Universidad Tecnológica Nacional
Facultad Regional Bahía Blanca

Ingeniería Electrónica

Proyecto final de grado

Sobre el comportamiento del Método de Volúmenes Finitos para resolver distribuciones discontinuas de campos magnéticos en discretizaciones espaciales basadas en mallas no-ortogonales y no-estructuradas

Autor
Augusto Riedinger

Director o Tutor
Dr. Martín Saravia

Codirector
Dr. José Ramírez

Bahía Blanca | 08 de Agosto de 2024

Índice general

Índice de figuras

Resúmen

Este informe presenta un estudio detallado del Método de Volúmenes Finitos (FVM) aplicado a la resolución de distribuciones discontinuas de campos magnéticos en discretizaciones espaciales basadas en mallas no-estructuradas y no-ortogonales. Se desarrolla un *solver* específico, denominado `multiRegionMagneticFoam`, implementado en el marco de OpenFOAM, para modelar sistemas electromagnéticos en geometrías complejas con alta precisión y eficiencia computacional. El método propuesto se valida mediante la comparación de los resultados obtenidos con soluciones de otros métodos numéricos, destacando su capacidad para capturar variaciones en el campo magnético a pesar de las interacciones complejas entre los medios.

Palabras claves

Método de Volúmenes Finitos, mallas no-estructuradas, mallas no-ortogonales, campos magnéticos, OpenFOAM, modelado electromagnético, simulación numérica.

Abstract

This report presents a detailed study of the Finite Volume Method (FVM) applied to the resolution of discontinuous magnetic field distributions on unstructured and non-orthogonal mesh-based spatial discretizations. A numerical solver, named `multiRegionMagneticFoam`, is developed within the OpenFOAM framework to model electromagnetic systems in complex geometries with high accuracy and computational efficiency. The proposed method is validated by comparing the obtained results with solutions from other numerical methods, highlighting its ability to capture variations in the magnetic field despite complex interactions between media.

Keywords

Finite Volume Method, unstructured meshes, non-orthogonal meshes, magnetic fields, OpenFOAM, electromagnetic modeling, numerical simulation.

Introducción 1

1.1. Estado del arte y motivación

Los campos magnéticos han cautivado a los investigadores desde su descubrimiento, y últimamente las aplicaciones de ingeniería e industria que hacen uso de ellos han experimentado un aumento significativo. Por lo tanto, el campo de la magnetostática ha presenciado un progreso sustancial en los últimos años. Sin embargo, abordar las geometrías complejas encontradas en aplicaciones del mundo real siempre ha representado un desafío importante debido a la inherente complejidad en las ecuaciones de Maxwell; es decir, las ecuaciones diferenciales parciales fundamentales que rigen la dinámica de los campos magnéticos (Maxwell, 1865). Por esta razón, la aplicación de métodos numéricos para predecir campos magnéticos se ha vuelto crucial para abordar este problema intrincado y buscar soluciones aplicables para uso práctico.

Uno de los elementos clave recurrentes en los diseños de ingeniería moderna es la incorporación de superficies complejas; ya sea para mejorar la eficiencia aerodinámica en aeronaves y optimizar problemas de transferencia de calor (Wright y Schobeiri, 1999), o incluso para mejorar el rendimiento de dispositivos electromagnéticos como antenas y sistemas de comunicación (Senior y Volakis, 1995). Si bien éstas técnicas de optimización han demostrado mejorar la eficiencia de los dispositivos, la simulación numérica de geometrías tan sofisticadas se ha vuelto cada vez más elaborada debido a la no-ortogonalidad inherente que surge al discretizar espalcialmente dichas geometrías.

En el caso del Método de Elementos Finitos (FEM, por sus siglas en inglés), la investigación sobre los efectos numéricos en elementos curvos se ha llevado a cabo desde finales de la década de 1960 por parte de J. H. Argyris y otros (Argyris, 1968; Argyris y Scharpf, 1969). Luego, P. G. Ciarlet y otros continuaron este trabajo en (Ciarlet y Raviart, 1972), donde los autores investigaron las complejidades matemáticas del problema de teoría de aproximación para la convergencia sobre elementos curvos. En los últimos años, la investigación se ha centrado en el análisis de elementos curvos de alto orden, como en (Sevilla y col., 2011), donde los autores compararon la influencia del número de integración en la precisión de varias técnicas derivadas de elementos finitos.

La mencionada complejidad de tratar superficies complejas no solo es una preocupación en el análisis de elementos finitos (FE), sino que también ha sido un tema de suma importancia en el Método de Volumenes Finitos (FVM). En 2009, D. A. Calhoun et al. (Calhoun y Helzel, 2010) investigaron la solución de una ecuación de difusión parabólica de coeficiente constante en superficies paramétricas complejas utilizando un esquema de FV. El trabajo fue continuado por S. R. Sabbagh-Yazdi et al. (Sabbagh-Yazdi y col., 2012) para un FVM no estructurado sin matriz aplicado a ecuaciones de Cauchy en límites curvos con elementos triangulares. Luego, J. C. Chassaing et al. (Chassaing y col., 2013) desarrollaron una evaluación para un enfoque aplicado en el análisis de geometrías complejas, donde se investigaron flujos compresibles en un método basado en FV de orden superior de mínimos cuadrados. Otra aplicación fue investigada por C. Demuth et al. (Demuth y col., 2014); los autores evaluaron el rendimiento del FVM en la conductividad térmica de materiales compuestos aplicados a interfaces curvas. Se realizaron investigaciones más abstractas por A. Boularas et al. (Boularas y col., 2017) y R. Costa et al. (Costa y col., 2018; Costa y col., 2019; Costa y col., 2022; Costa y col., 2023), donde se evaluaron esquemas de FV de orden superior en diversas condiciones.

Además, el tema de los elementos curvos se observa en enfoques más modernos, como el Método del Elemento Virtual (VEM, por sus siglas en inglés). Veiga et al. (Da Veiga y col., 2019) han realizado una extensa investigación sobre el VEM en caras curvas para dos dimensiones, mostrando procedimientos para una tasa óptima de convergencia. El trabajo fue continuado por F. Dassi et al. (Dassi y col., 2021), extendiendo la investigación para lograr tasas de convergencia óptimas con elementos de borde curvilíneos.

Sin embargo, no solo el análisis numérico es más riguroso en geometrías complejas, sino que también el proceso de mallado se vuelve bastante arduo. S. H. Lo ha realizado una extensa investigación sobre este tema (S. Lo, 1988), centrándose en la generación de mallas en dominios cilíndricos, cónicos y esféricos con

elementos triangulares. En 1996, T. S. Lan et al. extendieron su trabajo (Lan y Lo, 1996), desarrollando un esquema automático de generación de mallas triangulares donde generaron elementos en las superficies curvas mediante el uso de una técnica de frente avanzado. En años más recientes, Lo et al. continuaron trabajando en el tema (S. H. Lo y Wang, 2005), optimizando el proceso de generación de mallas en las intersecciones entre superficies curvas.

En lo que respecta al Electromagnetismo Computacional (CEM, por sus siglas en inglés) específicamente, la investigación en el campo es extensa, comenzando con T. K. Sarkar et al. (Sarkar y col., 1981) desarrollando esquemas numéricos para lograr soluciones a ecuaciones matriciales lineales para problemas de dispersión electromagnética y radiación; y continuando con amplios avances a lo largo de los años (Salon y Chari, 1999). Incluso se han realizado algunas investigaciones aplicadas en el tema; una revisión de un modelo electromagnético particular se puede leer en (Niu y col., 2020). Sin embargo, la literatura es escasa cuando se trata de modelado numérico de campos electromagnéticos en curvas.

Vale la pena revisar en detalle los avances en CEM a lo largo de los años, centrándose en el FEM y el FVM. Comenzando con los esquemas FE, algunos modelos numéricos tempranos fueron desarrollados por J. H. Coggon et al. en 1971 (Coggon, 1971), y una introducción general al análisis numérico aplicado a electromagnetismo fue realizada por Sadiku en (Sadiku, 1989), culminando incluso en un libro (Sadiku, 2018). Otra gran literatura en el FEM en electromagnetismo fue realizada por J. Jin et al. (Jin, 2015) y A. C. Polycarpou et al. (Polycarpou, 2022); y una revisión reciente realizada por M. Augustyniak et al. se puede encontrar en (Augustyniak y Usarek, 2016). El FVM tomó más años en ser aplicado al electromagnetismo, comenzando con C.-D. Munz et al. (Munz y col., 2000) desarrollando un esquema FV para las ecuaciones de Maxwell, y con T. S. Chung et al. (Chung y Zou, 2001) extendiendo el trabajo a dominios poliédricos generales en 3D con coeficientes físicos discontinuos. Luego, Y. Liu et al. (Liu y col., 2006) investigaron el FVM espectral en mallas no estructuradas donde desarrollaron un método que es computacionalmente más efectivo que los métodos estructurados convencionales. Enfoques más aplicados también tuvieron lugar; por ejemplo, X. Ferrieres et al. (Ferrieres y col., 2004) resolvieron problemas de compatibilidad electromagnética automotriz utilizando un método híbrido de diferencia finita con FV, y E. Haber et al. (Haber y Ruthotto, 2014) y L. A. C. Mata et al. (Caudillo-Mata y col., 2017) desarrollaron técnicas para reducir el costo computacional de resolver las ecuaciones de Maxwell a bajas frecuencias, con aplicaciones directas a entornos geofísicos.

En 2021, Saravia (Saravia, 2021) presentó un marco para resolver distribuciones discontinuas de campos magnéticos, basado en un enfoque de múltiples regiones utilizando la biblioteca OpenFOAM. El autor desarrolló una formulación escrita en términos del potencial vector magnético para resolver eficientemente esquemas de múltiples cuerpos, incluyendo medios permeables, permanentemente magnetizados y portadores de corriente. Mostraron la precisión del solucionador magnetostático de múltiples regiones comparándolo en experimentos numéricos con varios métodos establecidos. Luego, Riedinger et al. (Riedinger y Saravia, 2023) extendieron el marco para esquemas de región única, donde demostraron que el campo magnético podía ser capturado con precisión con un método continuo utilizando técnicas de mejora de malla, como la malla de capa límite. Sin embargo, los autores en (Saravia, 2021) y (Riedinger y Saravia, 2023) se enfocaron solo en mallas ortogonales y uniformes, y sus métodos no han sido probados para superficies complejas no-ortogonales y no-uniformes.

La literatura sobre el tratamiento numérico de distribuciones discontinuas de campos magnéticos es escasa, y la aplicación del FVM para resolver las ecuaciones diferenciales parciales magnetostáticas es aún más rara. Por lo tanto, la aplicación de dicho método para encontrar la solución a las ecuaciones de Maxwell en superficies curvas nunca ha sido estudiada en detalle. Los numerosos ejemplos del FVM aplicado al análisis de otras áreas de la física en superficies complejas, como la mecánica de fluidos y la transferencia de calor, lo convierten en un candidato atractivo para modelar la magnetostática en mallas no uniformes y no ortogonales. Sin embargo, las descripciones eulerianas del método hacen que el campo magnético sea discontinuo en la interfaz. Esto, junto con la no ortogonalidad de los elementos curvos, dificulta lograr la convergencia sin relajación del campo en los términos del lado derecho de la ecuación magnetostática.

En este trabajo, profundizamos en el tratamiento numérico de distribuciones discontinuas de campos magnéticos en superficies curvas discretizadas con mallas no-ortogonales y no-uniformes. Sostenemos que la aplicación del FVM para resolver dicho problema nos permite encontrar soluciones precisas en las interfaces

discontinuas entre diferentes medios. Por lo tanto, hemos evaluado un esquema FV con un algoritmo de corrección no-ortogonal desarrollado en OpenFOAM.

El documento está estructurado de la siguiente manera: en el Capítulo 2 se describe la formulación del Método de Volumenes Finitos para resolver de forma numérica ecuaciones diferenciales parciales. Luego, en el Capítulo 3 se describe la reformulación de las ecuaciones magnetostáticas para la aplicación del FVM de forma efectiva, así como también se realiza una demostración de los esquemas de discretización y corrección no-ortogonal utilizados para aplicar el método. En el Capítulo 4 se realiza una descripción del marco de OpenFOAM que se utiliza para implementar el Método de Volumenes Finitos. Subsecuentemente, en el Capítulo 5 se explica cómo se programó la formulación del Capítulo 3 en el marco de OpenFOAM. Para finalizar, en el Capítulo 6 se muestran los resultados y la validación de la formulación, y en el Capítulo 7 se presentan las conclusiones del trabajo.

El método de Volumenes Finitos 2

El Método de Volumenes Finitos (FVM, por sus siglas en inglés) es una técnica numérica que transforma las ecuaciones diferenciales parciales que representan leyes de conservación sobre volumenes diferenciales en ecuaciones algebraicas discretas sobre volumenes finitos (o elementos o celdas). De manera similar al método de diferencias finitas o al método de elementos finitos, el primer paso en el proceso de solución es la discretización del dominio geométrico, que, en el FVM, se discretiza en elementos o volumenes finitos no superpuestos. Las ecuaciones diferenciales parciales son entonces discretizadas/transformadas en ecuaciones algebraicas mediante su integración sobre cada elemento discreto. El sistema de ecuaciones algebraicas se resuelve luego para calcular los valores de la variable dependiente para cada uno de los elementos. En el método de volumen finito, algunos de los términos en la ecuación de conservación se convierten en flujos de cara y se evalúan en las caras de los volumenes finitos. Debido a que el flujo que entra en un volumen dado es idéntico al que sale del volumen adyacente, el FVM es estrictamente conservativo. Esta propiedad inherente de conservación del FVM lo convierte en el método preferido en la dinámica de fluidos computacional (CFD, por sus siglas en inglés). Otra atributo importante del FVM es que puede formularse en el espacio físico en mallas poligonales no estructuradas. Finalmente, en el FVM es bastante fácil implementar una variedad de condiciones de contorno de manera no invasiva, ya que las variables desconocidas se evalúan en los centroides de los elementos de volumen, no en sus caras de límite.

Estas características han hecho que el Método de Volumen Finito sea bastante adecuado para la simulación numérica de una variedad de aplicaciones que involucran flujo de fluidos y transferencia de calor y masa, y los avances en el método han estado estrechamente vinculados con los avances en la CFD. Desde un potencial limitado en su inicio confinado a resolver física y geometría simples sobre mallas estructuradas, el FVM ahora es capaz de lidiar con todo tipo de física y aplicaciones complejas. Estas cuestiones hacen al método atractivo para aplicaciones de magnetoestática, donde se encuentas discontinuidades y grandes variaciones del campo magnético en un determinado dominio.

2.1. El proceso de discretización

La solución numérica de una ecuación diferencial parcial consiste en encontrar los valores de la variable dependiente ϕ en puntos especificados a partir de los cuales se puede construir su distribución sobre el dominio de interés. Estos puntos se llaman elementos de la malla o nodos de la malla y resultan de la discretización de la geometría original en un conjunto de elementos discretos no superpuestos, un proceso conocido como mallado. Los nodos o variables resultantes generalmente se colocan en los centroides de las celdas o en los vértices dependiendo del procedimiento de discretización adoptado. En todos los métodos, el enfoque está en reemplazar la solución exacta continua de la ecuación diferencial parcial con valores discretos. La distribución de ϕ se discretiza, y es apropiado referirse a este proceso de convertir la ecuación gobernante en un conjunto de ecuaciones algebraicas para los valores discretos de ϕ como el proceso de discretización y a los métodos específicos empleados para llevar a cabo esta conversión como los métodos de discretización. Los valores discretos de ϕ típicamente se calculan resolviendo un conjunto de ecuaciones algebraicas que relacionan los valores en elementos de la malla vecinos entre sí; estas ecuaciones discretizadas o algebraicas se derivan de la ecuación de conservación que gobierna ϕ. Una vez que se calculan los valores de ϕ, los datos se procesan para extraer cualquier información necesaria.

La intención es introducir una técnica numérica para resolver los procesos físicos de interés y dado que el método debe implementarse en un programa de computadora, el proceso de discretización se explicará en ese sentido. Por ejemplo, se hará referencia a cómo almacenar valores en un código relacionado con elementos interiores, elementos de borde, variables, etc.

2.1.1. Paso 1: Modelado geométrico y físico

La modelización de fenómenos físicos está, en cierto modo, en el corazón de la ciencia. Un fenómeno físico generalmente no puede considerarse como comprendido a menos que pueda ser formulado matemáticamente y esta formulación probada y validada. Para nuestro propósito, se realizan dos niveles de modelado, uno en relación con la geometría del dominio físico y otro en relación con los fenómenos físicos de interés. En ambos niveles, se ignoran o simplifican los detalles que no son relevantes ni de interés. Por ejemplo, un dominio tridimensional podría convertirse en una representación bidimensional, o la simetría puede tenerse en cuenta para disminuir el tamaño del dominio de estudio. En algunos casos, los componentes físicos pueden ser eliminados y reemplazados con representaciones matemáticas apropiadas.

Si se considera a manera de ejemplo un microprocesador que está conectado a un disipador de calor con una base de cobre que actúa como esparcidor de calor, un primer modelo de este sistema simplifica tanto su física como su geometría. El disipador de calor y el procesador son reemplazados por condiciones de contorno que especifican la temperatura estimada del disipador de calor y la temperatura de funcionamiento esperada del procesador, respectivamente. El dominio físico se modela como un dominio computacional bidimensional ya que la variación de temperatura a través del grosor del disipador de calor será mínima. Para una solución en estado estacionario del flujo de calor y la distribución de temperatura en la base de cobre, solo se considera la conducción de calor. El resultado del proceso de modelado es un sistema de ecuaciones diferenciales parciales lineales (o no lineales si k depende de T), que en este caso implica una forma simplificada de la ecuación de energía dada por

$$-\nabla \cdot (k \nabla T) = \dot{q}, \tag{2.1}$$

donde k es la conductividad de la base del esparcidor de calor, y q es la fuente/pérdida de calor por unidad de volumen.

2.1.2. Paso 2: Discretización del dominio

La discretización geométrica del dominio físico resulta en una malla en la cual las ecuaciones de conservación son eventualmente resueltas. Esto requiere la subdivisión del dominio en celdas o elementos discretos no superpuestos que llenan completamente el dominio computacional para obtener un sistema de rejilla o malla. Esto se logra mediante una variedad de técnicas que resultan en una amplia gama de tipos de mallas. Estas mallas se clasifican según varias características: estructura, ortogonalidad, bloques, forma de celda, disposición de variables, etc. En todos los casos, la malla está compuesta por elementos discretos definidos por un conjunto de vértices y limitados por caras. Para que la malla sea una plataforma útil para la discretización de ecuaciones, se necesita información relacionada con la topología de los elementos de la malla, además de cierta información geométrica derivada. Esto incluye relaciones entre elementos, relaciones entre caras y elementos, información geométrica de las superficies, centroide y volumen del elemento, centroide de la cara, área y dirección normal, etc. Esta información generalmente se infiere a partir de los datos básicos de la malla. Para ciertas topologías de malla, los detalles sobre la malla se pueden deducir fácilmente a partir de los índices de los elementos como en las mallas estructuradas, mientras que para otras se debe construir y almacenar en listas para su posterior recuperación, como es el caso de las mallas no estructuradas.

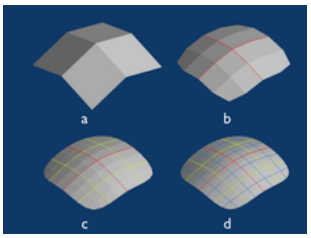

Figura 2.1: Dominio discretizado espacialmente por una malla refinada de forma más fina desde a) hasta d). (contributors, 2023)

Consideremos el dominio simple mostrado en la Figura 2.1. El dominio consiste en un volumen (área para el caso bidimensional) y fronteras. Para continuar el ejemplo anterior de temperatura por un microprocesador dado por la Ecuación 2.1, se puede pensar que las fronteras representan el calentamiento o enfriamiento del microprocesador, un disipador de calor y la base del esparcidor de calor. La frontera de la malla se divide en parches de caras de frontera a los que se les asignan números, es decir, Parche 1, Parche 2 y Parche 3. Estos parches se utilizan para definir las condiciones de contorno físicas para el problema en cuestión. La malla consta de una determinada cantidad de elementos no superpuestos cuya geometría está definida por un número equivalente de puntos (vértices de las celdas). Los elementos también están limitados por un determinado número de caras (líneas en un caso bidimensional), de las cuales también se cuentan las caras interiores. Las ecuaciones algebraicas que resultan de la discretización de las ecuaciones gobernantes, como se explicará en la sección siguiente, se describen para cada elemento en el dominio computacional con la solución expresada como un campo de elementos con valores definidos en el centroide de cada elemento. En este ejemplo, los elementos tienen una forma cuadrada, aunque podrían haberse utilizado otras formas (por ejemplo, elementos triangulares).

La malla se puede describir desde diferentes perspectivas. En el nivel más elemental, es una lista de vértices o puntos que representan ubicaciones en espacios unidimensionales, bidimensionales o tridimensionales. La malla también representa el dominio discretizado subdividido en elementos no superpuestos, que pueden ser de formas poliédricas convexas arbitrarias. Los elementos están completamente limitados por caras que generalmente son compartidas por elementos vecinos, excepto en las fronteras. Los elementos pueden definirse tanto en términos de los puntos que los delimitan como en términos de las caras que los limitan. Las caras de la malla, que se almacenan en una lista, son de dos tipos: (i) caras interiores que son compartidas por (o conectan) dos elementos, y (ii) caras de frontera que coinciden con el límite del dominio; estas caras de frontera tienen solo un elemento contiguo. Mientras que las caras interiores se derivan de información relacionada con la topología del elemento, es esencial proporcionar caras de frontera ya que estas definen la frontera física del dominio. En dos dimensiones, las caras se describen en términos de sus puntos definitorios. En tres dimensiones, los puntos definitorios describen bordes que limitan la cara. La dirección de la normal a una cara interior generalmente se define según la topología de los elementos vecinos. Por otro lado, la dirección de la normal a una cara de frontera siempre apunta hacia afuera del dominio. Además, las caras de frontera se organizan en listas de caras según el parche de frontera al que pertenecen.

Durante la discretización, las ecuaciones diferenciales parciales se integran sobre cada elemento en la malla, lo que resulta en un conjunto de ecuaciones algebraicas en las que cada una vincula el valor de la variable en un elemento con los valores en sus vecinos. Las ecuaciones algebraicas se ensamblan luego en matrices y vectores globales, y los coeficientes de cada ecuación se almacenan en las ubicaciones de fila y columna correspondientes a los diversos índices de elementos. La integración de las ecuaciones sobre cada elemento se denomina ensamblaje local, mientras que la construcción del sistema general de ecuaciones a partir de estas contribuciones se denomina ensamblaje global. Por lo tanto, mientras que la discretización de las ecuaciones se deriva en términos de elementos vecinos, el ensamblaje de las ecuaciones en la matriz global tiene en cuenta los índices reales de los elementos. Este procedimiento se detallará en capítulos posteriores, sin embargo, los ingredientes habilitadores de este procedimiento se introducen brevemente a continuación en su nivel más elemental, que es en forma de información topológica sobre elementos, caras y vértices que se representan en términos de listas de conectividad.

La conectividad de elementos relaciona la matriz de ensamblaje local con la matriz global de manera que las ecuaciones formadas para un elemento sean consistentes con las ecuaciones formadas para los otros elementos en el dominio computacional. Generalmente se establecen conectividades de elemento a elemento, de elemento a cara y de elemento a vértice. Estas conectividades relacionan el elemento con los elementos vecinos, las caras limitantes y los vértices

Generalmente, para elementos arbitrarios, es más eficiente ensamblar términos de flujo recorriendo las caras. En este caso, es esencial que la información sobre los vecinos del elemento de la cara esté fácilmente disponible; esto se define en la conectividad de la cara. Para las caras, se almacena información sobre los elementos que comparten la cara para su uso durante los cálculos. La orientación de la cara es tal que el vector normal a la cara apunta desde un elemento denominado elemento 1 o propietario hacia el segundo elemento

denominado elemento 2 o vecino. Las caras de límite limitan solo un elemento, definido como elemento 1, por lo tanto, el vector normal de las caras de límite siempre está orientado hacia afuera del dominio. La conectividad de vértices es útil para el posprocesamiento y para el cálculo de gradiente y generalmente implica listas de elementos y caras que comparten el vértice.

2.1.3. Paso 3: Discretización de las ecuaciones fundamentales

En el paso III, las ecuaciones diferenciales parciales gobernantes se transforman en un conjunto de ecuaciones algebraicas, una para cada elemento en el dominio computacional. Estas ecuaciones algebraicas luego se ensamblan en una matriz global y vectores que pueden expresarse en la forma

$$\mathbf{A}\,[T] = \mathbf{b}, \tag{2.2}$$

donde la variable desconocida T está definida en cada elemento interior y en el límite del dominio computacional. Los valores de contorno para T generalmente se obtienen a partir de las condiciones de contorno especificadas. Para ello, se debe definir un campo de elemento para T, y generalmente para cada ecuación gobernante.

El campo de elemento consiste en una matriz de valores definidos en el centroide de cada elemento, designado por el campo de elemento interior, que está representado por una matriz de tamaño igual al número total de elementos interiores y de límite.

El paso de discretización de ecuaciones se realiza sobre cada elemento del dominio computacional para obtener una relación algebraica que conecta el valor de una variable en un elemento con los valores de la variable en los elementos vecinos. Esta ecuación algebraica se deriva discretizando la ecuación diferencial, que para el ejemplo considerado es la ecuación de energía escrita en términos de temperatura T (es decir, T es la variable desconocida). Como se muestra a continuación, en el método de volumen finito la discretización de la ecuación se realiza primero integrando la ecuación diferencial sobre un volumen de control o celda para obtener una forma semidiscretizada de la ecuación, y luego aproximando la variación de la variable dependiente entre los elementos de la malla mediante perfiles impuestos para obtener la forma discretizada final. El hecho de que solo unos pocos elementos de la malla participen en una ecuación de discretización dada es una consecuencia de la naturaleza por partes de los perfiles elegidos. El valor de T en un punto de la malla influye así en la distribución de T solo en su vecindario inmediato. A medida que aumenta el número de elementos de la malla, se espera que la solución de las ecuaciones discretizadas se acerque a la solución exacta de la ecuación diferencial correspondiente. Esto se debe a la consideración de que, a medida que los elementos de la malla se acercan entre sí, los cambios en T entre elementos vecinos se vuelven pequeños, y luego los detalles reales de la suposición del perfil se vuelven irrelevantes.

Para una ecuación diferencial dada, las posibles ecuaciones de discretización no son de ninguna manera únicas, aunque se espera que todos los tipos de técnicas de discretización, en el límite de un número muy grande de elementos de malla, den la misma solución. Los diferentes tipos surgen de las diferencias en las suposiciones de perfil y los métodos de derivación. Como ejemplo del paso de discretización de ecuaciones utilizando el método de volumen finito, se busca la forma discretizada de la ecuación de energía sobre el volumen de control C. El proceso comienza integrando la Ec. 2.1 sobre el elemento C, lo que permite recuperar su forma de balance integral como

$$-\iint_{V_C} \nabla \cdot (k \nabla T)\ \mathrm{d}V = \iint_{V_C} \dot{q}\,\mathrm{d}V. \tag{2.3}$$

Entonces, utilizando el teorema de la divergencia, la integral de volumen se transforma en una integral de superficie que da como resultado

$$-\int_{S_C} (k \nabla T) \cdot \mathrm{d}\mathbf{S} = \dot{q}_C\, V_C. \tag{2.4}$$

Esta ecuación es en realidad un balance de calor sobre el elemento C. Básicamente, es la forma integral de la ecuación diferencial original y no involucra ninguna aproximación. Reemplazando la integral de superficie por una suma sobre las caras del volumen de control, la Ec. 2.4 se convierte en

$$-\sum_{f\sim nb(C)} (k\nabla T)_f \cdot \mathbf{S}_f = \dot{q}_C \, V_C, \tag{2.5}$$

donde f representa el punto de integración en el centroide de la cara del límite. Esta transformación es la primera aproximación introducida. Por lo tanto, la integral en la Ec. 2.4 se aproxima numéricamente por los flujos en los centroides de las caras. Esta es una aproximación de segundo orden como se demostrará en una sección posterior.

Expandiendo la suma, la Ec. 2.5 se puede escribir como

$$-(k\nabla T)_{f_1} \cdot \mathbf{S}_{f_1} - (k\nabla T)_{f_2} \cdot \mathbf{S}_{f_2} - (k\nabla T)_{f_3} \cdot \mathbf{S}_{f_3} - (k\nabla T)_{f_4} \cdot \mathbf{S}_{f_4} = \dot{q}_C \, V_C. \tag{2.6}$$

Considerando la cara f_1, el vector de superficie y el gradiente de temperatura en la Ec. 2.6 están dados por

$$\begin{aligned} \mathbf{S}_{f_1} &= \Delta y_{f_1} \, \hat{\mathbf{x}}, \\ \nabla T_{f_1} &= \left(\frac{\partial T}{\partial x}\right)_{f_1} \hat{\mathbf{x}} + \left(\frac{\partial T}{\partial y}\right)_{f_2} \hat{\mathbf{y}}, \end{aligned} \tag{2.7}$$

donde Δy_{f_1} es el área de la cara f_1. Luego, por substitución, el primer término de la Ec. 2.6 resulta

$$\nabla T_{f_1} \cdot \mathbf{S}_{f_1} = \left(\frac{\partial T}{\partial x}\hat{\mathbf{x}} + \frac{\partial T}{\partial y}\hat{\mathbf{y}}\right)_{f_1} \cdot \Delta y_{f_1}\hat{\mathbf{x}} = \left(\frac{\partial T}{\partial x}\right)_{f_1} \Delta y_{f_1}. \tag{2.8}$$

Para proceder, se necesita un perfil approximando la variación de T entre C y F_1. Asumiendo una variación lineal de T, la componente x del gradiente en la cara f_1 puede ser escrito como

$$\left(\frac{\partial T}{\partial x}\right)_{f_1} = \frac{T_{f_1} - T_C}{\delta x_{f_1}}, \tag{2.9}$$

por tanto Ec. 2.8 puede ser aproximada como

$$\nabla T_{f_1} \cdot \mathbf{S}_{f_1} = \frac{T_{f_1} - T_C}{\delta x_{f_1}} \Delta y_{f_1}, \tag{2.10}$$

o, más generalmente como

$$-(k\nabla T)_{f_1} \cdot \mathbf{S}_{f_1} = a_{F_1} \, (T_{F_1} - T_C), \tag{2.11}$$

donde

$$a_{F_1} = -k\frac{\Delta y_{f_1}}{\delta x_{f_1}}. \tag{2.12}$$

Repitiendo el proceso de forma similar para las demás caras, se obtienen los coeficientes

$$\begin{aligned} a_{F_2} &= -k\frac{\Delta y_{f_2}}{\delta x_{f_2}}, \\ a_{F_3} &= -k\frac{\Delta y_{f_3}}{\delta x_{f_3}}, \\ a_{F_4} &= -k\frac{\Delta y_{f_4}}{\delta x_{f_4}}, \end{aligned} \tag{2.13}$$

los cuáles, al ser substituidos en la Ec. 2.6 resulta en

$$\begin{aligned} -\sum_{f\sim nb(C)} (k\nabla T)_f \cdot \mathbf{S}_f &= \sum_{F\sim NB(C)} a_F \left(T_F - T_C\right); \\ &= -\left(a_{F_1} + a_{F_2} + a_{F_3} + a_{F_4}\right) T_C + a_{F_1} T_{F_1} + a_{F_2} T_{F_2} + a_{F_3} T_{F_3} + a_{F_4} T_{F_4}; \\ &= \dot{q}_C \, V_C. \end{aligned} \tag{2.14}$$

O, escrito de forma más compacta

$$a_C T_C + \sum_{F\sim NB(C)} a_F T_F = b_C, \tag{2.15}$$

donde

$$a_C = -\sum_{F\sim NB(C)} a_F \tag{2.16}$$

y

$$b_C = \dot{q}_C \, V_C. \tag{2.17}$$

Ecuaciones similares a la Ec. 2.15 pueden derivarse para todas las celdas en el dominio, dando como resultado un conjunto de ecuaciones algebraicas que pueden resolverse utilizando una variedad de métodos directos o iterativos. Centrándonos en el elemento C, la Ec. 2.15 implica una relación entre T_C y las temperaturas en sus cuatro vecinos, a saber, T_{F_1}, T_{F_2}, T_{F_3}, T_{F_4}.

Ecuaciones similares también se derivan para los elementos de límite y su colección da como resultado el conjunto de ecuaciones , que pueden representarse en forma de matriz como se muestra en la Ec. 2.2, donde $\mathbf{A}$ es la matriz de coeficientes, T el vector solución, y $\mathbf{b}$ es un vector compuesto por términos que no pueden incluirse en $\mathbf{A}$.

Finalmente, debe afirmarse que las propiedades del método de volumen finito en relación con la precisión, robustez y otras características serán revisadas en capítulos posteriores. Esto incluye examinar con más detalle la discretización del término de difusión del volumen finito, que se presentó anteriormente para una malla cartesiana rectangular.

2.1.4. Paso 4: Solución de las ecuaciones discretizadas

La discretización de la ecuación diferencial resulta en un conjunto de ecuaciones algebraicas discretas, que deben resolverse para obtener los valores discretos de T. Los coeficientes de estas ecuaciones pueden ser independientes de T (es decir, lineales) o depender de T (es decir, no lineales). Las técnicas para resolver este sistema algebraico de ecuaciones son independientes del método de discretización, y representan las diferentes trayectorias que pueden seguirse para obtener una solución. Para los conjuntos algebraicos lineales encontrados en este libro, la unicidad de la solución está garantizada. Por lo tanto, si el método de solución adoptado proporciona una solución, será la solución deseada. Todos los métodos de solución (es decir, todos los caminos hacia la solución) que lleguen a una solución darán la misma solución para el mismo conjunto de ecuaciones discretas.

Los métodos de solución para resolver sistemas de ecuaciones algebraicas pueden clasificarse ampliamente como directos o iterativos y se revisan brevemente a continuación.

2.1.4.1. Métodos directos

En un método directo, la solución al sistema de ecuaciones (por ejemplo, Ec. 2.2) se obtiene aplicando un algoritmo relativamente complejo, en comparación con un método iterativo, solo una vez para obtener la

solución para un conjunto dado de coeficientes. Un ejemplo de un método directo es la inversión de matrices, mediante la cual la solución se obtiene como

$$[T] = \mathbf{A}^{-1}\,\mathbf{A}. \tag{2.18}$$

Por lo tanto, se garantiza una solución para $[T]$ si se puede encontrar $\mathbf{A}^{-1}$. Sin embargo, el número de operaciones para la inversión de una matriz $N \times N$ es $\mathcal{O}(N^3)$, lo que resulta computacionalmente costoso. En consecuencia, la inversión casi nunca se emplea en problemas prácticos. Hay métodos más eficientes para sistemas lineales disponibles. Para los métodos de discretización de interés aquí, $\mathbf{A}$ es dispersa, y para mallas estructuradas es de banda. Para ciertos tipos de ecuaciones (por ejemplo, difusión pura), la matriz es simétrica. La manipulación de matrices puede tener en cuenta la estructura especial de $\mathbf{A}$ para diseñar técnicas de solución eficientes. En general, los métodos directos rara vez se usan en la dinámica de fluidos computacional debido a sus grandes requisitos computacionales y de almacenamiento. La mayoría de los problemas hoy en día involucran cientos de miles de celdas, con 5 a 10 incógnitas por celda incluso para problemas simples. Por lo tanto, la matriz $\mathbf{A}$ suele ser muy grande, y la mayoría de los métodos directos se vuelven imprácticos para estos problemas grandes. Además, la matriz $\mathbf{A}$ suele ser no lineal, por lo que el método directo debe estar incrustado dentro de un bucle iterativo para actualizar las no linealidades en $\mathbf{A}$. Por lo tanto, el método directo se aplica una y otra vez, lo que lo hace aún más lento.

2.1.4.2. Métodos iterativos

Los métodos iterativos siguen un procedimiento de adivinar y corregir para refinar gradualmente la solución estimada al resolver repetidamente el sistema discreto de ecuaciones. Consideremos un método iterativo extremadamente simple de Gauss-Seidel. El bucle de solución general para este método se puede escribir de la siguiente manera:

1. Adivinar los valores discretos de T en todos los elementos de la cuadrícula en el dominio.
2. Visitar cada elemento de la cuadrícula por turno. Actualizar T usando

$$T_C = \frac{-\sum_{F\sim NB(C)} a_F\, T_F + b_C}{a_C}, \tag{2.19}$$

   donde los valores vecinos son necesarios para la actualización de T. Se asume que estos son conocidos en los valores actuales. Así, los elementos de la cuadrícula que ya han sido visitados tendrán valores actualizados de T y aquellos que no lo han sido tendrán valores antiguos.
3. Barrer el dominio hasta que todos los elementos de la cuadrícula estén cubiertos. Esto completa una iteración.
4. Comprobar si se cumple un criterio de convergencia apropiado. El requisito, por ejemplo, podría ser que el cambio máximo en los valores de los puntos de la cuadrícula de T sea inferior al 1 %. Si se cumple el criterio, detenerse. De lo contrario, volver al paso 2) y repetir.

El procedimiento de iteración descrito aquí no garantiza converger a una solución para combinaciones arbitrarias de C y NB. La convergencia del proceso está garantizada para problemas lineales si se cumple el criterio de Scarborough. El criterio de Scarborough requiere que un C y un NB satisfagan

$$\frac{-\sum_{F\sim NB(C)} a_F\, T_F + b_C}{a_C} \begin{cases} \leq 1 \ , & \text{para todos los puntos de la malla,} \\ < 1 \ , & \text{para al menos un punto de la malla.} \end{cases} \tag{2.20}$$

Las matrices que satisfacen el criterio de Scarborough tienen dominancia diagonal.

El esquema de Gauss-Seidel se puede implementar con muy poco almacenamiento. Todo lo que se necesita es almacenamiento para los valores discretos de T en los elementos de la cuadrícula. Los coeficientes pueden calcularse sobre la marcha si se desea, ya que no se requiere toda la matriz de coeficientes para el dominio al actualizar el valor de T en cualquier punto de la cuadrícula. Además, la naturaleza iterativa del esquema lo hace particularmente adecuado para problemas no lineales. Si los coeficientes dependen de T, pueden

actualizarse utilizando los valores prevalecientes de T a medida que avanzan las iteraciones. Sin embargo, el esquema de Gauss-Seidel rara vez se usa en la práctica para resolver sistemas complejos. La tasa de convergencia del esquema disminuye a niveles inaceptablemente bajos si el sistema de ecuaciones es grande. Se utilizan más comúnmente métodos multigrid algebraicos para acelerar la tasa de convergencia de los esquemas iterativos y mejorar su rendimiento.

2.2. La malla de Volumenes Finitos

Un ingrediente clave en la implementación del método de Volumenes Finitos es establecer el marco de soporte geométrico para el problema en cuestión. Este proceso comienza con la generación de la malla, que reemplaza el dominio continuo por uno discreto formado por un conjunto contiguo de elementos o celdas no superpuestas delimitadas por un conjunto de caras, y la definición de los límites físicos mediante la marcación de las caras de frontera. Continúa con el cálculo de la información geométrica relevante para los diversos componentes de la malla computacional y se completa capturando la topología de estos componentes, es decir, cómo están relacionados y ubicados uno con respecto al otro. Por lo tanto, el resultado del paso de discretización del dominio no solo son los elementos no superpuestos y otras entidades geométricas relacionadas y la información generada sobre sus propiedades geométricas, sino también la información topológica sobre su disposición y relaciones. Es esta información combinada la que define la malla de volumen finito. El objetivo de este capítulo es clarificar los requisitos topológicos y geométricos de la malla de volumen finito.

La discretización del dominio físico, o generación de la malla, produce una malla computacional en la que posteriormente se resuelven las ecuaciones gubernamentales. Los métodos y técnicas utilizados para la discretización del dominio han cambiado drásticamente en las últimas décadas, y en la actualidad, se han automatizado en su mayoría. Antes de revisar los tipos de elementos comúnmente utilizados en una malla computacional, se describen primero las características y atributos que el sistema de mallas debe poseer para ser empleado en el contexto del método de volumen finito. Estos atributos se presentarán en el contexto del cálculo del gradiente de una variable ϕ tanto en una malla estructurada como en una malla triangular no estructurada.

En general, un dominio geométrico puede ser discretizado utilizando un sistema de rejilla estructurada o no estructurada. En una malla estructurada, los elementos tridimensionales se definen por sus índices locales (i, j, k). Un sistema de rejilla estructurada tiene muchas ventajas de codificación y rendimiento pero sufre de una flexibilidad geométrica limitada. Se puede lograr flexibilidad adicional en la generación de mallas estructuradas empleando múltiples bloques para definir la geometría, con una malla estructurada generada para cada bloque de manera independiente de otros bloques o conjuntamente.

Otra forma de hacer que la generación de mallas sea más flexible es evitar el uso de rejillas estructuradas con su información topológica implícita, y adoptar una malla no estructurada con información topológica explícita basada en tablas de conectividad y numeración de entidades geométricas.

Las rejillas estructuradas siguieron siendo el estándar de la simulación numérica durante mucho tiempo y solo en las últimas dos décadas las rejillas no estructuradas se volvieron más populares. A partir de principios de la década de 1970, el interés en la generación automática de mallas aumentó a medida que el tamaño del problema aumentaba y la generación manual de mallas se volvía demasiado consumidora de tiempo. Los primeros métodos eran semi-automáticos, con un operador colocando manualmente puntos en el dominio computacional y luego, en un segundo paso, utilizando una computadora para generar la malla. Hoy en día, todo el proceso está completamente automatizado, con puntos y elementos generados automáticamente.

La mayoría de los códigos modernos tienen la capacidad de utilizar rejillas no estructuradas además de una variedad de rejillas híbridas multibloque. OpenFOAM utiliza rejillas no estructuradas pero también puede usar rejillas multibloque conformes y no conformes. El método de volumen finito se presentará aquí en el contexto de un sistema de rejilla no estructurada. Sin embargo, a medida que se definen los requisitos de la

malla de volumen finito no estructurada, se compararán sus características con las de un sistema de rejilla estructurada.

La discusión sobre los requisitos de la malla de volumen finito se contextualizará en términos de un problema simple, a saber, el cálculo del gradiente de un campo de elementos. El gradiente primero se calculará en una rejilla estructurada y luego en una rejilla no estructurada; las diferencias ayudarán a aclarar varios problemas.

2.2.1. Soporte de malla para el cálculo del gradiente

Se pueden utilizar muchas técnicas para calcular el gradiente de un campo de elementos. El método adoptado en esta sección se basa en el teorema de Green-Gauss. Es relativamente sencillo y se puede utilizar para una variedad de topologías y mallas (estructuradas/no estructuradas, ortogonales/no ortogonales, etc.). El punto de partida es definir el gradiente promedio sobre un elemento de volumen finito de centroide C y volumen V_C como

$$\overline{\nabla \phi_C} = \frac{1}{V_C} \int_{V_C} \nabla \phi \, \mathrm{d}V. \tag{2.21}$$

Luego, utilizando el teorema de la divergencia, la integral de volumen se transforma en una integral de superficie que produce

$$\overline{\nabla \phi_C} = \frac{1}{V_C} \int_{\partial V_C} \phi \, \mathrm{d}\mathbf{S}, \tag{2.22}$$

donde dS es el vector de superficie orientado hacia afuera. En presencia de caras discretas, la ecuación superior se puede escribir como

$$\overline{\nabla \phi_C} \, V_C = \sum_{\partial V_C} \phi \, \mathrm{d}\mathbf{S}. \tag{2.23}$$

A continuación, la integral sobre una cara de celda se aproxima utilizando la regla de integración de punto medio para que sea igual al valor interpolado del campo en el centroide de la cara multiplicado por el área de la cara, lo que resulta en

$$\overline{\nabla \phi_C} = \frac{1}{V_C} \sum_{f=nb(C)} \overline{\phi_f} \, \mathbf{S}_f. \tag{2.24}$$

Al revisar la ecuación superior, queda claro que para calcular el promedio del gradiente sobre el elemento de control C, se requiere información sobre el área de la cara y la dirección ($\mathbf{S}_f$), así como información sobre los elementos vecinos y los valores ϕ en los centroides de los elementos (ϕ_C, ϕ_{F_k}). Esta información es necesaria para calcular el valor de ϕ en la interfaz (ϕ_f), que tendrá que interpolarse de alguna manera.

Se asume un perfil para la variación de la variable dependiente ϕ entre los valores nodales, lo que básicamente introduce una aproximación en la evaluación del gradiente. En todos los casos, el valor de ϕ_f deberá calcularse en cada centroide de cara. Suponiendo un perfil lineal para la variación de ϕ entre los elementos C y F que atraviesan la interfaz f, un valor aproximado para ϕ_f, denotado por $\overline{\phi_f}$, se puede calcular como

$$\overline{\phi_f} = g_F \, \phi_F + g_C \, \phi_C. \tag{2.25}$$

Una forma de calcular los valores de peso g_C y g_F está dada por

$$g_F = \frac{V_C}{V_C + V_F} \qquad g_C = \frac{V_F}{V_C + V_F} = 1 - g_F. \tag{2.26}$$

2.3. Mallas estructuradas

Para una malla estructurada regular, cada celda interior en el dominio está conectada al mismo número de celdas vecinas. Estas celdas vecinas se pueden identificar utilizando los índices i, j y k en las direcciones de

coordenadas x, y, y z, respectivamente, y se pueden acceder directamente incrementando o decrementando los índices respectivos. Esto permite un menor uso de memoria ya que la información topológica está incrustada en la estructura de la malla a través del sistema de indexación. Esto también conduce a una mayor eficiencia en la codificación, utilización de caché y vectorización. Las rejillas estructuradas fueron ampliamente utilizadas en el desarrollo de los métodos de Volumen Finito y Diferencias Finitas.

En una rejilla estructurada, se puede asociar a cada celda computacional un conjunto ordenado de índices (i, j, k), donde cada índice varía en un rango fijo, independientemente de los valores de los otros índices, y donde las celdas vecinas tienen índices asociados que difieren en uno más o uno menos. Por lo tanto, si hay N_i, N_j y N_k elementos en las direcciones de índice i, j y k, respectivamente, entonces el número total de elementos en el dominio es $N_i \times N_j \times N_k$. En espacios tridimensionales, los elementos son hexágonos con 6 caras y 8 vértices, y cada elemento interior tiene 6 vecinos. En dos dimensiones, los elementos son cuadriláteros con 4 caras y 4 vértices, y cada elemento interior tiene 4 vecinos.

2.3.1. Información topológica

Los índices globales generalmente se utilizan para construir el sistema completo de ecuaciones sobre el dominio computacional, mientras que los índices locales se emplean para definir el stencil local de un elemento, información que es útil durante el proceso de discretización. En los sistemas de rejilla estructurada, los índices locales se utilizan indistintamente como índices globales porque se pueden traducir fácilmente a índices globales y viceversa. En espacios bidimensionales, la relación entre los índices locales (i, j) y el índice global (n) se da por

$$n = i + (j-1)N_i \;\forall\; 1 \leq i \leq N_i\,,\; 1 \leq j \leq N_j, \tag{2.27}$$

y los índices globales correspondientes para los vecinos de la celda (i, j) se obtienen como

$$\begin{aligned} &(i,j) \to n \\ (i+1,j) &\to n+1 \quad (i-1,j) \to n-1 \\ (i,j+1) &\to n+N_i \;\; (i,j-1) \to n-N_i. \end{aligned} \tag{2.28}$$

Esto simplifica en gran medida el acceso a los coeficientes y la solución del sistema de ecuaciones, ya que los coeficientes construidos sobre el stencil local de una celda se utilizan básicamente en el sistema general de ecuaciones sin necesidad de un paso de traducción entre índices locales y globales, ya que el mapeo entre ellos está disponible fácilmente. Esto también se aplica a los campos geométricos y a los diversos campos de conservación que se están resolviendo.

2.3.2. Información geométrica

Acceder a la información geométrica local alrededor de un elemento es bastante simple. Si para el elemento (i, j) las caras almacenadas circundantes son $\mathbf{S}_1\,(i,j)$, $\mathbf{S}_2\,(i,j)$, $\mathbf{S}_1\,(i+1,j)$ y $\mathbf{S}_2\,(i,j+1)$. Dado que para cualquier elemento las caras deben apuntar hacia afuera, entonces

$$\begin{aligned} \mathbf{S}_{i-1/2,j} &= -\mathbf{S}_1\,(i,j)\,; \\ \mathbf{S}_{i,j-1/2} &= -\mathbf{S}_2\,(i,j)\,, \end{aligned} \tag{2.29}$$

mientras que para otras caras, se aplica lo siguiente:

$$\begin{aligned} \mathbf{S}_{i+1/2,j} &= -\mathbf{S}_1\,(i+1,j)\,; \\ \mathbf{S}_{i,j+1/2} &= -\mathbf{S}_2\,(i,j+1)\,. \end{aligned} \tag{2.30}$$

El campo de elementos en una malla estructurada bidimensional o tridimensional también se define como una matriz de tamaño $[N_x]\;[N_y]$ o $[N_x]\;[N_y]\;[N_z]$, respectivamente. Por lo tanto, acceder al valor del elemento y sus vecinos es igualmente simple. El campo de elementos de un sistema multidimensional también puede

definirse utilizando una matriz unidimensional, que en dos dimensiones tendrá un tamaño de $[N_i \times N_j]$ y en tres dimensiones un tamaño de $[N_x \times N_y \times N_z]$. Representar un campo multidimensional mediante una matriz unidimensional, con valores almacenados utilizando el sistema de indexación global, ahorra mucha memoria de computadora cuando se adopta un sistema de multirrejilla.

2.3.3. Acceso al campo de un elemento

Acceder al campo en una malla estructurada es tan simple como usar los índices del elemento. Por lo tanto, en un espacio bidimensional, $\phi\,(i,j)$ o $\phi_{i,j}$ es el valor del campo ϕ en el elemento (i,j). Los valores de ϕ en las celdas vecinas a (i,j) son, respectivamente, $\phi_{i+1,j}$, $\phi_{i-1,j}$, $\phi_{i,j+1}$ y $\phi_{i,j-1}$.

Como se mencionó anteriormente, el cálculo del gradiente usando la Ecuación 2.24 requiere calcular el valor de ϕ en cada cara del volumen finito (para nuestros pseudo elementos, las caras delanteras y traseras tienen el mismo valor y por lo tanto no se incluirán en el cálculo). Por lo tanto, además del valor de ϕ en el punto (i,j), también se necesitan los valores de ϕ en los puntos vecinos $(i+1,j)$, $(i-1,j)$, $(i,j+1)$ y $(i,j-1)$. En una malla estructurada, esta información está disponible fácilmente y el valor en la cara se calcula mediante interpolación simple entre los valores de ϕ en los centroides de los dos volumenes que comparten la cara. Usando la Ecuación 2.25, el valor interpolado en la cara $(i+1/2,j)$ en términos de índices locales se puede escribir como

$$\overline{\phi}_{i+1/2,j} = g_{i+1/2,j}\;\phi_{i+1,j} + \left(1 - g_{i+1/2,j}\right)\;\phi_{i,j}. \tag{2.31}$$

El gradiente en el elemento (i,j) se puede calcular usando indexación local como

$$\overline{\nabla\phi_{i,j}} = \frac{1}{V_{ij}}\left(\overline{\phi}_{i+1/2,j}\,\mathbf{S}_{i+1/2,j} + \overline{\phi}_{i-1/2,j}\,\mathbf{S}_{i-1/2,j} + \overline{\phi}_{i,j+1/2}\,\mathbf{S}_{i,j+1/2} + \overline{\phi}_{i,j-1/2}\,\mathbf{S}_{i,j-1/2}\right), \tag{2.32}$$

mientras que usando un sistema de indexación global, se convierte en

$$\overline{\nabla\phi_n} = \frac{1}{V_n}\left(\overline{\phi}_{n+1/2}\,\mathbf{S}_{n+1/2} + \overline{\phi}_{n-1/2}\,\mathbf{S}_{n-1/2} + \overline{\phi}_{n+N^i/2}\,\mathbf{S}_{n+N^i/2} + \overline{\phi}_{n-N^i/2}\,\mathbf{S}_{n-N^i/2}\right). \tag{2.33}$$

Cabe mencionar que $\mathbf{S}$ es el vector exterior normal a la superficie en la cara del volumen de control. Excepto en los límites del dominio, las caras del volumen de control son compartidas por dos elementos. Por lo tanto, la dirección exterior para un elemento representará la dirección interior para el otro elemento. Por lo tanto, para evitar duplicar vectores superficiales en interfaces, solo se calcula y almacena un vector en una interfaz; aquel en la dirección de aumento de i o j. Las características incrustadas en un sistema de cuadrícula estructurada permiten que se elija la dirección correcta sin necesidad de almacenar información adicional. Para cualquier elemento (i,j), la superficie con un índice mayor que i o j es positiva mientras que la superficie con un índice menor que i o j se multiplica por un signo negativo. Esto explica los signos negativos en las ecuaciones superiores.

2.4. Mallas no-estructuradas

Mallas no estructuradas ofrecen más flexibilidad en el mallado de un dominio tanto en términos de los tipos de elementos que se pueden usar como en términos de dónde pueden concentrarse los elementos. Sin embargo, esta flexibilidad conlleva un costo de complejidad adicional.

En un sistema de malla no estructurada, los elementos no están numerados secuencialmente, al igual que las caras, nodos y otras cantidades geométricas. Esto significa que no hay una manera directa de vincular varias entidades basadas solo en sus índices. Por lo tanto, la conectividad local debe definirse explícitamente comenzando por determinar las cantidades geométricas para un elemento en particular. Por lo tanto, se necesita información topológica detallada sobre elementos, caras y nodos vecinos para complementar la indexación global de la malla.

2.4.1. Información topológica y conectividad

La información topológica se desarrolla mediante la construcción explícita de los índices locales y globales que definen las conectividades de los componentes geométricos (elemento a elemento, elemento a caras, caras a elementos, elemento a nodos, etc.). Con este fin, la estructura de datos de elementos, caras y nodos ahora incluye información sobre conexiones vecinas en términos de índices locales y globales.

El algoritmo de cálculo del gradiente ahora puede expresarse en términos de los índices de discretización para un par de elementos triangulares como

$$\overline{\nabla\phi_C} = \frac{1}{V_C}\left(\overline{\phi}_{f_1}\,\mathbf{S}_{f_1} + \overline{\phi}_{f_2}\,\mathbf{S}_{f_2} + \overline{\phi}_{f_3}\,\mathbf{S}_{f_3} + \overline{\phi}_{f_4}\,\mathbf{S}_{f_4} + \overline{\phi}_{f_5}\,\mathbf{S}_{f_5} + \overline{\phi}_{f_6}\,\mathbf{S}_{f_6}\right), \tag{2.34}$$

o en términos de los números locales de las caras del elemento

$$\overline{\nabla\phi_0} = \frac{1}{V_C}\left(\overline{\phi}_1\,\mathbf{S}_1 + \overline{\phi}_2\,\mathbf{S}_2 + \overline{\phi}_3\,\mathbf{S}_3 + \overline{\phi}_4\,\mathbf{S}_4 + \overline{\phi}_5\,\mathbf{S}_5 + \overline{\phi}_6\,\mathbf{S}_6\right), \tag{2.35}$$

donde los números indican el índice local de la cara. La relación del gradiente también se puede escribir en términos de índices globales que representan los valores almacenados en las caras del elemento.

Mientras que el vector de superficie local S siempre se asume que está en la dirección hacia afuera, esto no es necesariamente cierto, ya que solo se almacena un vector normal en cualquier cara. Estos vectores de superficie almacenados específicos realmente apuntan hacia adentro del elemento, por lo tanto, el signo es negativo. A diferencia de los sistemas de malla estructurada donde la dirección correcta se puede obtener fácilmente, en una malla no estructurada la dirección de la normal a la superficie debe almacenarse de alguna manera. Esto se detallará en la siguiente sección. Para tener en cuenta la dirección del vector, se utiliza una función de signo y la ecuación para el gradiente se modifica como

$$\overline{\nabla\phi_k} = \frac{1}{V_k}\left(-\sum_{n\leftarrow\langle f\sim nb(k)\rangle<k}\overline{\phi}_n\,\mathbf{S}_n + \sum_{n\leftarrow\langle f\sim nb(k)\rangle>k}\overline{\phi}_n\,\mathbf{S}_n\right). \tag{2.36}$$

Para las caras, la información sobre los elementos contiguos es lo que define la topología de la cara. Además, la orientación de la cara se puede definir de manera estándar indexando los elementos en un orden específico. En este caso, el vector normal en la interfaz entre dos elementos está orientado desde el elemento 1 al elemento 2, que también se denotan en OpenFOAM como los elementos propietario y vecino, respectivamente.

El cálculo del gradiente se puede realizar para cada elemento. Sin embargo, dado que el flujo $\overline{\phi}_f\,\mathbf{S}_f$ en cada interfaz es el mismo para los elementos contiguos, con la diferencia siendo su signo, el cálculo del gradiente podría proceder de manera más eficiente calculando el gradiente en todo el dominio (por ejemplo, todo el dominio) en lugar de elemento por elemento. Esto se hace recorriendo todas las caras en el dominio computacional y actualizando directamente el valor del gradiente para los elementos que rodean la interfaz mediante el incremento y decremento del flujo calculado a partir del gradiente del elemento 1 y del elemento 2, respectivamente.

Por lo tanto, el algoritmo en una malla no estructurada para calcular un campo de gradiente se convierte en

```
1. Declarar un arreglo de gradientes e inicializarlo en cero.
2. Iterar sobre las caras interiores:
    > Calcular el flujo flux_f = phi_f * S_f.
    > Sumar flux_f al gradiente del elemento propietario y -flujo_f al gradiente del
  ↪ elemento vecino.
3. Iterar sobre las caras de frontera:
    > Calcular el flujo flux_f = phi_f * S_f
    > Sumar flux_f al gradiente del elemento propietario.
4. Iterar sobre los elementos y dividir el gradiente por el volumen del elemento.
```

Esto básicamente proporciona el gradiente en cada elemento en el dominio computacional según la ecuación 2.36. El mismo algoritmo podría usarse con una malla estructurada para reducir el costo computacional.

Formulación del fenómeno físico 3

3.1. Ecuaciones magnetostáticas fundamentales

3.1.1. Leyes de conservación

En sistemas donde las corrientes no cambian con el tiempo, los campos magnéticos son estáticos. A partir de las ecuaciones de Maxwell (Maxwell, 1865), si asumimos que las cargas se mueven como una corriente constante $\mathbf{J}_f$, las ecuaciones se pueden separar en dos conjuntos: uno para el campo eléctrico y otro para el campo magnético. Estos campos son independientes entre sí; por lo tanto, podemos enfocarnos únicamente en el campo magnético y las ecuaciones se reducen a

$$\nabla \cdot \mathbf{B} = 0, \tag{3.1}$$

y

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{J}_f, \tag{3.2}$$

donde (3.1) es conocida como la ley de Gauss para el magnetismo mientras que (3.2) es conocida como la ley de Ampère. En las ecuaciones anteriores, $\mathbf{B}$ representa la densidad del flujo magnético vectorial y $\nabla \cdot \mathbf{v}$ denota la divergencia de un vector dado $\mathbf{v}$; luego, $\mathbf{J}_f$ representa la corriente libre mientras que $\nabla \times \mathbf{v}$ denota el rotacional de un vector dado $\mathbf{v}$. Además, la corriente libre $\mathbf{J}_f$ obedece su propia ley de conservación

$$\nabla \cdot \mathbf{J}_f = 0, \tag{3.3}$$

y su relación con los campos magnéticos se da a través de la ley de Ampère, como se muestra en la Ecuación (3.2). Estas últimas tres ecuaciones son conocidas como las leyes de conservación para la magnetostática (Griffiths, 2021; Jackson, 1999; Sommerfeld, 2013). Nuestro objetivo es, basándonos en las leyes fundamentales, desarrollar una ecuación de equilibrio para tratar múltiples medios. Por lo tanto, en la siguiente sección proponemos una variante de las ecuaciones constitutivas en términos del vector potencial magnético, que denotaremos como $\mathbf{A}$.

3.1.2. Formulación de la ley de Ampère

Los tipos principales de medios que pretendemos abordar son dominios permeables y magnetizados permanentemente. Específicamente, nuestro enfoque en esta sección es derivar ecuaciones de equilibrio para ambos medios por separado, seguidas de la proposición de una ecuación unificada para facilitar el cálculo de la densidad del flujo magnético en ambos medios.

3.1.2.1. Ley de Ampère para medios permeables

Para lograr este objetivo, comenzaremos nuestro análisis examinando la distribución del campo magnético dentro de un dominio permeable, denotado como $\mathbf{B}_p$. Como se describe en (Griffiths, 2021), esta distribución está gobernada por una variación de la ley de Ampère dada por

$$\nabla \times \mathbf{B}_p = \mu_0 \left(\mathbf{J}_f + \mathbf{J}_i\right), \tag{3.4}$$

donde $\mu_0 \approx 4,\pi \times 10^{-7}$ $[\mathrm{H/m}]$ es la permeabilidad magnética del espacio libre, y $\mathbf{J}_i$ denota el vector de corriente inducida.

Luego, si interpretamos $\mathbf{J}_i$ como el resultado de un vector de magnetización inducida $\mathbf{M}_i$, entonces la siguiente relación se cumple:

$$\mathbf{J}_i = \nabla \times \mathbf{M}_i. \tag{3.5}$$

Dada la expresión de $\mathbf{J}_i$ en la Ecuación (3.5), podemos reescribir la Ecuación (3.4) en la forma

$$\nabla \times \mathbf{B}_p = \mu_0 \left(\mathbf{J}_f + \nabla \times \mathbf{M}_i\right), \tag{3.6}$$

que, reorganizando términos, da como resultado

$$\nabla \times \left(\frac{1}{\mu_0}\mathbf{B}_p - \mathbf{M}_i\right) = \mathbf{J}_f. \tag{3.7}$$

Es crucial determinar una expresión para la corriente libre $\mathbf{J}_f$, ya que representa una cantidad física real que puede controlarse fácilmente en la mayoría de los experimentos. Para simplificar esta expresión, introducimos el campo auxiliar $\mathbf{H}$, definido como

$$\mathbf{H} \equiv \frac{1}{\mu_0}\mathbf{B}_p - \mathbf{M}_i + \nabla\phi, \tag{3.8}$$

donde ϕ es un campo escalar arbitrario, y el operador $\nabla\mathbf{v}$ denota el gradiente de un campo vectorial dado $\mathbf{v}$. Es destacable que ϕ es arbitrario debido a que el rotacional de un gradiente es siempre cero. Por lo tanto, podemos seleccionar un campo de tal manera que $\nabla\phi = 0$, y la Ecuación (3.8) se simplifica a

$$\mathbf{H} = \frac{1}{\mu_0}\mathbf{B}_p - \mathbf{M}_i. \tag{3.9}$$

El término del lado derecho (RHS) en la Ecuación (3.9) es igual al término entre paréntesis en la Ecuación (3.7), lo que nos permite reescribirlo en una forma simplificada,

$$\nabla \times \mathbf{H} = \mathbf{J}_f. \tag{3.10}$$

Luego, recordamos la ley constitutiva para campos magnéticos en medios permeables (Sommerfeld, 2013),

$$\mathbf{B}_p = \mu\,\mathbf{H}, \tag{3.11}$$

siendo μ la permeabilidad absoluta del medio específico. Podemos expandir la ecuación anterior utilizando el resultado de la Ecuación (3.9), lo que nos da

$$\mathbf{B}_p = \mu\left(\frac{1}{\mu_0}\mathbf{B}_p - \mathbf{M}_i\right). \tag{3.12}$$

A partir de esto, podemos derivar una expresión para la magnetización $\mathbf{M}_i$, como función de la densidad del flujo magnético $\mathbf{B}_p$,

$$\mathbf{M}_i = \left(\frac{\mu - \mu_0}{\mu, \mu_0}\right)\mathbf{B}_p. \tag{3.13}$$

Definiendo la cantidad $\chi = f(\mu)$ como la susceptibilidad magnética normalizada en la forma

$$\chi = \frac{\mu - \mu_0}{\mu\,\mu_0} = \frac{\mu_r - 1}{\mu_r\,\mu_0}, \tag{3.14}$$

donde $\mu_r = {}^{\mu}/_{\mu_0}$ es la permeabilidad relativa, podemos simplificar la Ecuación (3.13) a

$$\mathbf{M}_i = \chi\,\mathbf{B}_p. \tag{3.15}$$

Sustituyendo las Ecuaciones (3.10) y (3.15) en la Ecuación (3.6), derivamos una expresión expandida para la ley de Ampère en medios permeables en la forma

$$\nabla \times \mathbf{B}_p = \mu_0\left[\nabla \times \mathbf{H} + \nabla \times (\chi\,\mathbf{B}_p)\right]. \tag{3.16}$$

Consecuentemente, redefinimos el vector de corriente inducida $\mathbf{J}_i$ como

$$\mathbf{J}_i = \nabla \times (\chi\, \mathbf{B}_p)\,. \tag{3.17}$$

3.1.2.2. Ley de Ampère para medios magnetizados permanentemente

Para determinar la densidad del flujo magnético en un medio magnetizado permanentemente $\mathbf{B}_m$, debemos resolver la siguiente variante de la ley de Ampère:

$$\nabla \times \mathbf{B}_m = \mu_0\, \mathbf{J}_b. \tag{3.18}$$

Aquí, un medio magnetizado resulta en una corriente vinculada $\mathbf{J}_b$ (Griffiths, 2021), definida como

$$\mathbf{J}_b = \nabla \times \mathbf{M}, \tag{3.19}$$

donde $\mathbf{M}$ denota el vector de magnetización. En contraste con la corriente libre $\mathbf{J}_f$, esta corriente vinculada es el resultado de la modelización continua de la magnetización, y por lo tanto no es un parámetro controlable. Sin embargo, podemos expandir la Ecuación (3.18) incorporando la Ecuación (3.19); esto nos da

$$\nabla \times \mathbf{B}_m = \mu_0\, \nabla \times \mathbf{M}, \tag{3.20}$$

que representa la forma final de la ley de Ampère para medios magnetizados.

3.1.2.3. Forma combinada de la ley de Ampère

Si consideramos un medio compuesto por cuerpos permeables y magnetizados, la densidad resultante del flujo magnético $\mathbf{B}$ comprenderá contribuciones tanto de $\mathbf{B}_p$ como de $\mathbf{B}_m$. En otras palabras, la ley de Ampère produce

$$\nabla \times \mathbf{B} = \nabla \times \mathbf{B}_p + \nabla \times \mathbf{B}_m = \mu_0\, (\mathbf{J}_f + \mathbf{J}_i + \mathbf{J}_b)\,. \tag{3.21}$$

Incorporando las expresiones para las densidades de corriente de las Ecuaciones (3.17) y (3.19), podemos reescribir la ecuación anterior como

$$\nabla \times \mathbf{B} = \mu_0\, [\mathbf{J}_f + \nabla \times (\chi\, \mathbf{B}_p) + \nabla \times \mathbf{M}]\,. \tag{3.22}$$

Si asumimos que la permeabilidad relativa del medio magnetizado satisface

$$\mathbf{B}_p = \mathbf{B}, \tag{3.23}$$

entonces, podemos derivar la ley de Ampère generalizada para todo el dominio

$$\nabla \times \mathbf{B} = \mu_0\, [\mathbf{J}_f + \nabla \times (\chi\, \mathbf{B}) + \nabla \times \mathbf{M}]\,. \tag{3.24}$$

Es destacable que la relación en la Ecuación (3.23) es verdadera si, por ejemplo, la permeabilidad relativa del cuerpo magnetizado es igual a 1 y la magnetización del medio permeable es cero. Esta afirmación sugiere que no es posible que los materiales permeables y los medios magnetizados ocupen el mismo espacio simultáneamente; si bien esta suposición no tiene implicaciones prácticas, simplifica significativamente el proceso de encontrar una solución, razón por la cual se está considerando en este trabajo.

3.2. Formulación en función del vector potencial magnético

3.2.1. Ecuación de balance en función del vector potencial magnético

Dado que OpenFOAM resolverá la Ecuación (3.24) utilizando el rotacional del campo magnético, necesitamos modificarla para cumplir con este requisito. Dado que el campo magnético es solenoidal, puede expresarse como el rotacional de un campo vectorial alternativo, el potencial vectorial magnético $\mathbf{A}$, tal que

$$\mathbf{B} = \nabla \times \mathbf{A}, \tag{3.25}$$

lo cual satisface su propia ley de conservación en la forma

$$\nabla \cdot \mathbf{B} = \nabla \cdot (\nabla \times \mathbf{A}) = 0, \tag{3.26}$$

ya que el gradiente del rotacional de un campo vectorial siempre da cero, terminamos con

$$\nabla \cdot \mathbf{A} = 0. \tag{3.27}$$

Con la definición para el potencial vectorial magnético en la Ecuación (3.25), podemos reescribir la Ecuación (3.24) como

$$\nabla \times (\nabla \times \mathbf{A}) = \mu_0 \left[\mathbf{J}_f + \nabla \times (\chi\, \mathbf{B}) + \nabla \times \mathbf{M}\right]. \tag{3.28}$$

Luego, utilizando la identidad vectorial (A.1), obtenemos

$$\nabla \left(\nabla \cdot \mathbf{A}\right) - \nabla^2 \mathbf{A} = \mu_0 \ \left[\mathbf{J}_f + \nabla \times (\chi\, \mathbf{B}) + \nabla \times \mathbf{M}\right], \tag{3.29}$$

donde $\nabla^2 \mathbf{v}$ denota el laplaciano de un campo vectorial dado $\mathbf{v}$. Sin embargo, debido a la conservación del potencial vectorial magnético dado en la Ecuación (3.27), la Ecuación (3.29) se simplifica a

$$\nabla^2 \mathbf{A} = -\mu_0 \ \left[\mathbf{J}_f + \nabla \times (\chi\, \mathbf{B}) + \nabla \times \mathbf{M}\right]. \tag{3.30}$$

Ahora, expandiendo el término $\nabla \times (\chi\, \mathbf{B})$ utilizando la identidad vectorial (A.2), tenemos

$$\nabla^2 \mathbf{A} = -\mu_0 \ \left[\mathbf{J}_f + \chi \ (\nabla \times \mathbf{B}) + (\nabla \chi) \times \mathbf{B} + \nabla \times \mathbf{M}\right]. \tag{3.31}$$

Luego, reemplazando la Ecuación (3.25) en la expresión anterior, obtenemos

$$\nabla^2 \mathbf{A} = -\mu_0 \left[\mathbf{J}_f + \chi \ (\nabla \times \nabla \times \mathbf{A}) + (\nabla \chi) \times (\nabla \times \mathbf{A}) + \nabla \times \mathbf{M}\right], \tag{3.32}$$

lo que, utilizando nuevamente la identidad vectorial (A.1) para el término $(\nabla \times \nabla \times \mathbf{A})$, se reduce a

$$\nabla^2 \mathbf{A} = -\mu_0 \left[\mathbf{J}_f + \chi\, \nabla^2 \mathbf{A} + (\nabla \chi) \times (\nabla \times \mathbf{A}) + \nabla \times \mathbf{M}\right]. \tag{3.33}$$

Al combinar los términos del laplaciano, podemos obtener la siguiente expresión para la ecuación de balance del potencial vectorial:

$$\nabla^2 \mathbf{A} = \frac{-\mu_0}{1 - \chi\, \mu_0} \ \left[\mathbf{J}_f + (\nabla \chi) \times (\nabla \times \mathbf{A}) + \nabla \times \mathbf{M}\right]. \tag{3.34}$$

3.2.2. Forma en divergencia de la ecuación de balance

El RHS de la Ecuación (3.34) incluye el rotacional de la solución, es decir, el rotacional del potencial vectorial magnético $\nabla \times \mathbf{A}$. Este término no se evaluaría eficientemente en la implementación FV a nivel discreto. Por lo tanto, nuestro objetivo en esta sección es reescribir la Ecuación (3.34) para abordar este problema.

Utilizando identidades vectoriales, el término que contiene el rotacional del potencial vectorial magnético puede expresarse como

$$(\nabla\chi)\times(\nabla\times\mathbf{A})=\left(\nabla\mathbf{A}-\nabla\mathbf{A}^{T}\right)\nabla\chi. \tag{3.35}$$

Si definimos $\nabla\widetilde{A}=\left(\nabla\mathbf{A}-\nabla\mathbf{A}^{T}\right)$, entonces la ecuación anterior se simplifica a

$$(\nabla\chi)\times(\nabla\times\mathbf{A})=\nabla\widetilde{A}^{T}\nabla\chi. \tag{3.36}$$

Luego, utilizando la identidad vectorial A.3, este término toma la forma

$$\nabla\widetilde{A}^{T}\nabla\chi=\nabla\cdot\left(\chi\nabla\widetilde{A}\right)-\chi\nabla\cdot\nabla\widetilde{A}. \tag{3.37}$$

Dado que $\nabla\widetilde{A}$ es un tensor antisimétrico, se sigue que $\nabla\widetilde{A}=-\nabla\widetilde{A}$. Por lo tanto, la Ecuación(3.35) se puede reformular como

$$(\nabla\chi)\times(\nabla\times\mathbf{A})=-\nabla\cdot\left(\chi\nabla\widetilde{A}\right)+\chi\nabla\cdot\left(\nabla\widetilde{A}\right). \tag{3.38}$$

A continuación, dado que las identidades vectoriales (A.4) y (A.5), el RHS de la expresión anterior resulta en

$$(\nabla\chi)\times(\nabla\times\mathbf{A})=-\nabla\cdot\left(\chi\nabla\widetilde{A}\right)+\chi\left[\nabla^{2}\mathbf{A}-\nabla\left(\nabla\cdot\mathbf{A}\right)\right], \tag{3.39}$$

lo que, utilizando la conservación del potencial vectorial magnético dado en la Ecuación (3.27), toma la forma simplificada

$$(\nabla\chi)\times(\nabla\times\mathbf{A})=-\nabla\cdot\left(\chi\nabla\widetilde{A}\right)+\chi\nabla^{2}\mathbf{A}. \tag{3.40}$$

La Ecuación (3.40) es el resultado que estábamos buscando, ya que el rotacional del potencial vectorial magnético se ha descompuesto en dos términos, un gradiente y un laplaciano. Argumentamos que la computación de volumen finito de estos dos nuevos términos a nivel discreto es más efectiva que resolver el rotacional del campo de solución ya que permite el uso del teorema de Gauss. Esto significa que si insertamos la expresión anterior en la ley de Ampère combinada en la Ecuación (3.34), obtenemos

$$\nabla^{2}\mathbf{A}=\frac{-\mu_{0}}{1-\chi\,\mu_{0}}\left[\mathbf{J}_{f}-\nabla\cdot\left(\chi\nabla\widetilde{A}\right)+\chi\nabla^{2}\mathbf{A}+\nabla\times\mathbf{M}\right], \tag{3.41}$$

y agrupando los laplacianos, esto produce

$$\nabla^{2}\mathbf{A}=-\mu_{0}\left[\mathbf{J}_{f}-\nabla\cdot\left(\chi\nabla\widetilde{A}\right)+\nabla\times\mathbf{M}\right], \tag{3.42}$$

que tiene el gradiente de un término de divergencia en el RHS en lugar del rotacional de $\mathbf{A}$, como se esperaba.

3.3. Condiciones de contorno de interfaz

Como se mencionó anteriormente, el campo magnético exhibe una discontinuidad en la interfaz entre diferentes materiales. En consecuencia, es esencial establecer una formulación capaz de manejar esta discontinuidad. Para mantener la concisión, no profundizaremos en los detalles de la derivación de la condición de contorno de interfaz; los lectores pueden consultar la derivación completa en (Saravia, 2021). El enfoque para derivar las condiciones de contorno de interfaz implica el uso de cajas de pillboxes gaussianas y bucles de Stokesian para la integración de la ecuación de balance. Esta integración es la operación matemática que conecta los campos en ambos lados de la interfaz. Como se demuestra en (Saravia, 2021), se ha establecido que el potencial vectorial y la componente normal del campo magnético permanecen continuos a través de la interfaz, asegurando una conexión sin problemas; por lo tanto, se cumple

$$\Delta\mathbf{A}=0, \tag{3.43}$$

y

$$\Delta \mathbf{B} \cdot \mathbf{e}_n = 0, \tag{3.44}$$

donde Δ representa el operador para denotar saltos a través de la interfaz, y $\mathbf{e}_n$ es el versor normal a la superficie. Por el contrario, la componente tangencial del potencial vectorial, $\mathbf{A}$, experimenta una discontinuidad en la interfaz, y por lo tanto la magnitud de este salto se define en consecuencia

$$\Delta \mathbf{B} = \mu_0 \mathbf{K} \times \mathbf{e}_n, \tag{3.45}$$

para que $\mathbf{K}$ sea una corriente superficial definida de la siguiente manera

$$\mathbf{K} = -\mathbf{K}_f + \Delta \mathbf{B}_s \times \mathbf{e}_n, \tag{3.46}$$

con $\mathbf{B}_s$ siendo la densidad de flujo magnético en la superficie. Debido a la Ecuación 3.46, la discontinuidad en la componente tangencial del campo magnético conduce a una discontinuidad correspondiente en el gradiente normal del potencial vectorial, que se manifiesta de la siguiente manera

$$\frac{\partial \Delta \mathbf{A}}{\partial e_n} = -\mu_0 \mathbf{K} = \mu_0 \left[\mathbf{K}_f - \Delta \mathbf{B}_s \times \mathbf{e}_n \right], \tag{3.47}$$

siendo e_n la dirección normal a la interfaz, de modo que lo siguiente se cumple

$$\frac{\partial \Delta \mathbf{A}}{\partial e_n} = \left(\mathbf{e}_n \cdot \nabla \right) \Delta \mathbf{A}. \tag{3.48}$$

Las ecuaciones anteriores proporcionan información sobre la dinámica de la densidad de flujo magnético y el potencial vectorial en ambos lados de la interfaz. Estos cambios se expresan en términos de la corriente libre de interfaz, $\mathbf{K}$. En (Saravia, 2021), el autor demuestra que la condición de contorno se vuelve discontinua en la interfaz, con una expresión dada por

$$\frac{\partial \Delta \mathbf{A}}{\partial e_n} = \mu_0 \left[\mathbf{K}_f - \Delta \left(\chi \mathbf{B} + \mathbf{M} \right) \times \mathbf{e}_n \right]. \tag{3.49}$$

3.4. Esquemas numéricos

3.4.1. Discretización en Volumes Finitos

La solución numérica descrita en la Ecuación 3.42 requiere determinar el potencial magnético vectorial $\mathbf{A}$ en puntos específicos dentro del dominio. Estos puntos corresponden a los centroides de las celdas y sirven como base para construir la distribución de $\mathbf{A}$ a lo largo de los elementos de la malla. Este proceso implica reemplazar la solución continua y exacta de la ecuación diferencial parcial con valores discretos, específicamente discretizando la distribución de $\mathbf{A}$ (Moukalled y col., 2016). Por lo tanto, esta sección se adentrará en el procedimiento de transformar la ecuación gobernante en un conjunto de ecuaciones algebraicas para representar los valores discretos de $\mathbf{A}$.

El proceso comienza integrando la Ecuación 3.42 sobre un volumen de control V de forma

$$\int_V \nabla^2 \mathbf{A} \, \mathrm{d}V = -\mu_0 \int_V \left[\mathbf{J}_f - \nabla \cdot \left(\chi \nabla \widetilde{A} \right) + \nabla \times \mathbf{M} \right] \, \mathrm{d}V. \tag{3.50}$$

Se observa que la Ecuación 3.50 representa la forma integral de la ecuación gobernante y no implica ninguna aproximación ni discretización. Ahora nos enfocaremos en los diferentes términos de la Ecuación 3.50. El término del Laplaciano puede reescribirse utilizando el teorema de la divergencia de Gauss como

$$\int_V \nabla^2 \mathbf{A} \, \mathrm{d}V = \int_V \nabla \cdot \left(\nabla \mathbf{A} \right) \, \mathrm{d}V. \tag{3.51}$$

Luego, podemos reemplazar la integral con una suma sobre las caras del volumen de control f, y la expresión anterior se convierte en

$$\int_V \nabla \cdot (\nabla \mathbf{A}) \, \mathrm{d}V \approx \sum_f \mathbf{s} \cdot \nabla \, \mathbf{A}_f, \tag{3.52}$$

donde $\mathbf{s}$ es el vector de superficie y $\mathbf{A}_f$ es el potencial magnético vectorial en la cara f. A continuación, el gradiente de la cara puede evaluarse mediante la interpolación de los gradientes de los centroides de las celdas; esto resulta en

$$\nabla \mathbf{A}_f = f_x \nabla \mathbf{A}_C + (1 - f_x) \nabla \mathbf{A}_E. \tag{3.53}$$

En la expresión anterior, $\mathbf{A}_C$ y $\mathbf{A}_E$ representan el potencial magnético vectorial en los centroides de celdas vecinas C y E respectivamente, como se muestra en la Figura 3.1.

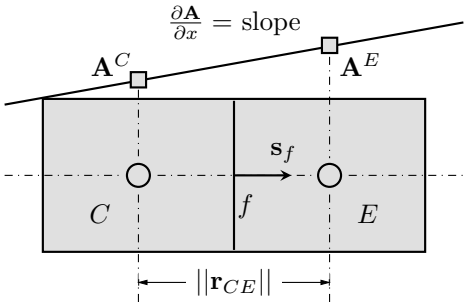

Figura 3.1: Cálculo del gradiente del potencial vectorial entre celdas vecinas.

Por lo tanto, la discretización del gradiente resulta en

$$\nabla \mathbf{A}^C \approx \frac{1}{V} \sum_f \mathbf{A}_f. \tag{3.54}$$

Sin embargo, dentro del marco de OpenFOAM, el enfoque fundamental para calcular el gradiente en una cara implica restar el valor en el centro de la celda en un lado de la cara, C, del valor en el centro en el otro lado, E, y luego dividir por la distancia entre ellos; todas las cantidades se muestran en la Figura 3.1. La precisión de este cálculo es de segundo orden cuando se mide el gradiente normal a la cara, específicamente cuando el vector que conecta los centros de las celdas es perpendicular a la cara, indicando una relación de ángulo recto. Este método se conoce como el esquema ortogonal. En esencia, la Ecuación 3.54 representa una expresión de segundo orden, aunque tiene un término de error principal que es cuatro veces más grande que al calcular el Laplaciano discreto como sigue

$$\mathbf{s}_f \cdot \nabla \mathbf{A}_f = |\mathbf{s}_f| \, \frac{\mathbf{A}^C - \mathbf{A}^E}{|\mathbf{r}_{EC}|}, \tag{3.55}$$

siendo $\mathbf{s}_f$ el vector de superficie de la cara f, y $\mathbf{r}_{EC}$ la distancia entre los centros de las celdas; ver Figura 3.1. Para aprovechar mejor la precisión de la segunda forma, podemos dividir el proceso de discretización en una contribución ortogonal y no ortogonal de la siguiente manera

$$\sum_f \mathbf{s}_f \cdot \nabla \mathbf{A}_f = |\mathbf{g}| \, \frac{\mathbf{A}^C - \mathbf{A}^E}{|\mathbf{r}_{EC}|} + \mathbf{k} \cdot (f_x \nabla \mathbf{A}^C + (1 - f_x) \nabla \mathbf{A}^E). \tag{3.56}$$

La ortogonalidad requiere una malla regular, típicamente alineada con el sistema de coordenadas cartesianas, que normalmente no ocurre en mallas para geometrías de ingeniería del mundo real. Por lo tanto, para mantener la precisión de segundo orden, se puede agregar explícitamente una corrección no ortogonal a la componente ortogonal, conocida como el esquema corregido que se verá más adelante. La corrección aumenta de tamaño a medida que aumenta la no ortogonalidad, con el ángulo entre el vector de celda a celda y el vector normal de la cara aumentando.

El segundo término en el RHS de la Ecuación 3.50 se puede calcular explícitamente utilizando un esquema

de Gauss; esto da como resultado

$$\int_V \nabla \cdot \left(\chi \nabla \widetilde{A}\right) \, \mathrm{d}V \approx \sum_{i=1}^{F} \chi_f \left(\nabla \widetilde{A^i}\right)_f \cdot \mathbf{s}_f^i, \tag{3.57}$$

donde χ_f es la susceptibilidad en la cara f, y F denota el número total de caras. Esta susceptibilidad de cara se puede calcular utilizando un esquema de interpolación lineal. Para el enfoque discontinuo, la susceptibilidad para caras interfaciales se calcula como una función de la celda propietaria solamente, evitando así el efecto de difuminación generado por el proceso de promediado de celda; es decir,

$$\chi_f = (1 - w_f) \; \chi_C + w_f \, \chi_E, \tag{3.58}$$

siendo w_f el factor de interpolación, definido como

$$w_f = \frac{||\mathbf{r}_{Cf}||}{||\mathbf{r}_{EC}||}. \tag{3.59}$$

Por último, el término del rizo en el RHS de la Ecuación 3.50 se puede discretizar como

$$\int_V \nabla \times \mathbf{M} \, \mathrm{d}V \approx 2 \, \mathrm{skew} \left(\frac{1}{V} \sum_{i=1}^{F} \mathbf{M}_f^i \otimes \mathbf{s}_f^i \right). \tag{3.60}$$

Sin embargo, como se ha señalado antes, la mayoría de las aplicaciones del mundo real se modelan con mallas no ortogonales y oblicuas estructuradas o no estructuradas. Por lo tanto, el proceso de discretización descrito anteriormente no es preciso, ya que ahora el vector de superficie $\mathbf{s}_f$ y el vector $||\mathbf{r}_{CE}||$ que une los centroides de los elementos que abarcan la interfaz no son colineales. Esta situación particular se muestra en la Figura 3.2. Como se puede ver en la figura, el vector normal a la superficie ahora tiene una componente en la dirección perpendicular a $||\mathbf{r}_{CE}||$. Por lo tanto, el gradiente no se puede escribir en términos de $\mathbf{A}^C$ y $\mathbf{A}^E$ como antes. Además, la dirección del gradiente que resulta en una expresión que incorpora tanto $\mathbf{A}^C$ como $\mathbf{A}^E$ debe alinearse con la línea que conecta los puntos C y E.

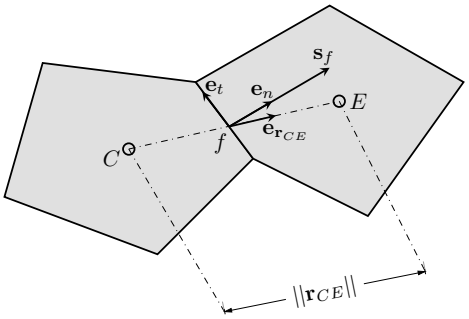

Figura 3.2: Representación de elementos vecinos en una malla no ortogonal.

Denotaremos $\mathbf{e}_{\mathbf{r}CE}$ como el vector unitario a lo largo de la dirección de $\mathbf{r}_{CE}$, y podemos definirlo como

$$\mathbf{e}_{\mathbf{r}_{CE}} = \frac{\mathbf{r}_C - \mathbf{r}_E}{||\mathbf{r}_{CE}||}. \tag{3.61}$$

De esta manera, podemos encontrar una expresión para el gradiente en la dirección $\mathbf{e}_{\mathbf{r}CE}$ como

$$(\nabla \mathbf{A} \cdot \mathbf{e}_{\mathbf{r}_{CE}})_f = \left(\frac{\partial \mathbf{A}}{\partial e}\right)_f = \frac{\mathbf{A}^C - \mathbf{A}^E}{||\mathbf{r}_{CE}||}. \tag{3.62}$$

Luego, el vector de superficie $\mathbf{s}_f$ se puede descomponer como la suma de dos vectores de la forma

$$\mathbf{s}_f = \mathbf{E}_f + \mathbf{T}_f, \tag{3.63}$$

donde $\mathbf{E}_f$ está en la dirección de $\mathbf{e}_{\mathbf{r}_{CE}}$ y $\mathbf{T}_f$ está en la dirección tangencial $\mathbf{e}_t$. Al escribir $\mathbf{s}_f$ de esta manera, pretendemos linealizar la malla no ortogonal. En otras palabras, debido a que $\mathbf{E}_f$ está en la dirección de $\mathbf{e}_{\mathbf{r}_{CE}}$, podemos escribir parte del gradiente del potencial vectorial como una función de los valores del centro de la celda $\mathbf{A}^C$ y $\mathbf{A}^E$ de tal manera que

$$(\nabla \mathbf{A})_f \cdot \mathbf{s}_f = (\nabla \mathbf{A})_f \cdot \mathbf{E}_f + (\nabla \mathbf{A})_f \cdot \mathbf{T}_f. \tag{3.64}$$

El primer término en el RHS de la Ecuación 3.64 corresponde a una contribución similar a la observada en mallas ortogonales, ya que implica la participación de los valores nodales $\mathbf{A}^C$ y $\mathbf{A}^E$. Podemos expandir este término como

$$(\nabla \mathbf{A})_f \cdot \mathbf{E}_f = E_f \left(\frac{\partial \mathbf{A}}{\partial e} \right)_f = E_f \frac{\mathbf{A}^C - \mathbf{A}^E}{||\mathbf{r}_{CE}||}. \tag{3.65}$$

En contraste, el segundo término en el RHS se define como una difusión no ortogonal, derivada de la naturaleza no ortogonal de la malla. Por lo tanto, se necesita un método de corrección no ortogonal para abordar la no ortogonalidad de la malla; es decir, la componente no ortogonal del gradiente debe moverse al RHS y evaluarse explícitamente.

3.4.2. Esquemas de solución

La discretización numérica de problemas que involucran medios permeables de múltiples regiones resulta en conjuntos de ecuaciones algebraicas no lineales acopladas que implican variables correspondientes a diferentes dominios. Se pueden utilizar dos enfoques para resolver estos problemas: *a)* abordar simultáneamente todos los dominios, generando así un solo conjunto de ecuaciones algebraicas que abarque todas las variables relevantes o *b)* resolver el sistema de ecuaciones de cada dominio individual y secuencialmente, y tratar los términos de acoplamiento asociados a las interacciones entre regiones de manera explícita.

El enfoque *a)* genera variables que exhiben heterogeneidad, ya que representan la discretización en diversos dominios, mientras que el *b)* produce múltiples conjuntos de ecuaciones algebraicas. Este último generalmente es más efectivo, ya que se evita la solución de un sistema grande a expensas de iteraciones entre sistemas más pequeños; por lo tanto, en este documento elegimos seguir el enfoque *b)*.

Dado que implementamos la formulación actual en OpenFOAM, las principales dificultades asociadas con la adopción de un esquema multirregión escalonado son: la transferencia de datos de campo entre las regiones; el algoritmo de verificación de convergencia; y la sincronización de la solución de la región que se superan intrínsecamente por las capacidades de la biblioteca. El esquema de solución es el siguiente: supongamos un sistema magnético de dos regiones, entonces la matriz del sistema correspondiente a la versión discretizada de la Ecuación 3.42 es

$$\begin{bmatrix} \mathbf{L}_{11} + \mathbf{N}_{11}(\mathbf{A}_1) & \mathbf{N}_{12}(\mathbf{A}_2) \\ \mathbf{N}_{21}(\mathbf{A}_1) & \mathbf{L}_{22} + \mathbf{N}_{22}(\mathbf{A}_2) \end{bmatrix} \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{F}_1 \\ \mathbf{F}_2 \end{bmatrix}, \tag{3.66}$$

donde $\mathbf{L}_{jj}$ son operadores lineales, $\mathbf{N}_{jj}$ son operadores no lineales correspondientes al segundo término en el RHS de la Ecuación (3.50), $\mathbf{A}_1$ y $\mathbf{A}_2$ son los potenciales vectoriales para cada región, y $\mathbf{F}_1$ y $\mathbf{F}_2$ son los términos del RHS en la Ecuación 3.42.

Una solución directa de la Ecuación (3.66) se puede obtener utilizando esquemas de linealización disponibles para la solución de problemas no lineales, como el método de Newton-Raphson, el método de Picard, etc. En cambio, optamos por usar un esquema iterativo de bloque de la forma

$$\mathbf{L}_{11}\mathbf{A}_1^i = \mathbf{F}_1 - \mathbf{N}_{11}\left(\mathbf{A}_1^{i-1}\right)\mathbf{A}_1^{i-1} - \mathbf{N}_{12}\left(\mathbf{A}_2^{i-1}\right)\mathbf{A}_2^{i-1} \tag{3.67}$$

y

$$\mathbf{L}_{22}\mathbf{A}_2^i = \mathbf{F}_2 - \mathbf{N}_{22}\left(\mathbf{A}_2^{i-1}\right)\mathbf{A}_2^{i-1} - \mathbf{N}_{21}\left(\mathbf{A}_1^{i}\right)\mathbf{A}_1^{i}, \tag{3.68}$$

donde i es el índice del contador de iteraciones. El uso de la última solución disponible para la región 1 en el último término en el RHS de la Ecuación (3.68) indica que este bucle de iteración es un esquema de Gauss-Seidel en bloque (BGS).

Este algoritmo es efectivo para la solución de sistemas acoplados de múltiples regiones; sin embargo, la convergencia del proceso de solución se ve muy afectada por la naturaleza secuencial del solucionador. Con mallas altamente no ortogonales o sesgadas y sistemas mal condicionados, ocurren grandes variaciones en $\mathbf{A}$ durante el proceso de solución. Además, en algunos casos, este comportamiento puede causar pérdida de convergencia. Para promover la convergencia y estabilizar el proceso iterativo, un enfoque práctico es sub-relajar ciertos campos entre iteraciones.

Dado que el RHS del sistema de una región es una función de segundas derivadas de $\mathbf{A}$, la relajación de $\mathbf{A}$ no siempre estabiliza la solución. No es raro encontrar que para casos que involucran materiales altamente permeables, la solución diverge. Además, como muestra la Ecuación 3.49, la condición límite de la interfaz es una función de un salto en el campo magnético en ambos lados de la interfaz. Dado que el esquema BGS resuelve cada región a la vez, durante el bucle secuencial de la región, se actualiza uno de los valores del centro de la celda vecina de la interfaz, mientras que los otros no lo hacen. Esto a menudo genera grandes oscilaciones en los saltos de la interfaz. En la implementación numérica, la condición límite de la interfaz alimenta al operador Laplaciano con el gradiente normal de la cara del potencial vectorial,

$$\frac{\partial \mathbf{A}_e^C}{\partial e_n} = a_C \mathbf{A}^C + a_E \mathbf{A}^E + a_K \mathbf{K}, \tag{3.69}$$

siendo a_C el factor interno de gradiente y a_E y a_K factores de gradiente de frontera (Moukalled y col., 2016) y la corriente superficial generalizada dada por 3.46.

Dado que la corriente superficial libre $\mathbf{K}_f$ y la magnetización permanente $\mathbf{M}$ son generalmente constantes y uniformemente distribuidas, la oscilación de la corriente superficial es generada por el término $\Delta(\chi \mathbf{B})$; esto sugiere que su sub-relajación ciertamente ayudaría a la convergencia. El primer término en el RHS de la Ecuación (3.69) se suma a la diagonal de la matriz del sistema, mientras que los otros dos se suman al RHS. Para el caso de las caras de interfaz, el segundo término no es un término de matriz porque el valor del centro de la celda vecina pertenece a una región diferente. En cuanto al tercer término, debe evaluarse explícitamente porque la corriente de interfaz no puede escribirse en términos de $\mathbf{A}^C$ ya que el rotor acopla las componentes vectoriales. Por lo tanto, tanto el segundo como el tercer término pueden generar cambios bruscos en el RHS del sistema en la región, lo que conduce a inestabilidades en sistemas mal condicionados.

En la misma dirección, el término de divergencia en el RHS de la Ecuación (3.42) se discretiza explícitamente como se indica en la Ecuación 3.57.

Así, el gradiente del potencial vectorial puede generar valores con una magnitud no físicamente grande en mallas malas. Un valor de gradiente grande en una celda puede contribuir lo suficiente al término fuente de una ecuación como para causar la no acotación de la solución.

Para aliviar los problemas mencionados, utilizamos dos técnicas, limitación del gradiente y relajación del campo para la magnetización inducida y $\Delta(\chi \mathbf{B})$. La relajación de un campo $\mathbf{A}$ se ejecuta de la siguiente manera; siendo la ecuación de discretización general

$$a_C \mathbf{A}^C + \sum_f a_E \mathbf{A}^E = b_C, \tag{3.70}$$

el valor actual del centro de la celda es

$$\mathbf{A}^C = \frac{-\sum_f a_E \mathbf{A}^E + b_C}{a_C}. \tag{3.71}$$

Ahora, sea $\mathbf{A}{*}^C$ el valor de $\mathbf{A}^C$ en la iteración anterior. Si $\mathbf{A}{*}^C$ se suma y resta al RHS de la ecuación anterior,

entonces se obtiene

$$\mathbf{A}^C = \mathbf{A}_*^C + \left(\frac{-\sum_f a_E \mathbf{A}^E + b_C}{a_C} - \mathbf{A}_*^C \right) = \mathbf{A}_*^C + \widehat{\mathbf{A}^C}, \tag{3.72}$$

donde $\widehat{\mathbf{A}^C}$ representa el cambio en $\mathbf{A}^C$ producido por la iteración actual. Este cambio podría modificarse mediante la introducción de un factor de relajación $\lambda^{\mathbf{A}}$, de modo que

$$\mathbf{A}^C = \mathbf{A}_*^C + \lambda^{\mathbf{A}} \widehat{\mathbf{A}^C}. \tag{3.73}$$

En la convergencia, los valores de $\mathbf{A}^C$ y $\mathbf{A}_*^C$ se vuelven iguales independientemente del valor de $\lambda^{\mathbf{A}}$ utilizado.

Los experimentos numéricos muestran que, para ayudar a la convergencia de la ecuación de balance, un factor de sub-relajación de la magnetización inducida $\lambda^{\mathbf{M}_I} \approx 0{,}8$ parece ser óptimo. Notablemente, la convergencia es menos sensible a las variaciones en $\Delta(\chi \mathbf{B})$, por lo que se pueden usar valores de relajación más altos.

En el contexto del esquema multirregión, existen dos estrategias principales para implementar el proceso de sub-relajación. La primera implica una aplicación por región, mientras que la segunda implica que la relajación se aplique globalmente a todas las regiones después de la solución. Optar por el primer enfoque introduce una dependencia significativa del orden en el que se resuelven las regiones, ya que una solución sub-relajada puede propagarse a regiones vecinas a través de las condiciones de contorno de la interfaz. Por el contrario, tales preocupaciones se mitigaron en el último enfoque, ya que la relajación del lado derecho de la ecuación de equilibrio se aplica uniformemente a todas las regiones antes de comenzar el bucle de solución iterativa.

El marco de OpenFOAM 4

4.1. Introducción

OpenFOAM es una librería de `C++` utilizada para resolución numérica de ecuaciones diferenciales. Incluye una colección de aplicaciones que realizan una variedad de tareas. Las aplicaciones utilizan funcionalidades empaquetadas contenidas en más de 150 bibliotecas. Además de realizar cálculos numéricos, hay aplicaciones que configuran e inicializan simulaciones, manipulan la geometría del caso, generan mallas computacionales y procesan y visualizan resultados. Las aplicaciones principalmente se dividen en dos categorías: *solvers*, que realizan los cálculos numéricos; y utilidades, que realizan las otras tareas descritas anteriormente. Antes de la versión 11 de OpenFOAM, se escribían solvers individuales para numerosos tipos específicos de sistemas dinámicos físicos. Con tantas combinaciones de tipos de sistemas y física adicional, OpenFOAM incluía casi 100 *solvers* en un momento dado. Solvers con nombres como `simpleFoam` y `pimpleFoam` han sido la piedra angular de OpenFOAM desde principios de la década de 1990. Sin embargo, la versión 11 de OpenFOAM introduce solvers modulares como una mejora importante a los solvers de aplicaciones originales. Los solvers de aplicaciones son reemplazados por un único solver `foamRun` que describe los pasos de un algoritmo general para cálculos de dinámica de fluidos. `foamRun` carga un módulo de solver, que define cada paso para caracterizar un tipo particular de flujo.

Los solvers modulares son más simples de usar y mantener que los solvers de aplicaciones. Su código fuente es más fácil de navegar, lo que promueve un mejor entendimiento. Son más flexibles; en particular, también hay un solver `foamMultiRun` que puede tomar dos o más regiones de dominio y aplicar un módulo de solver diferente a cada región. En particular, los módulos para una o más dinámicas pueden acoplarse para casos multifásico.

4.1.1. El lenguaje de programación de OpenFOAM

Esta sección proporciona información para ayudar a comprender cómo se compilan las aplicaciones y bibliotecas de OpenFOAM; y ofrece algunos conocimientos básicos sobre `C++`, el lenguaje base de OpenFOAM. Henry Weller eligió `C++` como el lenguaje de programación principal de OpenFOAM cuando lo creó a finales de la década de 1980. La idea era utilizar la programación orientada a objetos para expresar conceptos abstractos de manera eficiente, al igual que el lenguaje verbal y las matemáticas pueden hacerlo. Por ejemplo, en dinámica de fluidos, usamos el término *campo de velocidad*, que tiene significado sin ninguna referencia a la naturaleza del flujo o a datos de velocidad específicos. El término encapsula la idea de movimiento con dirección y magnitud y se relaciona con otras propiedades físicas. En matemáticas, *campo de velocidad* puede ser reemplazado por un solo símbolo, por ejemplo U, y otros símbolos expresan operaciones y funciones, por ejemplo, *el campo de magnitud de velocidad* por $|U|$.

Las herramientas de resolución de ecuaciones diferenciales de OpenFOAM se hicieron muy populares para resolver fenómenos de dinámica de fluidos. La dinámica de fluidos computacional trata con ecuaciones diferenciales parciales en 3 dimensiones de espacio y tiempo. Las ecuaciones contienen: campos de escalares, vectores y tensores; álgebra tensorial; cálculo tensorial; y unidades dimensionales. La solución a estas ecuaciones implica procedimientos de discretización, matrices, solucionadores y algoritmos de solución.

En lugar de programar CFD en términos de entidades intrínsecas conocidas por una computadora, por ejemplo, bits, bytes, enteros, números de punto flotante, OpenFOAM proporciona clases que definen las entidades encontradas en CFD. Por ejemplo, un campo de velocidad puede ser definido por una clase `vectorField`, permitiendo que un programador cree una instancia, u objeto, de esa clase. El objeto puede ser creado con el nombre `U` para imitar el símbolo utilizado en matemáticas. Las funciones asociadas pueden ser creadas con nombres que también intentan emular la simplicidad de las matemáticas, por ejemplo, `mag(U)` puede representar $|U|$.

La estructura de clases concentra el desarrollo de código en regiones contenidas del código, es decir, las propias clases, lo que hace que el código sea más fácil de manejar. Se pueden derivar nuevas clases o heredar propiedades de otras clases, por ejemplo, el `vectorField` puede derivarse de una clase `vector` y una clase `Field`. `C++` proporciona el mecanismo de clases de plantilla de manera que la clase de plantilla `Field<Type>` puede representar un campo de cualquier `<Type>`, por ejemplo, escalar, vector, tensor. Las características generales de la clase de plantilla se transfieren a cualquier clase creada a partir de la plantilla. El templado y la herencia reducen la duplicación de código y crean jerarquías de clases que imponen una estructura general en el código.

Un tema del diseño de OpenFOAM es que tiene una sintaxis que se asemeja estrechamente a las ecuaciones diferenciales parciales que se están resolviendo. Por ejemplo, la ecuación

$$\frac{\partial (\rho \mathbf{U})}{\partial t} + \nabla \cdot \phi \, \mathbf{U} - \nabla \cdot \mu \, \nabla \mathbf{U} = -\nabla p \tag{4.1}$$

se puede representar por el Código 4.1.1.

Listing 4.1.1: Representación de Ecuación

```
solve
(
      fvm::ddt(rho, U)
    + fvm::div(phi, U)
    - fvm::laplacian(mu, U)
      ==
    - fvc::grad(p)
);
```

La sintaxis de las ecuaciones es más evidente en el código de los *solvers*, tanto en los módulos como en las aplicaciones de solvers. Los usuarios no necesitan un profundo conocimiento de programación en `C++` para interpretar las ecuaciones escritas en un solver.

Ayuda tener un entendimiento rudimentario de los principios detrás de la orientación a objetos y las clases, y tener un conocimiento básico de la sintaxis de algunos códigos en `C++`. Para programar en OpenFOAM, a menudo hay poca necesidad para que un usuario se sumerja en el código de cualquiera de las clases de OpenFOAM. La esencia de la orientación a objetos es que el usuario no debería tener que revisar el código de cada clase que utiliza; simplemente el conocimiento de la existencia de la clase y su funcionalidad son suficientes para usarla.

4.2. Compilación de apliaciones y *solvers*

La compilación es una parte integral del desarrollo de código que requiere una gestión cuidadosa, ya que cada pieza de código requiere sus propias instrucciones para acceder a los componentes dependientes de la biblioteca OpenFOAM. En los sistemas Linux existen varias herramientas para ayudar a automatizar el proceso de gestión, comenzando con la utilidad estándar `make`. OpenFOAM utiliza su propio `script` de compilación wmake que se basa en `make`. Está específicamente diseñado para el gran número de componentes individuales que se compilan por separado en OpenFOAM (aproximadamente 150 aplicaciones y 150 bibliotecas).

Para entender el proceso de compilación, primero necesitamos explicar ciertos aspectos de `C++` y su estructura de archivos, que se muestra esquemáticamente en la Figura 4.1. Una clase se define a través de un conjunto de instrucciones como la construcción de objetos, almacenamiento de datos y funciones miembro de la clase. El archivo que define estas funciones —la definición de la clase— tiene una extensión `.C`, por ejemplo, una clase `nc` se escribiría en el archivo `nc.C`. Este archivo puede ser compilado de forma independiente de otro código en un archivo binario ejecutable de biblioteca conocido como una biblioteca de objetos compartidos con la extensión de archivo `.so`, es decir, `nc.so`. Cuando se compila un fragmento de código, por ejemplo,

newApp.C, que utiliza la clase nc, nc.C no necesita ser recompilado, sino que newApp.C llama a la biblioteca nc.so en tiempo de ejecución. Esto se conoce como enlace dinámico.

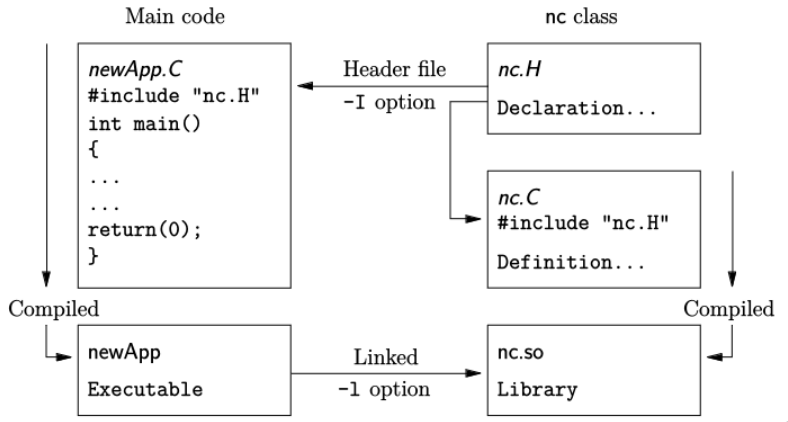

Figura 4.1: Archivos de encabezado, source, compilación y linking. (Greenshields y Weller, 2022)

4.2.1. Archivos de encabezado o header .H

Como medio de verificar errores, el fragmento de código que se está compilando debe saber que las clases que utiliza y las operaciones que realizan realmente existen. Por lo tanto, cada clase requiere una declaración de clase, contenida en un archivo de encabezado con una extensión de archivo .H, por ejemplo, nc.H, que incluye los nombres de la clase y sus funciones. Este archivo se incluye al principio de cualquier fragmento de código que utilice la clase, utilizando la directiva include descrita a continuación, incluyendo el código de declaración de clase en sí mismo. Cualquier fragmento de código .C puede hacer referencia a cualquier número de clases y debe comenzar incluyendo todos los archivos .H requeridos para declarar estas clases. A su vez, esas clases pueden hacer referencia a otras clases y, por lo tanto, también comienzan incluyendo los archivos .H relevantes. Al buscar recursivamente en la jerarquía de clases, podemos producir una lista completa de archivos de encabezado para todas las clases en las que finalmente depende el código de nivel superior .C; estos archivos .H se conocen como las dependencias. Con una lista de dependencias, un compilador puede verificar si los archivos fuente se han actualizado desde su última compilación y compilar selectivamente solo aquellos que lo necesitan.

Los archivos de encabezado se incluyen en el código mediante la directiva include, por ejemplo como se ve a continuación:

```
1 include "nc.H"
```

Esto hace que el compilador suspenda la lectura del archivo actual para leer el archivo incluido. Este mecanismo permite que cualquier fragmento de código autocontenido se incluya en un archivo de encabezado y se incluya en la ubicación relevante en el código principal para mejorar la legibilidad del código. Por ejemplo, en la mayoría de las aplicaciones de OpenFOAM, el código para crear campos y leer datos de entrada de campo se incluye en un archivo createFields.H que se llama al principio del código. De esta manera, los archivos de encabezado no se utilizan únicamente como declaraciones de clases.

Es wmake el que realiza la tarea de mantener listas de dependencias de archivos, entre otras funciones enumeradas a continuación:

- Generación automática y mantenimiento de listas de dependencias de archivos, es decir, listas de archivos que se incluyen en los archivos fuente y, por lo tanto, de los que dependen.
- Compilación y enlace multiplataforma, manejados a través de una estructura de directorios apropiada.
- Compilación y enlace multilingüe, por ejemplo, C, C++, Java.

- Compilación y enlace multiopción, por ejemplo, depuración, optimizado, paralelo y perfilado.
- Soporte para programas de generación de código fuente, por ejemplo, `lex`, `yacc`, `IDL`, `MOC`.
- Sintaxis simple para listas de archivos fuente.
- Creación automática de listas de archivos fuente para nuevos códigos.
- Manejo simple de múltiples bibliotecas compartidas o estáticas.

4.2.2. Compilación con `wmake`

Las aplicaciones de OpenFOAM se organizan utilizando una convención estándar según la cual el código fuente de cada aplicación se coloca en un directorio cuyo nombre es el de la aplicación. El archivo fuente de nivel superior toma entonces el nombre de la aplicación con la extensión `.C`. Por ejemplo, el código fuente de una aplicación llamada `newApp` residiría en un directorio `newApp` y el archivo de nivel superior sería `newApp.C`, como se muestra en la Figura 4.2. Luego, `wmake` requiere que el directorio contenga un subdirectorio llamado `Make` que contenga 2 archivos, `options` y `files`, que se describen en las secciones siguientes.

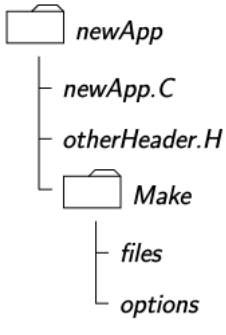

Figura 4.2: Estructura de directorios de una aplicación. (Greenshields y Weller, 2022)

4.2.3. Inclusión de encabezados

El compilador busca los archivos de encabezado incluidos en el siguiente orden, especificado con la opción -I en wmake:

1. el directorio `$WM-PROJECT-DIR/src/OpenFOAM/lnInclude;`,
2. un directorio `lnInclude` local, es decir, `newApp/lnInclude;`,
3. el directorio local, es decir, `newApp`,
4. rutas dependientes de la plataforma establecidas en archivos en el directorio `$WM-PROJECT-DIR/wmake/rules`, por ejemplo, `/usr/include/X11`,
5. otros directorios especificados explícitamente en el archivo `Make/options` con la opción `-I`.

El archivo `Make/options` contiene las rutas completas de los directorios para localizar archivos de encabezado utilizando la sintaxis:

Listing 4.2.1: Sintaxis del archivo `Make/options`.

```
EXE_INC = \
    -I<directoryPath1> \
    -I<directoryPath2> \
    ... \
    -I<directoryPathN>
```

Es importante notar primero que los nombres de los directorios están precedidos por el flag `-I` y que la sintaxis usa el \ para continuar `EXE-INC` a través de varias líneas, sin \ después de la entrada final.

4.2.4. Archivos fuente a ser compilados

El compilador requiere una lista de archivos fuente `.C` que deben ser compilados. La lista debe contener el archivo `.C` principal, pero también cualquier otro archivo fuente que se cree para la aplicación específica pero que no esté incluido en una biblioteca de clases. Por ejemplo, los usuarios pueden crear una nueva clase o alguna nueva funcionalidad para una clase existente para una aplicación en particular.

La lista completa de archivos fuente `.C` debe incluirse en el archivo `Make/files`. Para muchas aplicaciones, la lista solo incluye el nombre del archivo `.C` principal, por ejemplo, `newApp.C` en el caso de nuestro ejemplo anterior. El archivo `Make/files` también incluye una ruta completa y el nombre del ejecutable compilado, especificado por la sintaxis `EXE =`. La convención estándar estipula que el nombre es el de la aplicación, es decir, `newApp` en nuestro ejemplo. La versión de OpenFOAM ofrece dos opciones útiles para la ruta: las aplicaciones de la versión estándar se almacenan en `$FOAM-APPBIN`; las aplicaciones desarrolladas por el usuario se almacenan en `$FOAM-USER-APPBIN`.

Si el usuario está desarrollando sus propias aplicaciones, es recomondable que se cree un subdirectorio de aplicaciones en su directorio `$WM-PROJECT-USER-DIR` que contenga el código fuente de las aplicaciones personales de OpenFOAM. Al igual que con las aplicaciones estándar, el código fuente de cada aplicación de OpenFOAM debe almacenarse dentro de su propio directorio. La única diferencia entre una aplicación de usuario y una de la versión estándar es que el archivo `Make/files` debe especificar que los ejecutables del usuario se escriben en su directorio `$FOAM-USER-APPBIN`.

El archivo `Make/files` para nuestro ejemplo aparecería de la siguiente manera:

Listing 4.2.2: Sintaxis del archivo `Make/files`.

```
newApp.C
EXE = \(FOAM_USER_APPBIN)/newApp
```

4.2.5. Ejecución de `wmake`

El script `wmake` se ejecuta generalmente escribiendo:

```
wmake <optionalDirectory>
```

El `<optionalDirectory>` es la ruta del directorio de la aplicación que se está compilando.

Normalmente, `wmake` se ejecuta desde dentro del directorio de la aplicación que se está compilando, en cuyo caso se puede omitir `<optionalDirectory>`.

Para información, a continuación se enumeran los ajustes generales de variables de entorno utilizados por `wmake`.

- `$WM-PROJECT-INST-DIR`: ruta completa al directorio de instalación, por ejemplo, `$HOME/OpenFOAM`.
- `$WM-PROJECT`: nombre del proyecto en compilación, es decir, OpenFOAM.
- `$WM-PROJECT-VERSION`: versión del proyecto en compilación, es decir, 11.
- `$WM-PROJECT-DIR`: ruta completa al directorio principal de la versión de OpenFOAM, por ejemplo, `$HOME/OpenFOAM/OpenFOAM-11`.
- `$WM-PROJECT-USER-DIR`: ruta completa al directorio equivalente para desarrollos personalizados en la cuenta de usuario, por ejemplo, `$HOME/OpenFOAM/$USER-11`.
- `$WM-THIRD-PARTY-DIR`: ruta completa al directorio del software de terceros, por ejemplo como se puede ver a continuación `$HOME/OpenFOAM/ThirdParty-11`.

A continuación se enumeran los ajustes de variables de entorno para la compilación con `wmake`.

- `$WM-ARCH`: arquitectura de la máquina, por ejemplo, `linux`, `linux64`, `linuxArm64`, `linuxARM7`, `linuxPPC64`, `linuxPPC64le`.
- `$WM-ARCH-OPTION`: arquitectura de 32 o 64 bits.
- `$WM-DIR`: ruta completa del directorio `wmake`.
- `$WM-LABEL-SIZE`: tamaño de 32 o 64 bits para etiquetas (enteros).
- `$WM-LABEL-OPTION`: compilación de etiquetas `Int32` o `Int64`.
- `$WM-LINK-LANGUAGE`: compilador utilizado para enlazar bibliotecas y ejecutables, `C++`.
- `$WM-MPLIB`: biblioteca de comunicaciones paralelas, `SYSTEMOPENMPI` = versión del sistema de `openMPI`, las alternativas incluyen `OPENMPI`, `SYSTEMMPI`, `MPICH`, `MPICH-GM`, `HPMPI`, `MPI`, `QSMPI`, `INTELMPI` y `SGIMPI`.
- `$WM-OPTIONS`, por ejemplo, `linux64GccDPInt32Opt`, formado combinando `$WM-ARCH`, `$WM-COMPILER`, `$WM-PRECISION-OPTION`, `$WM-LABEL-OPTION` y `$WM-COMPILE-OPTION`.
- `$WM-PRECISION-OPTION`: precisión de punto flotante de los binarios compilados, `SP` = precisión simple, `DP` = precisión doble.

A continuación se enumeran los ajustes de variables de entorno relacionados con la elección del compilador y las opciones con `wmake`.

- `$WM-CC`: elección del compilador `C`, `gcc`.
- `$WM-CFLAGS`: banderas adicionales para el compilador `C`, por ejemplo, `-m64 -fPIC`.
- `$WM-CXX`: elección del compilador `C++`, `g++`.
- `$WM-CXXFLAGS`: banderas adicionales para el compilador `C++`, por ejemplo, `-m64 -fPIC -std=c++0x`.
- `$WM-COMPILER`: compilador utilizado, por ejemplo, `Gcc = gcc`, `Clang = LLVM Clang`.
- `$WM-COMPILE-OPTION`: opción de compilación, `Debug` = depuración, `Opt` = optimizado.
- `$WM-COMPILER-LIB-ARCH`: arquitectura de la biblioteca del compilador, por ejemplo, 64.
- `$WM-COMPILER-TYPE`: elección del compilador, sistema o `ThirdParty`, es decir, compilado en el directorio `ThirdParty`.
- `$WM-LDFLAGS`: banderas adicionales para el enlazador, por ejemplo, `-m64`.
- `$WM-LINK-LANGUAGE`: lenguaje del enlazador, por ejemplo, `C++`.
- `$WM-OSTYPE`: sistema operativo, `=POSIX c++`.

Cuando se ejecuta, `wmake` construye un archivo de lista de dependencias con una extensión de archivo `.dep`, por ejemplo, `newApp-C.dep` en nuestro ejemplo, en un subdirectorio `$WM-OPTIONS` del directorio `Make`, por ejemplo, `Make/linuxGccDPInt64Opt`. Si el usuario desea eliminar estos archivos, por ejemplo, después de realizar cambios en el código, el usuario puede ejecutar el script `wclean` escribiendo:

```
1 wclean <optionalDirectory>
```

Nuevamente, el `<optionalDirectory>` es una ruta al directorio de la aplicación que se está compilando. Típicamente, wclean se ejecuta desde dentro del directorio de la aplicación, en cuyo caso la ruta puede omitirse.

Luego, al compilar una biblioteca, hay 2 diferencias críticas en la configuración del archivo en el directorio `Make`:

- en el archivo files, `EXE` = es reemplazado por `LIB` = y el directorio de destino para la entidad compilada cambia de `$FOAM-APPBIN` a `$FOAM-LIBBIN` (y un directorio equivalente `$FOAM-USER-LIBBIN`);
- en el archivo options, `EXE-LIBS` = es reemplazado por `LIB-LIBS` = para indicar las bibliotecas enlazadas a la biblioteca que se está compilando.

Cuando se ejecuta `wmake`, además crea un directorio llamado `lnInclude` que contiene enlaces simbólicos a todos los archivos de la biblioteca. El directorio `lnInclude` es eliminado por el script `wclean` al limpiar el código fuente de la biblioteca.

4.2.6. Debugging y switches de optimización

OpenFOAM proporciona un sistema de mensajería que se escribe durante el tiempo de ejecución, la mayoría de los cuales son para ayudar a depurar problemas encontrados durante la ejecución de un caso de OpenFOAM. Los interruptores se enumeran en el archivo `$WM-PROJECT-DIR/etc/controlDict`; si el usuario desea cambiar la configuración, debería hacer una copia en su directorio `$HOME`, es decir, el archivo `$HOME/.OpenFOAM/11/controlDict`. La lista de posibles interruptores es extensa, relacionada con una clase o rango de funcionalidad, y se pueden activar incluyéndolos en el archivo `controlDict`, y estableciéndolos en `1`. Por ejemplo, OpenFOAM puede realizar la verificación de unidades dimensionales en todos los cálculos estableciendo el interruptor `dimensionSet` en `1`.

Un pequeño número de interruptores controlan la mensajería en tres niveles, `0`, `1` y `2`, especialmente el interruptor de nivel general y `lduMatrix` que proporciona mensajes para la convergencia del solucionador durante una ejecución.

Hay algunos interruptores que controlan ciertos problemas operativos y de optimización. De particular importancia es `fileModificationSkew`. OpenFOAM escanea el tiempo de escritura de los archivos de datos para verificar la modificación. Cuando se ejecuta sobre un `NFS` con alguna disparidad en la configuración de reloj en diferentes máquinas, los archivos de datos de campo parecen ser modificados antes de tiempo. Esto puede causar un problema si OpenFOAM considera los archivos como recién modificados e intenta volver a leer estos datos. La palabra clave `fileModificationSkew` es el tiempo en segundos que OpenFOAM restará del tiempo de escritura del archivo al evaluar si el archivo ha sido modificado recientemente. Los principales interruptores de optimización se enumeran a continuación:

- `fileModificationSkew`: un tiempo en segundos que debería establecerse más alto que el retraso máximo en las actualizaciones de `NFS` y la diferencia de reloj para ejecutar OpenFOAM sobre un `NFS`.
- `fileModificationChecking`: método para comprobar si los archivos han sido modificados durante una simulación, ya sea leyendo la marca de tiempo o usando `inotify`; también existen versiones que leen solo datos de nodo maestro, denominadas `timeStampMaster` e `inotifyMaster`.
- `commsType`: tipo de comunicación paralela, no bloqueante, programada o bloqueante.
- `floatTransfer`: si es `1`, compactará números a precisión flotante antes de transferirlos; el valor predeterminado es `0`.
- `nProcsSimpleSum`: optimiza la suma global para el procesamiento paralelo, estableciendo el número de procesadores por encima del cual se realiza una suma jerárquica en lugar de una suma lineal.

4.2.7. Enlace dinámico en tiempo de ejecución

Puede surgir la situación de que un usuario cree una nueva biblioteca, digamos `new1`, y desee que las características dentro de esa biblioteca estén disponibles en una variedad de aplicaciones. Por ejemplo, el usuario puede crear una nueva condición de contorno, compilada en `new1`, que necesitaría ser reconocida por una variedad de aplicaciones de *solvers*, utilidades de pre y post-procesamiento, herramientas de malla, etc. En circunstancias normales, el usuario necesitaría recompilar cada aplicación con `new1` vinculado a ella.

En cambio, existe un mecanismo simple para vincular una o más bibliotecas de objetos compartidos dinámicamente en tiempo de ejecución en OpenFOAM. El usuario simplemente puede agregar la entrada de palabra clave `optional libs` al archivo `controlDict` para un caso e ingresar los nombres completos de las bibliotecas dentro de una lista (como entradas de cadena entre comillas). Por ejemplo, si un usuario quisiera vincular las bibliotecas `new1` y `new2` en tiempo de ejecución, simplemente debería agregar lo siguiente al archivo `controlDict` del caso:

Listing 4.2.3: Configuración de enlaces dinámicos en `controlDict`.

```
libs
(
    "libnew1.so"
    "libnew2.so"
);
```

4.3. Ejecución de aplicaciones

Cada aplicación está diseñada para ejecutarse desde una línea de comandos de terminal, típicamente leyendo y escribiendo un conjunto de archivos de datos asociados con un caso particular. Los archivos de datos para un caso se almacenan en un directorio con el nombre del caso ; el nombre del directorio con la ruta completa se da aquí con el nombre genérico `<caseDir>`.

Para cualquier aplicación, la forma de la entrada de línea de comandos para cualquier aplicación se puede encontrar simplemente ingresando el nombre de la aplicación en la línea de comandos con la opción `-help`, por ejemplo, escribiendo

```
foamRun -help
```

devuelve el uso

```
Usage: foamRun [OPTIONS]
  options:
    -case <dir>     specify alternate case directory, default is cwd
    -fileHandler <handler>
                    override the fileHandler
    -hostRoots <((host1 dir1) .. (hostN dirN))>
                    slave root directories for distributed running
    -libs '("lib1.so" ... "libN.so")'
                    pre-load libraries
    -noFunctionObjects
                    do not execute functionObjects
    -parallel       run in parallel
    -roots <(dir1 .. dirN)>
                    slave root directories for distributed running
    -solver <name>  Solver name
    -srcDoc display source code in browser
    -doc            display application documentation in browser
    -help           print the usage
```

Si la aplicación se ejecuta desde un directorio de caso, operará en ese caso. Alternativamente, la opción `-case <caseDir>` permite especificar el caso directamente para que la aplicación se pueda ejecutar desde cualquier lugar en el sistema de archivos.

Al igual que cualquier ejecutable UNIX/Linux, las aplicaciones se pueden ejecutar como un proceso en segundo plano, es decir, uno que no tiene que completarse antes de que el usuario pueda dar comandos adicionales al `shell`. Si el usuario desea ejecutar el ejemplo `foamRun` como un proceso en segundo plano y enviar el progreso del caso a un archivo de registro, podrían ingresar:

```
foamRun > log &
```

4.4. Ejecución de aplicaciones en paralelo

Esta sección describe cómo ejecutar OpenFOAM en paralelo en procesadores distribuidos. El método de cómputo paralelo utilizado por OpenFOAM se conoce como descomposición de dominio, en el cual la geometría y los campos asociados se dividen en piezas y se asignan a procesadores separados para la solución. El proceso de computación paralela implica: la descomposición de la malla y los campos; ejecutar la aplicación en paralelo; y, postprocesar el caso descompuesto como se describe en las siguientes secciones. La ejecución paralela utiliza la implementación de dominio público `openMPI` de la interfaz de paso de mensajes estándar (MPI) de forma predeterminada, aunque se pueden utilizar otras bibliotecas.

4.4.1. Descomposición de la malla y de los datos iniciales de campos

La malla y los campos se descomponen utilizando la utilidad `decomposePar`. El objetivo subyacente es dividir el dominio con el menor esfuerzo posible pero de manera que garantice una solución económica. La geometría y los campos se dividen según un conjunto de parámetros especificados en un diccionario llamado `decomposeParDict` que debe estar ubicado en el directorio del sistema del caso de interés. Un ejemplo de diccionario `decomposeParDict` se puede copiar en un directorio de sistema del caso utilizando el script `foamGet`.

```
foamGet decomposeParDict
```

Las entradas del diccionario se reproducen a continuación:

```
numberOfSubdomains 8;

 /*
 Main methods are:
 1) Geometric: "simple"; "hierarchical", with ordered sorting, e.g. xyz, yxz
 2) Scotch: "scotch", when running in serial; "ptscotch", running in parallel
 */

 method hierarchical;

 simpleCoeffs
 {
 n (4 2 1); // total must match numberOfSubdomains
 }

 hierarchicalCoeffs
 {
 n (4 2 1); // total must match numberOfSubdomains
 order xyz;
 }
```

El usuario tiene la opción de cuatro métodos de descomposición, especificados por la palabra clave `method` como se describe a continuación.

- `simple`: Descomposición geométrica simple en la que el dominio se divide en piezas por dirección, por ejemplo, 2 piezas en la dirección x, 1 en y, etc.
- `hierarchical`: Descomposición geométrica jerárquica que es igual a `simple` excepto que el usuario especifica el orden en que se realiza la división direccional, por ejemplo, primero en la dirección y, luego en la dirección x, etc.
- `scotch`: Descomposición Scotch que no requiere entrada geométrica del usuario e intenta minimizar el número de límites del procesador. El usuario puede especificar un ponderación para la descomposición entre procesadores, a través de una palabra clave opcional `processorWeights` que puede ser útil en máquinas con diferentes rendimientos entre procesadores. También hay una entrada de palabra clave opcional `strategy` que controla la estrategia de descomposición a través de una cadena compleja suministrada a `Scotch`.

Para cada método, hay un conjunto de coeficientes especificados en un sub-diccionario de `decompositionDict`, llamado `<method>Coeffs` como se muestra en la lista del diccionario. El conjunto completo de entradas de palabras clave en el diccionario `decomposeParDict` se explica a continuación:

- `numberOfSubdomains`: número total de subdominios `N`.
- `method`: método de descomposición, `simple`, jerárquica, `scotch`.
- `n`: para `simple` y jerárquica, número de subdominios en x, y, z (`nx ny nz`)
- `order`: orden de la descomposición jerárquica, `xyz/xzy/yxz`...
- `processorWeights`: opción para scotch: lista de factores de ponderación (`<wt1>...<wtN>`) para la asignación de celdas a procesadores; `<wt1>` es el factor de ponderación para el procesador 1, etc.; los pesos están normalizados, por lo que pueden tomar cualquier rango de valores.

La utilidad decomposePar se ejecuta de la manera habitual escribiendo:

```
1 decomposePar
```

4.4.2. Archivos de entrada/salida en paralelo

Usando la finalización estándar de entrada/salida de archivos, se habrán creado un conjunto de subdirectorios, uno para cada procesador, en el directorio del caso. Los directorios se denominan `processorN` donde $N = 0, 1, \ldots$ representa un número de procesador y contiene un directorio de tiempo, que contiene las descripciones de campo descompuestas, y un directorio `constant/polyMesh` que contiene la descripción de la malla descompuesta.

Si bien esta estructura de archivos está bien organizada, para casos paralelos grandes, genera una gran cantidad de archivos. En simulaciones muy grandes, los usuarios pueden experimentar problemas, incluyendo alcanzar límites en el número de archivos abiertos impuestos por el sistema operativo.

Como alternativa, se introdujo el formato de archivo consolidado en OpenFOAM en el que los datos para cada campo (y malla) descompuesto se consolidan en un solo archivo que se escribe (y lee) en el procesador maestro. Los archivos se almacenan en un solo directorio llamado processors.

La escritura de archivos se puede realizar en varios hilos, lo que permite que la simulación continúe mientras los datos se escriben en el archivo; vea más abajo para detalles. `NFS` (Sistema de Archivos de Red) no es necesario al usar el formato consolidado y, además, existe una opción `masterUncollated` para escribir datos con el formato original sin consolidar sin `NFS`.

Los controles para el manejo de archivos están en las `OptimisationSwitches` del archivo global `etc/controlDict`:

```
OptimisationSwitches { ... //- Parallel IO file handler // uncollated
    (default), collated or masterUncollated fileHandler uncollated; //-
    collated: thread buffer size for queued file writes. // If set to 0 or not
    sufficient for the file size threading is not used. // Default: 2e9
    maxThreadFileBufferSize 2e9; //- masterUncollated: non-blocking buffer size.
    // If the file exceeds this buffer size scheduled transfer is used. //
    Default: 2e9 maxMasterFileBufferSize 2e9; }
```

El fileHandler se puede configurar para una simulación específica de las siguientes maneras:

- anulando las `OptimisationSwitches` globales `fileHandler ...;` en el archivo `controlDict` del caso;
- utilizando el argumento de línea de comandos `-fileHandler` para el *solver*;
- configurando la variable de entorno `$FOAM-FILEHANDLER`.

Una utilidad `foamFormatConvert` permite a los usuarios convertir archivos entre los formatos consolidado y sin consolidar, por ejemplo:

```
mpirun -np 2 foamFormatConvert -parallel -fileHandler uncollated
```

Se proporciona un caso de ejemplo que demuestra los métodos de manejo de archivos en:

```
FOAM_TUTORIALS/heatTransfer/buoyantFoam/iglooWithFridges
```

El manejo de archivos consolidados se ejecuta más rápido con el enhebrado, especialmente en casos grandes. Pero requiere que el soporte de enhebrado esté habilitado en el MPI subyacente. Sin él, la simulación se "bloqueará.º fallará. Para `openMPI`, el soporte de enhebrado no está habilitado de forma predeterminada antes de la versión 2, pero generalmente se activa a partir de la versión 2 en adelante. El usuario puede verificar si openMPI está compilado con soporte de enhebrado mediante el siguiente comando:

```
ompi_info -c | grep -oE "MPI_THREAD_MULTIPLE[^,]*"
```

Cuando se utiliza el manejo de archivos consolidados, se asigna memoria para los datos en el hilo. `maxThreadFileBufferSize` establece el tamaño máximo de memoria que se asigna en bytes. Si los datos superan este tamaño, la escritura no utiliza el enhebrado.

Nota: si el enhebrado no está habilitado en el MPI, debe deshabilitarse para el manejo de archivos consolidados configurando en el archivo global `etc/controlDict`:

```
maxThreadFileBufferSize 0;
```

Cuando se utiliza el manejo de archivos `masterUncollated`, la comunicación MPI no bloqueante requiere un búfer de memoria lo suficientemente grande en el nodo maestro. `maxMasterFileBufferSize` establece el tamaño máximo del búfer. Si los datos superan este tamaño, el sistema utiliza la comunicación programada.

4.4.3. Ejecución de un caso descompuesto

Un caso descompuesto de OpenFOAM se ejecuta en paralelo utilizando la implementación `openMPI` de MPI.

`openMPI` se puede ejecutar en una máquina multiprocesador local de manera muy sencilla, pero cuando se ejecuta en máquinas a través de una red, se debe crear un archivo que contenga los nombres de host de las máquinas. El archivo puede tener cualquier nombre y ubicarse en cualquier ruta. En la siguiente descripción nos referiremos a dicho archivo con el nombre genérico, incluida la ruta completa, `<machines>`.

El archivo `<machines>` contiene los nombres de las máquinas listados una máquina por línea. Los nombres deben corresponder a un nombre de host completamente resuelto en el archivo `/etc/hosts` de la máquina en la que se ejecuta el `openMPI`. La lista debe contener el nombre de la máquina que ejecuta el `openMPI`. Cuando un nodo de máquina contiene más de un procesador, el nombre del nodo puede ir seguido de la entrada `cpu=n` donde `n` es el número de procesadores en los que `openMPI` debería ejecutarse en ese nodo.

Por ejemplo, imaginemos que un usuario desea ejecutar `openMPI` desde la máquina `aaa` en las siguientes máquinas: `aaa`; `bbb`, que tiene 2 procesadores; y `ccc`. El archivo `<machines>` contendría:

```
1 aaa
2 bbb cpu=2
3 ccc
```

Una aplicación es ejecutada en paralelo usando `mpirun` de la siguiente forma:

```
1 mpirun --hostfile <machines> -np <nProcs> <foamExec> <otherArgs> -parallel > log &
```

donde: `<nProcs>` es el número de procesadores; `<foamExec>` es el ejecutable, por ejemplo, `foamRun`; y, la salida se redirige a un archivo llamado `log`. Por ejemplo, si `foamRun` se ejecuta en 4 nodos, especificados en un archivo llamado `machines`, entonces se debe ejecutar el siguiente comando:

```
1 mpirun --hostfile machines -np 4 foamRun -parallel > log &
```

4.4.4. Distribución de datos entre varios discos

Es posible que los archivos de datos deban ser distribuidos si, por ejemplo, solo se utilizan discos locales para mejorar el rendimiento. En este caso, el usuario puede encontrarse con que la ruta raíz al directorio del caso puede diferir entre las máquinas. Entonces, las rutas deben ser especificadas en el diccionario `decomposeParDict` utilizando las palabras clave `distributed` y `roots`. La entrada distributed debería leerse como

```
1 distributed yes;
```

y la entrada roots es una lista de rutas raíz, `<root0>`, `<root1>`, . . ., para cada nodo

```
roots
<nRoots>
(
    "<root0>"
    "<root1>"
    ...
);
```

donde `<nRoots>` es el número de raíces.

Cada uno de los directorios processorN debe ser colocado en el directorio del caso en cada una de las rutas raíz especificadas en el diccionario `decomposeParDict`. El directorio del sistema y los archivos dentro del directorio constante también deben estar presentes en cada directorio de caso. Nota: los archivos en el directorio constante son necesarios, pero el directorio `polyMesh` no lo es.

4.4.5. Postprocesamiento de casos en paralelo

Cuando se realiza el postprocesamiento de casos que se han ejecutado en paralelo, el usuario tiene dos opciones:

- Reconstruir la malla y los datos de campo para recrear el dominio completo y los campos, los cuales pueden ser postprocesados como de costumbre.
- Postprocesar cada segmento del dominio descompuesto de manera individual.

Después de que un caso ha sido ejecutado en paralelo, puede ser reconstruido para postprocesamiento.

El caso se reconstruye fusionando los conjuntos de directorios de tiempo de cada directorio `processorN` en un solo conjunto de directorios de tiempo. La utilidad `reconstructPar` realiza dicha reconstrucción ejecutando el comando:

```
reconstructPar
```

El usuario puede postprocesar casos descompuestos utilizando el posprocesador `paraFoam`. Toda la simulación puede ser postprocesada reconstruyendo el caso o, alternativamente, es posible postprocesar un segmento del dominio descompuesto de manera individual simplemente tratando el directorio individual del procesador como un caso por sí mismo.

4.5. Casos de OpenFOAM

Esta sección trata sobre la estructura de archivos y la organización de casos en OpenFOAM. Normalmente, un usuario asignaría un nombre a un caso, por ejemplo, el caso de tutorial de aerodinámica de una motocicleta se llama simplemente `motorBike`. Este nombre se convierte en el nombre de un directorio en el que se almacenan todos los archivos y subdirectorios del caso.

Cuando se ejecuta una simulación, un directorio de caso puede estar ubicado en cualquier lugar del sistema de archivos del usuario. Sin embargo, se recomienda colocar los casos dentro de un subdirectorio de ejecución del sistema de archivos del usuario, es decir, `$HOME/OpenFOAM/$USER-11`. La variable de entorno `$FOAM-RUN` se establece en `$HOME/OpenFOAM/$USER-11/run` de forma predeterminada y el usuario puede moverse rápidamente a ese directorio ejecutando un alias predefinido, `run`, en la línea de comandos.

Los casos de tutorial que acompañan a la distribución de OpenFOAM proporcionan ejemplos útiles de las estructuras de directorios de casos. Los tutoriales están ubicados en el directorio `$FOAM-TUTORIALS`, al que se puede acceder rápidamente ejecutando el alias `tut` en la línea de comandos.

4.5.1. Estructura de archivos de un caso

La estructura básica de directorios de un caso de OpenFOAM, que contiene el conjunto mínimo de archivos necesarios para ejecutar una aplicación, se muestra en la Figura 4.3 y se describe de la siguiente manera:

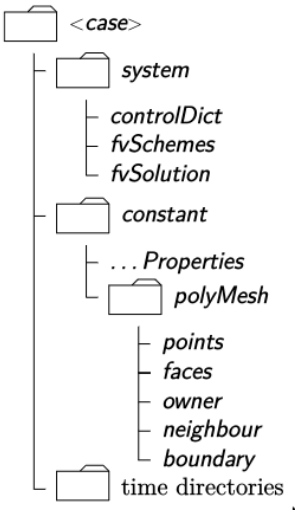

Figura 4.3: Estructura de un caso. (Greenshields y Weller, 2022)

- El directorio `constant` que contiene una descripción completa de la malla del caso en un subdirectorio polyMesh y archivos que especifican propiedades y modelos para la aplicación en cuestión, por ejemplo, `physicalProperties` y `momentumTransport`.
- El directorio `system` para establecer parámetros asociados con el procedimiento de solución en sí. Contiene al menos los siguientes tres archivos: `controlDict` donde se establecen los parámetros de control de ejecución, incluidos el tiempo de inicio/fin, el paso de tiempo y los parámetros para la salida de datos; `fvSchemes` donde se seleccionan los esquemas de discretización utilizados en la solución; y, `fvSolution` donde se establecen los solucionadores de ecuaciones, tolerancias y otros controles de algoritmo para la ejecución.
- Directorios `time` que contienen archivos individuales de datos para campos particulares, por ejemplo, velocidad y presión. Los datos pueden ser: o bien, valores iniciales y condiciones de contorno que el usuario debe especificar para definir el problema; o, resultados escritos en archivos por OpenFOAM.

Los campos siempre deben inicializarse, incluso cuando la solución no lo requiera estrictamente, como en problemas de estado estacionario. El nombre de cada directorio de tiempo se basa en el tiempo simulado en el que se escriben los datos. Dado que generalmente comenzamos nuestras simulaciones en el tiempo $t = 0$, las condiciones iniciales generalmente se almacenan en un directorio llamado `0`. Por ejemplo, en el tutorial de `motorBike`, el campo de velocidad `U` y el campo de presión `p` se inicializan a partir de los archivos `0/U` y `0/p` respectivamente.

4.5.2. Formato de archivos de entrada/salida

OpenFOAM necesita leer una variedad de estructuras de datos como cadenas, palabras, escalares, vectores, tensores, listas y campos. El formato de entrada/salida (E/S) de los archivos es extremadamente flexible, siguiendo un conjunto consistente de reglas que hacen que los archivos sean fáciles de interpretar. El formato de archivo de OpenFOAM se describe en las siguientes secciones.

El formato se asemeja al código `C++`, siguiendo los principios generales a continuación:

- Los archivos tienen formato libre, sin ningún significado particular asignado a ninguna columna y sin necesidad de indicar la continuación a través de líneas.

- Las líneas no tienen un significado particular excepto para un delimitador de comentario `//` que hace que OpenFOAM ignore cualquier texto que le siga hasta el final de la línea.
- Un comentario que abarca varias líneas se realiza encerrando el texto entre delimitadores `/*` y `*/`.

4.5.2.1. Diccionarios

OpenFOAM principalmente utiliza diccionarios para especificar datos, en los cuales las entradas de datos se recuperan mediante palabras clave. Cada entrada de palabra clave sigue el formato general, comenzando con la palabra clave y terminando en un punto y coma (`;`).

```
<palabraClave> <entradaDeDatos1> ... <entradaDeDatosN>;
```

Muchas entradas incluyen solo una entrada de datos, como se muestra a continuación.

```
<palabraClave> <entradaDeDatos>;
```

La mayoría de los archivos de datos, por ejemplo, `controlDict`, son ellos mismos diccionarios ya que contienen una serie de entradas de palabras clave. Cualquier diccionario puede contener uno o más subdiccionarios, generalmente indicados por un nombre de diccionario y sus entradas de palabras clave contenidas entre llaves de la siguiente manera.

```
<nombreDiccionario>
  {
    ... entradas de palabras clave ...
  }
```

Los subdiccionarios pueden estar anidados dentro de otros, como se muestra en el Código 4.5.1. El extracto, de un archivo de diccionario `fvSolution`, contiene dos diccionarios, `solvers` y `PIMPLE`. El diccionario `solvers` contiene subdiccionarios anidados para diferentes ecuaciones matriciales basadas en diferentes variables de solución, por ejemplo, `p`, `U` y `k`.

4.5.2.2. El encabezado de archivos

Todos los archivos de datos que son leídos y escritos por OpenFOAM comienzan con un diccionario llamado `FoamFile` que contiene un conjunto estándar de entradas de palabras clave, enumeradas a continuación:

- `version`: versión del formato de E/S, opcional, por defecto `2.0`;
- `format`: formato de datos, ascii o binario,
- `class`: clase relacionada con los datos, ya sea un diccionario o un campo, por ejemplo, `volVectorField`;
- `object`: nombre del archivo, por ejemplo, `controlDict` (obligatorio, pero no utilizado);
- `location`: ruta al archivo (opcional).

Un ejemplo de encabezado para un archivo `controlDict` se muestra a continuación.

```
FoamFile
  {
    format ascii;
    class dictionary;
    location "system";
    object controlDict;
  }
```

Listing 4.5.1: Ejemplo de un diccionario con sub-diccionarios.

```
solvers
{
  p
  {
    solver GAMG;
    tolerance 1e-7;
    relTol 0.01;

    smoother DICGaussSeidel;
  }

  pFinal
  {
    p;
    relTol 0;
  }

  "(U|k|epsilon)"
  {
    solver smoothSolver;
    smoother symGaussSeidel;
    tolerance 1e-05;
    relTol 0.1;
  }

  "(U|k|epsilon)Final"
  {
    U;
    relTol 0;
  }
}

PIMPLE
{
  nNonOrthogonalCorrectors 0;
  nCorrectors 2;
}
```

4.5.2.3. Listas

Las aplicaciones de OpenFOAM contienen listas, por ejemplo, una lista de coordenadas de vértices para una descripción de malla. Las listas se encuentran comúnmente en la E/S y tienen un formato propio en el que las entradas están contenidas entre paréntesis redondos. Cuando un usuario especifica una lista en un archivo

de entrada, por ejemplo, la lista de vértices en un archivo `blockMeshDict`, simplemente incluye la palabra clave `vertices` y los datos entre paréntesis, por ejemplo,

```
vertices
  (
  ... entradas ...
  );
```

Cuando OpenFOAM escribe una lista, invariablemente la precede con el número de elementos en la lista. Por ejemplo, el archivo points para la malla en el caso `pizDailySteady` contiene la siguiente lista (abreviada), donde `25012` denota el número de puntos de vértice en la malla.

```
25012
  (
    (-0.0206 0 -0.0005)
    (-0.01901716308 0 -0.0005)
    ... entradas ...
  );
```

En algunos casos, cuando OpenFOAM escribe una lista, la precede aún más con el nombre de clase de la lista. Por ejemplo, la entrada inGroups en un archivo `boundary` de una malla contiene una lista donde cada nombre de grupo es una palabra. La entrada para el *patch* `lowerWall` del caso `pizDailySteady` se muestra a continuación, indicando la clase `List<word>` con un solo elemento.

```
lowerWall
  {
    type wall;
    inGroups List<word> 1(wall); // Note!
    nFaces 250;
    startFace 24480;
  }
```

4.5.2.4. Escalares, vectores y tensores

Un escalar es un solo número representado como tal en un archivo de datos. Un vector contiene tres valores, expresados usando el formato de lista simple, de modo que el vector `(1.0, 1.1, 1.2)` se escribe:

```
(1.0 1.1 1.2)
```

En OpenFOAM, un tensor contiene nueve elementos, de modo que el tensor identidad se puede escribir:

```
(1 0 0 0 1 0 0 0 1)
```

El usuario puede escribir la entrada en varias líneas para dar la .ªpariencia"de un tensor como una entidad 3 $\times$ 3.

```
(
1 0 0
0 1 0
0 0 1
)
```

4.5.2.5. Unidades dimensionales

En la mecánica de medios continuos, las propiedades se representan en unidades elegidas, por ejemplo, la masa en kilogramos (kg), el volumen en metros cúbicos (m^3), y la presión en Pascales (kg m^{-1} s^{-2}). Las operaciones algebraicas deben realizarse en estas propiedades utilizando unidades de medida consistentes; en particular, la adición, sustracción e igualdad solo tienen sentido físico para propiedades con las mismas unidades dimensionales. Como salvaguardia contra la implementación de una operación sin sentido, OpenFOAM adjunta dimensiones a los datos de campo y propiedades físicas y realiza verificación de dimensiones en cualquier operación.

Las dimensiones se describen mediante la clase dimensionSet, que tiene su propia sintaxis única de E/S utilizando corchetes cuadrados, por ejemplo,

```
(0 2 -1 0 0 0 0)
```

donde cada uno de los valores corresponde a la potencia de cada una de las unidades base de medida listadas en secuencia a continuación:

1. masa, por ejemplo, kilogramo (kg), libra-masa (lbm);
2. longitud, por ejemplo, metro (m), pie (ft);
3. tiempo, por ejemplo, segundo (s);
4. temperatura, por ejemplo, Kelvin (K), grado Rankine (°R);
5. cantidad, por ejemplo, mol (mol);
6. corriente, por ejemplo, amperio (A);
7. intensidad luminosa, por ejemplo, candela (cd).

La lista incluye unidades base para el Sistema Internacional (SI) y el Sistema de Medidas Estadounidense (USCS), pero OpenFOAM se puede usar con cualquier sistema de unidades. Todo lo que se requiere es que los datos de entrada sean correctos para el conjunto de unidades elegido. Los datos de entrada pueden incluir constantes físicas, por ejemplo, la Constante Universal de los Gases R, que se especifican en un subdiccionario `DimensionedConstant` del archivo principal `controlDict` de la instalación de OpenFOAM (`$WM-PROJECT-DIR/etc/controlDict`). Por defecto, estas constantes se establecen en unidades del SI. Aquellos que deseen usar el USCS o cualquier otro sistema de unidades deben modificar estas constantes según su conjunto de unidades elegido.

Las propiedades físicas suelen especificarse con sus dimensiones asociadas. A menudo se describen mediante la clase `dimensioned`, que incluye tres componentes: un nombre de palabra; un `dimensionSet` y un valor (escalar, vector, etc.).

La E/S para una entrada dimensionada puede incluir los tres componentes, por ejemplo:

```
nu nu [0 2 -1 0 0 0 0] 1e-5;
```

Nótese que el primer `nu` es la palabra clave; el segundo `nu` es el nombre de palabra almacenado en la clase palabra; la siguiente entrada es el `dimensionSet` y la entrada final es el valor escalar. Sin embargo, muy a

menudo, la palabra y el `dimensionSet` se especifican en el código con valores predeterminados, por lo que pueden omitirse de la E/S como se muestra a continuación.

```
nu 1e-5;
```

4.5.2.6. Campos

Los archivos de campo, por ejemplo, `U` y `p`, que se leen y escriben en los directorios de tiempo, poseen su propia E/S personalizada con las siguientes tres entradas clave.

- `dimensions`: las dimensiones del campo, por ejemplo, `(1 1 -2 0 0 0 0)`.
- `internalField`: valores dentro del campo interno, por ejemplo, dentro de cada celda de una malla.
- `boundaryField`: condición (tipo) y datos para cada parche del límite de la malla.

El `internalField` se puede especificar de dos maneras. Primero, cuando el usuario edita un archivo de campo para inicializarlo, generalmente especifica un valor único en todos los elementos, es decir, las celdas (o caras, puntos, según el tipo de campo) de la malla. Un solo valor de `0` se denota por la palabra clave `uniform` como se muestra a continuación.

```
internalField uniform 0;
```

Cuando se escriben los resultados, los campos generalmente no pueden representarse por un valor único. La salida utiliza la palabra clave `nonuniform`, seguida de una lista adecuada de valores. El ejemplo abreviado a continuación es de un archivo de salida `p` para una malla de `12225` celdas.

```
internalField nonuniform List<scalar>
12225
(
    -4.92806
    -5.42676
    ...
);
```

El `boundaryField` es un diccionario que contiene un conjunto de entradas correspondientes a cada parche enumerado en el archivo de límites en el directorio `polyMesh`. Cada entrada es un sub-diccionario que contiene una lista de entradas de palabras clave. La entrada obligatoria, `type`, describe la condición del campo de parche especificada para el campo. Las entradas restantes corresponden al tipo de condición del campo de parche seleccionado y típicamente pueden incluir datos de campo que especifican condiciones iniciales en las caras del parche.

Listing 4.5.2: Ejemplo de entradas para un diccionario del campo de velocidades *U*

```
dimensions [0 1 -1 0 0 0 0];

internalField uniform (0 0 0);

boundaryField
{
  inlet
  {
    type fixedValue;
    value uniform (10 0 0);
  }

  outlet
  {
    type zeroGradient;
  }

  upperWall
  {
    type noSlip;
  }

  lowerWall
  {
    type noSlip;
  }

  frontAndBack
  {
    type empty;
  }
}
```

4.5.2.7. Control de tiempo y datos de entrada/salida

Los *solvers* de OpenFOAM comienzan todas las ejecuciones configurando una base de datos. La base de datos controla la E/S y, dado que la salida de datos generalmente se solicita en intervalos de tiempo durante la ejecución, el tiempo es una parte inextricable de la base de datos. El diccionario `controlDict` establece los parámetros de entrada esenciales para la creación de la base de datos. Las entradas de palabras clave en `controlDict` se enumeran en las secciones siguientes. Solo las entradas de control de tiempo y `writeInterval` son obligatorias, y la base de datos utiliza valores predeterminados para cualquiera de las entradas opcionales que se omiten. A continuación, se presentan ejemplos de entradas de un diccionario `controlDict`:

Listing 4.5.3: Ejemplo de entradas para un diccionario `controlDict`

```
application foamRun;

solver incompressibleFluid;

startFrom latestTime;

startTime 0;

stopAt endTime;

endTime 0.3;

deltaT 0.0001;

writeControl adjustableRunTime;

writeInterval 0.01;

purgeWrite 0;

writeFormat ascii;

writePrecision 6;

writeCompression off;

timeFormat general;

timePrecision 6;

runTimeModifiable yes;

adjustTimeStep yes;

maxCo 5;

functions
{
  #includeFunc patchAverage(patch=inlet, fields=(p U))
}
```

4.6. Esquemas numéricos en OpenFOAM

El diccionario `fvSchemes` en el directorio `system` establece los esquemas numéricos para términos, como derivadas en ecuaciones diferenciales parciales, que se calculan durante una simulación. Esta sección describe cómo especificar los esquemas en el diccionario `fvSchemes`.

El objetivo de `fvSchemes` es proporcionar una elección sin restricciones de esquemas al usuario para todo, desde derivadas, por ejemplo, el gradiente ∇, hasta interpolaciones de valores de un conjunto de puntos a otro. OpenFOAM utiliza el método de Volumenes Finitos, por lo que las derivadas espaciales se basan en la integración Gaussiana que suma valores en las caras de las celdas, que deben interpolarse desde los centros de las celdas. El usuario tiene una amplia gama de opciones para los esquemas de interpolación, con ciertos

esquemas diseñados específicamente para términos de derivadas particulares, especialmente los términos de divergencia de advección, $\nabla\cdot$.

El conjunto de términos, para los cuales se deben especificar esquemas numéricos, se subdividen dentro del diccionario `fvSchemes` en las siguientes categorías, usando ψ como una variable de campo de ejemplo.

- `timeScheme`: primeras y segundas derivadas temporales, por ejemplo, $\partial\psi/\partial t$, $\partial^2\psi/\partial t^2$;
- `gradSchemes`: gradiente $\nabla\psi$;
- `divSchemes`: divergencia $\nabla\cdot\psi$;
- `laplacianSchemes`: Laplaciano $\nabla^2\psi$;
- `interpolationSchemes`: interpolaciones de celda a cara de valores;
- `snGradSchemes`: componente del gradiente normal a una cara de celda;
- `wallDist`: cálculo de distancia a la pared, cuando sea necesario.

Cada palabra clave es el nombre de un subdiccionario que contiene términos de un tipo particular, por ejemplo, `gradSchemes` contiene todos los términos de derivadas de gradiente como `grad(p)` (que representa ∇p). Se muestran más ejemplos en el extracto de un diccionario `fvSchemes` en el Código 4.7.1.

El ejemplo muestra `fvSchemes` con 6 subdiccionarios de `...Schemes`, cada uno conteniendo entradas de palabras clave que incluyen: una entrada predeterminada; otras entradas para el término específico especificado, por ejemplo, `div(U, k)` para $\nabla\cdot(\mathbf{U}k)$. Si se especifica un esquema predeterminado en un subdiccionario específico de `...Schemes`, se asigna a todos los términos a los que se refiere el subdiccionario, por ejemplo, especificar un predeterminado en `gradSchemes` establece el esquema para todos los términos de gradiente en la aplicación, por ejemplo, ∇p, $\nabla\mathbf{U}$. Con un predeterminado especificado, los términos específicos no son necesarios en ese subdiccionario, es decir, las entradas para `grad(p)`, `grad(U)` se omiten en este ejemplo. Sin embargo, especificar un término particular anulará el esquema predeterminado.

El usuario puede especificar que no hay un esquema predeterminado con la entrada `none`, como en `divSchemes` en el ejemplo anterior. El usuario entonces está obligado a especificar todos los términos en ese subdiccionario individualmente. Establecer el predeterminado en `none` garantiza que el usuario especifique todos los términos individualmente, lo que es común para `divSchemes` que requiere una configuración precisa.

OpenFOAM incluye una vasta cantidad de esquemas de discretización, de los cuales solo algunos se recomiendan típicamente para aplicaciones de ingeniería del mundo real. El usuario puede obtener ayuda con la selección del esquema interrogando los casos de tutorial para ver ejemplos de configuraciones de esquemas. Deberían examinar los esquemas utilizados en casos relevantes, por ejemplo, para ejecutar una simulación de gran escala (LES), buscar esquemas utilizados en tutoriales que ejecutan LES. Además, `foamSearch` es una herramienta útil para listar los esquemas utilizados en todos los tutoriales. Por ejemplo, para imprimir todas las entradas predeterminadas para ddtSchemes para casos en el directorio `$FOAM-TUTORIALS`, el usuario puede escribir:

```
foamSearch $FOAM_TUTORIALS fvSchemes ddtSchemes/default
```

lo que devuelve:

```
default backward;
default CrankNicolson 0.9;
default Euler;
default localEuler;
default none;
default steadyState;
```

Los esquemas enumerados usando foamSearch se describen en las secciones siguientes.

Listing 4.6.1: Ejemplo de entradas para un diccionario `fvSchemes`

```
ddtSchemes
{
  default Euler;
}

gradSchemes
{
  default Gauss linear;
}

divSchemes
{
  default none;

  div(phi,U) Gauss linearUpwind grad(U);
  div(phi,k) Gauss upwind;
  div(phi,epsilon) Gauss upwind;
  div(phi,R) Gauss upwind;
  div(R) Gauss linear;
  div(phi,nuTilda) Gauss upwind;

  div((nuEff*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
  default Gauss linear corrected;
}

interpolationSchemes
{
  default linear;
}

snGradSchemes
{
  default corrected;
}
```

4.6.1. Esquemas temporales

Los términos de primera derivada temporal ($\partial/\partial t$) se especifican en el subdiccionario `ddtSchemes`. Los esquemas de discretización para cada término pueden seleccionarse entre los siguientes:

- `steadyState`: establece las derivadas temporales en cero.
- `Euler`: transitorio, implícito de primer orden, acotado.
- `backward`: transitorio, implícito de segundo orden, potencialmente ilimitado.
- `CrankNicolson`: transitorio, implícito de segundo orden, acotado; requiere un coeficiente de descentramiento ψ donde:

$$\psi = \begin{cases} 1 \ , & \text{corresponde a } \texttt{CrackNicolson} \text{ puro;} \\ 0 \ , & \text{corresponde a } \texttt{Euler}. \end{cases} \tag{4.2}$$

  generalmente se usa $\psi = 0{,}9$ para estabilizar el esquema para problemas de ingeniería práctica.

- `localEuler`: pseudo-transitorio para acelerar una solución a estado estacionario utilizando pasos de tiempo locales; implícito de primer orden.

Cualquier término de segunda derivada temporal ($\partial^2/\partial t^2$) se especifica en el subdiccionario `d2dt2Schemes`. Solo está disponible el esquema de Euler para `d2dt2Schemes`.

4.6.2. Esquemas de gradiente

El subdiccionario `gradSchemes` contiene términos de gradiente. El esquema de discretización predeterminado que se utiliza principalmente para los términos de gradiente es:

```
default Gauus linear;
```

Esta entrada especifica la discretización de volumen finito estándar con integración Gaussiana, que requiere la interpolación de valores desde los centros de celda hasta los centros de cara. El esquema de interpolación se especifica como `linear`, lo que significa interpolación lineal o diferenciación central.

En algunos casos de tutoriales, especialmente aquellos que involucran mallas de calidad inferior, la discretización de términos de gradiente específicos se anula para mejorar la limitación y la estabilidad. Los términos que se anulan en esos casos son el gradiente de velocidad

```
grad(U) cellLimited Gauss linear 1;
```

y, con menos frecuencia, el gradiente de campos de turbulencia, por ejemplo,

```
grad(k) cellLimited Gauss linear 1;
grad(epsilon) cellLimited Gauss linear 1;
```

Utilizan el esquema `cellLimited` que limita el gradiente de tal manera que cuando los valores de la celda se extrapolan a las caras utilizando el gradiente calculado, los valores de la cara no caen fuera de los límites de los valores en las celdas circundantes. Se especifica un coeficiente de limitación después del esquema subyacente, donde `1` garantiza la limitación y `0` no aplica ninguna limitación; `1` se utiliza invariablemente.

Otros esquemas que rara vez se utilizan son los siguientes.

- `leastSquares`: un cálculo de distancia de segundo orden por mínimos cuadrados utilizando todas las celdas vecinas.
- `Gauss cubic`: esquema de tercer orden que aparece en ejemplos de `solidDisplacement` y `dnsFoam`.

4.6.3. Esquemas de divergencia

El sub-diccionario `divSchemes` contiene términos de divergencia, es decir, términos de la forma $\nabla \cdot \ldots$, excluyendo términos laplacianos (de la forma $\nabla^2 \ldots$). Esto incluye tanto términos de advección, por ejemplo, $\nabla \cdot (\mathbf{U}k)$, donde la velocidad $\mathbf{U}$ proporciona el flujo advectivo, como otros términos, que son a menudo difusivos por naturaleza, por ejemplo, $\nabla \cdot \nu(\nabla \mathbf{U})T$.

El hecho de que términos fundamentalmente diferentes residan en un sub-diccionario significa que el esquema predeterminado generalmente se establece en `none` en `divSchemes`. Los términos no advectivos generalmente utilizan la integración Gaussiana con interpolación lineal, por ejemplo,

```
div(U) Gauss linear;
```

El tratamiento de los términos advectivos es uno de los principales desafíos en la numeración de CFD, por lo que las opciones son más extensas. El identificador de palabras clave para los términos advectivos suele tener la forma `div(phi,...)`, donde `phi` denota el flujo (volumétrico) de velocidad en las caras de las celdas para flujos de densidad constante y el flujo de masa para flujos compresibles, por ejemplo, `div(phi,U)` para la advección de velocidad, `div(phi,e)` para la advección de energía interna, etc. Para la advección de la velocidad, el usuario puede ejecutar el script `foamSearch` para extraer la palabra clave `div(phi,U)` de todos los tutoriales.

```
foamSearch $FOAM_TUTORIALS fvSchemes "divSchemes/div(phi,U)"
```

Los esquemas se basan en la integración Gaussiana, utilizando el flujo `phi` y el campo advectado que se interpola en las caras de las celdas por uno de una selección de esquemas, por ejemplo, `linear`, `linearUpwind`, etc. Hay una variante limitada de la discretización, discutida más adelante. Ignorando los esquemas `V` (con palabras clave que terminan en `V`), y esquemas raramente utilizados como Gauss cúbico y `vanLeerV`, los esquemas de interpolación utilizados en los tutoriales son los siguientes.

- `lineal`: de segundo orden, no acotado.
- `linearUpwind`: de segundo orden, sesgado hacia arriba, no acotado (pero mucho menos que lineal), que requiere que se especifique la discretización del gradiente de velocidad.
- `LUST`: esquema mezclado 75 % `linear` / 25 % `linearUpwind`, que requiere que se especifique la discretización del gradiente de velocidad.
- `limitedLinear`: esquema lineal que se limita hacia arriba en regiones de gradiente que cambian rápidamente; requiere un coeficiente, donde `1` es la limitación más fuerte, tendiendo hacia lineal a medida que el coeficiente tiende a `0`.
- `upwind`: de primer orden acotado, generalmente demasiado inexacto para la velocidad pero se usa más a menudo para el transporte de campos escalares.

La sintaxis de ejemplo para estos esquemas es la siguiente.

```
div(phi,U) Gauss linear;
div(phi,U) Gauss linearUpwind grad(U);
div(phi,U) Gauss LUST grad(U);
div(phi,U) Gauss LUST unlimitedGrad(U);
div(phi,U) Gauss limitedLinear 1;
div(phi,U) Gauss upwind;
```

Los esquemas `V` son versiones especializadas de esquemas diseñados para campos vectoriales. Difieren de los esquemas convencionales al calcular un único limitador que se aplica a todas las componentes de los vectores, en lugar de calcular limitadores separados para cada componente del vector. El único limitador de los esquemas `V` se calcula en función de la dirección del gradiente que cambia más rápidamente, lo que resulta en el cálculo del limitador más fuerte que es más estable pero, argumentablemente, menos preciso. La sintaxis de ejemplo es la siguiente.

```
div(phi,U) Gauss limitedLinearV 1;
div(phi,U) Gauss linearUpwindV grad(U);
```

Las variantes `bounded` de los esquemas se relacionan con el tratamiento de la derivada material del tiempo que puede expresarse en términos de una derivada temporal espacial y convección, por ejemplo, para el campo e en un flujo incompresible

$$\frac{De}{Dt} = \frac{\partial e}{\partial t} + \mathbf{U} \cdot \nabla e = \frac{\partial e}{\partial t} + \nabla \cdot (\mathbf{U}e) - (\nabla \cdot \mathbf{U})\, e. \tag{4.3}$$

Para la solución numérica de flujos incompresibles, $\nabla \cdot \mathbf{U} = 0$ en la convergencia, momento en el que el tercer término del lado derecho es cero. Antes de que se alcance la convergencia, sin embargo, $\nabla \cdot \mathbf{U} \neq 0$ y en algunas circunstancias, especialmente en simulaciones en estado estacionario, es mejor incluir el tercer término dentro de una solución numérica para ayudar a mantener la limitación de la variable de solución y promover una mejor convergencia. La variante acotada del esquema Gaussiano proporciona esto, incluyendo automáticamente la discretización del tercer término con el término de advección. La sintaxis de ejemplo es la siguiente, como se ve en los archivos `fvSchemes` para casos en estado estacionario.

```
1 div(phi,U) bounded Gauss limitedLinearV 1;
2 div(phi,U) bounded Gauss linearUpwindV grad(U);
```

Los esquemas utilizados para la advección de campos escalares son similares a los utilizados para la advección de velocidad, aunque en general se hace mayor énfasis en la limitación que en la precisión al seleccionar los esquemas. Por ejemplo, una búsqueda de esquemas para la advección de la energía interna (e) revela lo siguiente.

```
1 foamSearch $FOAM_TUTORIALS fvSchemes "divSchemes/div(phi,e)"
```

```
1 div(phi,e) bounded Gauss upwind;
2 div(phi,e) Gauss limitedLinear 1;
3 div(phi,e) Gauss linearUpwind limited;
4 div(phi,e) Gauss LUST grad(e);
5 div(phi,e) Gauss upwind;
6 div(phi,e) Gauss vanAlbada;
```

En comparación con la advección de velocidad, no hay casos configurados para usar `linear`. Los esquemas `limitedLinear` y `upwind` se utilizan comúnmente, con la aparición adicional de `vanLeer`, otro esquema limitado, con limitaciones menos fuertes que `limitedLinear`.

Existen versiones especializadas de los esquemas limitados para campos escalares que comúnmente están acotados entre `0` y `1`, por ejemplo, la variable de regresión de velocidad de llama `laminar b`. Una búsqueda de la discretización utilizada para la advección en la ecuación de transporte de llama laminar arroja:

```
1 div(phiSt,b) Gauss limitedLinear01 1;
```

El esquema subyacente es `limitedLinear`, especializado para una limitación más fuerte entre `0` y `1` agregando `01` al nombre del esquema.

El mecanismo de selección multivariada también existe para agrupar varios términos de ecuaciones juntos y aplicar los mismos limitadores a todos los términos, utilizando el limitador más fuerte calculado para todos los términos. Un buen ejemplo de esto se encuentra en un conjunto de ecuaciones de transporte de masa para especies de fluidos, donde es una buena práctica aplicar la misma discretización a todas las ecuaciones para una consistencia. El siguiente ejemplo proviene del tutorial `smallPoolFire3D` en `$FOAM-TUTORIALS/multicompon/entFluid`, en el que se incluye la ecuación para la entalpía `h` con las ecuaciones de transporte de masa de las especies en el cálculo de un solo limitador.

```
div(phi,Yi_h) Gauss multivariateSelection
{
  O2 limitedLinear01 1;
  CH4 limitedLinear01 1;
  N2 limitedLinear01 1;
  H2O limitedLinear01 1;
  CO2 limitedLinear01 1;
  h limitedLinear 1 ;
}
```

4.6.4. Esquemas de Laplacianos

El sub-diccionario `laplacianSchemes` contiene términos laplacianos. Un término laplaciano típico es $\nabla^2(\nu\mathbf{U})$, el término de difusión en las ecuaciones de momento, que corresponde a la palabra clave `laplacian(nu,U)` en `laplacianSchemes`. El esquema `Gauss` es la única opción de discretización y requiere una selección tanto de un esquema de interpolación para el coeficiente de difusión, es decir, ν en nuestro ejemplo, como de un esquema de gradiente normal superficial, es decir, $\nabla\mathbf{U}$. Para resumir, las entradas requeridas son:

```
Gauss <esquemaDeInterpolación> <esquemaDeGradienteNormal>
```

El usuario puede buscar el esquema predeterminado para `laplacianSchemes` en todos los casos en el directorio `$FOAM-TUTORIALS`.

```
foamSearch $FOAM_TUTORIALS fvSchemes laplacianSchemes/default
```

Revela las siguientes entradas.

```
default Gauss linear corrected;
default Gauss linear limited corrected 0.33;
default Gauss linear limited corrected 0.5;
default Gauss linear orthogonal;
default Gauss linear uncorrected;
```

En todos los casos, se utiliza el esquema de interpolación lineal para la interpolación de la difusividad.

El sub-diccionario `interpolationSchemes` contiene términos que son interpolaciones de valores típicamente desde los centros de las celdas hasta los centros de las caras, utilizados principalmente en la interpolación de la velocidad a los centros de las caras para el cálculo del flujo `phi`. Hay numerosos esquemas de interpolación en OpenFOAM, pero una búsqueda del esquema predeterminado en todos los casos de tutorial revela que se utiliza la interpolación lineal en casi todos los casos, excepto en un ejemplo de análisis de estrés que utiliza interpolación cúbica.

4.7. Solución y algoritmos de control

Una vez que se construye una ecuación matricial de acuerdo con los esquemas en la sección anterior, se aplica un solucionador lineal. Un conjunto de ecuaciones también se enmarca dentro de un algoritmo que contiene bucles y controles.

Los controles para algoritmos y solucionadores se encuentran en el diccionario `fvSolution` en el directorio del sistema. A continuación se muestra un conjunto de entradas de ejemplo del diccionario `fvSolution` para un caso que utiliza el *solver* modular `incompressibleFluid`, como se ve en el Código 4.7.1.

fvSolution contiene un conjunto de subdiccionarios, descritos en el resto de esta sección, que incluyen: solvers; relaxationFactors; y, SIMPLE para casos de estado estacionario o PIMPLE para casos transitorios o pseudo-transitorios.

Listing 4.7.1: Ejemplo de entradas para un diccionario `fvSolution`

```
solvers
{
  p
  {
    solver GAMG;
    tolerance 1e-7;
    relTol 0.01;

    smoother DICGaussSeidel;
  }

  pFinal
  {
    p;
    relTol 0;
  }

  "(U|k|epsilon)"
  {
    solver smoothSolver;
    smoother symGaussSeidel;
    tolerance 1e-05;
    relTol 0.1;
  }

  "(U|k|epsilon)Final"
  {
    U;
    relTol 0;
  }
}

PIMPLE
{
  nNonOrthogonalCorrectors 0;
  nCorrectors 2;
}
```

4.7.1. Control del *solver* lineal

El primer subdiccionario en nuestro ejemplo es *solvers*. Especifica cada solucionador lineal que se utiliza para cada ecuación discretizada; aquí, el término solucionador lineal se refiere al método de procesamiento numérico para resolver una ecuación matricial, en oposición a un solucionador modular, como `incompressibleFluid`, que describe el conjunto completo de ecuaciones y algoritmos para resolver un problema particular. El término 'solucionador lineal' se abrevia como 'solucionador' en gran parte de lo que sigue; esperemos que el contexto del término evite cualquier ambigüedad.

La sintaxis para cada entrada dentro de los solucionadores comienza con una palabra clave que representa la variable que se resuelve en la ecuación particular. Por ejemplo, `incompressibleFluid` resuelve ecuaciones para la presión p, la velocidad $\mathbf{U}$ y a menudo campos de turbulencia, de ahí las entradas para `p`, `U`, `k` y `epsilon`. La palabra clave introduce un subdiccionario que contiene el tipo de solucionador y los parámetros que utiliza el solucionador. El solucionador se selecciona mediante la palabra clave del `solver` de las opciones que se enumeran a continuación. Los parámetros, incluida la tolerancia, `relTol`, precondicionador, etc. se describen en las secciones siguientes.

- `PCG/PBiCGStab`: Gradiente conjugado precondicionado (bi-)estabilizado, tanto para matrices simétricas como asimétricas;
- `PCG/PBiCG`: Gradiente conjugado precondicionado (bi-)sin estabilizar, con `PCG` para matrices simétricas, `PBiCG` para matrices asimétricas;
- `smoothSolver:` solucionador que utiliza un suavizador;
- `GAMG`: malla multi-grid algebraico-geométrica generalizada;
- `diagonal`: solucionador diagonal para sistemas explícitos.

Los solucionadores distinguen entre matrices simétricas y asimétricas. La simetría de la matriz depende de los términos de la ecuación que se está resolviendo, por ejemplo, las derivadas temporales y los términos laplacianos forman coeficientes de una matriz simétrica, mientras que una derivada advectiva introduce asimetría. Si el usuario especifica un solucionador simétrico para una matriz asimétrica, o viceversa, se escribirá un mensaje de error para aconsejar al usuario en consecuencia, por ejemplo.

```
--> FOAM FATAL IO ERROR : Unknown asymmetric matrix solver PCG
Valid asymmetric matrix solvers are :
4
(
GAMG
PBiCG
PBiCGStab
smoothSolver
)
```

4.7.2. Tolerancia de la solución

El método de volumen finito generalmente resuelve ecuaciones de manera segregada y desacoplada, lo que significa que cada ecuación matricial resuelve solo una variable. Las matrices son en consecuencia dispersas, lo que significa que predominantemente incluyen coeficientes de `0`. Los solucionadores generalmente son iterativos, es decir, se basan en reducir el residuo de la ecuación en soluciones sucesivas. El residuo es ostensiblemente una medida del error en la solución, por lo que cuanto menor sea, más precisa será la solución. Más precisamente, el residuo se evalúa sustituyendo la solución actual en la ecuación y tomando la magnitud de la diferencia entre los lados izquierdo y derecho; también se normaliza para hacerlo independiente de la escala del problema que se está analizando.

Antes de resolver una ecuación para un campo particular, se evalúa el residuo inicial en función de los valores actuales del campo. Después de cada iteración del solucionador, se reevalúa el residuo.

El solucionador se detiene si se alcanza alguna de las siguientes condiciones:

- el residuo cae por debajo de la tolerancia del solucionador, `tolerance`;
- la relación entre los residuos actual e inicial cae por debajo de la tolerancia relativa del solucionador, `relTol`;
- el número de iteraciones excede un número máximo de iteraciones, `maxIter`.

La tolerancia del solucionador debería representar el nivel en el que el residuo es lo suficientemente pequeño como para que la solución pueda considerarse suficientemente precisa. La tolerancia relativa del solucionador limita la mejora relativa desde la solución inicial hasta la final. En simulaciones transitorias, es habitual establecer la tolerancia relativa del solucionador en `0` para forzar que la solución converja a la tolerancia del solucionador en cada paso de tiempo. Las tolerancias, `tolerance` y `relTol`, deben especificarse en los diccionarios para todos los solucionadores; `maxIter` es opcional y por defecto tiene un valor de `1000`.

Las ecuaciones a menudo se resuelven múltiples veces dentro de un paso de solución o paso de tiempo. Por ejemplo, al utilizar el algoritmo `PIMPLE`, se resuelve una ecuación de presión según el número especificado por `nCorrectors`. Cuando esto ocurre, el solucionador se configura muy a menudo para usar diferentes ajustes al resolver la ecuación particular por última vez, especificada por una palabra clave que agrega `Final` al nombre del campo. Por ejemplo, en el ejemplo transitorio `pitzDaily` para el solucionador `incompressibleFluid`, los ajustes del solucionador para la presión son los siguientes.

```
p
{
  solver GAMG;
  tolerance 1e-07;
  relTol 0.01;
  smoother DICGaussSeidel;
}
pFinal
{
  $p;
  relTol 0;
}
```

Si se especifica que el caso debe resolver la presión 4 veces dentro de un paso de tiempo, entonces las primeras 3 soluciones usarían los ajustes para `p` con `relTol` de `0.01`, para que el costo de resolver cada ecuación sea relativamente bajo. Solo cuando se resuelve la ecuación la última (cuarta) vez, se resuelve a un nivel de residuo especificado por la tolerancia (ya que `relTol` es 0, desactivándolo efectivamente) para obtener una mayor precisión, pero a un mayor costo.

4.7.3. Solucionadores conjugados precondicionados

Existen una variedad de opciones para la precondición de matrices en los solucionadores conjugados gradientes, representadas por la palabra clave `preconditioner` en el diccionario del solucionador, listadas a continuación. Es importante notar que los precondicionadores `DIC/DILU` están exclusivamente especificados en los tutoriales de OpenFOAM.

- `DIC/DILU`: diagonal incompleta-Cholesky (simétrica) e incompleta-LU (asimétrica);
- `FDIC`: diagonal incompleta-Cholesky ligeramente más rápida (DIC con almacenamiento en caché, simétrica);
- `GAMG`: multi-rejilla geométrico-algebraica;
- `diagonal`: precondicionamiento diagonal, generalmente no utilizado;
- `none`: sin precondicionamiento.

4.7.4. Solucionadores de multi-rejilla geométrico-algebraica (GAMG)

El método de multi-rejilla geométrico-algebraica (GAMG) utiliza el principio de: generar una solución rápida en una malla con un pequeño número de celdas; mapear esta solución en una malla más fina; utilizarla como una conjetura inicial para obtener una solución precisa en la malla fina. GAMG es más rápido que los métodos estándar cuando el aumento de velocidad al resolver primero en mallas más gruesas supera los

costos adicionales de refinamiento de malla y mapeo de datos de campo. En la práctica, GAMG comienza con la malla original y la refina en etapas. El engrosamiento está limitado por un número mínimo de celdas especificado en el nivel más grueso. La aglomeración de celdas se realiza mediante el método especificado por la palabra clave agglomerator. Los tutoriales utilizan el método predeterminado `faceAreaPair`. La aglomeración puede ser controlada utilizando las siguientes entradas opcionales, la mayoría de las cuales son predeterminadas en los tutoriales.

- `cacheAgglomeration`: interruptor que especifica el almacenamiento en caché de la estrategia de aglomeración (predeterminado verdadero);
- `nCellsInCoarsestLevel`: tamaño aproximado de la malla en el nivel más grueso en términos del número de celdas (predeterminado 10);
- `directSolveCoarset`: utilizar un solucionador directo en el nivel más grueso (predeterminado falso);
- `mergeLevels`: la palabra clave controla la velocidad a la que se realiza el engrosamiento o refinamiento; el valor predeterminado es 1, que es el más seguro, pero para mallas simples, la velocidad de la solución puede aumentar al engrosar/refinar 2 niveles a la vez, es decir, configurando `mergeLevels 2`.

El número de pasadas utilizadas por el suavizador en diferentes niveles de densidad de malla se especifican mediante las siguientes entradas opcionales.

4.7.5. Subrelajación de la solución

El archivo `fvSolution` generalmente incluye un subdiccionario `relaxationFactors` que controla la subrelajación. La subrelajación se utiliza para mejorar la estabilidad de un cálculo, especialmente para problemas en estado estacionario. Funciona limitando la cantidad en que una variable cambia de una iteración a la siguiente, ya sea manipulando la ecuación matricial antes de resolver un campo, o modificando el campo después. Un factor de subrelajación $\alpha : 0 < \alpha \leq 1$, especifica la cantidad de subrelajación, como se describe a continuación.

- Sin α especificado: no hay subrelajación.
- $\alpha = 1$: igualdad/dominio diagonal de la matriz garantizada.
- α disminuye, la subrelajación aumenta.
- $\alpha = 0$: la solución no cambia con las iteraciones sucesivas.

Una elección óptima de α es aquella que es lo suficientemente pequeña como para garantizar un cálculo estable pero lo suficientemente grande como para avanzar rápidamente en el proceso iterativo; valores de α tan altos como $0{,}9$ pueden garantizar estabilidad en algunos casos y cualquier valor mucho menor, digamos, $0{,}2$, puede ser prohibitivamente restrictivo para ralentizar el proceso iterativo.

Los factores de relajación para la subrelajación de campos se especifican dentro de un subdiccionario de campo; los factores de relajación para la subrelajación de ecuaciones están dentro de un subdiccionario de ecuaciones. A continuación se muestra un ejemplo de un caso con el módulo de solucionador `incompressibleFluid` funcionando en modo estacionario. Los factores se especifican para la presión p, la velocidad $\mathbf{U}$ y los campos turbulentos agrupados mediante una expresión regular.

```
relaxationFactors
{
  fields
  {
    p 0.3;
  }
  equations
  {
    U 0.7;
    "(k|omega|epsilon).*" 0.7;
  }
}
```

Para un caso transitorio con el módulo `incompressibleFluid`, la subrelajación simplemente ralentizaría la convergencia de la solución dentro de cada paso de tiempo. En su lugar, generalmente se adopta la siguiente configuración para garantizar la igualdad diagonal que generalmente se requiere para la convergencia de una ecuación matricial.

```
relaxationFactors
{
  equations
  {
    ".*" 1;
  }
}
```

4.7.6. Algoritmos PIMPLE y SIMPLE

La mayoría de los módulos solucionadores de fluidos utilizan algoritmos para acoplar ecuaciones de conservación de masa y momentum conocidos como:

- `SIMPLE` (método semi-implícito para ecuaciones vinculadas a la presión);
- `PIMPLE`, una combinación de PISO (operador dividido implícito para la presión) y SIMPLE.

Dentro de un paso de tiempo, o solución, los algoritmos resuelven una ecuación de presión para hacer cumplir la conservación de masa, con una corrección explícita a la velocidad para satisfacer la conservación de momentum. Opcionalmente, comienzan cada paso resolviendo la ecuación de momentum, el llamado predictor de momentum.

Aunque todos los algoritmos resuelven las mismas ecuaciones gobernantes (aunque en formas diferentes), principalmente difieren en cómo iteran sobre las ecuaciones. La iteración está controlada por parámetros de entrada que se enumeran a continuación. Se establecen en un diccionario nombrado según el algoritmo, es decir, `SIMPLE` o `PIMPLE`.

- `nCorrectors`: utilizado por `PIMPLE`, establece el número de veces que el algoritmo resuelve la ecuación de presión y el corrector de momentum en cada paso; típicamente establecido en 2 o 3.
- `nNonOrthogonalCorrectors`: utilizado por todos los algoritmos, especifica soluciones repetidas de la ecuación de presión, utilizadas para actualizar la corrección no ortogonal explícita del término Laplaciano ; típicamente establecido en 0 para casos en estado estacionario y en 1 para casos transitorios.
- `nOuterCorrectors`: utilizado por `PIMPLE`, habilita la iteración sobre todo el sistema de ecuaciones dentro de un paso de tiempo, representando el número total de veces que se resuelve el sistema; debe ser mayor o igual a 1 y típicamente se establece en 1.
- `momentumPredictor`: interruptor que controla la resolución del predictor de momentum; típicamente se establece en desactivado para algunos flujos, incluidos los de bajo número de Reynolds y `multiphase`.

4.8. Discretización espacial en mallas

Esta sección proporciona una especificación de cómo OpenFOAM describe una malla. La malla es una parte integral de la solución numérica y debe cumplir ciertos criterios para garantizar una solución válida y, por lo tanto, precisa. OpenFOAM define una malla de celdas polihédricas arbitrarias en 3D, delimitadas por caras poligonales arbitrarias, es decir, las celdas pueden tener un número ilimitado de caras donde, para cada cara, no hay límite en el número de bordes ni ninguna restricción en su alineación. Esta descripción flexible de una malla ofrece una gran libertad en la generación y manipulación de la malla cuando la geometría del dominio es compleja.

Una malla de OpenFOAM comienza con puntos (o vértices). Cada punto es una ubicación en el espacio 3D, definida por un vector. El conjunto de puntos forma una lista donde cada punto puede ser indexado por su posición en la lista, comenzando desde cero. La lista no contiene puntos que no se utilicen.

Los puntos se utilizan para formar caras de malla, donde cada cara se define como una lista ordenada de puntos, descritos por sus etiquetas donde un punto se refiere a su etiqueta. El ordenamiento de las etiquetas de puntos en una cara es tal que cada par de puntos vecinos está conectado por un borde, es decir, sigues los puntos mientras viajas alrededor de la circunferencia de la cara. El conjunto de caras forma una lista donde cada cara se refiere por su etiqueta, representando su posición en la lista. Cada cara puede ser caracterizada por un vector cuya dirección es normal a la cara. El vector normal sigue la regla de la mano derecha, es decir, mirando hacia una cara, si la numeración de los puntos sigue un recorrido en sentido horario, el vector normal apunta hacia afuera, como se muestra en la Figura 4.4. Tenga en cuenta que las caras pueden estar deformadas, es decir, los puntos de la cara pueden no estar necesariamente en un plano. Hay dos tipos de caras, descritos a continuación.

- Caras internas, que conectan dos celdas (y nunca pueden ser más de dos). Para cada cara interna, el ordenamiento de las etiquetas de puntos es tal que el vector normal de la cara apunta hacia la celda con la etiqueta más grande, es decir, para celdas etiquetadas como '2' y '5', el normal apunta hacia '5'.
- Caras de frontera, que pertenecen a una sola celda ya que coinciden con el límite del dominio. Por lo tanto, una cara de frontera es abordada por una celda (solo) y un parche de frontera. El ordenamiento de las etiquetas de puntos es tal que el vector normal de la cara apunta fuera del dominio computacional.

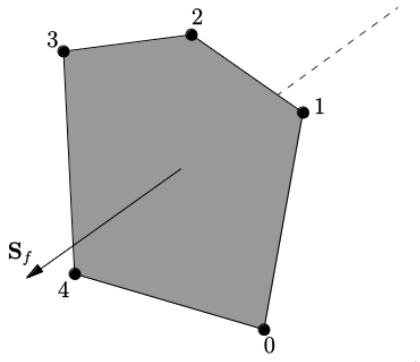

Figura 4.4: Vector de area de cara a partir de numeración de puntos en la cara. (Greenshields y Weller, 2022)

Una celda es una lista de caras en orden arbitrario. En circunstancias normales, las celdas deben tener las propiedades enumeradas a continuación.

- Las celdas deben ser contiguas, es decir, cubrir completamente el dominio computacional y no deben superponerse unas con otras.
- cada celda debe estar cerrada geométricamente, de modo que cuando todos los vectores de área de las caras estén orientados hacia afuera de la celda, su suma debe ser igual al vector cero con precisión de la máquina;
- cada celda debe estar cerrada topológicamente de modo que todos los bordes en una celda sean utilizados por exactamente dos caras de la celda en cuestión.

El límite está formado por las caras de límite. Debe estar cerrado, es decir, la suma de todos los vectores de área de las caras de límite debe ser igual a cero con tolerancia de la máquina. Se divide en regiones conocidas como parches para que se puedan aplicar diferentes condiciones de contorno a diferentes partes del límite. Un parche se define por las etiquetas de las caras que contiene.

Cuando una malla es escrita por OpenFOAM, los archivos de datos se colocan en un subdirectorio llamado `polyMesh`. Normalmente, el directorio `polyMesh` se escribe en el directorio constante, pero las simulaciones con mallas dinámicas (por ejemplo, movimiento de malla, refinamiento, etc.) escriben las mallas modificadas en directorios de tiempo junto con los archivos de datos de campo.

Los archivos de datos están basados en caras en lugar de celdas. Por lo tanto, a cada cara se le asigna una celda 'propietaria' y una celda 'vecina', de modo que la conectividad a través de una cara dada puede ser simplemente descrita por las etiquetas de las celdas propietaria y vecina. En el caso de los límites, no hay celda vecina. Con esto en mente, la especificación de E/S consiste en los siguientes archivos:

- `points`: una lista de vectores que describe los vértices de la celda, donde el primer vector en la lista representa el vértice `0`, el segundo vector representa el vértice `1`, etc.;
- `faces`: una lista de caras, donde cada cara es una lista de índices de vértices en la lista de puntos, donde nuevamente, la primera entrada en la lista representa la cara `0`, etc.;
- `owner`: una lista de etiquetas de celdas propietarias, comenzando con la celda propietaria de la cara `0`, luego `1`, `2`, etc.;
- `neighbour`: una lista de etiquetas de celdas vecinas;
- `boundary`: una lista de parches, que contiene una entrada de diccionario para cada parche, declarado usando el nombre del parche.

Críticamente, la lista de caras está ordenada de modo que todas las caras internas se enumeran primero, seguidas de las caras de los límites. Las caras de los límites están ordenadas de tal manera que comienzan con las caras en el primer parche, seguidas por el segundo, etc. Como consecuencia, las entradas de parches en el archivo de límites son muy compactas, por ejemplo:

```
1 inlet
2 {
3   type patch;
4   nFaces 30;
5   startFace 24170;
6 }
```

Debido al ordenamiento de las caras, las caras del parche se describen simplemente mediante: `startFace`, el índice en la lista de caras de la primera cara en el parche; y, `nFaces`, el número de caras en el parche.

El límite del dominio está definido por los parches dentro de la malla, enumerados dentro del archivo de límites de la malla. Cada parche incluye una entrada de tipo que puede aplicar una restricción geométrica al parche. Estas restricciones geométricas incluyen condiciones que representan una aproximación geométrica, por ejemplo, un plano de simetría, y condiciones que forman conexiones numéricas entre parches, por ejemplo, un límite cíclico (o periódico). A continuación se muestra un ejemplo de archivo de límites que incluye algunos parches con restricciones geométricas.

Se especifica una entrada de tipo para cada parche (entrada, salida, etc.), con tipos asignados que incluyen parche, pared, plano de simetría y vacío. Algunos parches también incluyen un `inGrupos`.

4.9. Condiciones de borde

Las condiciones de contorno se especifican en archivos de campo, por ejemplo, `0/p`, `0/U`, en directorios de tiempo. Incluyen tres entradas: dimensiones para las unidades dimensionales; `internalField` para los

valores iniciales del campo interno; y `boundaryField` donde se especifican las condiciones de contorno. El `boundaryField` requiere una entrada para cada parche en la malla. Los parches se especifican en el archivo de límites.

El `boundaryField` es un subdiccionario que contiene una entrada para cada parche en la malla. Cada entrada comienza con el nombre del parche y configura la condición de contorno a través de entradas en un subdiccionario. Se requiere una entrada de tipo para cada parche que especifique el tipo de condición de contorno. Los ejemplos anteriores incluyen condiciones `zeroGradient` y `fixedValue` correspondientes a parches genéricos definidos en el archivo de límites. También incluyen tipos `symmetry` y `empty` correspondientes a parches de restricción equivalentes, por ejemplo, el parche up se define como simetría en la malla y utiliza una condición de simetría en el archivo de campo.

Hay tres formas diferentes en las que se puede especificar una entrada para un parche en el `boundaryField` de un archivo de campo: 1) por nombre de parche; 2) por nombre de grupo; 3) mediante la coincidencia de un nombre de parche con una expresión regular. Se enumeran aquí en orden de precedencia que se obedece si múltiples entradas son válidas para un parche en particular. Las diferentes especificaciones se pueden ilustrar imaginando una malla con los siguientes parches.

- `inlet`: un parche genérico.
- `lowerWall` y `upperWall`: dos parches de pared.
- `outletSmall`, `outletMedium` y `outletLarge`: tres parches de salida de tipo genérico, todos en un grupo de parches llamado `outlet`.

Luego imagina el siguiente `boundaryField` para un campo, por ejemplo, `p`, correspondiente a los parches anteriores.

```
boundaryField { inlet { type zeroGradient; } ".*Wall" { type zeroGradient; }
    outletSmall { type fixedValue; value uniform 1; } outlet { type fixedValue;
      value uniform 0; } }
```

En este ejemplo, la entrada del campo de entrada se lee para el parche de entrada, siguiendo la regla 1 anterior (coincidencia de nombre de parche). Del mismo modo, la entrada de `outletSmall` se leerá para el parche del mismo nombre.

Los parches `outletMedium` y `outletLarge` no tienen entradas coincidentes en el archivo de campo, por lo que en su lugar se aplicará la entrada de salida (regla 2), ya que coincide con el nombre del grupo al que pertenecen los parches. Tenga en cuenta que el parche `outletSmall` no utiliza la entrada de salida porque una entrada de parche coincidente tiene prioridad sobre una entrada de grupo coincidente.

Finalmente, los parches `lowerWall` y `upperWall` coinciden con la expresión regular `".Wall"`. Estas expresiones regulares deben incluirse entre comillas dobles. El componente `"."` coincide con cualquier expresión (incluyendo nada), por lo que coincide con los nombres de parche de pared aquí. La expresión regular podría usar agrupaciones de palabras para proporcionar una coincidencia más precisa con los nombres de parche, por ejemplo.

```
"(lower|upper)Wall" { type zeroGradient; }
```

Alternativamente, una entrada de parche podría cubrir los parches de pared aprovechando el hecho de que cada parche no genérico se coloca automáticamente en un grupo del mismo nombre que su tipo. En este caso, todos los parches de pared se colocan en un grupo llamado `wall`, por lo que la siguiente entrada se leería para ambos parches.

```
wall
{
  type zeroGradient;
}
```

Implementación de la formulación magnetostática en OpenFOAM 5

5.1. Introducción

La simulación computacional se ha convertido en una herramienta invaluable para el estudio y diseño de sistemas electromagnéticos complejos en una variedad de aplicaciones industriales y científicas. En este contexto, OpenFOAM ha ganado popularidad debido a su flexibilidad y capacidad para abordar una amplia gama de problemas físicos mediante el Método de Volumenes Finitos.

Este capítulo se centra en la implementación en OpenFOAM de la formulación presentada en el Capítulo 3, con el objetivo de simular campos magnéticos en sistemas físicos complejos. La formulación magnetostática se basa en las ecuaciones de Maxwell, que describen la interacción de campos eléctricos y magnéticos en medios materiales. En particular, nos enfocamos en la resolución numérica de estas ecuaciones en sistemas con múltiples regiones de diferentes propiedades magnéticas.

El *solver* desarrollado, denominado `multiRegionMagneticFoam`, es una implementación en el marco de trabajo de OpenFOAM que permite modelar sistemas electromagnéticos en geometrías complejos con una alta precisión y eficiencia computacional. Este *solver* utiliza el Método de Volumenes Finitos para discretizar las ecuaciones de Maxwell en cada región del dominio, permitiendo así la resolución numérica de campos magnéticos en condiciones estáticas.

El código implementado se estructura en torno a la clase creada por los autores `solidMagnetostaticModel`, que encapsula los modelos físicos y numéricos necesarios para resolver las ecuaciones de Maxwell en medios sólidos. Además, se emplea una estrategia de control de convergencia `SIMPLE` para garantizar la estabilidad y precisión de la simulación, lo que simplifica significativamente el proceso de ajuste de parámetros y la optimización del rendimiento computacional.

En este capítulo, presentamos una descripción detallada del solver `multiRegionMagneticFoam`, destacando su estructura, funcionamiento y capacidades. Además, discutimos las principales características del modelo magnetostático implementado, incluyendo la discretización espacial de las ecuaciones, los esquemas de relajación y corrección no ortogonal, y las estrategias de control de convergencia utilizadas.

En la Figura 5.1 se puede observar una representación en diagrama de flujo del *solver* `multiRegionMagneticFoam`. En las secciones siguientes se procederá a detallar cada una de las rutinas de la implementación en OpenFOAM según la Fig. 5.1, mientras que el código completo se puede encontrar en la Sección B.1.

5.2. Inicialización

En esta sección se detallarán las clases y encabezados que se encargan de inicializar y leer los archivos de configuración del *case*, como lo son la geometría o *mesh*, los controles de solución y discretización, los valores iniciales y condiciones de borde, etc. Esto se corresponde con la preparación anterior al bucle exterior que se muestra en la Figura 5.1, y con las primeras 23 lineas en el Código B.1.1.

La sección del código comienza con la inclusión de archivos de encabezado para la implementación y resolución de la formulación magnetostática. Estos archivos proporcionan las constantes electromagnéticas necesarias, así como las propiedades específicas de las regiones en las que se llevará a cabo la simulación.

La inclusión de `electromagneticConstants.H` establece las bases fundamentales para la descripción matemática de los fenómenos electromagnéticos involucrados en el problema a resolver en el marco de OpenFOAM. Luego, `regionProperties.H` permite la creación de objetos que almacenan las propiedades específicas de cada región en el dominio de la simulación. Estas propiedades pueden incluir características

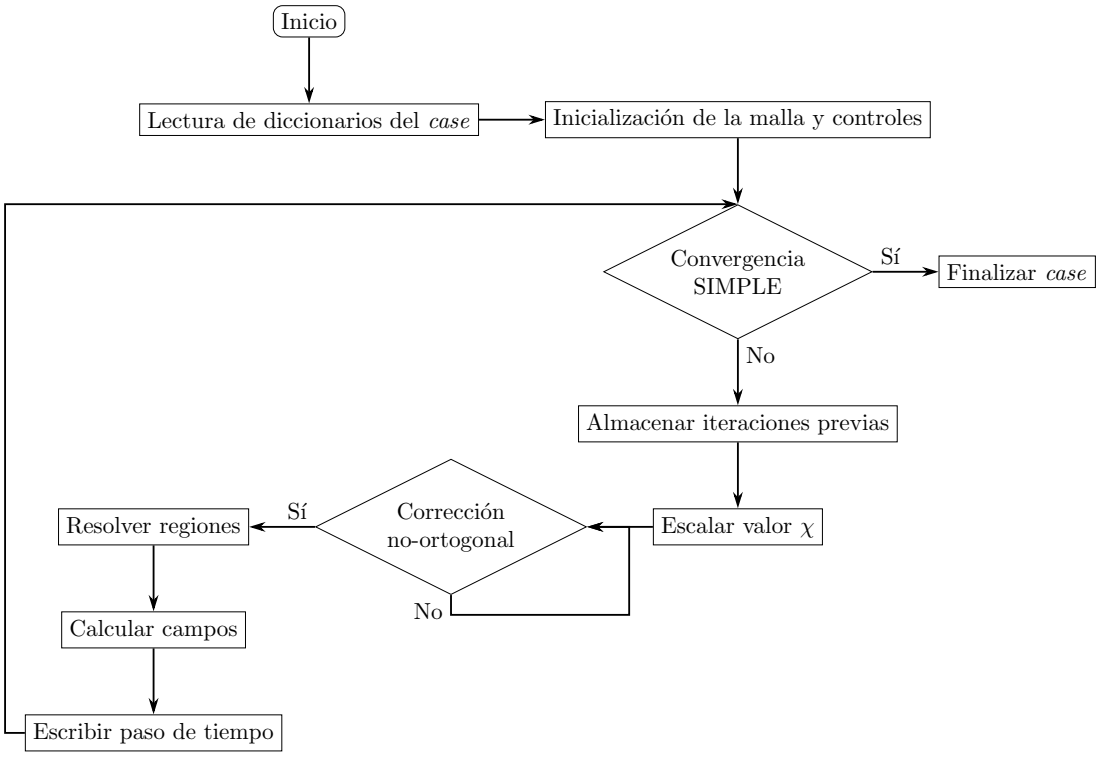

Figura 5.1: Diagrama del *solver*.

geométricas, materiales, y condiciones de contorno, entre otros aspectos relevantes para la simulación magnetostática.

Se procede incluyendo los archivos `solidMagnetostaticModel.H` y `simpleControl.H`, los cuales son se utilizan para la implementación del modelo físico y el control de la convergencia en el proceso de solución, respectivamente. El primero define la estructura y comportamiento del modelo magnetostático sólido a resolver, mientras que el segundo proporciona herramientas para gestionar y controlar la convergencia numérica del proceso de simulación a través del algoritmo SIMPLE descrito en la Sección 4.7.6. Aún más, `solidMagnetostaticModel.H` define la clase `solidMagnetostaticModel` como se muestra en los Códigos B.1.3-B.1.4.

Luego, la función principal `main` inicia la ejecución del programa, donde se procede a la configuración inicial del caso de simulación. Se inicializan listas de casos, se definen condiciones temporales y se generan los elementos básicos necesarios para la simulación, como mallas y campos en los encabezados `createTime.H`, `creteMeshes.H`, `createMagneticControls.H` y `readMagneticControls.H`. Además, se establece un control simplificado de la convergencia numérica utilizando un el *solver* SIMPLE que proporciona una única estrategia de convergencia para todas las regiones del dominio; i.e.

```
1 fvMesh& mesh = allRegions[0];
2 simpleControl simple(mesh);
```

5.3. Bucle principal `SIMPLE`

El bucle principal del programa se encarga de ejecutar las iteraciones necesarias para resolver el problema electromagnético en estudio. Este bucle está diseñado para ajustar el parámetro χ, que controla la respuesta magnética del material permeable, a lo largo de varias iteraciones externas. Durante cada iteración externa, se realiza un conjunto de correcciones no ortogonales para asegurar la convergencia del sistema. El bucle

continúa según las opciones de convergencia del algoritmo SIMPLE. Esto es, el proceso de cálculo por volumenes finitos continuará hasta que se consiga la convergencia de los valores de $\mathbf{A}$ o hasta que se llegue al número máximo de iteraciones establecido por el usuario en el diccionario del case `system/controlDict`.

Primero, se almacenan las iteraciones previas de las magnitudes magnéticas para tener la posibilidad de aplicar el coeficiente de relajación durante la corrección no-ortogonal.

```
forAll(allRegions, i)
{
        MIs[i].storePrevIter();
}
```

Si se ha activado la opción de escalado de χ, se lleva a cabo dicho escalado según las especificaciones definidas. Esta opción se puede definir en el case en el diccionario `system/fvSolution` simplemente como `scaleChi yes;`. Luego, en `readMagneticControls.H` se busca dicha palabra clave según

```
Switch scaleChi = vmfControl.lookupOrDefault<Switch>("scaleChi", "on");
```

y si no se encuentra se activa por defecto. Luego, el proceso de escalado de χ se realiza de la siguiente manera:

```
scalar ppf = 0.0; // Parameter Proportional Factor
scalar iStop(runTime.endTime().value()-chiIterCutOff); // Iteration of Stop
scalar iDelta(iStop-chiIterCutIn);

if (runTime.value() > chiIterCutIn && runTime.value() <= iStop)
{
    ppf = 1.0/(iDelta)*runTime.value() - chiIterCutIn/(iDelta);
}
else if(runTime.value() > iStop)
{
    ppf = 1.0;
    scaleChi = false;
}

forAll(allRegions, i)
{
    volScalarField& chii = chis[i];
    chii = ppf * (murs[i]-1)/(murs[i]*mu0);
}

Info << "Scaling chi with factor: " << ppf << endl;
```

Posteriormente, se establece la tolerancia para las últimas iteraciones, lo que permite ajustar los parámetros mediante el parámetro `finalIterations` en el diccionario `system/fvSolution` con el objetivo de encontrar una convergencia más precisa hacia el final del cálculo. Si se alcanza el número de iteración correspondiente a la tolerancia de las últimas iteraciones, se ajustan los parámetros de convergencia y se desactiva esta opción para evitar modificaciones posteriores.

```
if (runTime.value() >= runTime.endTime().value() - finalIterations)
{
        Info<< "Switching to final correction solution parameters..." << nl << endl;
        solverControl.set("tolerance", finalTolerance);
        solDict.regIOobject::write(true);
        nCorrNonOrth = finalCorrectors;
        correctFinal = false;
}
```

A continuación, se realiza un conjunto de correcciones no ortogonales para cada región del dominio. Estas correcciones se llevan a cabo para garantizar la convergencia del campo magnético en presencia de geometrías complejas y no-ortogonales. Durante este proceso, se aplican métodos de relajación para actualizar las magnitudes magnéticas en cada región del dominio.

```
for (int k = 1; k<=nCorrNonOrth; k++)
{
        Info<< "Non-orthogonal corrector: " << k << nl << endl;
        forAll(allRegions, i)
        {
                MIs[i].relax();
        }
        forAll(allRegions, i)
        {
            Info<< nl <<"--> Solving region: " << allRegions[i].name() << endl;
                #include "setRegionFields.H"
                #include "solveRegion.H"
                #include "calculateFields.H"

                if (calcForces)
                {
                        Info<< nl <<"--> Calculating forces ..."<< endl;
                        #include "calculateForces.H"
                }
        }
}
```

Después de cada corrección no-ortogonal, se resuelven las ecuaciones para cada región del dominio, se calculan los campos magnéticos y se actualizan las fuerzas magnéticas en los encabezados `setRegionFields.H`, `solveRegion.H` y `calculateFields`, como se explicará en la sección siguiente.

Una vez completadas todas las correcciones no ortogonales, se registra el tiempo de ejecución del cálculo para evaluar el rendimiento del programa, y si no se llega a encontrar convergencia y no se alcanzó el número máximo de iteraciones establecido por el usuario, se vuelve a iterar.

5.4. Algoritmos de resolución

Los algoritmos de resolución se encargan de efectivamente encontrar una solución numérica de la ecuación de la ecuación de balance 3.42. El primer encabezado que se ejecuta en el Código B.1.2 es `setRegionFields.H`, el cual se encarga de establecer los campos de las regiones del dominio, lo que implica asignar valores iniciales a las variables físicas relevantes para cada región. Este proceso se utiliza para iniciar adecuadamente la simulación electromagnética y garantizar que los campos magnéticos se calculen consistentemente.

```
Info<<"        Setting region fields... "  << endl;

volVectorField& Ai = As[i];

volScalarField& mui = mus[i];
volScalarField& chii = chis[i];
volVectorField& Ji = Js[i];

volVectorField& Bi = Bs[i];
volVectorField& Bri = Brs[i];
volVectorField& Hi = Hs[i];
volVectorField& Hri = Hrs[i];
volVectorField& Mi = Ms[i];
volVectorField& MIi = MIs[i];
volVectorField& MIri = MIrs[i];
volVectorField& MTi = MTs[i];
volVectorField& MTri = MTrs[i];

volTensorField& Ti = Ts[i];
volTensorField& Tri = Trs[i];
volVectorField& Fli = Fls[i];

volVectorField& Fki = Fks[i];
volVectorField& Fkri = Fkrs[i];
volVectorField& Fk2i = Fk2s[i];
volVectorField& Fk2ri = Fk2rs[i];
```

Se accede a los campos de variables físicas asociados con la región actual. Estos campos incluyen tanto escalares como vectores, y representan magnitudes como la densidad de corriente $\mathbf{J}$, la permeabilidad magnética μ, la susceptibilidad magnética χ, entre otros. Para cada campo físico, se utiliza una convención de nomenclatura para identificar la región a la que pertenece. Por ejemplo, `Ai` representa el potencial vectorial magnético, `Bi` representa el campo magnético, `Mi` representa la magnetización, y así sucesivamente.

Una vez que se han accedido a todos los campos relevantes, se procede a asignar valores iniciales o condiciones iniciales apropiadas a cada uno de ellos. Estos valores iniciales pueden ser cero, constantes o pueden provenir de datos de entrada previamente definidos en el case por el usuario.

Luego, en `solveRegion.H` se produce cálculo y la resolución de la ecuación diferencial parcial que describe la evolución del potencial vectorial magnético $\mathbf{A}$ en el dominio del problema, i.e. Ecuación 3.42. Esto se realiza mediante la construcción de una matriz de vectores `tAEqn` que representa la ecuación diferencial en forma matricial. La ecuación está compuesta por un operador laplaciano aplicado al potencial vectorial magnético $\nabla^2\mathbf{A}$ o `fvm::laplacian(Ai)` en el lado izquierdo de 3.42, igualado a la contribución de las fuentes magnéticas en el dominio. Estas fuentes incluyen la densidad de corriente $\mathbf{J}$ o `Ji`, la magnetización $-\nabla\cdot\left(\chi\nabla\widetilde{A}\right)$ o `MIi` y el rotor de la magnetización inducida $\nabla\times\mathbf{M}$ o `fvc::curl(Mi)`. Los coeficientes de la ecuación, como la permeabilidad magnética del vacío μ_0, se incorporan según las leyes del electromagnetismo.

Una vez que se ha formulado la ecuación diferencial en términos matriciales, se procede a resolverla utilizando el método numérico iterativo. En este caso, se utiliza el método de resolución implementado en el objeto `AEqn`, que se refiere a la matriz de vectores `tAEqn`. El proceso de resolución se lleva a cabo utilizando un controlador de solución `solverControl`, que supervisa la convergencia del método numérico.

```
Info << "Bi norm" << Bi.internalField().average() << endl;

tmp<fvVectorMatrix> tAEqn
        (
           fvm::laplacian(Ai)
        ==
           mu0*(
                - Ji
                - MIi
                - fvc::curl(Mi)
                  )
        );

fvVectorMatrix& AEqn = tAEqn.ref();

AEqn.solve(solverControl);
```

Finalmente en `calculateFields.H` se aborda la determinación de los campos de solución.

```
const surfaceScalarField& magSfi = allRegions[i].magSf();

Bi = fvc::curl(Ai);
MIi = fvc::div(chii*2.0*skew(fvc::grad(Ai)));
```

Primeramente, se accede al vector de superficie $\mathbf{S}$ o `magSfi` en cada uno de los elementos de la región asociada, el cual es necesario para calcular el comportamiento magnético en dicha región. Posteriormente, se procede a calcular el campo magnético $\mathbf{B}$ o `Bi` y la magnetización $-\nabla \cdot \left(\chi \nabla \widetilde{\mathbf{A}}\right)$ o `MIi`. En primer lugar, el cálculo del campo magnético se realiza mediante el rotacional del potencial vectorial magnético $\mathbf{A}$. Este procedimiento se realiza utilizando la función `fvc::curl()`, que evalúa el rotacional de un campo vectorial en el dominio. El campo magnético resultante se asigna a la variable `Bi`, lo que proporciona la distribución espacial y la intensidad del campo magnético en la región.

En segundo lugar, se calcula la magnetización mediante la divergencia del producto de la susceptibilidad magnética χ y el gradiente del potencial vectorial magnético $\nabla \mathbf{A}$. La magnetización resultante se escala por un factor de 2 ya que la función `skew()` devuelve el resultado multiplicado por un factor de $0{,}5$. Este proceso permite capturar la contribución de los materiales ferromagnéticos al campo magnético.

Resultados numéricos y evaluación de la formulación 6

Desarrollamos diversos tests de referencia para determinar el rendimiento y la precisión de la formulación en superficies curvas. Inicialmente, comparamos la eficacia de la formulación de Volumen Finito actual, implementada en OpenFOAM, con un *solver* de Elementos Finitos establecido utilizando COMSOL Multiphysics. Posteriormente, evaluamos la capacidad del método para capturar distribuciones discontinuas de campo magnético en configuraciones que incluyen combinaciones de medios permeables, magnetizados permanentemente y portadores de corriente.

Todos los casos de prueba son pseudo-bidimensionales, y la discretización del dominio se ha realizado con mallas triangulares. Se han elegido casos bidimensionales en lugar de tridimensionales para ahorrar tiempo de computación y facilitar el proceso de mallado, ya que se ha demostrado que los resultados obtenidos con casos tridimensionales corresponden correctamente con los obtenidos con casos bidimensionales (Riedinger y Saravia, 2023). La densidad de la malla se especifica en cada caso de prueba.

6.1. Prueba del imán cuadrado y material permeable curvo

En la comparación inicial entre el enfoque de VF y el solucionador establecido de FE, resolvimos la configuración mostrada en la Figura 6.1. La configuración consiste en un material permeable curvo (ferro) de diámetro $\phi = 0{,}075$ m y un material magnetizado permanentemente cuadrado (imán) con lados de longitud $0{,}1$ m. El ferro se estableció con una permeabilidad relativa $\mu_r = 30$ y la magnetización permanente del imán se fijó en $\mathbf{M} = 9{,}75 \times 10^5$ A m^{-1} $\mathbf{e}y$, siendo $\mathbf{e}y$ el versor que apunta hacia la dirección vertical. Tanto el imán como el ferro estaban inmersos en un volumen de aire de dimensiones 1 m $\times$ 1 m $\times$ 1 m. Las condiciones de contorno (BCs) se especificaron como $\mathbf{A} = 0$ tanto para el método de VF actual como para el solucionador de FE.

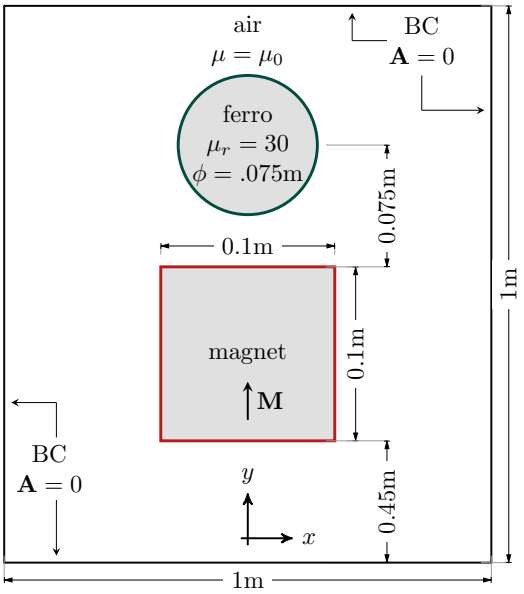

Figura 6.1: Configuración del caso de prueba de referencia con un material permeable circular y un imán permanente cuadrado.

La densidad de la malla alrededor del ferro y el imán se controló con un tamaño de borde de 5×10^{-3} m, más gruesa en los límites del dominio de aire 1×10^{-2} m, asegurando la conectividad de la malla en todo el dominio. Esto se muestra en la Figura 6.2, y representa la cuadrícula utilizada tanto con los solucionadores de FE como de VF. Esta malla no ideal, aunque generalmente gruesa, facilita la evaluación de la formulación

de VF, particularmente su rendimiento con elementos de alto orden y alta no ortogonalidad. Los elementos marcados en magenta en la Figura 6.2 se utilizarán para la comparación posterior del campo magnético entre los dos solucionadores; esto se explicará en detalle más adelante en este artículo.

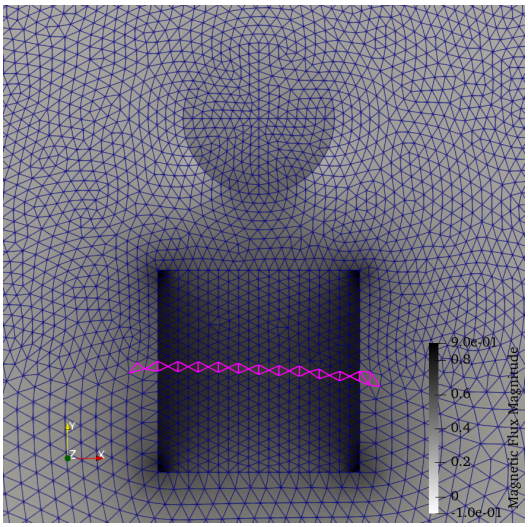

Figura 6.2: Representación de la cuadrícula utilizada para discretizar espacialmente el dominio en la Figura 6.1.

Los resultados obtenidos con ambas formulaciones se pueden ver en las Figuras 6.3 y 6.4. Las figuras muestran la convolución integral de línea de superficie para visualizar el campo magnético vectorial, no solo en magnitud sino también en dirección. De esta manera, las discontinuidades en ambas cantidades del campo vectorial pueden ser apreciadas y comparadas. Desde un punto de vista cualitativo, es claro a partir de las figuras que tanto los enfoques de FE como de VF coinciden bien en la predicción de la distribución del campo magnético. Las discontinuidades en las interfaces entre diferentes medios son capturadas correctamente con la presente formulación, y coinciden con la solución encontrada por el solucionador de FE. Podemos ver cómo la superficie curva del material ferroso afecta la dirección de la componente tangencial de $\mathbf{B}$; sin embargo, el solucionador de VF es capaz de capturar este cambio discontinuo en la dirección. El gradiente en la magnitud observado en la superficie del imán no solo es comparable al resultado obtenido con el solucionador de FE, sino que también corresponde a los resultados mostrados en (Riedinger y Saravia, 2023; Saravia, 2021).

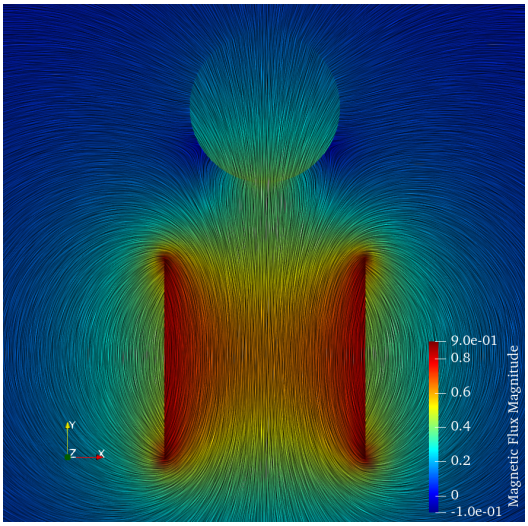

Figura 6.3: Convolución integral de línea de superficie del campo magnético $\mathbf{B}$ en la prueba del imán cuadrado y el material permeable curvo. Solución encontrada por el solucionador FEM de COMSOL Multiphysics.

Luego, en la Figura 6.5 mostramos una evaluación cuantitativa para corroborar los resultados obtenidos cualitativamente de las Figuras 6.3 y 6.4. Evaluamos la componente y del campo magnético, $\mathbf{B} \cdot \mathbf{e}_y$, con ambos

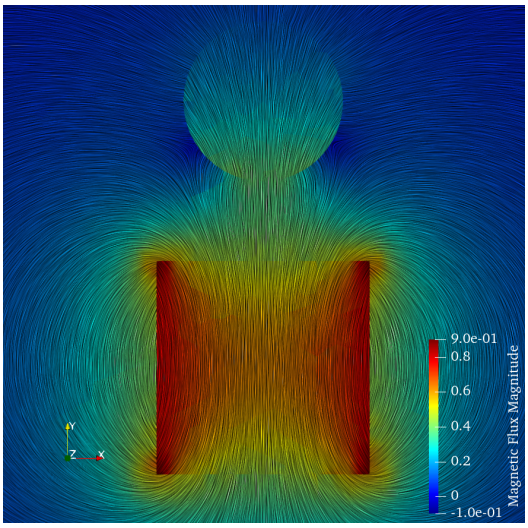

Figura 6.4: Convolución integral de línea de superficie del campo magnético $\mathbf{B}$ en la prueba del imán cuadrado y el material permeable curvo. Solución encontrada por el presente solucionador FVM.

el solucionador comercial de FE y el método de VF presente. Solo evaluamos la componente vertical de $\mathbf{B}$, ya que es la única que presenta una discontinuidad, como se ha demostrado en (Saravia, 2021). La medida se realiza en los centroides de las celdas de los elementos marcados en magenta en la Figura 6.2, y coincide para ambos métodos ya que se ha utilizado la misma cuadrícula para encontrar las respectivas soluciones. Sin embargo, dado que el FEM calcula los valores del campo en los nodos de la celda, es necesario utilizar técnicas de interpolación para evaluar los mismos puntos espaciales que en la formulación de VF presente, donde la solución se calcula principalmente en los centroides de las celdas. Por lo tanto, se ha utilizado la interpolación lineal para interpolar datos de punto a datos de celda dentro de un elemento triangular en la solución de FEM.

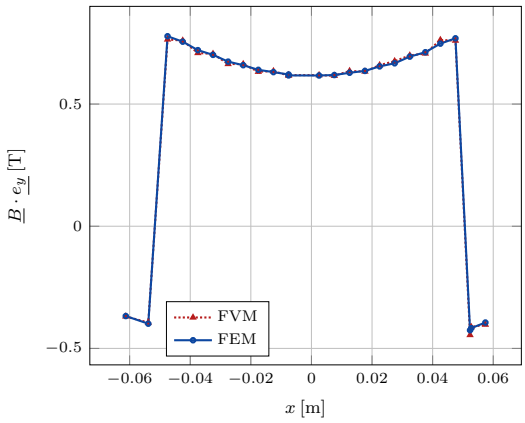

Figura 6.5: Valores centrados en celda de $\mathbf{B} \cdot \mathbf{e}_y$. Comparación entre FVM (línea roja punteada) y FEM (línea azul).

La Figura 6.5 muestra que la formulación presente, representada como FVM en el gráfico, puede encontrar la misma solución en los centroides de las celdas que el solucionador comercial COMSOL; FEM en la figura. Como se puede ver, la formulación presente puede capturar correctamente las distribuciones discontinuas de campo magnético en mallas no ortogonales. Sin embargo, es importante destacar que esta comparación entre los enfoques de FE y VF depende de técnicas externas al método, ya que los dos miden los datos del campo en puntos diferentes del espacio. Sin embargo, las evaluaciones son válidas para verificar la formulación si se utilizan valores interpolados en el solucionador de FE.

6.2. Prueba del imán curvo y material permeable curvo

Expandiendo la comparación inicial, probamos el método de VF con la configuración mostrada en la Figura 6.6, que presenta un dominio compuesto por un imán circular y un material permeable, con diámetros de $\phi = 0{,}1$ m y $\phi = 0{,}075$ m respectivamente. Luego, los materiales están rodeados por un volumen de aire de dimensiones $1\text{ m} \times 1\text{ m} \times 1\text{ m}$. La malla es más fina que en el caso anterior, con una densidad basada en elementos de tamaño 1×10^{-3} m alrededor del centro del dominio, aumentando a 1×10^{-2} m en los límites.

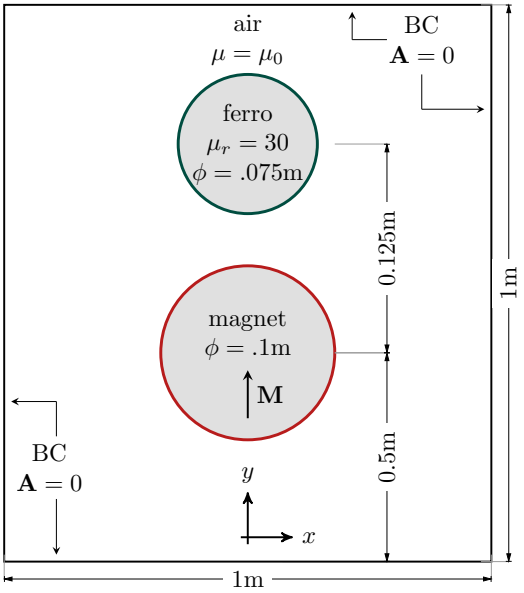

Figura 6.6: Configuración del caso de prueba de referencia con un material permeable circular y un imán permanente curvo.

Los resultados del experimento numérico se muestran en la Figura 6.7. La figura representa la convolución integral de línea del campo magnético. Se puede observar que las discontinuidades en la dirección y magnitud de $\mathbf{B}$ que surgen en la interfaz entre el imán y el aire son capturadas correctamente por el solucionador. Además, el cambio drástico en la dirección en el componente ferroso, como se muestra en la sección anterior, también se captura. Luego, lejos de las interfaces ferro-aire e imán-aire, los resultados son consistentes con (Saravia, 2021) y (Riedinger y Saravia, 2023), y con la predicción de bordes de FEM mostrada en la sección anterior.

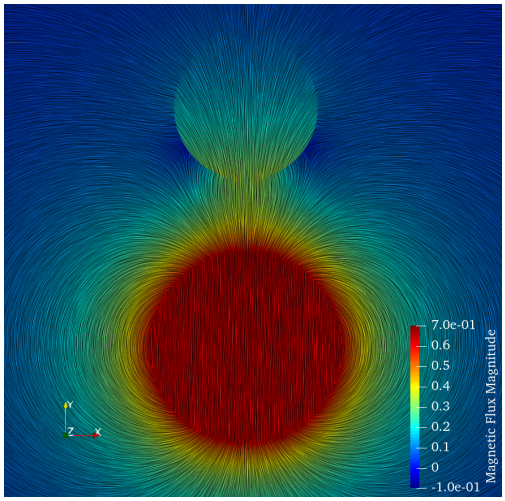

Figura 6.7: Convolución integral de línea del campo magnético $\mathbf{B}$ en la prueba del imán curvo y el material permeable curvo. Solución encontrada por el presente solucionador FVM.

6.3. Prueba de corriente volumétrica con un imán curvo y material permeable

A continuación, evaluamos la precisión del método en un caso con una interacción fuerte entre tres materiales curvos. La configuración se muestra en la Figura 6.8, con la prueba de referencia compuesta por un imán de diámetro $\phi = 0{,}1$ m y magnetización permanente $\mathbf{M} = 9{,}75 \times 10^5$ A m^{-1}, $\mathbf{e}_y$, un material permeable de diámetro $\phi = 0{,}075$ m y permeabilidad relativa $\mu_r = 30$, y finalmente un medio portador de corriente de diámetro $\phi = 0{,}075$ m y vector de corriente $\mathbf{J} = 2{,}5 \times 10^7$ A m^{-2} $\mathbf{e}_z$. Los tres medios están rodeados por un volumen de aire de dimensiones 1 m $\times$ 1 m $\times$ 1 m. La densidad de la malla y las condiciones de contorno se especificaron de manera similar a la prueba anterior.

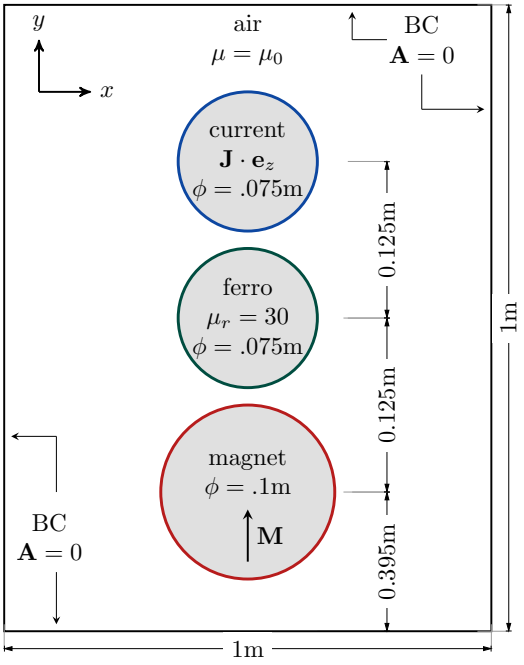

Figura 6.8: Configuración del caso de prueba de referencia con materiales permeables, magnéticos permanentemente magnetizados y portadores de corriente circulares.

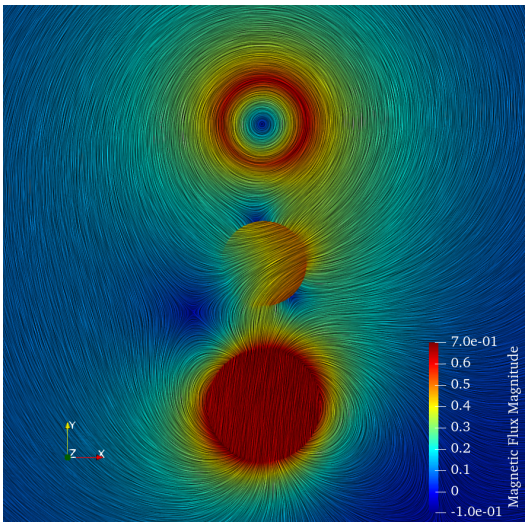

Figura 6.9: Convolución integral de línea del campo magnético $\mathbf{B}$ en la prueba de corriente volumétrica con un imán curvo y un material permeable curvo. Solución encontrada por el presente solucionador FVM.

El campo magnético resultante se puede observar en la Figura 6.9. El material portador de corriente se

caracteriza por la falta de corrientes superficiales, $\mathbf{K}_f$, y por lo tanto el RHS de la condición de contorno, la Ecuación 3.49, es nulo ya que $\Delta\chi\mathbf{B} = \Delta\mathbf{M}$. En consecuencia, las componentes normal y tangencial del potencial vectorial en la interfaz del material de corriente son continuas. Sin embargo, la interacción con el campo magnético generado por los medios magnéticos permanentemente magnetizados es fuerte, lo que lo convierte en un caso interesante para evaluar las diferentes discontinuidades en los materiales ferroso e imantado. Como se puede ver en la figura, la formulación actual es capaz de capturar los cambios en la dirección y magnitud del campo magnético vectorial en los tres materiales curvos.

6.4. Prueba de no ortogonalidad

Finalmente, resolvemos la configuración representada en la Figura 6.10. Esta configuración ya ha sido resuelta en (Riedinger y Saravia, 2023; Saravia, 2021), pero solo para mallas uniformes y ortogonales. Ahora, probamos los resultados con una malla no ortogonal compuesta exclusivamente de elementos tria. Un análisis exhaustivo de la no ortogonalidad en el FVM no es trivial, y una implementación de correctores no ortogonales para la condición de contorno todavía está en desarrollo. Por lo tanto, no es el propósito de esta sección analizar exhaustivamente la no ortogonalidad; un análisis exhaustivo se dejará como trabajo futuro. Más bien, el objetivo principal de esta sección es validar los resultados obtenidos en las secciones anteriores con mallas triangulares, comparando un caso similar al resuelto en (Riedinger y Saravia, 2023; Saravia, 2021) con una estrategia de malla diferente.

La configuración para el caso de prueba se muestra en la Figura 6.10, y consiste en dos materiales cuadrados: un imán de dimensiones $0{,}1$ m $\times$ $0{,}1$ m y un ferro de tamaño $0{,}1$ m $\times$ $0{,}05$ m, rodeados por un volumen de aire de dimensiones 1 m $\times$ 1 m. La malla triangular en este caso consiste en elementos de tamaño 1×10^{-3} m en el centro del dominio, aumentando hacia los límites donde el tamaño es 1×10^{-2} m.

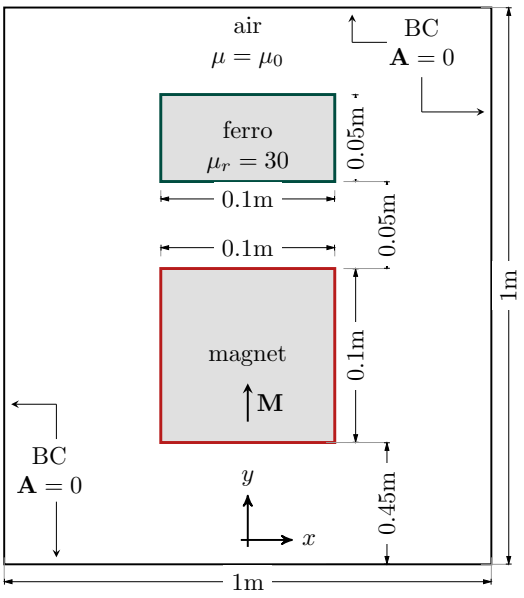

Figura 6.10: Configuración del caso de prueba de referencia con un material permeable cuadrado y un imán permanente cuadrado.

Los resultados de esta prueba se muestran en la Figura 6.11, mientras que la Figura 6.12 representa los mismos resultados pero con una malla ortogonal y uniforme, manteniendo una densidad de malla similar. Como se puede ver en la evaluación cualitativa, el cálculo del campo magnético coincide independientemente de la malla. Sin embargo, en el caso no ortogonal, la convergencia solo se logra teniendo en cuenta consideraciones de relajación del campo. Esto no es el caso para el experimento ortogonal, donde la relajación no es necesaria y, por lo tanto, no se implementa.

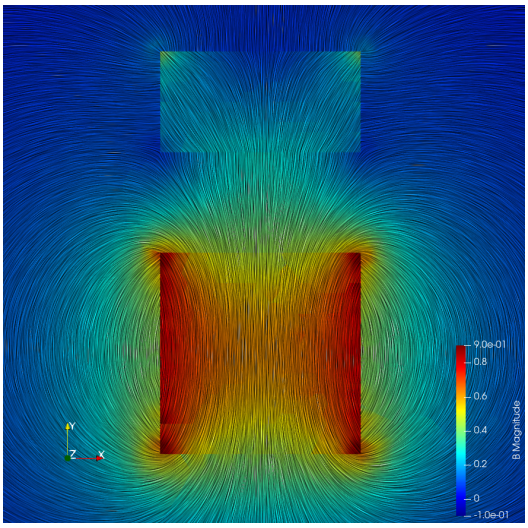

Figura 6.11: Convolución integral de línea del campo magnético $\mathbf{B}$ en la prueba de no ortogonalidad con malla no ortogonal basada en tria.

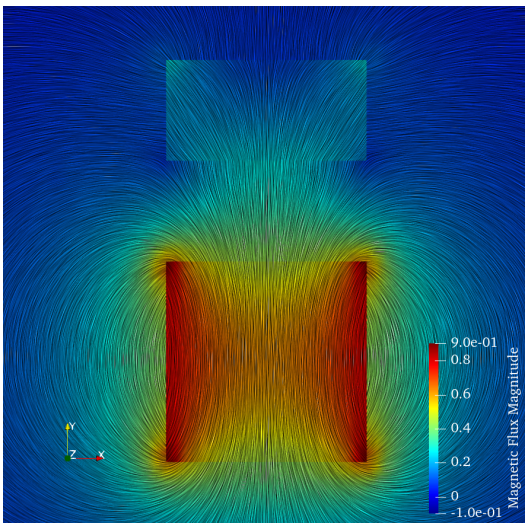

Figura 6.12: Convolución integral de línea del campo magnético $\mathbf{B}$ en la prueba de no ortogonalidad con malla ortogonal y uniforme.

6.5. Otros aspectos numéricos

Se han encontrado muchos aspectos numéricos dentro de la simulación de la geometría representada en la Figura 6.2, y se señalarán en esta sección.

En primer lugar, cabe destacar que el esquema de gradiente utilizado en el proceso de discretización es de suma importancia. Dentro del marco de OpenFOAM, hay dos esquemas de discretización de gradiente disponibles para que el usuario seleccione: Gauss-lineal, como se describe en la Sección 3.4.1, y el método de mínimos cuadrados. Se ha encontrado experimentalmente que el solucionador de Volumen Finito diverge usando el método de mínimos cuadrados, mientras que la convergencia se ha logrado con éxito con el enfoque Gauss-lineal. Esto se debe a la gran dependencia que el método de mínimos cuadrados tiene con el vector de distancia entre dos centroides de celdas. Esto es, en una malla no ortogonal donde los centroides de las celdas no son colineales y varían dentro de las coordenadas espaciales, las funciones de ponderación no son suficientes para compensar la disparidad dada por el término no ortogonal. Esto se debe principalmente al hecho de que la condición de contorno no tiene una corrección no ortogonal implementada en el solucionador de Volumen Finito, y por lo tanto la solución con este método de discretización de gradiente diverge. Esto todavía está en progreso y se dejará como trabajo futuro.

En segundo lugar, la sobrerrelajación como se define en la Sección 3.4.2 hace que el solucionador diverja. Se ha encontrado para este problema en particular que $\lambda^{\mathbf{A}} = 0{,}8$ es suficiente para lograr la convergencia. Los valores de $\lambda^{\mathbf{A}} > 0{,}8$ hacen que el solucionador diverja. Por el contrario, para $\lambda^{\mathbf{A}} < 0{,}8$ el solucionador aún converge pero el número de iteraciones necesarias para lograr dicha convergencia es mayor, y por lo tanto es ineficiente. Se reitera aquí que no hay un valor universal de $\lambda^{\mathbf{A}}$ para el cual el solucionador converja con el menor número posible de iteraciones, ya que depende en gran medida de las características del caso. Además, a diferencia de (Saravia, 2021), no estamos utilizando una función de escala en el parámetro χ para relajar los campos, sino únicamente factores de relajación para estabilizar la ecuación.

Volviendo al método de discretización de gradiente, cuando se utilizan el método de mínimos cuadrados y valores de $\lambda^{\mathbf{A}} \sim \mathcal{O}(1 \times 10^{-3})$ el solucionador todavía diverge. Por lo tanto, por las razones descritas en los párrafos anteriores, actualmente no es posible encontrar convergencia mientras se utiliza el método de mínimos cuadrados, incluso aplicando una sub-relajación extrema.

En tercer lugar, resolver el RHS de la ecuación de balance en términos de $\mathrm{div}\,(\chi\, 2 \times \mathrm{skew}\,(\nabla \mathbf{A}))$ en lugar de $\nabla \times (\chi\, \mathbf{B})$ se ha mostrado necesario para que el solucionador converja. Este es un resultado que se ha encontrado experimentalmente, y que solo es cierto para mallas no ortogonales. Es decir, los casos uniformes y ortogonales en (Riedinger y Saravia, 2023; Saravia, 2021) implementaron la forma convencional del RHS de la ecuación de balance, $\nabla \times (\chi\, \mathbf{B})$, y la convergencia se logró de todos modos.

7 Conclusiones

Presentamos un análisis de distribuciones discontinuas de densidad de flujo magnético dentro de medios curvados discretizados con mallas no ortogonales. Se ha desarrollado y probado una formulación donde la ecuación de balance se basa en el potencial vectorial magnético en diversos casos numéricos con mallas triangulares. Dado que el potencial magnético vectorial de interfaz es proporcionado por la solución en lugar de ser impuesto como una condición de contorno interna, se han desarrollado experimentos numéricos para evaluar la densidad de flujo magnético en la interfaz multimedia.

Una de las principales conclusiones que se han alcanzado a partir de los experimentos numéricos es cómo lograr la convergencia dentro del marco del Volumen Finito. Hemos determinado que la relajación de campo es imperativa para encontrar una solución; precisamente, es necesario relajar el RHS de la Ecuación 3.42 con un factor $\lambda^{\mathbf{A}} \sim \mathbb{O}(0{,}8)$ en lugar de escalar el parámetro χ como se ha hecho en (Saravia, 2021) para mallas ortogonales. Además, relajar la densidad de flujo magnético en términos de $\nabla \cdot (\chi\, 2 \times \text{skew}\,(\nabla \mathbf{A}))$ en lugar de $\nabla \times (\chi\, \mathbf{B})$ ha demostrado ser más efectivo. Esto se ha verificado comparando los valores del centro de celda del marco de Volumen Finito con los valores interpolados dados por el solucionador FE COMSOL.

Una vez que las predicciones fueron verificadas y contrastadas con el solucionador FE, se realizaron más experimentos numéricos de referencia para probar el enfoque en geometrías curvas más complejas y con interacciones de campo más fuertes. Los resultados sugieren que el solucionador es capaz de manejar este tipo de geometrías, encontrando soluciones convergentes independientemente de las técnicas de malla utilizadas para discretizar el espacio curvo.

Los efectos de la no ortogonalidad se han evaluado numéricamente comparando las parcelas de convolución integral de línea de superficie con mallas conformales triangulares y cuadráticas dentro de la misma geometría. Aunque esta comparación es cualitativa en lugar de cuantitativa, es útil para obtener una idea general de la precisión del solucionador con mallas no ortogonales. Una evaluación más detallada de la influencia de las mallas no ortogonales está siendo desarrollada por los autores y se dejará como trabajo futuro.

El enfoque ha demostrado ser efectivo para el cálculo de distribuciones discontinuas de campo magnético dentro de superficies curvas. Además, los requisitos de convergencia que se han encontrado son fáciles de implementar y no han mostrado un deterioro en la efectividad computacional. Por lo tanto, se concluye que usar un esquema de VF producirá resultados similares a un solucionador FE.

Apéndice

Identidades vectoriales A

Sea $c \in \mathbb{R}$ un escalar, $\mathbf{a}, \mathbf{b} \in \mathbb{R}$ vectores, $\mathbf{e}_n$ un vector unidad normal, $\mathbf{T}$ un tensor, las siguientes identidades se cumplen:

$$\nabla \times (\times \mathbf{a}) = \nabla (\nabla \cdot \mathbf{a}) - \nabla^2 \mathbf{a}; \tag{A.1}$$

$$\nabla \times (c\,\mathbf{a}) = c\,\nabla \times \mathbf{a} + \nabla c \times \mathbf{a}; \tag{A.2}$$

$$\nabla \cdot (\phi \mathbf{T}) = \phi \nabla \cdot \mathbf{T} + \mathbf{T}^T \nabla \phi; \tag{A.3}$$

$$\nabla \cdot (\nabla \mathbf{a}) = \nabla^2 \mathbf{a}; \tag{A.4}$$

$$\nabla \cdot \left((\nabla \mathbf{a})^T \right) = \nabla (\nabla \cdot \mathbf{a}). \tag{A.5}$$

Código B

B.1. `multiRegionMagneticFoam.C`

Listing B.1.1: `multiRegionMagneticFoam` Parte 1

```
#include "fvCFD.H"
#include "electromagneticConstants.H"
#include "regionProperties.H" // Allows creation of object to hold region properties
#include "solidMagnetostaticModel.H"
#include "simpleControl.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
                #include "setRootCaseLists.H" // Add some list functionality to the old
                ↪  setRootCase
                #include "createTime.H"
                #include "createMeshes.H"
                #include "createFields.H"
                #include "createMagneticControls.H"
                #include "readMagneticControls.H"

                // Use the simple solver to simplify convergence control (only 1 control for
                ↪  all regions)
                fvMesh& mesh = allRegions[0];
                simpleControl simple(mesh);

                int nCorrNonOrth(simple.nCorrNonOrth());


                // Outer loop for ramping chi
                while (simple.loop(runTime))
                {
            /* Store the initial iter of the magnetic field */
                        forAll(allRegions, i)
                        {
                                //Bs[i].storePrevIter();
                                MIs[i].storePrevIter();
                        }

                        Info<< "Iteration: " << runTime.value() << nl << endl;

                        // Scale chi
                        if (scaleChi)
                        {
                                #include "scaleChi.H"
                        }

```

Listing B.1.2: `multiRegionMagneticFoam` Parte 2

```
                        // Set the tolerance for the last two iterations (access through the
                        ↪  mesh)
                        if (correctFinal)
                        {
                                if (runTime.value() >= runTime.endTime().value() -
                                ↪  finalIterations)
                                {
                                        Info<<
                                        ↪  "Switching to final correction solution parameters..."
                                        ↪  << nl << endl;
                                        solverControl.set("tolerance", finalTolerance);
                                        solDict.regIOobject::write(true); // Call the base
                                        ↪  class method
                                        nCorrNonOrth = finalCorrectors;
                                        correctFinal = false;
                                }
                        }
                        for (int k = 1; k<=nCorrNonOrth; k++)
                        {
                                // Solve regions
                                Info<< "Non-orthogonal corrector: " << k << nl << endl;
                                forAll(allRegions, i)
                                {
                                        //Bs[i].relax();
                                        MIs[i].relax();
                                }
                                forAll(allRegions, i)
                                {
                                Info<< nl <<"--> Solving region: " << allRegions[i].name()
                                ↪  << endl;
                                        #include "setRegionFields.H"
                                        #include "solveRegion.H"
                                        #include "calculateFields.H"

                                        if (calcForces)
                                        {
                                                Info<< nl <<"--> Calculating forces ..."<<
                                                ↪  endl;
                                                #include "calculateForces.H"
                                        }
                                }
                        }

                        runTime.write();

                        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
                                << "  ClockTime = " << runTime.elapsedClockTime() << " s"
                                << nl << endl;
                return 0;
}
```

Listing B.1.3: solidMagnetostaticModel.C Parte 1

```
// * * * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * * * //

namespace Foam
{
    defineTypeNameAndDebug(solidMagnetostaticModel, 0);
}


// * * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * //

Foam::solidMagnetostaticModel::solidMagnetostaticModel
(
    const volVectorField& B
)
:
    IOdictionary
    (
        IOobject
        (
            "magneticProperties",
            B.time().constant(),
            B.db(),
            IOobject::MUST_READ_IF_MODIFIED,
            IOobject::NO_WRITE
        )
    ),

    permeabilityModelPtr_
          (
                  permeabilityModel::New
                  (
                          "mur",
                          subDict("permeability"),
                          B
                  )
          ),

    magnetizationModelPtr_
          (
                  magnetizationModel::New
                  (
                          "M",
                          subDict("magnetization"),
                          B
                  )
          ),

    currentModelPtr_
    (
      currentModel::New
      (
        "J",
        subDict("current"),
        B
      )
    )
{}
```

Listing B.1.4: `solidMagnetostaticModel.C` Parte 2

```
// * * * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * * * //

Foam::solidMagnetostaticModel::~solidMagnetostaticModel()
{}


// * * * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * * * //

Foam::tmp<Foam::volScalarField>
Foam::solidMagnetostaticModel::mur() const
{
    return permeabilityModelPtr_->mur();
}

Foam::tmp<Foam::volVectorField>
Foam::solidMagnetostaticModel::M() const
{
    return magnetizationModelPtr_->M();
}

Foam::tmp<Foam::volVectorField>
Foam::solidMagnetostaticModel::J() const
{
    Info<<"    point1 !\n"  << endl;

    return currentModelPtr_->J();
}

void Foam::solidMagnetostaticModel::correct()
{
  permeabilityModelPtr_->correct();
        magnetizationModelPtr_->correct();
  currentModelPtr_->correct();
}


bool Foam::solidMagnetostaticModel::read()
{
   return regIOobject::read() &&
           permeabilityModelPtr_->read(subDict("permeability")) &&
           magnetizationModelPtr_->read(subDict("magnetization")) &&
           currentModelPtr_->read(subDict("current"));
}
```

Bibliografía

Argyris, J. (1968). The LUMINA element for the matrix displacement method. *The Aeronautical Journal*, *72*(690), 514-517 (vid. pág. 2).

Argyris, J., & Scharpf, D. (1969). The curved tetrahedronal and triangular elements TEC and TRIG for the matrix displacement method, part I: Small displacements, part II: Large displacements. *The Aeronautical Journal of the Royal Aeronautical Society*, *73*, 55-65 (vid. pág. 2).

Augustyniak, M., & Usarek, Z. (2016). Finite element method applied in electromagnetic NDTE: A review. *Journal of Nondestructive Evaluation*, *35*, 1-15 (vid. pág. 3).

Boularas, A., Clain, S., & Baudoin, F. (2017). A sixth-order finite volume method for diffusion problem with curved boundaries. *Applied Mathematical Modelling*, *42*, 401-422 (vid. pág. 2).

Calhoun, D. A., & Helzel, C. (2010). A finite volume method for solving parabolic equations on logically cartesian curved surface meshes. *SIAM Journal on Scientific Computing*, *31*(6), 4066-4099 (vid. pág. 2).

Caudillo-Mata, L. A., Haber, E., & Schwarzbach, C. (2017). An oversampling technique for the multiscale finite volume method to simulate electromagnetic responses in the frequency domain. *Computational Geosciences*, *21*, 963-980 (vid. pág. 3).

Chassaing, J.-C., Khelladi, S., & Nogueira, X. (2013). Accuracy assessment of a high-order moving least squares finite volume method for compressible flows. *Computers & Fluids*, *71*, 41-53 (vid. pág. 2).

Chung, T. S., & Zou, J. (2001). A finite volume method for Maxwell's equations with discontinuous physical coefficients. *International Journal of Applied Mathematics*, *7*(2), 201-224 (vid. pág. 3).

Ciarlet, P. G., & Raviart, P.-A. (1972). Interpolation theory over curved elements, with applications to finite element methods. *Computer Methods in Applied Mechanics and Engineering*, *1*(2), 217-249 (vid. pág. 2).

Coggon, J. (1971). Electromagnetic and electrical modeling by the finite element method. *Geophysics*, *36*(1), 132-155 (vid. pág. 3).

contributors, W. (2023). *Mesh generation* [[Online; accessed 2-May-2023]]. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Mesh_generation. (Vid. pág. 6)

Costa, R., Clain, S., Loubère, R., & Machado, G. J. (2018). Very high-order accurate finite volume scheme on curved boundaries for the two-dimensional steady-state convection–diffusion equation with Dirichlet condition. *Applied Mathematical Modelling*, *54*, 752-767 (vid. pág. 2).

Costa, R., Clain, S., Machado, G. J., & Nóbrega, J. M. (2022). Very high-order accurate finite volume scheme for the steady-state incompressible Navier–Stokes equations with polygonal meshes on arbitrary curved boundaries. *Computer Methods in Applied Mechanics and Engineering*, *396*, 115064 (vid. pág. 2).

Costa, R., Clain, S., Machado, G. J., Nóbrega, J. M., da Veiga, H. B., & Crispo, F. (2023). Imposing slip conditions on curved boundaries for 3D incompressible flows with a very high-order accurate finite volume scheme on polygonal meshes. *Computer Methods in Applied Mechanics and Engineering*, *415*, 116274 (vid. pág. 2).

Costa, R., Nobrega, J. M., Clain, S., & Machado, G. J. (2019). Very high-order accurate polygonal mesh finite volume scheme for conjugate heat transfer problems with curved interfaces and imperfect contacts. *Computer Methods in Applied Mechanics and Engineering*, *357*, 112560 (vid. pág. 2).

Da Veiga, L. B., Russo, A., & Vacca, G. (2019). The virtual element method with curved edges. *ESAIM: Mathematical Modelling and Numerical Analysis*, *53*(2), 375-404 (vid. pág. 2).

Dassi, F., Fumagalli, A., Losapio, D., Scialò, S., Scotti, A., & Vacca, G. (2021). The mixed virtual element method on curved edges in two dimensions. *Computer Methods in Applied Mechanics and Engineering*, *386*, 114098 (vid. pág. 2).

Demuth, C., Mendes, M. A., Ray, S., & Trimis, D. (2014). Performance of thermal lattice Boltzmann and finite volume methods for the solution of heat conduction equation in 2D and 3D composite media with inclined and curved interfaces. *International Journal of Heat and Mass Transfer*, *77*, 979-994 (vid. pág. 2).

Ferrieres, X., Parmantier, J.-P., Bertuol, S., & Ruddle, A. R. (2004). Application of a hybrid finite difference/-finite volume method to solve an automotive EMC problem. *IEEE Transactions on Electromagnetic Compatibility*, *46*(4), 624-634 (vid. pág. 3).

Greenshields, C., & Weller, H. (2022). *Notes on Computational Fluid Dynamics: General Principles*. CFD Direct Ltd. (Vid. págs. 31, 32, 42, 61).

Griffiths, D. J. (2021). Introduction to Electrodynamics Fourth Edition (vid. págs. 18, 20).

Haber, E., & Ruthotto, L. (2014). A multiscale finite volume method for Maxwell's equations at low frequencies. *Geophysical Journal International*, *199*(2), 1268-1277 (vid. pág. 3).

Jackson, J. D. (1999). Classical electrodynamics. (Vid. pág. 18).

Jin, J.-M. (2015). *The finite element method in electromagnetics*. John Wiley & Sons. (Vid. pág. 3).

Lan, T., & Lo, S. (1996). Finite element mesh generation over analytical curved surfaces. *Computers & Structures*, *59*(2), 301-309 (vid. pág. 3).

Liu, Y., Vinokur, M., & Wang, Z. J. (2006). Spectral (finite) volume method for conservation laws on unstructured grids V: Extension to three-dimensional systems. *Journal of Computational Physics*, *212*(2), 454-472 (vid. pág. 3).

Lo, S. (1988). Finite element mesh generation over curved surfaces. *Computers & structures*, *29*(5), 731-742 (vid. pág. 2).

Lo, S. H., & Wang, W. X. (2005). Finite element mesh generation over intersecting curved surfaces by tracing of neighbours. *Finite elements in analysis and design*, *41*(4), 351-370 (vid. pág. 3).

Maxwell, J. C. (1865). VIII. A dynamical theory of the electromagnetic field. *Philosophical transactions of the Royal Society of London*, (155), 459-512 (vid. págs. 2, 18).

Moukalled, F., Mangani, L., Darwish, M., Moukalled, F., Mangani, L., & Darwish, M. (2016). *The finite volume method*. Springer. (Vid. págs. 23, 27).

Munz, C.-D., Schneider, R., & Voß, U. (2000). A finite-volume method for the Maxwell equations in the time domain. *SIAM Journal on Scientific Computing*, *22*(2), 449-475 (vid. pág. 3).

Niu, K., Li, P., Huang, Z., Jiang, L. J., & Bagci, H. (2020). Numerical methods for electromagnetic modeling of graphene: A review. *IEEE Journal on Multiscale and Multiphysics Computational Techniques*, *5*, 44-58 (vid. pág. 3).

Polycarpou, A. C. (2022). *Introduction to the finite element method in electromagnetics*. Springer Nature. (Vid. pág. 3).

Riedinger, A., & Saravia, M. (2023). A single–region Finite Volume framework for modeling discontinuous magnetic field distributions. *Computers & Structures*, *277*, 106960 (vid. págs. 3, 71, 72, 74, 76, 78).

Sabbagh-Yazdi, S., Ali-Mohammadi, S., & Pipelzadeh, M. (2012). Unstructured finite volume method for matrix free explicit solution of stress–strain fields in two dimensional problems with curved boundaries in equilibrium condition. *Applied Mathematical Modelling*, *36*(5), 2224-2236 (vid. pág. 2).

Sadiku, M. N. (1989). A simple introduction to finite element analysis of electromagnetic problems. *IEEE Transactions on education*, *32*(2), 85-93 (vid. pág. 3).

Sadiku, M. N. (2018). *Numerical techniques in electromagnetics with MATLAB*. CRC press. (Vid. pág. 3).

Salon, S., & Chari, M. (1999). *Numerical methods in electromagnetism*. Elsevier. (Vid. pág. 3).

Saravia, M. (2021). A finite volume formulation for magnetostatics of discontinuous media within a multi-region openfoam framework. *Journal of Computational Physics*, *433*, 110089 (vid. págs. 3, 22, 23, 72-74, 76, 78, 79).

Sarkar, T., Siarkiewicz, K., & Stratton, R. (1981). Survey of numerical methods for solution of large systems of linear equations for electromagnetic field problems. *IEEE Transactions on Antennas and Propagation*, *29*(6), 847-856 (vid. pág. 3).

Senior, T. B., & Volakis, J. L. (1995). *Approximate boundary conditions in electromagnetics*. Iet. (Vid. pág. 2).

Sevilla, R., Fernández-Méndez, S., & Huerta, A. (2011). Comparison of high-order curved finite elements. *International Journal for Numerical Methods in Engineering*, *87*(8), 719-734 (vid. pág. 2).

Sommerfeld, A. (2013). *Electrodynamics: lectures on theoretical physics, vol. 3* (Vol. 3). Academic Press. (Vid. págs. 18, 19).

Wright, L., & Schobeiri, M. (1999). The effect of periodic unsteady flow on aerodynamics and heat transfer on a curved surface (vid. pág. 2).