Implementación de un simulador 2D de elementos finitos en Julia

Mario Núñez Domínguez[(1)], Adrián Amor Martín[(1)], Luis E. García Castillo[(1)]
100429724@alumnos.uc3m.es, [aamor, legcasti]@ing.uc3m.es
[(1)] Dpto. de Teoría de la Señal y de las Comunicaciones. Universidad Carlos III de Madrid. 28911 Leganés (Madrid)

***Abstract*—Este articulo se centra en el desarrollo de un simulador 2D basado en el método de elementos finitos (FEM) implementado en el lenguaje Julia para resolver el campo electromagnético en el interior de una guía de ondas. Este desarrollo es una prueba de concepto para probar el entorno de trabajo y las prestaciones computacionales de Julia, que es una herramienta con cierta expansión en el campo de la computación científica pero no en el del electromagnetismo computacional. En este trabajo se describe la formulación del problema, la metodología de implementación, y se verifica el correcto funcionamiento del código con la resolución de los modos de funcionamiento de una guía de ondas rectangular, que tiene solución analítica.**

I. Introducción

La resolución de modelos electromagnéticos de alta frecuencia es un problema exigente que requiere de la interacción de varias disciplinas como el campo de las matemáticas, de la física o de la computación científica, [1]. Esto hace que algunos avances que se dan en estas disciplinas son especialmente útiles en el campo del electromagnetismo computacional y ayuda a la resolución de sus problemas.

Uno de los principales cambios que se han dado en el ámbito de la computación científica es la aparición de un lenguaje de programación especialmente diseñado para esta disciplina como es Julia, [2]. Julia nace en 2012 con el principal objetivo de disponer de un lenguaje de alto nivel como Matlab o Python (más ágil de depurar y con una menor barrera de entrada) con las prestaciones de un lenguaje compilado clásico como C o Fortran. Para ello, introduce un compilador avanzado (JIT, "Just In Time") junto con mecanismos para la ejecución en paralelo y distribuida y una extensa biblioteca de funciones matemáticas. Este compilador JIT nos permite compilar en tiempo de ejecución ciertas partes del código, para luego poder utilizarlo en futuras peticiones. Esto significa que el compilador produce código máquina específico para la plataforma una vez el programa ya esta en ejecución. Julia, aunque es relativamente nuevo, cuenta con un gran número de bibliotecas y módulos, además de beneficiarse de la capacidad de interoperabilidad con bibliotecas de otros lenguajes, lo que le hace una alternativa atractiva frente a Python ya que su rendimiento se ha demostrado mejor que Python y comparable (en órdenes de magnitud) a C o Fortran, gracias al JIT mencionado anteriormente y a la capacidad de generar código máquina altamente optimizado, [3]. De esta forma, Julia ya se ha hecho un hueco en el ranking de los lenguajes de programación más usados a nivel mundial, [4], y se usa en diferentes disciplinas como biología, [5], aprendizaje máquina, [6], o elementos finitos (en inglés, Finite Element Method, FEM), [7], [8].

Nuestro objetivo en este trabajo es desarrollar un software de elementos finitos desde cero en Julia usando su filosofía de programación. Traducir un código de un lenguaje de programación a otro es relativamente sencillo y puede ser automático, pero para sacar el máximo partido a un lenguaje de programación hay que utilizar su filosofía de programación. En concreto, consideramos especialmente útil para un software FEM la funcionalidad que en Julia describen como *multiple dispatch*, que facilita la expresión de muchos patrones de programación orientada a objetos y funcionales de forma que se puedan integrar de forma nativa múltiples implementaciones del mismo concepto. De esta manera, no se definen las funciones de una sola vez, sino que podemos definirlas por partes según su comportamiento específico. Se puede encontrar un ejemplo en la Figura 1, donde cada caja son diferentes implementaciones del operador suma. Un ejemplo sencillo y útil de esta funcionalidad en un código de elementos finitos es disponer de diferentes implementaciones de la obtención de la matriz de masa, de forma que en función de la familia de funciones de base que se utilice (ya sea por su forma o por su definición intrínseca de las funciones) se devuelva una matriz u otra.

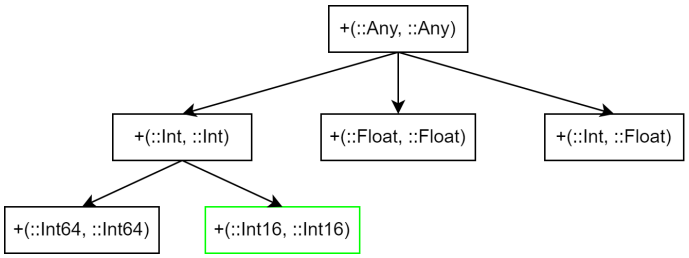

Fig. 1. Ejemplo de la implementación del operador + con la funcionalidad de multiple dispatch en Julia

Como prueba de concepto, el problema electromagnético a resolver en este trabajo es la obtención de los modos de propagación del campo electromagnético en el interior de una guía de ondas, lo que necesita de una formulación en dos dimensiones que incluye muchos de los términos que se necesitan en un software FEM de propósito general de tres dimensiones y que dispone de solución analítica para validar el desarrollo del código, [9], [10].

El artículo se ha dividido en las siguientes secciones: en primer lugar, se define la formulación en dos dimensiones para resolver el problema de propagación guiada en la Sección II; a continuación, se dan algunos detalles de programación en Julia en la Sección III; y finalmente, se valida el desarrollo en la Sección IV. Las conclusiones y líneas futuras del trabajo se exponen en la Sección V.

II. Formulación

A. Modelado electromagnético

Para obtener el campo electromagnético en el interior de un medio guiado se necesita resolver un problema de autovalores que dependen únicamente de la sección transversal del problema, resultando en un simulador 2D. En función de si el modo de propagación es Transversal Eléctrico (TE) o Transversal Magnético (TM), el campo longitudinal será magnético o eléctrico, respectivamente, y a partir de él se puede derivar la componente transversal de ese campo resolviendo todo el problema, [10], [11].

Partiendo de las ecuaciones de Maxwell asumiendo medios lineales, homogéneos e isótropos, llegamos a las ecuaciones de Helmholtz para el campo eléctrico y para el campo magnético,

$$\nabla^2 \vec{E} - \gamma_0^2 \vec{E} = 0 \tag{1a}$$

$$\nabla^2 \vec{H} - \gamma_0^2 \vec{H} = 0, \tag{1b}$$

donde $\gamma_0^2 = -\omega^2 \mu_0 \varepsilon_0$, y es la constante de propagación en un medio sin obstáculos; $\vec{E}$ es la intensidad de campo eléctrico; y $\vec{H}$ corresponde a la intensidad de campo magnético.

Tras esto aplicamos la separación entre los componentes longitudinales, $\vec{E_z}$, y transversales, $\vec{E_t}$, y también el método de separación de variables, obteniendo

$$\nabla_t^2 F - \gamma_c^2 F = 0, \tag{2}$$

donde F es una función de forma que puede representar campo eléctrico (en el caso de modos TM) o campo magnético (en el caso de modos TE), ∇_t es el operador ∇ restringido a las componentes transversales, y $\gamma_c^2 = \omega_c^2 \mu\varepsilon$, siendo ω_c el valor de la pulsación de corte para el modo. Esta pulsación hace que la propagación del modo se defina con

$$\gamma_c^2 + \gamma^2 = \gamma_0^2. \tag{3}$$

B. Funciones de base

En FEM, un elemento finito se define por su geometría, el espacio de funciones y los grados de libertad asociados. De esta forma, se definen unas funciones de base asociadas al elemento finito que aproximan la solución, en este caso, de F, que es un valor escalar. Por tanto, el espacio de funciones en el que se definen las funciones de base en un dominio Ω es

$$H^1(\Omega) := \left\{ v \in L_2(\Omega) \middle| \nabla v \in [L_2(\Omega)]^3 \right\}, \tag{4}$$

donde L_2 es el espacio de Hilbert de funciones cuadradas sobre Ω.

Los elementos que escogemos en este análisis son triángulos por lo que nos resultará muy útil definir funciones de base afines que, en el caso de primer orden para un triángulo definido en el plano ξ-η, son

$$\phi_1 = 1 - \xi - \eta \tag{5a}$$

$$\phi_2 = \xi \tag{5b}$$

$$\phi_3 = \eta. \tag{5c}$$

Estas funciones cumplen la propiedad de funciones interpolatorias, i.e.,

$$\phi_i(\xi_i, \eta_i) = 1, \quad \phi_i(\xi_i, \eta_i) = 0, \quad \forall i \neq j. \tag{6}$$

TABLE I
Funciones de base escalares para elementos triangulares.

Orden	Funciones de base	Asociaciones
1	ϕ_i	$\{i\}$
2	$\phi_i\phi_j$	$\{ij\}$
3	$\phi_i\phi_j(\phi_i - \phi_j)$	$\{ij\}$
	$\phi_i\phi_j\phi_k$	$\{ijk\}$

A la hora de usar funciones de orden más elevado, existen dos opciones: usar funciones interpolatorias, que cumplen (6), o usar funciones jerárquicas, que significa que las funciones de orden superior contienen a las funciones de orden inferior. La principal desventaja de este último tipo de funciones es su peor condicionamiento aunque permiten refinados de tipo hp. En este trabajo, se van a utilizar las funciones de [12], que son jerárquicas, y están incluidas en la Tabla I. En la columna asociaciones se indican los índices de los vértices involucrados, de forma que i, ij, y ijk indican que las funciones son nodales, de arista, y de cara, específicamente. En la Figura 2 se indican la distribución de las funciones de orden 2 como ejemplo, así como la numeración de las funciones incluida en el código.

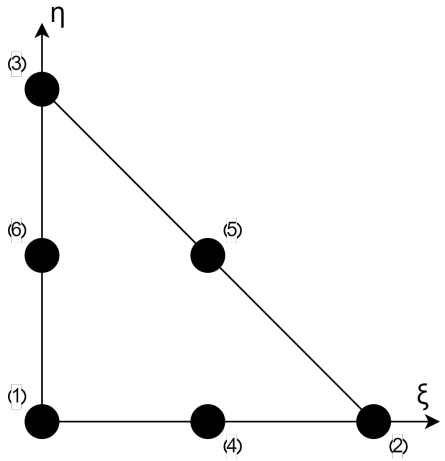

Fig. 2. Funciones de base de segundo y tercer orden sobre un triángulo

C. Análisis de elementos finitos en 2D

Para aplicar FEM necesitamos añadir a (4) las condiciones de contorno de nuestro problema, obteniendo la formulación variacional

$$\nabla_t^2 F - \gamma_c^2 F = 0 \quad \text{en} \quad \Omega \tag{7a}$$

$$F = 0 \quad \text{en} \quad \Gamma_1 \tag{7b}$$

$$\hat{n} \cdot \nabla_t F = 0 \quad \text{en} \quad \Gamma_2, \tag{7c}$$

donde F puede ser tanto F_E como F_H, dependiendo del modo elegido. Además, $\partial\Omega = \Gamma_1 \cup \Gamma_2$, considerando las condiciones de contorno esenciales (Dirichlet) y naturales (Neumann), y $\hat{n}$ es el vector unitario normal hacia fuera de $\partial\Omega$.

A continuación, multiplicamos (7a) por una función test w_i y la integramos sobre Ω:

$$\int_\Omega w_i \cdot (\nabla_t^2 F - \gamma_c^2 F)\, d\Omega = 0 \tag{8}$$

Después, teniendo en cuenta la identidad de Green:

$$\iiint_V (u\nabla^2 v + \nabla u \cdot \nabla v)\, dV = \iint_S (u\nabla v - v\nabla u)\, dS \quad (9)$$

Y adaptándola a nuestro problema 2D:

$$\int_\Omega w_i \nabla_t^2 F\, d\Omega = \oint_\Gamma (w_i \nabla_t F \cdot \hat{n})\, d\Gamma - \int_\Omega \nabla_t w_i \cdot \nabla_t F\, d\Omega \quad (10)$$

Donde $\oint_T (w_i \nabla_t F \cdot \hat{n}) dt$ es la integral sobre el contorno de la superficie Ω y, por lo tanto, fijándonos en las condiciones de contorno, se anula en ambos casos. Obtenemos la siguiente ecuación:

$$-\int_\Omega \nabla_t w_i \cdot \nabla_t F\, d\Omega - \gamma_c^2 \int_\Omega w_i \cdot F\, d\Omega = 0, \quad (11)$$

de igual forma que se obtiene en [10, Apéndice F.2.2, Sección 3.3.3.1]

Para llegar a un sistema algebraico de ecuaciones debemos realizar un procedimiento de ensamblando considerando que la solución $F(r)$ se expande en términos de las funciones de base:

$$F(\vec{r}) = \sum_{j=1}^{N_n} F_j \phi_j(\vec{r}), \quad (12)$$

que si incluimos en (11) y aplicamos el método de Galerkin eligiendo como funciones de test las mismas funciones de base ($w_i(\vec{r}) = \phi_i(\vec{r})$), llegamos a un sistema de ecuaciones $A\vec{x} = 0$ donde

$$A_{ij} = \int_\Omega \nabla_t \phi_i \cdot \nabla_t \phi_j\, d\Omega - k_c^2 \int_\Omega \phi_i \cdot \phi_j\, d\Omega \quad (13)$$

Los elementos A_{ij} de la matriz del sistema se calculan evaluando la integral (13) sobre el dominio Ω. A la hora de ejecutar el problema, debemos romper esta integral en integrales sobre cada elemento Ω^e, y sumar todas las contribuciones de todos los elementos (ensamblar en el sentido de los elementos finitos), i.e.,

$$A_{ij} = \sum_{e=1}^{N_e} \int_{\Omega^e} \nabla_t \phi_i \cdot \nabla_t \phi_j\, d\Omega^e - k_c^2 \int_{\Omega^e} \phi_i \cdot \phi_j\, d\Omega^e, \quad (14)$$

con N_e es el número total de elementos.

Por temas de practicidad y eficiencia computacional, definiremos las funciones en el elemento de referencia y las trasladaremos a cada elemento real por medio del cálculo del jacobiano, para lo que necesitaremos la definición de funciones isoparamétricas que, en el caso de elementos rectos, pueden ser directamente (5).

III. Esquema del código

El esquema seguido en el código implementado para la creación del simulador es el siguiente:

1) Leer informacion de entrada.
2) Definir una estructura de datos en la que se incluyan toda la información de los elementos finitos.
3) Calcular la matriz A para todos los nodos en el elemento de referencia con integracion numerica
4) Ensamblar las matrices locales A para cada elemento en el sentido de los elementos finitos
5) Obtener los autovalores y autovectores del problema que supondran γ_c y F respectivamente de cada modo.

Para leer la información de entrada, que supone la definición de los propiedades electromagnéticas para cada elemento y definir un mallado en el que aplicar (14), usamos la API de Gmsh, [13], que nos proporciona la información geométrica de cada elemento. Una de estas mallas se representa, a modo de ejemplo, en la Figura 3.

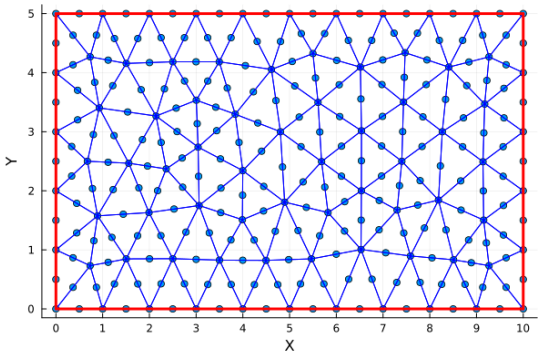

Fig. 3. Mallas desestructuradas obtenidas con la API de gmsh en Julia

IV. Resultados

En este trabajo se parte de (7a). Nuestro objetivo es poder calcular k_c^2 con la mayor exactitud posible. Tras aplicar los pasos generales del método de elementos finitos, llegamos a la denominada (11). Después de la discretización, usando nuestras funciones de base, hemos resuelto el problema general de autovalores,

$$A = S - k_c^2 M = 0 \quad (15)$$

Donde k_c^2 son los autovalores del problema en cuestión, mientras que S y M son las matrices de rigidez y masa.

El error relativo se ha calculado de la siguiente forma:

$$\varsigma_r = \frac{k_{c,\text{ref}}^2 - k_{c,\text{FEM}}^2}{k_{c,\text{ref}}^2} \quad (16)$$

Donde $k_{c,\text{ref}}^2$ es el número de onda analítico del modo propio y $k_{c,\text{FEM}}^2$ es la misma magnitud proporcionada por el código.

Como estructura de estudio para la validación de resultados, se ha utilizado una guía rectangular WR-90, i.e., $a = 0.02286\,\text{m}$ y $b = 0.01016\,\text{m}$, siendo posible calcular el valor analítico con, [14],

$$k_{c,\text{ref}}^2 = \left(\frac{m\pi}{a}\right)^2 + \left(\frac{n\pi}{b}\right)^2, \quad \text{para } m, n \geq 0 \quad (17)$$

Se realiza un análisis de convergencia donde se refina uniformemente la malla tanto en h (presente en el eje de ordenadas) como en p (presente en las diferentes figuras), representando en la Figura 4 el error relativo ς_r para los modos TE y en la Figura 5, el error relativo ς_r para los modos TM considerando la constante de onda del modo fundamental. En línea discontinua se muestra el error teórico $\mathcal{O}(h^{2p})$, [10], tomando como referencia el punto de la discretización más fina.

Se puede observar que el error sigue líneas rectas suaves que aproximan adecuadamente el valor de k_c^2, siguiendo asintóticamente el valor teórico del error. El valor concreto de las pendientes obtenidas se muestra en la Tabla II.

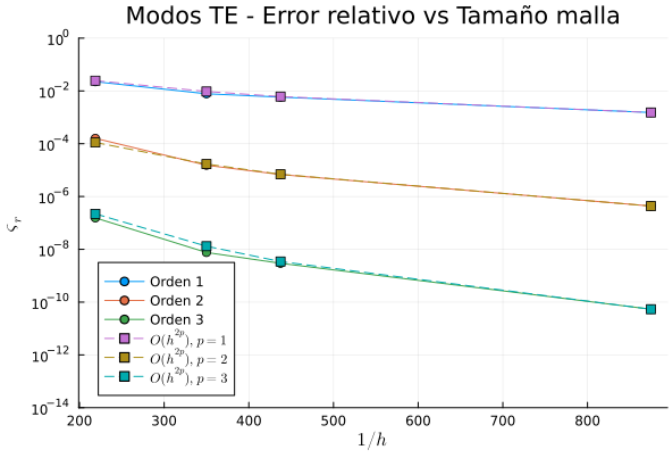

Fig. 4. Índices de convergencia para el modo fundamental TE_{01} para diferentes órdenes de aproximación y diferentes mallas para una guía rectangular

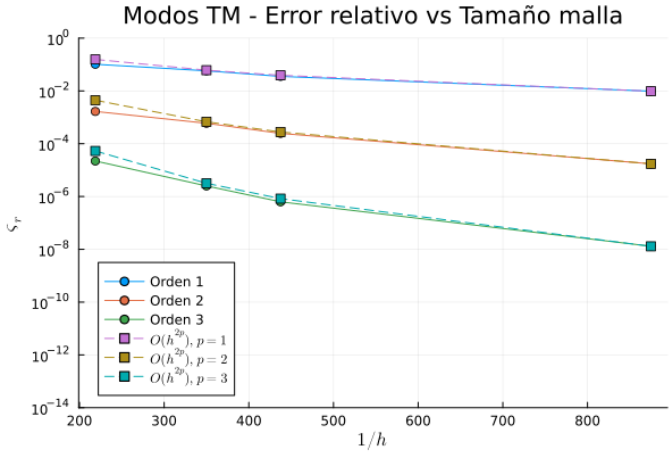

Fig. 5. Índices de convergencia para el modo fundamental TM_{11} para diferentes órdenes de aproximación y diferentes mallas para una guía rectangular

V. Conclusiones y líneas futuras

En este artículo se ha desarrollado desde cero un simulador 2D de elementos finitos que resuelve un problema de autovalores cuya principal aplicación está en el análisis de estructuras de guiaonda. Este software está integrado con librerías de Julia y no depende de lenguajes externos, habiendo integrado el mallador en el flujo de trabajo gracias a una API ya desarrollada en Julia de un mallador open-source como es Gmsh.

Se ha validado el resultado de este simulador por medio de una estructura analítica y se han obtenido resultados satisfactorios, por lo que se puede concluir que el desarrollo de la prueba de concepto es exitoso. Este trabajo abre la puerta a hacer comparaciones de prestaciones con otros lenguajes de programación, a usar este software de simulación aprovechando las principales ventajes de Julia como es *multiple dispatch* (por medio del uso simultáneo en una estructura de triángulos y cuadriláteros, por ejemplo), y a utilizarlo como base para alimentar modelos de Inteligencia Artificial donde se utilicen las librerías ya desarrolladas de aprendizaje máquina de forma nativa con un software programado integralmente en Julia. Este código se ha desarrollado siguiendo la filosofía de ciencia abierta, por lo que está disponible en

TABLE II
Pendientes modos TE.

Modos TE		**Modos TM**	
Orden 1	1.9319	Orden 1	1.8626
Orden 2	3.9639	Orden 2	3.8263
Orden 3	5.7751	Orden 3	5.5963

https://github.com/Mario22-MND/Codigo-TFG.

References

[1] N. Marsic, "Efficient methods for large-scale time-harmonic wave simulations," Ph.D. dissertation, Université de Liège, 2016.

[2] J. D. Team, *The Julia Programming Language*, 2024, accessed on April 6, 2024. [Online]. Available: https://docs.julialang.org/en/latest/

[3] A. J. Weiss and A. Z. Elsherbeni, "Performance of matlab and python for computational electromagnetic problems," *Applied Computational Electromagnetics Society Journal*, vol. 35, 2020.

[4] P. Krill, "Julia language cracks top 20 in tiobe popularity index," *InfoWorld*, 2023. [Online]. Available: https://www.infoworld.com/article/3704154/julia-language-cracks-top-20-in-tiobe-popularity-index.html

[5] E. Roesch, J. G. Greener, A. L. MacLean, H. Nassar, C. Rackauckas, T. E. Holy, and M. P. Stumpf, "Julia for biologists," *Nature methods*, vol. 20, no. 5, pp. 655–664, 2023.

[6] K. Gao, G. Mei, F. Piccialli, S. Cuomo, J. Tu, and Z. Huo, "Julia language in machine learning: Algorithms, applications, and open issues," *Computer Science Review*, vol. 37, p. 100254, 2020.

[7] S. Badia and F. Verdugo, "Gridap: An extensible finite element toolbox in julia," *Journal of Open Source Software*, vol. 5, no. 52, p. 2520, 2020.

[8] F. Verdugo and S. Badia, "The software design of gridap: a finite element package based on the julia jit compiler," *Computer Physics Communications*, vol. 276, p. 108341, 2022.

[9] P. Monk, "Finite element methods for maxwell's equations," *Oxford University Press*, 2003.

[10] M. Salazar-Palma, T. K. Sarkar, L. E. García-Castillo, T. Roy, and A. R. Djordjevic, *Iterative and Self-Adaptive Finite-Elements in Electromagnetic Modeling*. Norwood, MA: Artech House Publishers, Inc., 1998.

[11] J. P. de la Vega, "Propagación de ondas guiadas," vol. 4, pp. 1–10, 21–38, 1983.

[12] P. Ingelström, "A new set of h(curl)-conforming hierarchical basis functions for tetrahedral meshes," vol. 54, 2006.

[13] C. Geuzaine and J.-F. Remacle. (2023) Gmsh - a 2d and 3d finite element mesh generator. Sitio web consultado el 24 de septiembre de 2023. [Online]. Available: https://gmsh.info/

[14] J. Jin, "The finite element method in electromagnetics," vol. 2.