	3.00	3.00	3.00	3.00
P_3	4.04	4.02	4.01	4.00

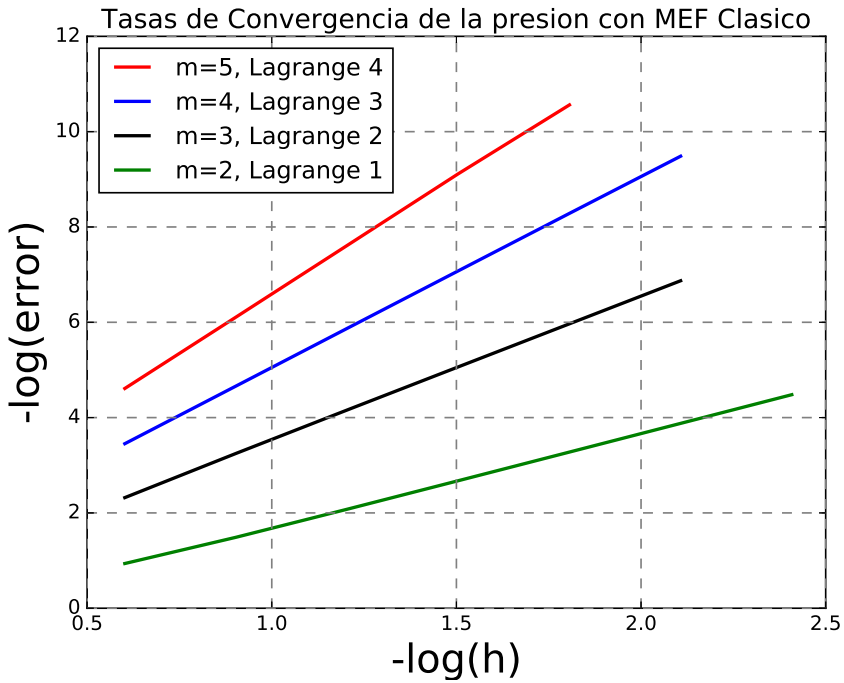
Una entrada como 1.99 para n=16 y P_1 significa que estimamos el orden de 1.99 en la comparación de dos mallas, con resoluciones n=16 y n=8, usando elementos P_1.

Cálculo del orden de convergencia

elemento	n = 8	n = 16	n = 32	n = 64
P_1	1.97	1.99	2.00	2.00
P_2	3.00	3.00	3.00	3.00
P_3	4.04	4.02	4.01	4.00

Las mejores estimaciones de los ordenes de convergencia aparecen en la columna de la derecha, ya que estas se basan en las resoluciones más finas y, por tanto, son más profundas en el régimen asintótico (hasta que alcanzamos un nivel en el que los errores de redondeo y la solución inexacta del sistema lineal empiezan a jugar un papel importante).

Cálculo del orden de convergencia



Por lo tanto, los valores m se aproximan al orden de convergencia esperado (típicamente el polinomio de *grado+1* para el error L^2) a medida que i aumenta.

Comentarios Finales

- Proyecto FEniCS
- Lenguaje de programación Python
- IDE y Notebooks
- Formulación variacional del problema de Poisson
- Código de FEniCS con Python
- Graficando la solución empleando ParaView
- Cálculo del orden de convergencia

Referencias


M. B. Allen, I. Herrera and G. F. Pinder, *Numerical modeling in science and engineering*, John Wiley & Sons., USA, (1988).


Z. Chen, G. Huan and Y. Ma, *Computational methods for multiphase flows in porous media*, SIAM, USA, (2007).


E. Linares and M. Díaz-Viera, *Modelo de flujo y transporte en medios porosos en FEniCS usando el método de elementos finitos mixtos. Aplicación a un caso de estudio para la simulación de un proceso de inyección de agua de baja salinidad a escala de laboratorio.*, Universidad Nacional Autónoma de México, México, (2018).


A. Logg, K. A. Mardal, G. Wells, *Automated Solution of Differential Equations by the Finite Element Method. The FEniCS Book*, Springer-Verlag, Berlin, (2012).

Gracias

Preguntas o Comentarios