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

Bursztyn, Gabriel;
Directores: Quinteros, Javier; Otero Alejandro

Fac. Ciencias Exactas y Naturales
Univesidad de Buenos Aires

Resumen En este trabajo se presenta una implementación realizada en lenguaje C++ del Método KLE (método de la ecuación laplaciana cinemática). Este método resuelve las ecuaciones de Navier–Stokes de fluidodinámica en base a la formulación vorticidad-velocidad de las mismas.
En esta implementación, se utilizó un conjunto de clases y bibliotecas diseñadas especialmente para el desarrollo de sistemas que resuelven problemas mediante el método de los elementos finitos. El framework adoptado fue extendido mediante la inclusión de código para el manejo de elementos espectrales y de condiciones de bordes variables en función de tiempo y espacio, entre otras. La mejora en el diseño de las clases junto con la consecuente reutilización de código, además de otras características, hacen que esta implementación presente mejores condiciones de escalabilidad respecto de versiones anteriores.
La implementación fue validada mediante la resolución de problemas con solución conocida. Las mediciones realizadas indican que se lograron mejoras importantes en cuanto a tiempo de procesamiento en comparación a implementaciones anteriores de este método.

1. Introducción

Existen fenómenos físicos en la naturaleza cuyo comportamiento, por lo general, no puede ser representado mediante una función analítica o bien existe dicha función pero no nos es conocida. Cuando no puede calcularse la solución matemática exactamente, es necesario utilizar métodos numéricos para aproximar la misma. En el caso que estos fenómenos físicos puedan ser expresados mediante ecuaciones en derivadas parciales (PDE), se pueden utilizar varios métodos como, por ejemplo, el método de elementos finitos. El método de elementos finitos (FEM) consiste en partir el problema que se quiere resolver en problemas más pequeños, dividiendo el dominio en lo que se llama elementos. Cada uno de estos elementos está definido mediante nodos y posee puntos de integración de donde se toman los valores que cada elemento aporta al problema.

Para la descripción de fluidos en movimiento se suele utilizar 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. Normalmente no existe solución analítica para dichas ecuaciones, salvo en el caso de algunos flujos simples en dominios poco complejos. Debido a esto, la solución numérica de la solución de las ecuaciones de Navier–Stokes resulta un activo campo de investigación y es utilizada en una amplia gama de especialidades. El método de la ecuación cinemática laplaciana, *Método KLE*, desarrollado por Ponta (2005), resuelve las ecuaciones de Navier-Stokes en base a la formulación vorticidad-velocidad de las mismas. El KLE calcula la evolución en el tiempo de la vorticidad como una ecuación diferencial ordinaria (ODE) 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 PDE 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 PDE por el método de los elementos finitos. 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 multifísica.

2. Método KLE

2.1. La Ecuación Laplaciana Cinemática

En el trabajo de Ponta (2005) se presentó un nuevo método basado en la formulación vorticidad-velocidad ($\boldsymbol{\omega}$, $\boldsymbol{v}$) para modelar las ecuaciones de Navier–Stokes . El método de la ecuación laplaciana cinemática (*Kinematic Laplacian Equation* - 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 con respecto a las seis que poseen las formulaciones clásicas. Además, esta descomposición del problema permite la utilización de algoritmos de integración de 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, es decir, la ecuación de transporte de vorticidad, que se resuelve como una ODE en cada nodo de la discretización. Ambas partes se realimentan mutuamente y en conjunto reciben la denominación de método KLE.

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

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

donde $\boldsymbol{v}$ es el campo de velocidades, $\boldsymbol{\omega}$ es la distribución del campo de vorticidades y $\alpha_{\mathcal{D}}$ y α_{ω} son las constantes de penalización correspondientes a las restricciones $\boldsymbol{\nabla}\cdot\boldsymbol{v} = \mathcal{D}$ y $\boldsymbol{\nabla}\times\boldsymbol{v} = \boldsymbol{\omega}$, siendo $\mathcal{D}$ la tasa de expansión.

2.2. Evolución Temporal del Flujo

El método KLE utiliza como ecuación para representar la dinámica del flujo las ecuaciones de Navier–Stokes en vorticidad,

$$\frac{\partial \boldsymbol{\omega}}{\partial t} = -\boldsymbol{v}\cdot\boldsymbol{\nabla\omega} + \nu\nabla^2\boldsymbol{\omega} + \boldsymbol{\omega}\cdot\boldsymbol{\nabla v}, \tag{2}$$

obtenidas a partir de considerar las ecuaciones de Navier–Stokes tridimensionales completas para flujo incompresible expresadas en vorticidad en un dominio Ω, con contorno sólido $\partial\Omega$ y el contorno *externo* suficientemente alejado, en un marco de referencia fijo al sólido.

Si en determinado instante se conoce el campo de velocidades $\boldsymbol{v}$ en todo el dominio Ω se puede reescribir la ecuación 2 como

$$\frac{\partial \boldsymbol{\omega}}{\partial t} = -\boldsymbol{v}\cdot\boldsymbol{\nabla}(\boldsymbol{\nabla}\times\boldsymbol{v}) + \nu\nabla^2(\boldsymbol{\nabla}\times\boldsymbol{v}) + (\boldsymbol{\nabla}\times\boldsymbol{v})\cdot\boldsymbol{\nabla v}, \tag{3}$$

de forma que se conoce la derivada temporal de la vorticidad $\boldsymbol{\omega}$ en ese instante en cada punto de discretización en el dominio Ω y se puede obtener $\boldsymbol{\omega}$ en un instante posterior integrando la ecuación (ec. 3 mediante un algoritmo de integración de ODE.

De esta forma, se utiliza la formulación variacional (ec. 1) de la ecuación Laplaciana cinemática como contraparte de la ecuación de transporte de la vorticidad (ec. 2) en una formulación híbrida *vorticidad–velocidad* para aproximar la solución de las ecuaciones de Navier–Stokes . La integración en el tiempo de la ecuación de transporte de vorticidad produce la distribución de vorticidad $\boldsymbol{\omega}$ en el dominio Ω. En flujos incompresibles, la tasa de deformación $\mathcal{D}$, i.e. la divergencia de la velocidad, se asume nula.

2.3. Discretización del método

La formulación variacional de la ecuación 1 se discretiza por medio del método de elementos espectrales desarrollado en el apéndice A de este trabajo. 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. 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). Una vez ubicados los nodos del elemento maestro según lo descripto en Otero (2008), 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. En el apéndice A se describe la discretización KLE según Otero y Ponta (2006); Otero (2008) que desencadena en el sistema global a resolver

$$\boldsymbol{K}\,\hat{\mathbf{V}} = \boldsymbol{R}\,\hat{\boldsymbol{\omega}}. \tag{4}$$

Para evaluar el lado derecho del sistema de ecuaciones diferenciales ordinarias que describe la evolución temporal, la ecuación 4 se reescribe como:

$$\frac{\partial \boldsymbol{\omega}}{\partial t} = \boldsymbol{\mathcal{F}}\left(\boldsymbol{\omega}, t\right) = \boldsymbol{\nabla} \times \left(\nu\, \boldsymbol{\nabla}\cdot\boldsymbol{\nabla}\boldsymbol{v} - \boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{v}\right), \tag{5}$$

y luego de discretizarla, resulta:

$$\boldsymbol{\mathcal{F}}(\hat{\boldsymbol{\omega}}, t) = \hat{\boldsymbol{C}}_{\boldsymbol{url}} \left(\nu\, \hat{\boldsymbol{D}}_{\boldsymbol{iv}} - \hat{\boldsymbol{V}}_{\boldsymbol{adv}}\right) \hat{\boldsymbol{G}}_{\boldsymbol{rad}} \hat{\mathbf{V}}. \tag{6}$$

3. Extensión del esquema e implementación del KLE

En este trabajo se utilizó el framework propuesto por Quinteros et al. (2007) para implementar modelos numéricos basados en FEM. Este framework está basado en un estructura de clases que representa cada una de las entidades que suelen formar parte de un problema resuelto por FEM. Una descripción detallada de las características del framework puede encontrarse en Quinteros et al. (2007); Quinteros (2008), mientras que aquí sólo se mencionan las características más importantes (ver Apéndice B).

En este trabajo se utiliza un tipo particular de elementos finitos llamados elementos espectrales para implementar el método KLE. Estos elementos no estaban incluidos en la implementación original del framework, por lo que se realizó una extensión en la estructura de clases, para incorporarlo. Una vez incorporados, los elementos espectrales, pueden utilizarse para resolver cualquiera de las ecuaciones ya implementadas. De esta forma, sólo se deberá implementar la formulación variacional utilizada y el método de integración elemental, utilizando todas las operaciones provistas.

3.1. Clases generadoras de nodos y puntos de Gauss

Se creó una clase llamada `Generator` la cual se encarga de calcular la distribución de puntos que se le solicite, junto con los pesos asociados a ellos, ya sea la distribución Gauss–Lobatto como la de Gauss–Legendre, mediante los métodos `getGaussLobattoPoints` y `getGaussLegendrePoints` respectivamente. Estos puntos son utilizados para ubicar en una grilla nodos o puntos de integración.

Para generar los nodos del elemento maestro que se corresponden con los puntos de una grilla Gauss–Lobatto se creó la clase `NodesGenerator`. Esta clase posee un constructor en el cual se le indica la cantidad de nodos que se desean (`nodesCount`) y si el resultado será en 1 ó 2 dimensiones (`dimension`). Luego, el método `getNodes` es el encargado de devolver el vector de nodos calculados. Los nodos devueltos están ordenados según una permutación definida de forma que la numeración sea análoga a la utilizada para los elementos finitos isoparamétricos clásicos en Bathe (1996).

De la misma forma, para crear los puntos de integración del elemento maestro se creó la clase `GaussPointGenerator`. Su constructor recibe la cantidad de puntos de Gauss solicitados (`iGaussPointCount`), si el resultado será en 1 ó 2 dimensiones (`dimension`) y, de forma opcional, se puede especificar la posición de los nodos en la cual se basará el cálculo (`pNodes`). En caso que no se especifique este último parámetro se utilizará por defecto la posición de los nodos correspondientes a la distribución Gauss–Lobatto. El método `getGaussPoints`, además de devolver los puntos de Gauss en un vector, posee un parámetro que se pasa por referencia llamado `pIDclosestGP`, que indica para cada nodo, el punto de integración más cercano. Esto resulta de utilidad para el método de derivación–promediación–reproyección expuesto en el apéndice A. Es posible elegir entre ambas formas de distribución de puntos de integración, la de Gauss–Lobatto o la de Gauss–Legendre mediante el parámetro `iGMethod`.

Ambas clases, `NodesGenerator` y `GaussPointGenerator`, utilizan una instancia de la clase `Generator` en sus cálculos y llaman al método `getPoints` definiendo el método de distribución de puntos.

3.2. Intérprete de condiciones de borde

Para calcular las condiciones de borde definidas por una función 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 el tiempo (t). Este agregado, brinda la posibilidad de definir problemas más complejos de manera más simple.

El modelo recibe como parámetro el nombre de un archivo de configuración donde están definidas la función de la vorticidad w, las condiciones de borde de velocidad y sus derivadas parciales como función del tiempo.

3.3. Extensión para elementos espectrales

A la clase `Element` del framework mencionado anteriormente se agregaron los métodos *calc_kle* y *calc_kleOP* que implementan el método KLE como puede verse en la figura 1. Se mantuvo la metodología mencionada en el apéndice B, donde existe un método `calc_equation` que invoca a otro método llamado `eval_equation` (donde `equation` es el nombre que define el problema).

A su vez, se creó una clase `Spectral`, la cual implementa los elementos espectrales con cantidad de nodos parametrizable. Esta clase, respetando la estructura del framework, hereda de `Element` todas sus propiedades y métodos.

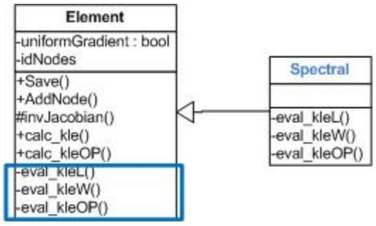


Figura 1: Clases `Element` y `Spectral`.

El método *calc_kle* se encarga de realizar la integración en los puntos de Gauss para obtener las matrices de la ecuación 4. Este método, para calcular las matrices elementales $\boldsymbol{K^e}$ y $\boldsymbol{R^e}$ de la ecuación 9 (ver apéndice A), utiliza los métodos `eval_kleL` y `eval_kleW` para poder dividir las operaciones correspondientes a la formulación laplaciana básica de las correspondientes a la penalización de las condiciones de vorticidad y divergencia de la velocidad respectivamente.

El método *calc_kleOP* también realiza integraciones en los puntos de Gauss, pero para obtener los operadores diferenciales necesarios para la ecuación 6. Este método llama al método `eval_kleOP` para calcular las matrices evaluadas en los puntos de integración necesarios para construir los operadores de derivación utilizados para evaluar el lado derecho de la función de integración de ODEs en sucesivos pasos de tiempo (Ponta, 2005; Otero y Ponta, 2006).

3.4. Implementación del método KLE

Se trabajó con un esquema del método KLE como el de la figura 2, el cual es detallado en las siguientes subsecciones.

Discretización del dominio

Como entrada se utiliza un archivo de texto generado con el *GID* (ver apéndice B) o con una rutina que se desarrolló en Matlab mediante la cual, especificando la cantidad de elementos y el orden de los mismos, se genera una

- Discretización del dominio
- Generación de la estructura de matrices
- Para cada elemento
  - Cálculo de las matrices elementales e integración en los puntos de Gauss
  - Ensamble de matrices globales
  - Cálculo de los operadores diferenciales
- Imposición de las condiciones de borde
- Resolución de las ecuaciones del sistema

Figura 2: Esquema del método KLE.

malla regular manteniendo el mismo formato que el archivo de salida de *GID*. Mediante este archivo se crea una instancia de la clase `Domain` que representa al dominio, con todos los elementos, nodos y condiciones de borde.

Por último, mediante la clase `GaussPointGenerator` se generan los puntos de Gauss y se calculan las funciones de forma y sus derivadas, interpolando sobre el dominio del elemento. Los puntos de Gauss y toda la información asociada es almacenada en las variables estáticas `gpsfull` y `gps` como se explica en el apéndice B.

Generación de la estructura de matrices

Las matrices que describen el sistema de ecuaciones, como $\boldsymbol{K}$ y $\boldsymbol{R}$ (ec. 4) suelen tener una cantidad pequeña de valores distintos de cero. Son representadas como instancias de la clase `SparseMatrix` y su estructura es construida en base a la matriz de conectividad que relaciona los elementos y los nodos.

De forma análoga, se construyen las estructuras de las matrices `BRot`, `BGrad` y `BDiv` que representan a los arreglos que calculan en forma discreta respectivamente los operadores diferenciales de la ecuación 6.

Integración elemental y ensamble global

Como se puede observar en el diagrama de la figura 2, para cada elemento se calculan las matrices elementales. Para esto, se debe integrar en los puntos de Gauss mediante el método `calc_kle`. Mediante los métodos `eval_kleL` y `eval_kleW` se realizan las integraciones correspondientes para calcular estas matrices (para más detalles ver ecuación 9 del apéndice A). Tomando como ejemplo 2 de estas matrices

$$\boldsymbol{K}_{\boldsymbol{L}}^{\boldsymbol{e}} = \int_{\Omega^e} \mathbf{B}^{eT}\mathbf{B}^e \,\mathrm{d}\Omega = \int_{-1}^{1}\int_{-1}^{1} \mathbf{B}^{eT}\mathbf{B}^e \,|\boldsymbol{J}|\,\mathrm{d}r\mathrm{d}s$$

$$\boldsymbol{R}_{\boldsymbol{L}}^{\boldsymbol{e}} = \int_{-1}^{1}\int_{-1}^{1} \mathbf{H}^{eT}\mathbf{B}_{\boldsymbol{\omega}}^{e} \,|\boldsymbol{J}|\,\mathrm{d}r\mathrm{d}s,$$

el método `calc_kle` utiliza para cada punto de Gauss al método `eval_kleL` para calcular $\mathbf{B}^{eT}\mathbf{B}^{e}\,|\boldsymbol{J}|$ y $\mathbf{H}^{eT}\mathbf{B}^{e}_{\boldsymbol{\omega}}\,|\boldsymbol{J}|$ y realiza la suma del resultado multiplicado por el peso correspondiente al punto de integración para realizar la doble integral de ambas ecuaciones para calcular $\boldsymbol{K}^{\boldsymbol{e}}_{\boldsymbol{L}}$ y $\boldsymbol{R}^{\boldsymbol{e}}_{\boldsymbol{L}}$. Mediante la suma de todas estas matrices se obtienen las matrices elementales.

Una vez obtenidas las matrices elementales $\boldsymbol{K}^{\boldsymbol{e}}$ y $\boldsymbol{R}^{\boldsymbol{e}}$ se ensamblan en las matrices ralas para calcular las matrices globales. La clase `SparseMatrix` posee una interface que facilita esta operación:

$$\boldsymbol{K} = \sum_{e} \boldsymbol{K}^{\boldsymbol{e}}$$

donde la sumatoria es un mapeo entre numeración elemental y global de los nodos. Lo mismo se realiza con la matriz global $\boldsymbol{R}$ y las matrices elementales $\boldsymbol{R}^{e}$.

Cálculo de los operadores diferenciales

Para la integración temporal se calculan los operadores diferenciales ($\hat{\boldsymbol{C}}_{\boldsymbol{url}}$, $\hat{\boldsymbol{G}}_{\boldsymbol{rad}}$ y $\hat{\boldsymbol{D}}_{\boldsymbol{iv}}$) utilizados en la evaluación del lado derecho de la ecuación de transporte de vorticidad (ec. 5). Estos operadores son calculados mediante una integración completa en los puntos de Gauss que se lleva a cabo mediante el método `calc_kleOP` de la clase `Element`. El método `calc_kleOP` tiene su correspondiente método `eval_kleOP`. Finalmente se ensamblan las matrices elementales dentro de las matrices globales correspondientes a los operadores. Con el propósito de optimizar los tiempos de procesamiento, este cálculo se realiza dentro del mismo ciclo elemental junto con el cálculo de las matrices $\boldsymbol{K}$ y $\boldsymbol{R}$ como puede observarse en el esquema de la figura 2 de forma tal de recorrer una sola vez todos los elementos.

Imposición de las condiciones de borde y resolución del sistema

A esta altura del programa, ya se tienen calculadas las matrices globales y resta imponer las condiciones de borde y resolver el sistema planteado por la ecuación 4. Tanto las condiciones de borde como el vector de vorticidad $\hat{\boldsymbol{\omega}}$ provienen de un archivo de configuración que es interpretado por el sistema. La matriz $\boldsymbol{R}$ fue calculada en el ensamble de las matrices junto con la matriz de rigidez $\boldsymbol{K}$. Para simplificar las operaciones, se define un vector $\boldsymbol{F} = \boldsymbol{R}\,\hat{\boldsymbol{\omega}}$. por lo que el sistema a resolver se convierte en

$$\boldsymbol{K}\;\hat{\mathbf{V}} = \boldsymbol{F}. \tag{7}$$

$\hat{\mathbf{V}}$ es el vector de incógnitas que queremos obtener luego de resolver el sistema.

El método `solve` de la clase `SparseMatrix` se encarga de resolver el sistema de ecuaciones utilizando algún solver como `Pardiso` (Pardiso, 2009), `MUMPS` (Mumps, 2009; Amestoy et al., 2001, 2006) o `SuperLU` (Demmel et al., 1999). Es importante notar que en esta sección no se utilizaron métodos de la clase `Spectral` que no estén definidos en la clase `Element`, por lo que el método KLE puede ser implementado con otro tipo de elemento de forma sencilla.

4. Resultados

Todos los experimentos fueron realizados en una PC con un procesador Intel Core 2 Duo de 1.86 GHz y 2GB de memoria RAM, bajo sistema operativo Gentoo Linux kernel 2.6.19.

En este trabajo se utilizaron mallas bidimensionales regulares con N_{el} elementos en cada dirección. Cada elemento, a su vez, de acuerdo a su orden p, tendrá N_{GL} nodos en cada dirección, con $N_{GL} = p + 1$, resultando un total de ${N_{GL}}^2$ nodos por elemento. La cantidad de nodos en total es indicada como NP y N^* es la inversa de la distancia internodal media. En todos los experimentos se resolvió el problema variando el orden de los elementos (refinamiento p), modificando la cantidad de nodos del mismo, como así también, la cantidad de elementos dentro de la malla (refinamiento h).

4.1. Validación de los resultados

Para validar los resultados obtenidos por medio de esta implementación se utilizó un campo de velocidades dado por la `función error` que corresponde a la solución del problema de la *placa plana infinita* que ha sido propuesto en Otero y Ponta (2006) como *benchmark* para métodos vorticidad-velocidad (ver Sec. 4.3 de Batchelor, 2000, entre otros). Primero, se definió este campo de velocidades del cual se calculó la vorticidad. Luego, imponiendo esa vorticidad en todo el dominio y las velocidades en el contorno de la malla, se resolvió la ecuación 4 de la KLE para recuperar las velocidades que se compararon con el campo original.

Precisión de la solución espacial de la velocidad

En los experimientos realizados, se utilizaron mallas de distinta definición modificando el número de intervalos en una dimensión N^* para distintos valores de un parámetro que depende del tiempo τ, con el fin de corroborar y verificar la convergencia con el aumento de N^*. Los mismos fueron realizados tanto para valores pequeños de τ que representan los primeros estadíos de la capa límite, donde las soluciones son menos suaves, como para valores de τ mayores donde las soluciones son más suaves.

En la figura 3 se muestran los resultados de los distintos experimentos. Se puede observar que a medida que N^* aumenta, el error disminuye. En la figura 3a se tomaron distintos valores del parámetro τ dentro del intervalo $[0{,}2; 0{,}9]$ y las líneas punteadas corresponden a los experimentos realizados para $\tau = 0{,}2$ y $\tau = 0{,}9$. Se puede observar que a partir de $\tau = 0{,}2$ todas las curvas muestran el comportamiento típico de la convergencia espectral, lo que permite validar el método y su implementación. Esto quiere decir, que las pendientes de las curvas aumentan en módulo al aumentar el orden p (Boyd, 2000). En otras palabras, a medida que la solución se vuelve más suave, al aumentar el valor de τ, la tasa de convergencia aumenta y las curvas de error se vuelven cada vez más suaves. Las curvas progresan con una disminución del error hasta un punto donde alcanzan

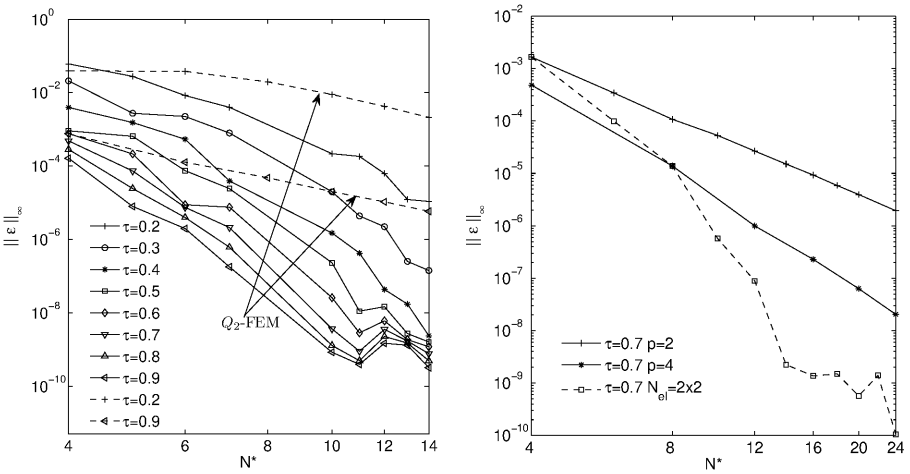
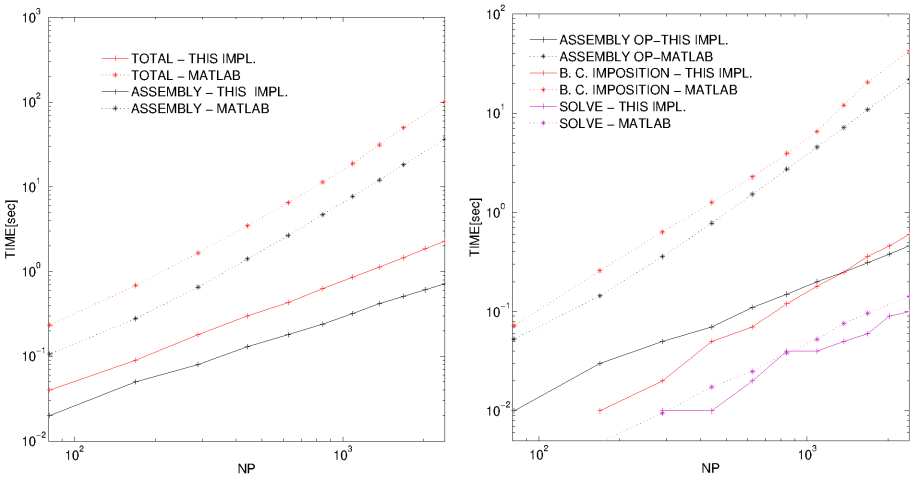


Figura 3: Error de la solución espacial para (a) una malla de 1 elemento comparado con el de $p = 2$ para $\tau \in [0{,}01, 0{,}15]$ y (b) comparando el refinamiento h con elementos de distinto orden y el refinamiento p con $\tau = 0{,}7$.

un límite inferior debido a la acumulación de errores numéricos que se generan durante el proceso de cálculo de las matrices y solución del sistema de ecuaciones.

A fin de poder comparar la convergencia del método cuando se realiza refinamiento p y refinamiento h, en la figura 3b se muestran los resultados para elementos de orden $p = 2$ y variando la cantidad de elementos y para una malla de 2×2 elementos variando su orden, para una solución correspondiente a un parámetro $\tau = 0{,}7$. Se puede observar la convergencia espectral y cómo a medida que se aumenta la cantidad de grados de libertad por refinamiento p, el método converge más rápidamente a la solución analítica que por refinamiento h. En el caso de los elementos finitos clásicos, el comportamiento normal resulta ser que al aplicar un refinamiento h se obtiene una recta de pendiente igual al orden p de los elementos en gráficos logarítmicos. Este comportamiento coincide con el obtenido por Otero y Ponta (2006) que es el esperado al utilizar elementos espectrales.

De forma parecida se realizaron experimentos para validar la precisión en el cálculo de los operadores diferenciales y se pudo observar que el refinamiento h da una recta cuya pendiente disminuye en módulo con el orden de las derivadas. En cambio, en las curvas de refinamiento p se pudo ver el comportamiento típico de la convergencia espectral. También, se comprobó que a medida que el cálculo precisa una derivada de orden mayor el error aumenta.



(a) (b)

Figura 4: Comparación de tiempos de procesamiento de esta implementación y la anterior con mallas de elementos de $N_{GL} = 5$. (a) total y ensamble y (b) ensamble de operadores.

4.2. Comparación con la implementación anterior del método KLE

Se analizó el tiempo de resolución del mismo problema en idénticas mallas, para la implementación realizada por Otero (2008) en Matlab (la cual denominaremos *implementación anterior*) y para esta implementación desarrollada en C++. Se consideraron mallas compuestas por elementos de distinto orden ($p = 2,\ 4$ y 8); con $\tau = 1$ como parámetro y cantidad de elementos variable. Se dividió el problema en 4 etapas: el ensamble de las matrices elementales (ASSEMBLY) para obtener las matrices de la ecuación 4, el ensamble de los operadores diferenciales (ASSEMBLY OP) necesarias para evaluar la ecuación 6, la imposición de las condiciones de borde (B.C. IMPOSITION) y la resolución del sistema (SOLVE) de la ecuación 4. En la figura 4 puede observarse que se obtuvieron importantes mejoras en el tiempo de ensamble (ASSEMBLY), en el tiempo de ensamble de los operadores diferenciales (ASSEMBLY OP) y en el tiempo total del sistema (TOTAL). Se puede observar que el tiempo de ensamble se encuentra cerca del tiempo total manteniendo una diferencia casi constante en el gráfico logarítmico, lo que implica que se mantendrá igual el orden de la diferencia al complejizar el problema. La diferencia en escala logarítmica entre el tiempo total de esta implementación y el tiempo total de la implementación anterior va aumentando a medida que se aumenta el orden de los elementos. Esto implica que esta implementación escalará mejor al aumentar el tamaño del problema. En los gráficos se observa que para todas las etapas, exceptuando los tiempos de resolución, hay una diferencia de al menos un orden de magnitud entre los tiempos de procesamiento de esta implementación y la anterior.

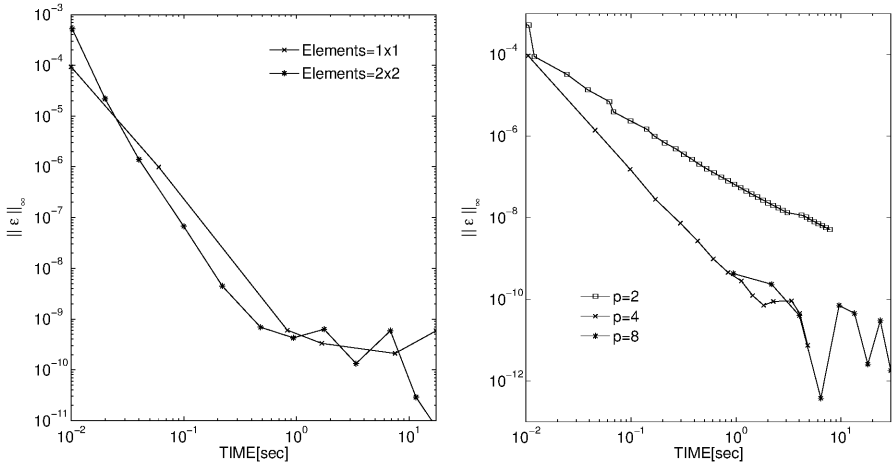
Una mejor medida para evaluar la escalabilidad se logra a través de la pendiente de las curvas presentadas en la figura 4. Las mismas fueron medidas para todas las etapas y casos estudiados y un resumen de éstas puede verse en el Cuadro 1. El orden de la dependencia del tiempo de cálculo con el tamaño del problema resulta aproximadamente la mitad en esta implementación respecto de la anterior, para todas las etapas salvo en la resolución del sistema de ecuaciones.

NGL	ASSEMBLY		ASSEMBLY OP		B.C. IMPOSITION		SOLVE	
	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
3	2.02	1.02	2.01	1.04	1.85	1.56	1.32	1.32
5	1.82	1.01	1.88	1.02	1.90	1.56	1.39	1.31
9	1.66	1.03	2.01	1.04	1.91	1.66	1.19	1.19

Cuadro 1: Pendientes de las curvas de tiempos de procesamiento. (a) Implementación anterior y (b) esta implementación

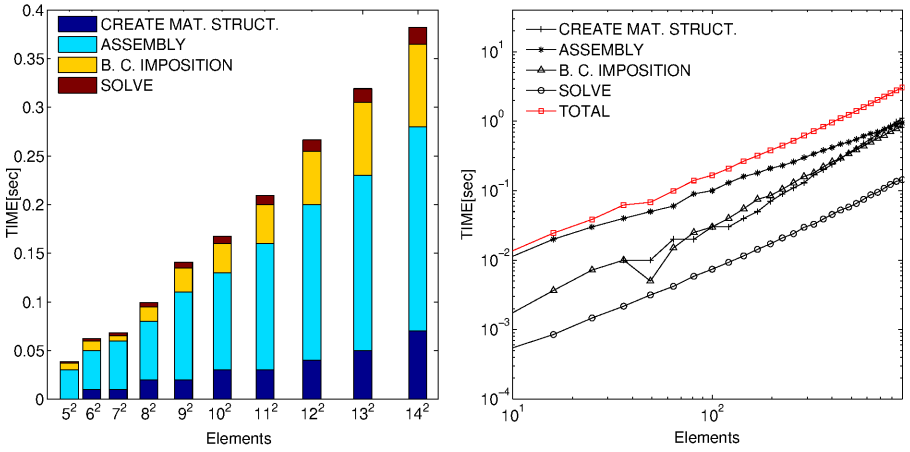
4.3. Influencia del orden p y la cantidad de elementos en la malla

Se realizaron una serie de experimentos con refinamiento p y refinamiento h para poder observar la forma en que cada uno de ellos afecta el tiempo de procesamiento y el error. En la figura 5b, donde se comparan mallas de elementos de diferente orden, se observa que para un mismo nivel de error se requiere más



(a) (b)

Figura 5: Comparación de error en función del tiempo de procesamiento (a) entre mallas de 1×1 y de 2×2 elementos y (b) mallas de $N_{GL} = 3,\ 5$ y 9.



(a) (b)

Figura 6: Tiempos de procesamiento en mallas de elementos de $N_{GL} = 3$ en (a) escala normal y (b) logarítmica.

esfuerzo computacional utilizando elementos de bajo orden ($p = 2$) que elementos de mediano y alto orden ($p = 4$ y $p = 8$). Un caso extremo puede verse en la figura 5a donde se compara una malla de 1 elemento con una de 2×2 elementos. Esto implica que los elementos en ambos casos tendrán ordenes muy elevados y para un esfuerzo equivalente las mallas de más elementos de ordenes moderados resultarán más precisas que las de ordenes más altos.

A fin de poder estudiar de forma más profunda el impacto que genera realizar un refinamiento p o un refinamiento h, se midió el tiempo de procesamiento de forma discriminada, separando el proceso total en el tiempo incurrido para crear la estructura de las matrices (CREATE MAT. STRUCT.) como se explica en la sección 3, ensamblar las matrices elementales para obtener las matrices globales de la ecuación 4 (ASSEMBLY), imponer las condiciones de borde (B. C. IMPOSITION) y resolver el sistema (SOLVE) de la ecuación 4. En todos los casos se tomó como parámetro un valor de $\tau = 1$.

En la figura 6 se observa la distribución de tiempos para experimentos realizados con mallas de elementos de orden $p = 2$. Se puede ver en el gráfico 6a que el tiempo de ensamble predomina por encima de los tiempos de procesamiento insumidos por el resto de las etapas. Es importante aclarar que los tiempos de la figura se refieren a la primera iteración para la resolución del problema y que las matrices una vez creadas no alterarán su estructura.

En este análisis no se mencionan los tiempos de creación de las estructuras de matrices ya que esto se considera *costo de start up* y los mismos son bajos en valor absoluto.

Un dato importante relacionado con el tiempo de procesamiento, es que la primera vez que se realiza la resolución del sistema se calculan los factores L

y U correspondientes a la matriz del problema pudiendo ser reutilizadas en las siguientes resoluciones. Es importante aclarar que, tanto en las comparaciones realizadas entre esta implementación y la anterior expuestas en la subsección 4.2 como en los gráficos anteriores de esta misma subsección, se tuvo en cuenta la primera resolución del sistema que incluye la factorización y la resolución propiamente dicha. Por lo que, si quisieramos extender dichas comparaciones para diversas resoluciones sucesivas, el tiempo insumido en esta implementación se reduciría notablemente.

5. Conclusiones y trabajos futuros

En este trabajo se presentó una nueva implementación del método KLE en C++ sobre un framework orientado a la resolución numérica de problemas por el método de elementos finitos (FEM). Se utilizó el framework propuesto por Quinteros et al. (2007) que fue extendido a fin de obtener en un diseño de clases toda la funcionalidad necesaria para poder implementar el KLE por medio de elementos espectrales.

Se amplió la funcionalidad del framework mediante la incorporación de elementos espectrales isoparamétricos con cantidad parametrizable de nodos mediante el desarrollo de la clase `Spectral` definida como subclase de `Element`. Se definieron nuevos métodos en la clase `Element` que sirven para el cálculo del método KLE y fueron implementados en la subclase `Spectral`. Dichos métodos, mantienen la misma metodología que planteaba el framework. Se recrearon las clases `Generator`, `NodesGenerator` y `GaussPointGenerator` para la generación de nodos y de puntos de integración. Asimismo, se incluyó un parser que permite definir condiciones de borde ampliando las capacidades del framework.

Se presentaron los resultados obtenidos mediante esta implementación. Primero se la validó mediante la resolución de un problema con su solución analítica conocida Los resultados expuestos muestran un total acuerdo desde el punto de vista numérico con los previamente publicados por Otero y Ponta (2006), mostrando una convergencia espectral.

Se lograron mejoras de casi un orden de magnitud en el tiempo de procesamiento para los distintos elementos y tamaños de problema con respecto a implementaciones previas. Las razones se deben en parte a la implementación mediante un lenguaje de alto rendimiento como C++ y, por otro lado, al prearmado de las estructuras de las matrices ralas y a la mejora en el diseño de las clases junto con la consecuente reutilización de código, evitando el recálculo en las operaciones. Esto asegura buenas características de escalabilidad del código y lo proyecta como una excelente base para futuras extensiones del método.

También, se realizó un estudio acerca de la influencia sobre el tiempo de ejecución al modificar la cantidad de elementos o variar su orden. Como era de esperar, se observó que a igual cantidad de nodos en una malla, el tiempo de procesamiento es menor para una malla con mayor cantidad de elementos, ya que el orden de los mismos es menor y por ende, la matriz contiene menos valores distintos de cero. Se ve que el tiempo total de procesamiento depende

fuertemente del tiempo de ensamble de las matrices. Sin embargo, por el valor aproximado de las pendientes en las curvas, se ve que cuando el número de elementos de la malla crezca, el tiempo de imposición de condiciones de borde influirá más sobre el tiempo total.

Se implementó el cálculo de los operadores diferenciales de la ecuación de transporte de vorticidad (ec. 5), lo que permitirá extender este trabajo a la resolución de pasos de tiempo sucesivos mediante un algoritmo de integración de ODE (ecuaciones diferenciales ordinarias).

Por otro lado, se planea utilizar esta implementación como base para una futura implementación tridimensional del método. En ese caso, resultará fundamental adaptarla para realizar cálculos en forma paralela mediante el método de las subestructuras sobre MPI. Estas extensiones abrirán un nuevo abanico de posibilidades de aplicación del método KLE a diversos problemas de multifísica.

Apéndice A

Método KLE

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

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

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

$$\hat{\mathbf{V}}^e = \left[\hat{v}_x^1\ \hat{v}_y^1\ \hat{v}_x^2\ \hat{v}_y^2 \cdots\ \hat{v}_x^{{N_{GL}}^2}\ \hat{v}_y^{{N_{GL}}^2} \right]^T,$$

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

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

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

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

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

$$\boldsymbol{\nabla} \times \boldsymbol{v} = \mathbf{r}\, \mathbf{B}^e\, \hat{\mathbf{V}}^e, \qquad\qquad \mathbf{r} = \begin{bmatrix} 0 & -1 & 1 & 0 \end{bmatrix}.$$

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

$$\boldsymbol{\omega} = \mathbf{H}^e_{\boldsymbol{\omega}}\, \hat{\boldsymbol{\omega}}^e, \qquad\qquad \boldsymbol{\nabla} \times \boldsymbol{\omega} = \begin{bmatrix} (\boldsymbol{\nabla} \times \boldsymbol{\omega})_x \\ (\boldsymbol{\nabla} \times \boldsymbol{\omega})_y \end{bmatrix} = \begin{bmatrix} \frac{\partial \boldsymbol{\omega}}{\partial y} \\ -\frac{\partial \boldsymbol{\omega}}{\partial x} \end{bmatrix} = \mathbf{B}^e_{\boldsymbol{\omega}}\, \hat{\boldsymbol{\omega}}^e,$$

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 un arreglo de las funciones de interpolación de la vorticidad y $\mathbf{B}^e_{\boldsymbol{\omega}}$ 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:

$$\begin{aligned}
\hat{\boldsymbol{\omega}}^e &= \left[\, \hat{\boldsymbol{\omega}}^1 \; \hat{\boldsymbol{\omega}}^2 \cdots \hat{\boldsymbol{\omega}}^{{N_{GL}}^2} \,\right]^T, \\
\mathbf{H}^e_{\boldsymbol{\omega}} &= \left[\, h_1 \quad h_2 \quad \cdots \quad h_{{N_{GL}}^2} \,\right], \\
\mathbf{B}^e_{\boldsymbol{\omega}} &= \begin{bmatrix} \frac{\partial h_1}{\partial y} & \frac{\partial h_2}{\partial y} & \cdots & \frac{\partial h_{{N_{GL}}^2}}{\partial y} \\ -\frac{\partial h_1}{\partial x} & -\frac{\partial h_2}{\partial x} & \cdots & -\frac{\partial h_{{N_{GL}}^2}}{\partial x} \end{bmatrix}.
\end{aligned}$$

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

$$\boldsymbol{\delta}\hat{\mathbf{V}}^{e^T} \underbrace{\left(\boldsymbol{K}^{\boldsymbol{e}}_{\boldsymbol{L}} + \boldsymbol{K}^{\boldsymbol{e}}_{\boldsymbol{\mathcal{D}}} + \boldsymbol{K}^{\boldsymbol{e}}_{\boldsymbol{\omega}}\right)}_{\boldsymbol{K^e}} \hat{\mathbf{V}}^e = \boldsymbol{\delta}\hat{\mathbf{V}}^{e^T} \underbrace{\left(\boldsymbol{R}^{\boldsymbol{e}}_{\boldsymbol{L}} + \boldsymbol{R}^{\boldsymbol{e}}_{\boldsymbol{\omega}}\right)}_{\boldsymbol{R^e}} \hat{\boldsymbol{\omega}}^e, \tag{9}$$

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 $\boldsymbol{\delta}\hat{\mathbf{V}}^e$ es el arreglo de valores nodales de las componentes del campo arbitrario $\boldsymbol{\delta v}$.

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,

$$\boldsymbol{K}\,\hat{\mathbf{V}} = \boldsymbol{R}\,\hat{\boldsymbol{\omega}}. \tag{10}$$

Dados el arreglo de valores de la vorticidad en los nodos de la malla $\hat{\boldsymbol{\omega}}$ y las condiciones de contorno correspondientes, se obtiene de la ecuación 10 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 el método de elementos espectrales, las matrices de la ecuación 9 se integran numéricamente.

Las matrices $\boldsymbol{K}$ y $\boldsymbol{R}$ dependen solamente de la malla, con lo cual, se pueden almacenar y reutilizar todas las veces que sea necesario repetir el cálculo o la solución del sistema de la ecuación 10 mientras la malla no se deforme o las conectividades no se modifiquen.

Evaluación del lado derecho del sistema de ecuaciones diferenciales ordinarias

Para la integración temporal, en el problema bidimensional, se reescribe la ecuación de transporte de vorticidad (ec. 3) de forma más conveniente para realizar los cálculos como

$$\frac{\partial \boldsymbol{\omega}}{\partial t} = \boldsymbol{\mathcal{F}}\left(\boldsymbol{\omega}, t\right) = \boldsymbol{\nabla} \times \left(\nu\, \boldsymbol{\nabla}\cdot\boldsymbol{\nabla}\boldsymbol{v} - \boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{v}\right). \tag{11}$$

En el método de elementos finitos suele ser necesario calcular las llamadas variables secundarias, i.e. aquellas obtenidos a partir de derivar las variables del problema. En el caso del KLE se requiere calcular las variables secundarias en los sucesivos pasos temporales. Por esto, se ensamblan las matrices que funcionan como operadores diferenciales, es decir, que al ser aplicados al vector de incógnitas devuelven las variables secundarias. Para obtener dichos operadores se ha utilizado una técnica de derivación, promediación y reproyección. Para ello, en cada punto de integración de cada elemento se define cuánto será el aporte sobre los nodos del dominio global. Esto es similar al procedimiento de cálculo de variables secundarias por *area-weighting* expuesto en Bathe (1996), pero como se ensamblan matrices globales que realizan el proceso de derivación–promediación–reproyección se evita recorrer nuevamente todos los elementos del problema una vez obtenida la solución de las variables principales para obtener las secundarias.

Una vez definidos los aportes, se calculan las matrices elementales que calculan las derivadas de las variables principales o combinaciones de éstas en cada punto de integración. Estas matrices se ensamblan en matrices globales pesadas de acuerdo a los pesos definidos en el nodo global correpondiente. De esta forma, las variables secundarias se obtienen multiplicando estas matrices globales de derivación–promediación–reproyección por los vectores solución que contienen a las variables principales.

Para evaluar el lado derecho de la ecuación 11 se ensamblan 3 arreglos ($\hat{\boldsymbol{G}}_{\boldsymbol{rad}}$, $\hat{\boldsymbol{D}}_{\boldsymbol{iv}}$ y $\hat{\boldsymbol{C}}_{\boldsymbol{url}}$) que realizan las operaciones discretas equivalentes a los operadores diferenciales gradiente, divergencia y rotor respectivamente, a partir del campo de velocidades discreto $\hat{\mathbf{V}}$ obtenido de resolver la KLE según la sección 2.2. Por ejemplo, la versión discreta del rotor de la velocidad $\boldsymbol{\nabla} \times \boldsymbol{v}$ se obtiene de la multiplicación $\hat{\boldsymbol{C}}_{\boldsymbol{url}}\hat{\mathbf{V}}$. De esta forma, el lado derecho de la ecuación 11 se evalúa en forma discreta como

$$\boldsymbol{\mathcal{F}}(\hat{\boldsymbol{\omega}}, t) = \hat{\boldsymbol{C}}_{\boldsymbol{url}} \left(\nu\, \hat{\boldsymbol{D}}_{\boldsymbol{iv}} - \hat{\boldsymbol{V}}_{\boldsymbol{adv}}\right) \hat{\boldsymbol{G}}_{\boldsymbol{rad}}\hat{\mathbf{V}}, \tag{12}$$

donde $\hat{\boldsymbol{V}}_{\boldsymbol{adv}}$ es un reordenamiento del arreglo de componentes nodales de la velocidad $\hat{\boldsymbol{V}}$ para realizar el producto escalar $\boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{v}$ en el término convectivo.

Como los operadores $\hat{\boldsymbol{C}}_{\boldsymbol{url}}$, $\hat{\boldsymbol{G}}_{\boldsymbol{rad}}$ y $\hat{\boldsymbol{D}}_{\boldsymbol{iv}}$ no dependen ni de la vorticidad $\hat{\boldsymbol{\omega}}$ ni del tiempo t, al igual que las matrices $\boldsymbol{K}$ y $\boldsymbol{R}$, se pueden calcular una vez para una malla dada, almacenarlos y usarlos todas las veces que resulte necesario evaluar la ecuación 12 dentro del proceso de integración temporal comandado por un *ODE solver*.

Apéndice B

Características generales de la estructura de clases

Uno de los propósitos más importantes del método de elementos finitos es definir una base de funciones de bajo orden, a partir de subdividir el dominio en subregiones llamadas elementos finitos. Cada elemento está compuesto por nodos que lo definen.

En el framework, el dominio está representado por la clase `Domain` la cual incluye toda la información necesaria para representar la geometría a modelar mediante un conjunto de elementos y nodos almacenados en los atributos `Elements` y `Nodes`.

Es importante notar que el elemento está determinado por un conjunto de nodos conectados. Sin embargo, no existe una relación de pertenencia entre los nodos 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. Esta información está representada por la clase `GaussPoint`.

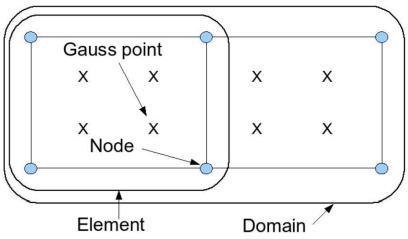


Figura 7: Dominio compuesto por 2 elementosde 4 nodos junto con sus puntos de Gauss (modificado de Quinteros et al., 2007).

En la figura 8 se muestra el diagrama de clases que provee el framework utilizado. Se pueden observar las clases `Domain`, `Element`, `Node`, `GaussPoint` y `Boundary` entre otras.

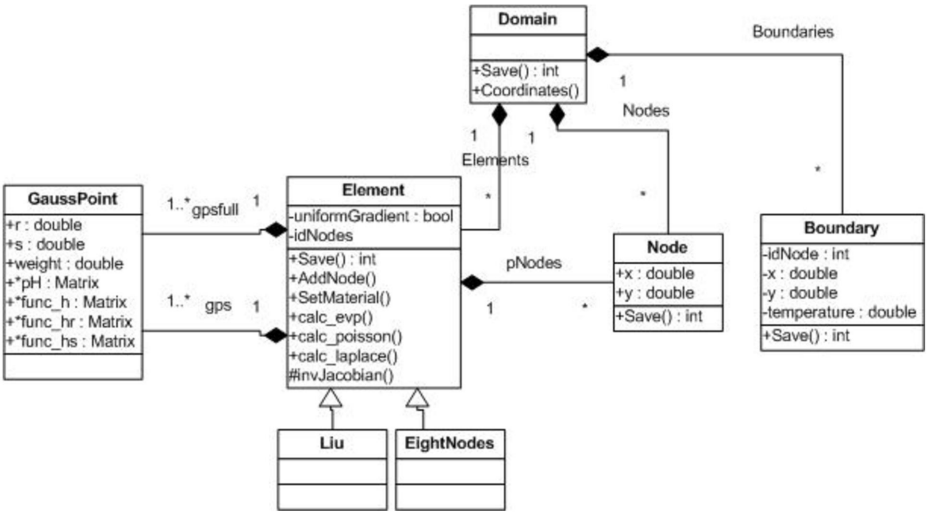


Figura 8: Diagrama de clases.

Operatoria del framework

El framework presenta una estructura de clases proveyendo la funcionalidad de cada una de las entidades presentes en problemas resueltos mediante el método de elementos finitos.

La clase `Domain` posee un método constructor que permite no sólo tomar los datos de un archivo en formato stream al igual que el resto de las clases, sino que también lee un archivo con extensión *.DAT* con la geometría que genera el mallador *GID* diseñado y desarrollado por CIMNE (2009). La clase `ELEM_IMPLEM` es una clase que se define con una instrucción al precompilador la cual especifica el tipo de elemento que se utilizará. Esta clase debe implementar algunos métodos puntuales de la clase `Element`. Como se ve en la sección 3, para este trabajo se implementó una clase que enriquece el framework agregando la funcionalidad propia de elementos espectrales isoparamétricos.

El diseño de la clase `Element` está definido como una base que provee una especificación común para que distintos tipos de elementos la implementen, como así también, incluye funciones ya codificadas y que heredan las subclases. Además, incluye la especificación de funciones que resuelven distintos tipos de ecuaciones integrando numéricamente. En el caso más común, la solución de PDEs 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 de peso y sumadas para obtener la matriz de rigidez elemental. Es por esta razón que se define un método llamado `calc_equation` (donde `equation` es la ecuación que define el problema) que luego de invocar a otro método llamado `eval_equation`, encargado de evaluar las matrices en los puntos de Gauss, realiza las operaciones correspondientes sobre dichas matrices.

La posición de cada uno de los puntos de Gauss 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. 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` también de forma estática. Luego, en la etapa de integración, sólo se deberá calcular la inversa de la matriz Jacobiana en cada punto de Gauss de acuerdo a la ecuación 8.

Bibliotecas utilizadas con las que interactúa el framework

En el framework se busca abstraer las operaciones de álgebra lineal de forma tal que las mismas sean realizadas por medio de bibliotecas específicas de desempeño óptimo independizándolas de las implementación del FEM en particular. Existen una serie de interfaces que aislan la implementación de las bibliotecas mencionadas de forma tal que éstas puedan ser reemplazadas en un futuro. La matriz de rigidez asociada a cada instancia de la clase `Element` es computada mediante operaciones algebraicas de matrices relacionadas con propiedades geométricas del elemento. Para esto, se utilizan bibliotecas que realizan las operaciones algebraicas utilizadas para las operaciones matriciales. 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, utilizando las técnicas de sobrecarga de operadores y polimorfismo. Mediante esta clase, el usuario se abstrae de la implementación de estas operaciones y sólo le debe importar la forma de llamar a los métodos públicos de la clase `Matrix`. En este caso, las operaciones algebraicas son realizadas por la biblioteca 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 suelen ser las matrices globales. Al igual que `Matrix`, esta clase abstrae al usuario de la implementación de las operaciones algebraicas, que en este caso son provistas por la biblioteca `SuperLU` (Demmel et al., 1999). La descomposición LU calculada para la resolución de un sistema puede ser almacenada y utilizada en sucesivas resoluciones, disminuyendo notoriamente el tiempo necesario para resolver el sistema a partir de la segunda llamada como se observa en la sección 4.

Bibliografía

P. R. Amestoy, I. S. Duff, J. Koster, y J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, y S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

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

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

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

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

J. P. Boyd. *Chebyshev and Fourier spectral methods*. Dover, Mineola, New York, USA, 2000.

CIMNE. International center for numerical methods in engineering - gid 9, 2009. URL http://www.gidhome.com/.

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

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

Mumps. Mumps, 2009. URL http://mumps.enseeiht.fr/.

A. D. Otero. *Análisis no-lineal del comportamiento estructural de sistemas avanzados de conversión eoloeléctrica*. Tesis de Doctorado, Universidad de Buenos Aires, 2008.

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

Pardiso. Pardiso, 2009. URL http://www.pardiso-project.org/.

F. L. Ponta. The kinematic Laplacian equation method. *J. Comput. Phys.*, 207: 405–426, 2005.

J. Quinteros. *Numerical modeling of an Andean system and its responseto climatic and rheological variations*. Tesis de Doctorado, Universidad de Buenos Aires, 2008.

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