Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXVII, págs. 2367-2383 (artículo completo)
Alberto Cardona, Mario Storti, Carlos Zuppa. (Eds.)
San Luis, Argentina, 10-13 Noviembre 2008

UNA NUEVA IMPLEMENTACIÓN PARA LAS ECUACIONES DE NAVIER-STOKES MEDIANTE KLE Y ELEMENTOS ESPECTRALES

Gabriel Bursztyn[a], Alejandro D. Otero[a,b] y Javier Quinteros[a,c]

[a]*Departamento de Computación, FCEN, Universidad de Buenos Aires, Intendente Güiraldes 2160 - Ciudad Universitaria, C1428EGA, Buenos Aires, Argentina, gabrielbursztyn@hotmail.com*
[b]*Grupo ISEP, Depto. de Electrotecnia, FI, Universidad de Buenos Aires, Paseo Colón 850, C1063ACV, Buenos Aires, Argentina, aotero@fi.uba.ar*
[c]*Laboratorio de Tectónica Andina - Depto. de Geología, FCEN, Universidad de Buenos Aires, Intendente Güiraldes 2160 - Ciudad Universitaria, C1428EGA, Buenos Aires, Argentina, jquinte@dc.uba.ar*

Resumen. El método de la ecuación cinemática laplaciana, *Método KLE*, resuelve las ecuaciones de Navier-Stokes en base a la formulación vorticidad-velocidad de las mismas. Este método se ha utilizado efectivamente para calcular el flujo, dependiente del tiempo, alrededor de cuerpos en movimiento y otros problemas de fluidodinámica. El KLE calcula la evolución en el tiempo de la vorticidad como una ecuación diferencial ordinaria en cada nodo de la discretización espacial. Los datos de entrada para la ecuación de transporte de vorticidad en cada paso de tiempo se calculan mediante una versión modificada de la ecuación diferencial en derivadas parciales lineal de Poisson para la velocidad, llamada *Ecuación KLE*.

Hasta el momento, las implementaciones existentes de este método varían entre prototipos funcionales o implementaciones sobre lenguajes interpretados. En este trabajo se presenta una implementación novedosa del método KLE sobre un framework de propósito general de alto desempeño para la resolución de ecuaciones en derivadas parciales (PDE) por el método de los elementos finitos. Asimismo, la funcionalidad del framework fue ampliada mediante una clase que implementa elementos espectrales con cantidad de nodos parametrizable. La totalidad del código fue programada en C++ e incorporada al framework respetando sus interfaces tanto para mantener las abstracciones elementales como las de la resolución del problema.

En este trabajo, se muestran varios casos de validación para distintas mallas y cantidad de nodos en los elementos, así como también los tiempos de resolución en cada caso.

1. INTRODUCCIÓN

Para la descripción de fluidos en movimiento en general se utilizan las denominadas ecuaciones de Navier–Stokes (Batchelor, 2000). Estas ecuaciones en derivadas parciales no lineales surgen de aplicar principios de conservación de la mecánica y la termodinámica a un volumen infinitesimal de fluido. La formulación clásica de dichas ecuaciones utiliza como variables el campo vectorial de la velocidad y el campo escalar de la presión. En general no existe solución analítica para dichas ecuaciones, salvo en el caso de algunos flujos simples en dominios poco complejos, por lo tanto la aproximación de la solución de las ecuaciones de Navier–Stokes por medio de simulación numérica resulta un activo campo de investigación.

En las últimas décadas se han propuesto varios métodos que utilizaban la representación de las ecuaciones de Navier-Stokes en términos de las denominadas variables no primitivas, el potencial-vector de la velocidad y la vorticidad, como variante a la formulación clásica. Estos métodos, conocidos generalmente como formulaciones vorticidad-función de corriente (ω, ψ), probaron ser efectivos en determinados problemas (ver referencias en Ponta and Aref, 2005).

Más recientemente se presentaron algunos métodos que utilizaban una formulación de tipo híbrida en términos de una variable primitiva, la velocidad, y una variable no primitiva, la vorticidad. Los métodos vorticidad-velocidad (ω, v), como se los denomina, presentaban ventajas respecto de la formulación clásica en variables primitivas y de la formulación en variables no primitivas. Más allá de las ventajas que presentaban, existía una desventaja en comparación con la formulación clásica en variables primitivas, se necesitaba trabajar con 6 funciones incógnitas en lugar de 4.

En este trabajo se presenta una nueva implementación del método de la ecuación cinemática laplaciana, *Método KLE (kinematic Laplacian equation)*, que está basado en la formulación vorticidad-velocidad. Asimismo, se detalla el diseño utilizado para esta nueva implementación del KLE en 2 dimensiones en lenguaje C++ a fin de obtener mejoras en el manejo de la memoria y los tiempos de cálculo, así como proveer un esquema que funcione como base para la implementación 3D y la combinación del KLE con ecuaciones que describan otros fenómenos en problemas de mutifísica.

En la próxima sección se presenta una breve descripción del método KLE, el cual se encuentra expuesto en detalle en los trabajos de Ponta and Aref (2005) y Otero and Ponta (2006). En la sección 3 se menciona el diseño de clases ideado por Quinteros et al. (2007), desarrollado y propuesto específicamente para la resolución y simulación de problemas en ecuaciones en derivadas parciales por medio del método de elementos finitos.

A partir del esquema general del diseño presentado, se detallan las ampliaciones de la funcionalidad del mismo, entre las cuales se encuentra una clase que implementa elementos espectrales con cantidad de nodos parametrizable, la cual se utiliza en esta implementación del método KLE.

Este nuevo desarrollo fue programado en lenguaje C++ e incorporado al framework respetando sus interfaces para mantener tanto las abstracciones elementales como las de la resolución del problema.

Se muestran varios casos de validación para distintas mallas y cantidad de nodos en los elementos, así como también los tiempos insumidos en cada etapa de la resolución para cada caso. A su vez, se comprueba que se obtienen mejoras con respecto al tiempo de cálculo que en la versión anterior presentada por Otero and Ponta (2006), la cual estaba programada en lenguaje MATLAB.

2. MÉTODO KLE

En el trabajo de Ponta and Aref (2005) se presentó un nuevo método basado en la formulación vorticidad-velocidad ($\boldsymbol{\omega}$, $\boldsymbol{v}$) que mejoró distintos aspectos respecto de las primeras implementaciones de esta formulación, como la reducción de la cantidad de incógnitas. El método KLE se caracteriza por un completo desacoplamiento de las dos variables, vorticidad en tiempo–velocidad en espacio. De esta forma, se reduce a tres el número de variables a resolver en el proceso de integración temporal. Además, esta descomposición del problema permite la utilización de algoritmos de integración de ecuaciones diferenciales ordinarias (ODE) de orden variable y paso temporal adaptativo que mejoran la eficiencia y robustez del proceso de integración.

El método KLE se compone de dos partes. En primer lugar, la denominada ecuación laplaciana cinemática, que es una versión modificada de la ecuación de Poisson para la velocidad, encargada de resolver la parte cinemática del problema. En segundo lugar, el algoritmo de integración de las ecuaciones de la dinámica del problema (Navier-Stokes en vorticidad), es decir, la ecuación de transporte de vorticidad, que se resuelve como una ecuación diferencial ordinaria en cada nodo de la discretización. Ambas partes se realimentan mutuamente y en conjunto reciben la denominación de método de la ecuación laplaciana cinemática o, brevemente, método KLE.

La expresión de la formulacion variacional de la KLE para flujo incompresible, sobre cuya resolución trata este trabajo, se puede expresar como

$$\int_\Omega \underline{\boldsymbol{\nabla v}} : \underline{\boldsymbol{\nabla \delta v}} + \alpha_{\mathcal{D}} (\underline{\boldsymbol{\nabla}} \cdot \underline{\boldsymbol{v}})(\underline{\boldsymbol{\nabla}} \cdot \underline{\boldsymbol{\delta v}}) + \alpha_\omega (\underline{\boldsymbol{\nabla}} \times \underline{\boldsymbol{v}}) \cdot (\underline{\boldsymbol{\nabla}} \times \underline{\boldsymbol{\delta v}}) \, \mathrm{d}\Omega = \int_\Omega (\underline{\boldsymbol{\nabla}} \times \underline{\boldsymbol{\omega}}) \cdot \underline{\boldsymbol{\delta v}} + \alpha_\omega \underline{\boldsymbol{\omega}} \cdot (\underline{\boldsymbol{\nabla}} \times \underline{\boldsymbol{\delta v}}) \, \mathrm{d}\Omega, \tag{1}$$

donde $\underline{\boldsymbol{v}}$ es el campo de velocidades, $\underline{\boldsymbol{\omega}}$ es la distribución del campo de vorticidades y $\alpha_{\mathcal{D}}$ y α_ω son las constantes de penalización correspondientes a las restricciones $\underline{\boldsymbol{\nabla}} \cdot \underline{\boldsymbol{v}} = \mathcal{D}$ y $\underline{\boldsymbol{\nabla}} \times \underline{\boldsymbol{v}} = \underline{\boldsymbol{\omega}}$, siendo $\mathcal{D}$ la tasa de expansión. En esta implementación, los valores utilizados para las constantes de penalización ($\alpha_{\mathcal{D}} = 10^3$ y $\alpha_\omega = 10^2$) son los mismos que en Ponta and Aref (2005) y Otero and Ponta (2006).

Discretización de la Ecuación Laplaciana Cinemática

La formulación variacional de la ecuación 1 se discretiza por medio del método de elementos espectrales (Otero and Ponta, 2006). Este método es una implementación particular de la versión *p* del método de elementos finitos donde los nodos de los elementos se ubican en los puntos de una grilla de Gauss–Lobatto (Patera, 1984; Karniadakis et al., 1985). Esta forma de ubicar los nodos de los elementos espectrales presenta una gran ventaja: los elementos de bajo orden, $p = 1$ y $p = 2$, se corresponden con los clásicos elementos finitos de 2 y 3 nodos. De esta manera, se tiene un método de orden variable adecuado para ser llevado hasta alto orden que contiene como casos particulares elementos ampliamente utilizados. En cuanto a los elementos de alto orden, esta forma de colocar los nodos resulta más económica que la que utiliza nodos espaciados en forma equidistante (Hourigan et al., 2001). Los puntos de la grilla de Gauss–Lobatto se ubican en los extremos, máximos o mínimos, del polinomio de Legendre del orden correspondiente al elemento y en los bordes del intervalo. Es decir, los extremos $r = \pm 1$ en el caso del intervalo natural y las raíces de la derivada del polinomio P_p de orden p ($dP_p/dr = 0$). La posición de dichos puntos puede calcularse a partir de las derivadas del polinomio de

Legendre del orden correspondiente utilizando el método de Newton–Raphson con las raíces del polinomio de Chebyshev del mismo orden como valores iniciales. Estas raíces se calculan como

$$r_{j+1}^{Cheb} = \cos\left(\frac{\pi j}{p}\right), \qquad j = 0...p. \tag{2}$$

Una vez ubicados los nodos del elemento maestro según lo descripto anteriormente, se construyen las funciones de interpolación como los polinomios de Lagrange asociados a esos nodos. A partir de esto se calculan sus derivadas respecto de las coordenadas naturales.

La implementación de esto último está explicada en la sección 4 y se basó en el esquema general de diseño expuesto en la sección 3.

Dado un elemento espectral de orden p, el número de nodos por elemento es ${N_{GL}}^2$ con $N_{GL} = p + 1$. Considerando un dominio bidimensional, las componentes del vector velocidad y el arreglo de las componentes de su gradiente expresadas en un sistema de coordenadas ortogonales (x, y) son interpoladas dentro de cada elemento como

$$\boldsymbol{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \mathbf{H}^e\,\hat{\mathbf{V}}^e, \qquad \boldsymbol{\nabla v} = \begin{bmatrix} \frac{\partial v_x}{\partial x} \\ \frac{\partial v_x}{\partial y} \\ \frac{\partial v_y}{\partial x} \\ \frac{\partial v_y}{\partial y} \end{bmatrix} = \mathbf{B}^e\,\hat{\mathbf{V}}^e, \tag{3}$$

donde $\hat{\mathbf{V}}^e$ es el arreglo de los valores nodales de las componentes del vector de velocidad, $\mathbf{H}^e$ el de las funciones de interpolación dentro del elemento y $\mathbf{B}^e$ el de las derivadas de las funciones de interpolación respecto de las coordenadas del problema.

$$\hat{\mathbf{V}}^e = \begin{bmatrix} \hat{v}_x^1 & \hat{v}_y^1 & \hat{v}_x^2 & \hat{v}_y^2 & \cdots & \hat{v}_x^{{N_{GL}}^2} & \hat{v}_y^{{N_{GL}}^2} \end{bmatrix}^T \tag{4}$$

$$\mathbf{H}^e = \begin{bmatrix} h_1 & 0 & h_2 & 0 & \cdots & h_{{N_{GL}}^2} & 0 \\ 0 & h_1 & 0 & h_2 & \cdots & 0 & h_{{N_{GL}}^2} \end{bmatrix} \tag{5}$$

$$\mathbf{B}^e = \begin{bmatrix} \frac{\partial h_1}{\partial x} & 0 & \frac{\partial h_2}{\partial x} & 0 & \cdots & \frac{\partial h_{{N_{GL}}^2}}{\partial x} & 0 \\ \frac{\partial h_1}{\partial y} & 0 & \frac{\partial h_2}{\partial y} & 0 & \cdots & \frac{\partial h_{{N_{GL}}^2}}{\partial y} & 0 \\ 0 & \frac{\partial h_1}{\partial x} & 0 & \frac{\partial h_2}{\partial x} & \cdots & 0 & \frac{\partial h_{{N_{GL}}^2}}{\partial x} \\ 0 & \frac{\partial h_1}{\partial y} & 0 & \frac{\partial h_2}{\partial y} & \cdots & 0 & \frac{\partial h_{{N_{GL}}^2}}{\partial y} \end{bmatrix} \tag{6}$$

Las derivadas parciales de las funciones de forma respecto de las coordenadas (x, y) del problema se calculan utilizando la matriz jacobiana

$$\mathbf{J}(r,s) = \begin{bmatrix} \frac{\partial x}{\partial r}(r,s) & \frac{\partial y}{\partial r}(r,s) \\ \frac{\partial x}{\partial s}(r,s) & \frac{\partial y}{\partial s}(r,s) \end{bmatrix} = \begin{bmatrix} \frac{\partial h_i}{\partial r}(r,s)\,x^i & \frac{\partial h_i}{\partial r}(r,s)\,y^i \\ \frac{\partial h_i}{\partial s}(r,s)\,x^i & \frac{\partial h_i}{\partial s}(r,s)\,y^i \end{bmatrix}, \tag{7}$$

donde $i = 1...\,(N_{GL})^2$, según

$$\begin{bmatrix} \frac{\partial h_k}{\partial x} \\ \frac{\partial h_k}{\partial y} \end{bmatrix} = \mathbf{J}^{-1} \begin{bmatrix} \frac{\partial h_k}{\partial r} \\ \frac{\partial h_k}{\partial s} \end{bmatrix}, \qquad \text{con } k = 1...N_{GL}^2. \tag{8}$$

En este caso, la divergencia del campo de velocidad puede calcularse a partir de las componentes del gradiente como

$$\boldsymbol{\nabla}\cdot\boldsymbol{v} = \mathbf{m}\,\mathbf{B}^e\,\hat{\mathbf{V}}^e, \qquad \mathbf{m} = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}, \tag{9}$$

y la única componente no nula del rotor de la velocidad $(\boldsymbol{\nabla} \times \boldsymbol{v})_z$ se obtiene como

$$\boldsymbol{\nabla} \times \boldsymbol{v} = \mathbf{r}\, \mathbf{B}^e\, \hat{\mathbf{V}}^e, \qquad \mathbf{r} = \left[\begin{array}{cccc} 0 & -1 & 1 & 0 \end{array} \right]. \tag{10}$$

En forma equivalente se procede a discretizar la vorticidad, llamando $\boldsymbol{\omega} = \boldsymbol{\omega}_z$ y las componentes x e y no nulas de su rotor según

$$\boldsymbol{\omega} = \mathbf{H}^e_{\boldsymbol{\omega}}\, \hat{\boldsymbol{\omega}}^e, \qquad \boldsymbol{\nabla} \times \boldsymbol{\omega} = \left[\begin{array}{c} (\boldsymbol{\nabla} \times \boldsymbol{\omega})_x \\ (\boldsymbol{\nabla} \times \boldsymbol{\omega})_y \end{array} \right] = \left[\begin{array}{c} \frac{\partial \boldsymbol{\omega}}{\partial y} \\ -\frac{\partial \boldsymbol{\omega}}{\partial x} \end{array} \right] = \mathbf{B}^e_{\boldsymbol{\omega}}\, \hat{\boldsymbol{\omega}}^e, \tag{11}$$

donde $\hat{\boldsymbol{\omega}}^e$ es el arreglo de los valores nodales de la componente $\boldsymbol{\omega}_z$ de la vorticidad, obtenidos de la integración de la ecuación de transporte de la vorticidad, $\mathbf{H}^e_{\boldsymbol{\omega}}$ es el arreglo de las funciones de interpolación de la vorticidad y $\mathbf{B}^e_{\boldsymbol{\omega}}$ el de las derivadas de las funciones de interpolación respecto de las coordenadas del problema para calcular las componentes del rotor de la vorticidad, dados por:

$$\hat{\boldsymbol{\omega}}^e = \left[\begin{array}{cccc} \hat{\boldsymbol{\omega}}^1 & \hat{\boldsymbol{\omega}}^2 & \cdots & \hat{\boldsymbol{\omega}}^{{N_{GL}}^2} \end{array} \right]^T, \tag{12}$$

$$\mathbf{H}^e_{\boldsymbol{\omega}} = \left[\begin{array}{cccc} h_1 & h_2 & \cdots & h_{{N_{GL}}^2} \end{array} \right], \tag{13}$$

$$\mathbf{B}^e_{\boldsymbol{\omega}} = \left[\begin{array}{cccc} \frac{\partial h_1}{\partial y} & \frac{\partial h_2}{\partial y} & \cdots & \frac{\partial h_{{N_{GL}}^2}}{\partial y} \\ -\frac{\partial h_1}{\partial x} & -\frac{\partial h_2}{\partial x} & \cdots & -\frac{\partial h_{{N_{GL}}^2}}{\partial x} \end{array} \right]. \tag{14}$$

Con estas interpolaciones se puede discretizar la formulación variacional de la KLE (ecuación 1) en cada subdominio elemental, reemplazando la velocidad y la vorticidad y sus derivadas por sus correspondientes discretizaciones como

$$\delta\hat{\mathbf{V}}^{eT} \underbrace{\left(\boldsymbol{K}^{\boldsymbol{e}}_{\boldsymbol{L}} + \boldsymbol{K}^{\boldsymbol{e}}_{\boldsymbol{\mathcal{D}}} + \boldsymbol{K}^{\boldsymbol{e}}_{\boldsymbol{\omega}}\right)}_{\boldsymbol{K^e}} \hat{\mathbf{V}}^e = \delta\hat{\mathbf{V}}^{eT} \underbrace{\left(\boldsymbol{R}^{\boldsymbol{e}}_{\boldsymbol{L}} + \boldsymbol{R}^{\boldsymbol{e}}_{\boldsymbol{\omega}}\right)}_{\boldsymbol{R^e}} \hat{\boldsymbol{\omega}}^e, \tag{15}$$

donde

$$\begin{aligned}
\boldsymbol{K}^{\boldsymbol{e}}_{\boldsymbol{L}} &= \int_{\Omega^e} \mathbf{B}^{eT}\mathbf{B}^e \, \mathrm{d}\Omega = \int_{-1}^{1}\int_{-1}^{1} \mathbf{B}^{eT}\mathbf{B}^e \, |\boldsymbol{J}| \, \mathrm{d}r\mathrm{d}s, \\
\boldsymbol{K}^{\boldsymbol{e}}_{\boldsymbol{\mathcal{D}}} &= \int_{-1}^{1}\int_{-1}^{1} \alpha_{\mathcal{D}}\, \mathbf{B}^{eT}\mathbf{m}^T\mathbf{m}\mathbf{B}^e \, |\boldsymbol{J}| \, \mathrm{d}r\mathrm{d}s, \\
\boldsymbol{K}^{\boldsymbol{e}}_{\boldsymbol{\omega}} &= \int_{-1}^{1}\int_{-1}^{1} \alpha_{\omega}\, \mathbf{B}^{eT}\mathbf{r}^T\mathbf{r}\mathbf{B}^e \, |\boldsymbol{J}| \, \mathrm{d}r\mathrm{d}s, \\
\boldsymbol{R}^{\boldsymbol{e}}_{\boldsymbol{L}} &= \int_{-1}^{1}\int_{-1}^{1} \mathbf{H}^{eT}\mathbf{B}^e_{\boldsymbol{\omega}} \, |\boldsymbol{J}| \, \mathrm{d}r\mathrm{d}s, \\
\boldsymbol{R}^{\boldsymbol{e}}_{\boldsymbol{\omega}} &= \int_{-1}^{1}\int_{-1}^{1} \alpha_{\omega}\, \mathbf{B}^{eT}\mathbf{r}^T\mathbf{H}^e_{\boldsymbol{\omega}} \, |\boldsymbol{J}| \, \mathrm{d}r\mathrm{d}s,
\end{aligned}$$

y $\delta\hat{\mathbf{V}}^e$ es el arreglo de valores nodales de las componentes del campo arbitrario $\underline{\boldsymbol{\delta v}}$ (Otero and Ponta, 2006).

Las matrices elementales $\boldsymbol{K^e}$ y $\boldsymbol{R^e}$ se ensamblan en las respectivas matrices globales llegando finalmente al sistema global que representa la discretización de la KLE,

$$\boldsymbol{K}\, \hat{\mathbf{V}} = \boldsymbol{R}\, \hat{\boldsymbol{\omega}}. \tag{16}$$

Dados el arreglo de valores de la vorticidad en los nodos de la malla ($\hat{\omega}$) y las condiciones de contorno correspondientes, se obtiene de la ecuación 16 el arreglo de las componentes de la velocidad en los nodos ($\hat{\mathbf{V}}$), i.e. la solución discreta de la ecuación laplaciana cinemática. Como es práctica común en ecuaciones espectrales, las matrices de la ecuación 15 se integran numéricamente utilizando la cuadratura de Gauss-Lobatto. Esto, combinado con la ubicación de los nodos utilizada, presenta ventajas pese a no ser la regla de integración exacta (Otero and Ponta, 2006; Patera, 1984; Henderson and Karniadakis, 1995).

3. ESQUEMA GENERAL DEL DISEÑO PARA RESOLUCIÓN MEDIANTE FEM

En cualquier desarrollo de software, es importante tener un buen diseño de la aplicación. Esto quiere decir, entre otras palabras, que la aplicación debe ser modular, de forma de poder reutilizar código, optimizando tiempos en la búsqueda de errores; ser escalable y flexible de manera de ahorrar tiempos ante una variación o ampliación del problema a resolver; y a su vez, que su performance en cuanto a tiempos de ejecución sea la mejor posible.

Cuando un problema debe ser resuelto mediante técnicas de métodos numéricos, el tiempo de programación, validación y ejecución deben ser factores a tener en cuenta a fin de poder reducir costos. Cuando se trata de un problema de simple resolución o cuya solución analítica es conocida, no se suele invertir mucho tiempo en el diseño. Pero en el caso en que dicha solución sea parte de un problema más complejo, es factible que aparezcan errores difíciles de detectar. En estos casos, es mejor contar con módulos validados anteriormente que puedan ser reutilizados. De esta forma, el código puede ser extendido para resolver problemas más complejos cuya solución no sea conocida, sin perder confiabilidad.

Por este motivo se decidió utilizar el framework propuesto por Quinteros et al. (2007) para implementar modelos numéricos basados en elementos finitos considerando que reúne las características previamente mencionadas. Este framework está basado en un diagrama de clases que representa cada una de las entidades que suelen formar parte de un problema resuelto por el Método de los Elementos Finitos (FEM). Según los conceptos de programación orientada a objetos, cada clase debe proveer una funcionalidad específica que represente de forma clara a un objeto y el diseñador del sistema debe cuidar que cada una de ellas brinde una funcionalidad acorde a la esperada en este tipo de implementaciones. Una descripción detallada de las características del framework puede encontrarse en Quinteros et al. (2007), mientras que aquí sólo se mencionan las carácteristicas más importantes.

3.1. Características generales del diagrama de clases

La clase `Domain` es la que incluye toda la información necesaria para describir el modelo mediante un conjunto de elementos y nodos almacenados en los atributos `Elements` y `Nodes`. Cada condición de borde es almacenada en una instancia de la clase `Boundary` formando un conjunto de condiciones de borde almacenado en la propiedad `Boundaries` de la clase `Domain`.

Es importante notar que el elemento está determinado por un conjunto de nodos conectados. Sin embargo, no existe una relación de pertenencia entre el nodo y el elemento, ya que algunos de ellos están ubicados en los bordes del elemento y, por lo tanto, pertenecerán a más de uno. Es por eso que cada elemento contendrá referencias a los nodos que lo componen, almacenando en el atributo `idNodes` los números de identificación globales que le correspondan.

Para calcular la matriz de rigidez de cada elemento, se deben evaluar las funciones de interpolación y sus derivadas en los puntos de Gauss que determinen el orden y tipo de elemento

utilizado. La posición de cada uno de estos puntos es idéntica en todos los elementos, ya que la integración numérica se realiza sobre un elemento no deformado, llamado *elemento maestro*. Los puntos de Gauss son almacenados de forma estática en el atributo `gps` de la clase `Element`, optimizando la memoria utilizada y el tiempo de cálculo, ya que se calcula una única vez y se reutiliza en todos los elementos. Además, el tiempo de acceso a la información se reduce debido a un mejor manejo de la memoria caché. En el caso de utilizar integración reducida, se pueden almacenar en `gps` sólo los puntos de Gauss necesarios. Para las funciones que requieran integración completa se define otro atributo llamado `gpsfull`, también declarado de forma estática, en el que deben incluirse el conjunto completo de puntos de Gauss.

Dado que la evaluación de las funciones de forma (h) y sus derivadas respecto de las coordenadas naturales ($\frac{\partial h}{\partial r}$ y $\frac{\partial h}{\partial s}$) se realizan siempre en el elemento maestro, son calculadas previamente y almacenadas en la clase `GaussPoint` de forma estática a fin de poder reutilizar dichos valores cada vez que se los necesite evitando el recálculo. Luego, en la etapa de integración, sólo se deberá calcular la inversa de la matriz Jacobiana en cada punto de Gauss, mejorándose así la perfomance en el cálculo.

Las matrices elementales son consideradas, en general, densas porque la mayoría de los nodos dentro del elemento están vinculados entre sí. En estos casos, las operaciones resultan ser eficientes ya que se debe operar sobre matrices donde el número de componentes está asociado con el cuadrado del número de nodos por elemento.

En el framework existe una clase llamada `Matrix` que provee una interfaz simple y fácil de utilizar para la manipulación de matrices densas, e incluye las operaciones más comunes de álgebra lineal necesarias en este tipo de implementaciones. Mediante esta clase, el usuario se abstrae de la implementación de estas operaciones, que en este caso son realizadas por la librería Lapack (Anderson et al., 1999).

Asimismo, se utiliza otra clase llamada `SparseMatrix` diseñada especialmente para mejorar el manejo de memoria y la perfomance en operaciones con matrices ralas de gran tamaño, como suele ser una matriz de rigidez global. Al igual que `Matrix`, esta clase abstrae al usuario de la implementación de las operaciones algebraicas, que en este caso son provistas por la librería `SuperLU` (Demmel et al., 1999). Mediante esta clase, se pueden también aprovechar las optimizaciones asociadas al caso de matrices simétricas, tanto en almacenamiento como en la resolución del sistema.

La descomposición LU calculada para la resolución de un sistema, puede ser almacenada y utilizada en sucesivas resoluciones, disminuyendo drásticamente el tiempo necesario para resolver el sistema.

Todas las clases mencionadas poseen un método Save que permite almacenar una instancia dentro de un stream de salida y un constructor con un parámetro de tipo stream de entrada para poder volver a crear la instancia en memoria. De esta forma, si la simulación requiere de muchos pasos de tiempo, se puede almacenar el estado en disco y volver a ejecutar luego desde el último *checkpoint*.

El diseño de la clase `Element` está definido como una base que provee una especificación común para que distintos tipos de elementos la implementen. Además incluye la especificación de diferentes funciones que resuelven distintos tipos de ecuaciones integrando numéricamente. En el caso más común, la solución para ecuaciones de derivadas parciales por el método de elementos finitos, implica que para cada elemento se calcule una serie de matrices evaluadas en los puntos de Gauss. Dichas matrices, son multiplicadas por un factor peso y sumadas para obtener la matriz de rigidez. Es por esta razón, que se define un método llamado `calc_equation` que luego de invocar a otro método llamado `eval_equation`, encargado de evaluar las matrices

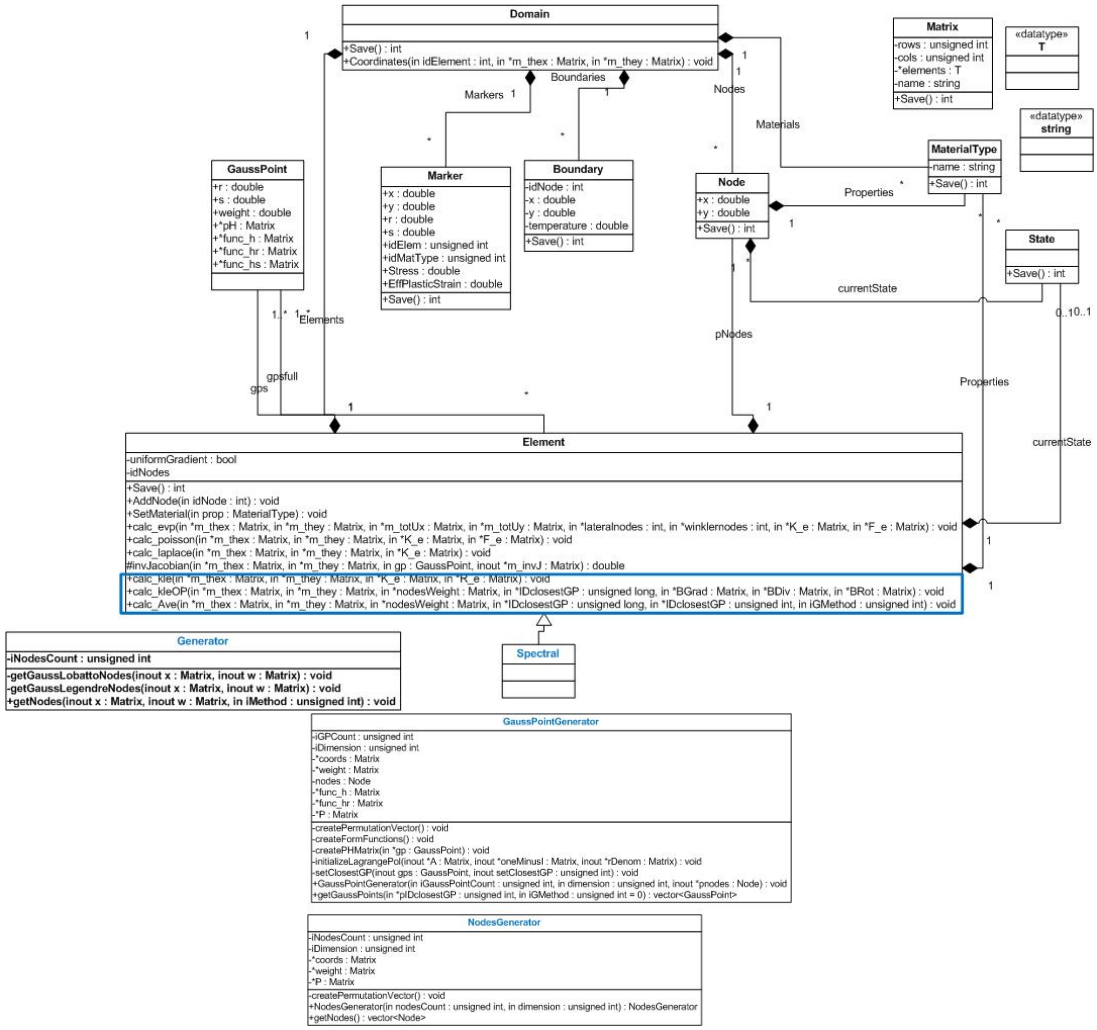


Figura 1: Diagrama de clases extendido

en los puntos de Gauss, realizará las operaciones correspondientes sobre dichas matrices.

4. EXTENSIÓN DEL ESQUEMA GENERAL PARA RESOLUCIÓN MEDIANTE FEM

A partir del framework mencionado en el capítulo anterior, se creó una clase `Spectral` correspondiente a las clases espectrales, que hereda de `Element` todas sus propiedades y métodos. Mediante esta clase se implementaron los elementos espectrales con cantidad de nodos variables.

A su vez, a la clase `Element` se le agregaron una serie de métodos que implementan el método KLE como puede verse en la figura 1.

El método `eval_kle` se utiliza para calcular las matrices en la totalidad de los puntos de integración, mientras que el método `eval_kle2` sólo sobre los puntos de integración reducida. Ambos métodos serán utilizados por el método `calc_kle` para calcular las matrices de rigidez (K y R) de la ecuación 16.

A su vez el método `eval_kleOP` calcula las matrices evaluadas en los puntos de integración necesarios para que el método `calc_kleOP` pueda construir los operadores de derivación que son utilizados para evaluar el lado derecho de la función de integración de ecuaciones diferenciales ordinarias (ODE) en sucesivos pasos de tiempo (Ponta and Aref, 2005; Otero and

```
 BOUNDARY_CONDITION_FUNCTION:
v_x=erf(y/t)
v_y=1
w=-2*exp(-1*((y/t)^2))/(sqrt(_PI)*t)
dv_x_x=0
dv_x_y=-2*exp(-1*((y/t)^2))/(sqrt(_PI)*t)
dv_y_x=0
dv_y_y=0
ddv_x_x_x=0
ddv_x_y_y=((-4*y)/(sqrt(_PI)*t^3))*exp(-1*((y/t)^2))
ddv_y_x_x=0
ddv_y_y_y=0
```

Figura 2: Ejemplo de archivo de configuración

Ponta, 2006).

Para ubicar los nodos del elemento maestro que se corresponden con los puntos de una grilla Gauss-Lobatto se creó la clase `NodesGenerator`, que utiliza el método correspondiente de la clase `Generator`. Los nodos devueltos estarán ordenados según una matriz de permutación definida de forma que la numeración sea análoga a la utilizada en el FEM (Bathe, 1996).

De la misma forma, para crear los puntos de integración del elemento maestro se creó la clase `GaussPointGenerator`. Mediante un parámetro de la función `getGaussPoints` se puede elegir entre ambas formas de distribución de puntos de integración.

Para calcular las condiciones de borde definidas por una funcion que depende del tiempo se creó una interfase con un parser llamado `muParser` (Berg, 2005), de forma de calcular en tiempo de ejecución el valor de la función a partir de su definición. `muParser` interpreta cualquier cadena de texto como una función, siempre y cuando se definan previamente las variables, constantes y funciones a utilizar. En este caso, se definieron como variables las coordenadas espaciales (x, y) y la variable de tiempo (t).

El modelo recibe como parámetro el nombre de un archivo de configuración donde están definididas la función de vorticidad Omega (w), la función de tiempo que define la condiciones de borde de velocidad y sus derivadas parciales. En la figura 2 puede verse un extracto del archivo de configuración utilizado para correr los casos de prueba, en el que están definidas tanto las velocidades como sus derivadas.

5. RESULTADOS

El método KLE es un modelo matemático más que un esquema de discretización. La primera implementación del método KLE realizada por Ponta and Aref (2005) utilizó elementos finitos clásicos de 9 nodos para la discretización del dominio espacial con muy buenos resultados. La generalidad del método permite explorar diferentes técnicas de discretización en tiempo y espacio. Un punto de particular interés es la calidad de la aproximación de la solución espacial y el cálculo de las derivadas espaciales en la evaluación del lado derecho de la ecuación de transporte de vorticidad. Para esto, en Otero and Ponta (2006) se implementó una discretización por elementos espectrales de la parte espacial del problema a fin de realizar un estudio sistemático contando con la posibilidad de variar el orden de la interpolación.

Para validar los resultados obtenidos por nuestra implementación se siguió la siguiente meto-

dología. Se define un campo de velocidades del cual se calcula la vorticidad y luego, imponiendo esa vorticidad y las velocidades en el contorno de la malla, se resuelve la KLE para recuperar las velocidades que se comparan con las impuestas. Para esto, se utiliza la función `error` que corresponde a la solución del problema de la placa plana infinita que ha sido propuesto en Otero and Ponta (2006) como *benchmark* en este tipo de métodos. Se utilizó una malla bidimensional y se resolvió el problema, variando la complejidad de cada elemento, modificando la cantidad de nodos del mismo, como así también, la cantidad de elementos dentro de la malla.

En el problema de la placa plana se considera una placa ubicada en el plano $y = 0$ que está rodeada por fluido en $y > 0$ y a la cual se le imprime una velocidad U que se mantiene en el tiempo. La solución analítica de este problema (ver Sec. 4.3 de Batchelor, 2000, entre otros) para el campo de velocidades descripto en el marco de referencia de la placa que se mueve en la dirección negativa de las x es

$$v_x = U \,\mathrm{erf}\left(\frac{y}{\sqrt{4\,\nu\,t}}\right), \tag{17}$$
$$v_y = 0,$$

donde erf es la función de error, y es la coordenada vertical y t es el tiempo. Reescribiendo la ecuación 17 en términos de la velocidad normalizada u/U, la coordenada vertical normalizada y/Y y el parámetro $\tau = \sqrt{4\,\nu\,t}/Y$ se llega a

$$\frac{v_x}{U} = \mathrm{erf}\left(\frac{y/Y}{\tau}\right), \tag{18}$$

donde Y es la altura de la malla.

La distribución de vorticidad correspondiente a la solución analítica se encuentra dada por

$$\frac{\omega}{U/Y} = \frac{-2}{\tau\,\sqrt{\pi}}\, e^{-\left(\frac{y/Y}{\tau}\right)^2}. \tag{19}$$

Para un instante de tiempo dado, la solución analítica del campo de velocidades y vorticidad están definidas, respectivamente, por las funciones gaussianas y de error de la coordenada espacial. Esto último previene el caso en que la solución del problema está contenida en el espacio generado por la base de funciones de interpolación asociada con la discretización por elementos espectrales, que resulta un problema trivial.

Precisión de la solución espacial

A fin de poder evaluar la precisión espacial se desarrollaron una serie de experimentos comparando el campo de velocidades dado por la ecuación 18 con la solución numérica obtenida por la KLE en dicho campo, tomando como término de cargas la correspondiente distribución de vorticidad según la ecuación 19. Al utilizar elementos espectrales, los resultados se presentan en función del número de intervalos en una dimensión N^*, que es 1 menos que la cantidad de nodos y la inversa de la media de la distancia internodal, tal como fue propuesto en Otero and Ponta (2006). Se resolvió el problema espacial en diferentes etapas de desarrollo de la capa límite calculando la norma infinito del error sobre los nodos de la discretización, utilizando los mismos valores del parámetro τ que en Otero and Ponta (2006). El comportamiento de la solución obtenida con esta implementación se comparó con los resultados de la implementación previa en MATLAB.

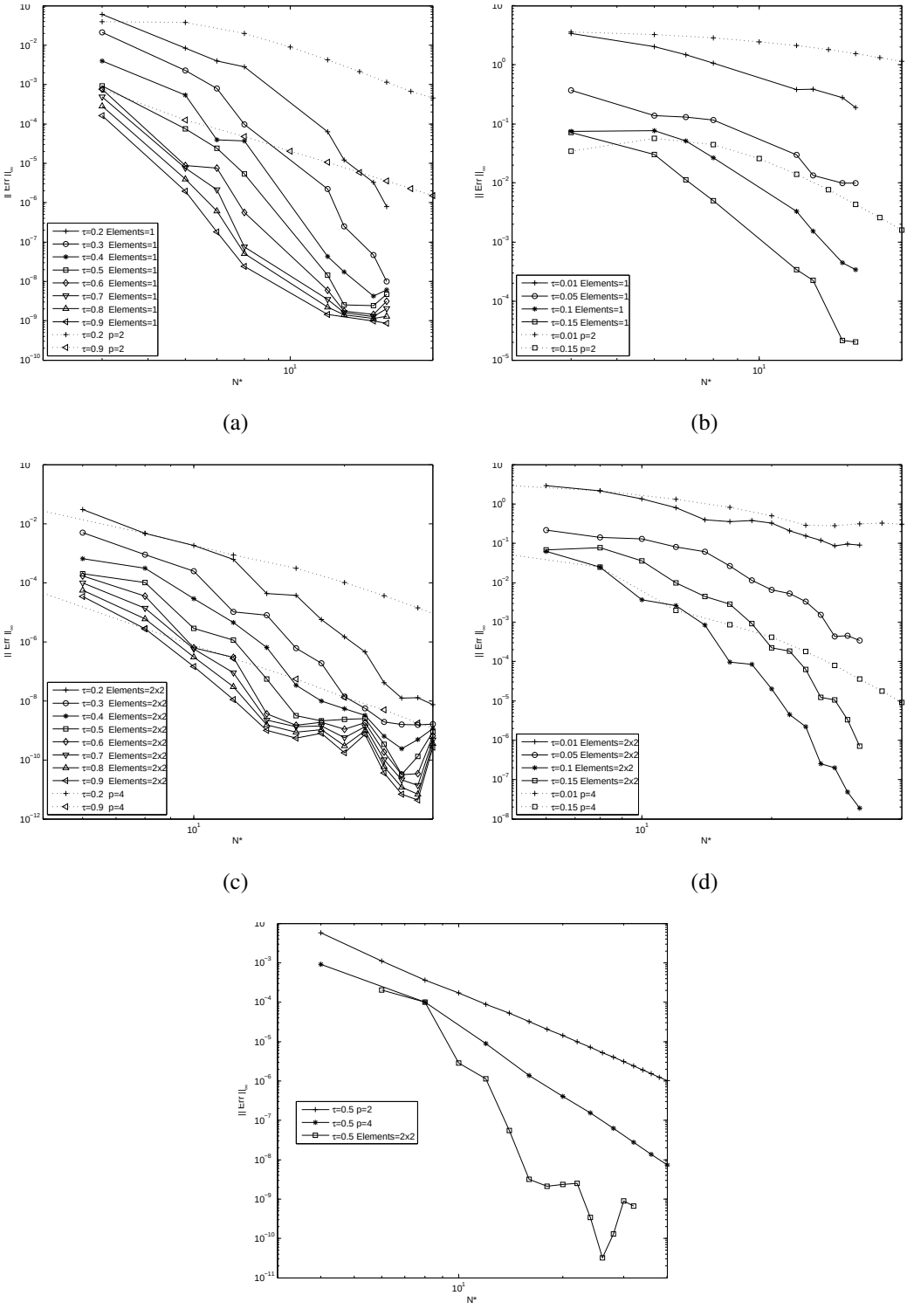


Figura 3: Norma infinito del error según $N*$. (a) $\tau \in [0{,}2, 0{,}9]$ para una malla de 1 elemento o $p = 2$. (b) $\tau \in [0{,}01, 0{,}15]$ para una malla de 1 elemento o $p = 2$. (c) $\tau \in [0{,}2, 0{,}9]$ para una malla de 2x2 elementos o $p = 4$. (d) $\tau \in [0{,}01, 0{,}15]$ para una malla de 2x2 elementos o $p = 4$. (e) $\tau = 0{,}5$, $p = 2$ o $p = 4$ o para una malla de 2x2 elementos.

En la figura 3 se muestran los resultados de los distintos experimentos. En la figura 3a se tomaron distintos valores del parámetro τ dentro del intervalo $[0{,}2; 0{,}9]$ con una malla de 1 elemento donde la cantidad de nodos que lo componen es variable. Las líneas punteadas corresponden a los experimentos realizados para $\tau = 0{,}2$ y $\tau = 0{,}9$ para elementos compuestos por 9 nodos ($p = 2$) variando la cantidad de elementos.

En la figura 3b se tomaron distintos valores del parámetro τ dentro del intervalo $[0{,}01; 0{,}15]$ con una malla de 1 elemento donde la cantidad de nodos que lo componen es variable. Las líneas punteadas corresponden a los experimentos realizados para $\tau = 0{,}01$ y $\tau = 0{,}15$ para elementos compuestos por 9 nodos ($p = 2$) variando la cantidad de elementos.

En la figura 3c se tomaron distintos valores del parámetro τ dentro del intervalo $[0{,}2; 0{,}9]$ con una malla de 2x2 elementos donde la cantidad de nodos que los componen es variable. Las líneas punteadas corresponden a los experimentos realizados para $\tau = 0{,}2$ y $\tau = 0{,}9$ para elementos compuestos por 25 nodos ($p = 4$) variando la cantidad de elementos.

En la figura 3d se tomaron distintos valores del parámetro τ dentro del intervalo $[0{,}01; 0{,}15]$ con una malla de 2x2 elementos donde la cantidad de nodos que los componen es variable. Las líneas punteadas corresponden a los experimentos realizados para $\tau = 0{,}01$ y $\tau = 0{,}15$ para elementos compuestos por 25 nodos ($p = 4$) variando la cantidad de elementos.

Para la figura 3e se tomó como parámetro $\tau = 0{,}5$ y se muestran los resultados para $p = 2$ variando la cantidad de elementos, para $p = 4$ variando la cantidad de elementos y para una malla de 2x2 elementos, variando la cantidad de nodos que los componen.

En todos los gráficos se toma como referencia a N^*, siendo $(N^* + 1)^2$ la cantidad de nodos en la malla y se muestra el error absoluto con respecto a la solución analítica.

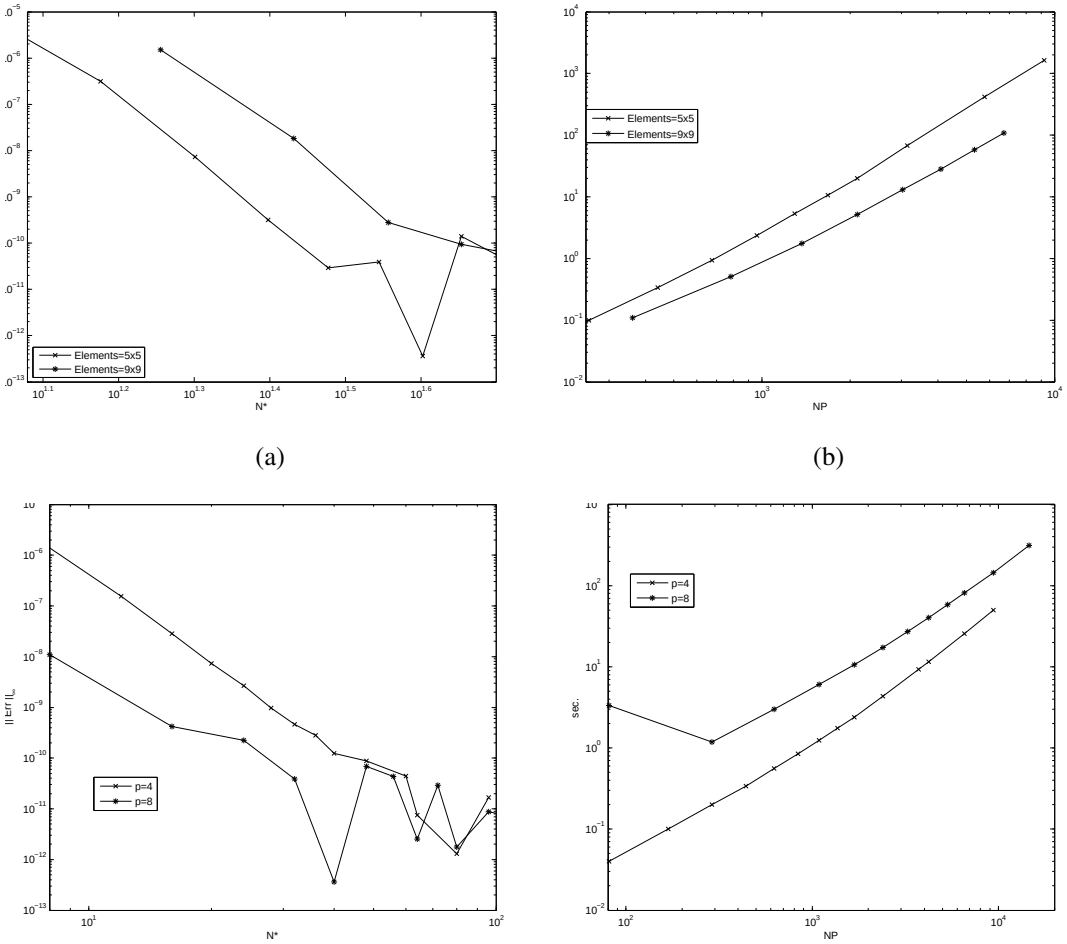
En los mismos se puede observar la convergencia espectral y cómo a medida que se aumenta la definición sobre p converge más rápidamente a la solución analítica que al aumentar la definición sobre h. Cabe destacar la convergencia lineal presente, típica del refinamiento sobre h en elementos finitos. Este comportamiento puede observarse también en los trabajos de Otero and Ponta (2006).

En la figura 4 se presentan 4 gráficos que comparan los refinamientos p y h en relación al error absoluto con respecto a la solución analítica y el tiempo total de procesamiento. En las figuras 4a y 4b se tomaron mallas de 5x5 elementos y de 9x9 elementos variando la cantidad de nodos que componen a cada elemento tomando como parámetro $\tau = 1$. En el primero se muestra el error absoluto con respecto a la solución analítica, mientras que en el segundo se muestra el tiempo total de procesamiento.

En las figuras 4c y 4d se tomaron mallas con elementos de 5x5 nodos y elementos de 9x9 nodos variando la cantidad de elementos dentro de la malla con parámetro $\tau = 1$. En el primero se muestra el error absoluto con respecto a la solución analílica, mientras que en el segundo se muestra el tiempo total de procesamiento.

Una característica que se puede deducir es que a medida que aumenta N^*, el tiempo insumido aumenta considerablemente. El tiempo medido en todos los casos corresponde al necesario para ensamblar las matrices elementales (ASSEMBLY TIME), imponer las condiciones de borde (BOUNDARY CONDITIONS IMPOSITION TIME) y resolver el sistema (SOLVE TIME). La figura 5 muestra cuánto tiempo se invierte en cada una de dichas etapas con respecto al tiempo total. Para esto, se tomó el tiempo de procesamiento para distintas resoluciones con mallas con elementos de 9x9 ($p = 8$), con $\tau = 1$ como parámetro y cantidad de elementos variable en la malla. Puede observarse que una proporción importante del tiempo total insumido para la resolución corresponde al ensamble de las matrices elementales.

Un dato importante relacionado con el tiempo de procesamiento, es que la primera vez que



(a) (b)

(c) (d)

Figura 4: (a): Error absoluto con respecto a la solución analílitca con mallas de 5x5 elementos y de 9x9 elementos; (b): Tiempo de procesamiento con mallas de 5x5 elementos y de 9x9 elementos; (c): Error absoluto con respectoa la solución analílitca con mallas de 5x5 nodos por elemento y de 9x9 nodos por elemento; (d): Tiempo de procesamiento con mallas de 5x5 nodos por elemento y de 9x9 nodos por elemento;

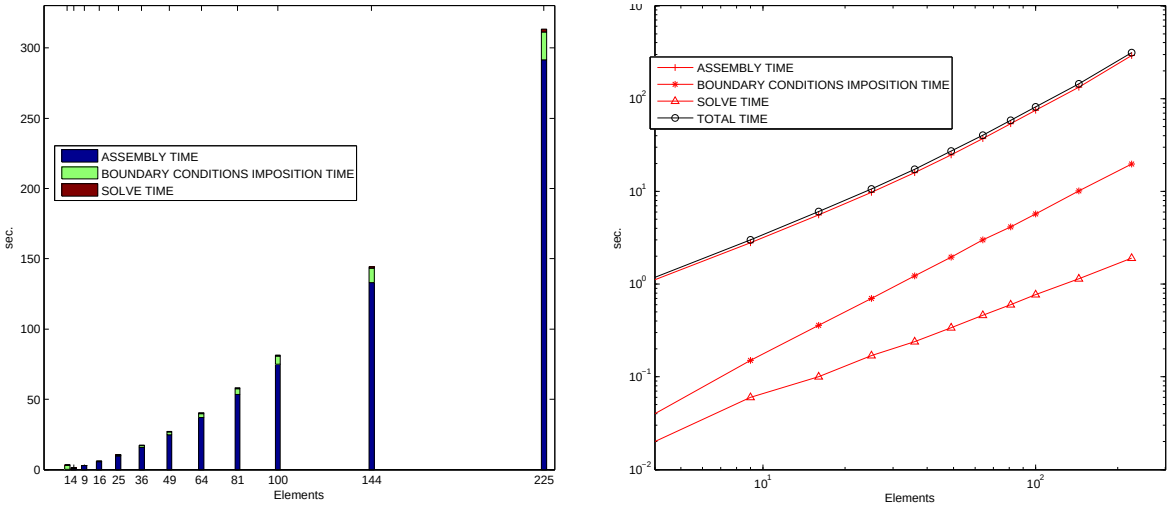


Figura 5: Tiempos de procesamiento en escalas normal y logarítmica.

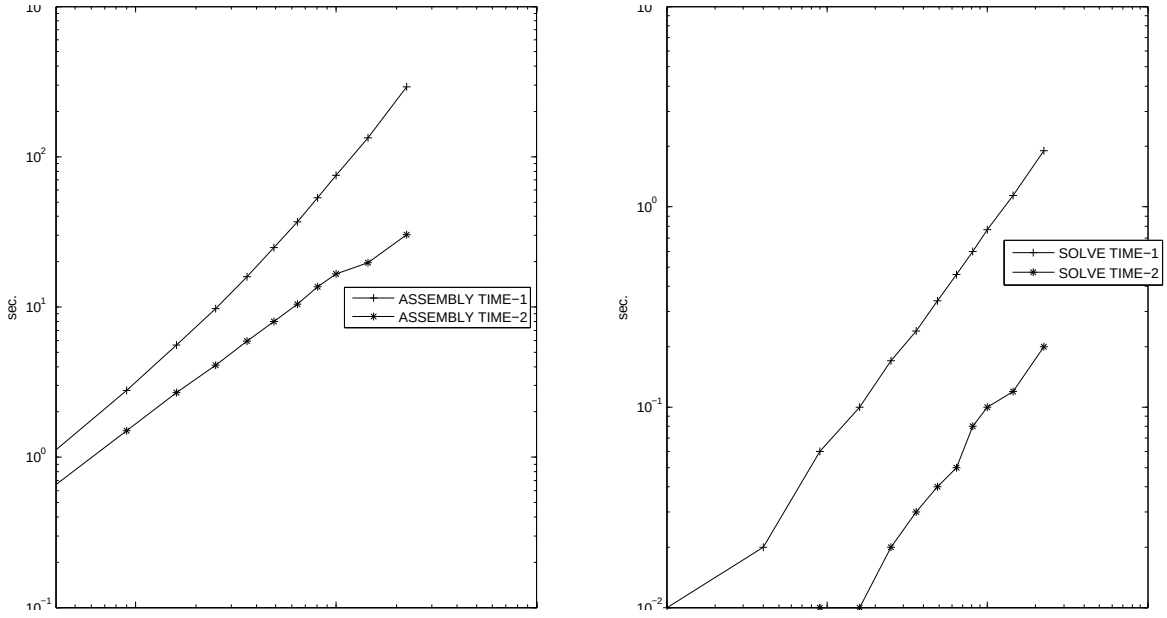


Figura 6: Comparación de tiempos de ensamblado y resolución del sistema.

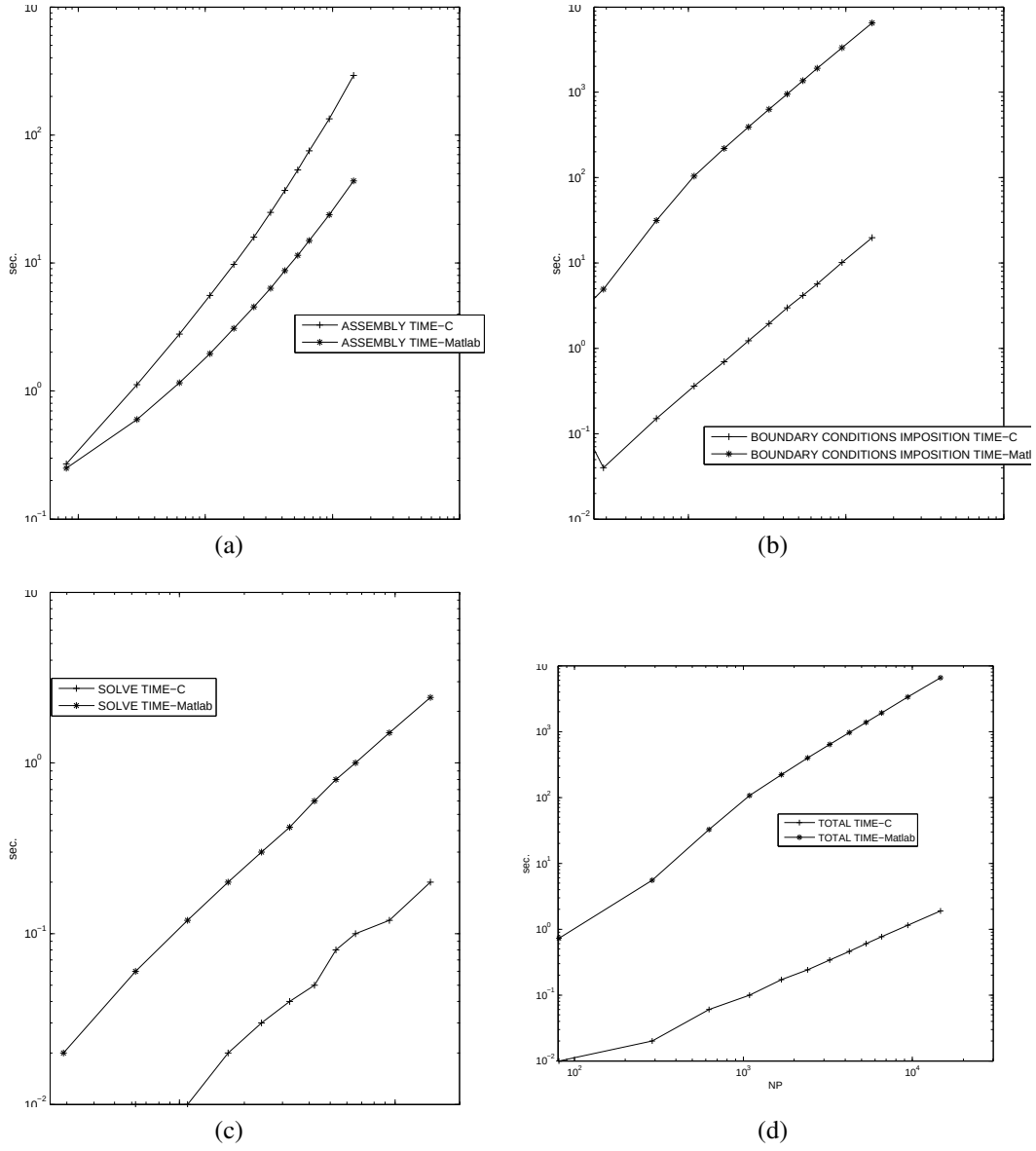


Figura 7: Comparación del tiempo de procesamiento en la implem. de C++ y la de MATLAB. (a): Ensamble. (b): Imposición de condiciones de borde. (c): Resolución del sistema. (d): Tiempo total.

se ensamblan las matrices elementales, se generan las estructuras de las matrices esparsas en memoria, algo que en los sucesivos ensambles no será necesario volver a hacer. Algo parecido ocurre con los tiempos de resolución del sistema, donde la primera vez se calculan las matrices L y U correspondientes a la matriz elemental pudiendo ser reutilizadas en las siguientes resoluciones. En la figura 6 se puede observar la difencia de tiempo con respecto al primer ensamblado (ASSEMBLY TIME 1) y los siguientes (ASSEMBLY TIME 2) y al tiempo de la primer resolución del sistema (SOLVE TIME 1) y los posteriores (SOLVE TIME 2). De ambos gráficos se puede deducir que el tiempo necesario para la primer resolución total del sistema será considerablemente mayor que en los sucesivas y se notará aún más la diferencia para problemas con mallas más complejas, es decir, con un valor de N^* grande. Para realizar dichos gráficos se tomó el tiempo de procesamiento para distintas resoluciones con mallas con elementos de 9x9 ($p = 8$), con $\tau = 1$ como parámetro y cantidad de elementos variable en la malla.

Por último, se consideró el tiempo de resolución del mismo problema en idénticas mallas tanto en MATLAB como en C++ para poder comprobar que la implementación realizada efectivamente mejora los tiempos de ejecución. Las mallas consideradas, están compuestas por elementos de 9x9 ($p = 8$), con $\tau = 1$ como parámetro y cantidad de elementos variable en la malla. En la figura 7 puede observarse que se obtuvieron importantes mejoras de tiempo en todas las etapas.

6. CONCLUSIONES Y TRABAJOS EN PROGRESO

En este trabajo se presentó una nueva implementación del método KLE en C++ sobre un framework orientado a la resolución de numérica de problemas por el FEM. Se presenta un análisis detallado en cuanto a la convergencia a la solución analítica de las distintas configuraciones según tipo y cantidad de elementos. Los resultados expuestos muestran un total acuerdo desde el punto de vista numérico con los previamente publicados.

Los tiempos de ejecución fueron notoriamente mejorados con respecto a implementaciones previas. Las razones de la disminución de los tiempos de ejecución se deben en parte a la implementación mediante un lenguaje de alto rendimiento como C++ y, por otro lado, a la mejora en el diseño de las clases y la consecuente reutilización de código.

Se amplió la funcionalidad del framework mediante la incorporación de elementos espectrales con cantidad parametrizable de nodos. Asimismo, la inclusión de un parser para permitir calcular las condiciones de borde dependientes de variales como tiempo o coordenadas espaciales en tiempo de ejecución, generaliza aún más las posibilidades de analizar casos con condiciones de borde complejas por medio de parámetros de entrada adecuados.

Tomando este trabajo como base, se puede extender fácilmente hacia la resolución de pasos de tiempo sucesivos mediante un algoritmo de integración de ecuaciones diferenciales ordinarias (ODE).

Asimismo, se está llevando a cabo en estos momentos la paralelización del código por el método de las subestructuras.

7. AGRADECIMIENTOS

Alejandro Otero y Javier Quinteros desean agradecer el apoyo brindado por el CONICET para la realización de este trabajo mediante las becas de Postdoctorado asignadas.

REFERENCIAS

Anderson E., Bai Z., Bischof C., Blackford S., Demmel J., Dongarra J., Du Croz J., Greenbaum A., Hammarling S., McKenney A., and Sorensen D. *LAPACK users' guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8 (paperback).

Batchelor G.K. *An introduction to fluid dynamics*. Cambridge University Press, Cambridge, UK, 2000.

Bathe K.J. *Finite element procedures*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1996.

Berg I. Muparser 1.28. http://muparser.sourceforge.net/, 2005.

Demmel J.W., Eisenstat S.C., Gilbert J.R., Li X.S., and Liu J.W.H. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Analysis and Applications*, 20(3):720–755, 1999.

Henderson R.D. and Karniadakis G.E. Unstructured spectral element methods for simulation of turbulent flows. *J. Comput. Phys.*, 122:191–217, 1995.

Hourigan K., Thompson M.C., and Tan B.T. Self–sustained oscilations in flows around long blunt plates. *J. Fluids Struct.*, 15:387–398, 2001.

Karniadakis G.E., Bullister E.T., and Patera A.T. A spectral element method for solution of two- and three-dimensional time-dependent incompressible navier-stokes equations. In *Finite Element Methods for Nonlinear Problems*, page 803. Springer-Verlag, New York/Berlin, 1985.

Otero A.D. and Ponta F.L. Spectral–element implementation of the KLE method: a $(\omega, \mathbf{v})$ formulation of the Navier–Stokes equations. *Mecánica Computacional*, 25:2649–2667, 2006.

Patera A.T. A spectral element method for fluid dynamics: laminar flow in a channel expansion. *J. Comput. Phys.*, 54:468–488, 1984.

Ponta F.L. and Aref H. Vortex synchronization regions in shedding from an oscillating cylinder. *Phys. Fluids*, 17:011703, 2005.

Quinteros J., Jacovkis P.M., and Ramos V.A. Diseño flexible y modular de modelos numéricos basados en elementos finitos. *Mecánica Computacional*, 26:1724–1740, 2007.