
This is the **published version** of the bachelor thesis:

del Rio Toledano, Javier. *Aceleración del software de diseño de filtros de microondas mediante GPUs y diferenciación automática*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Enginyeria Informàtica)

This version is available at <https://ddd.uab.cat/record/334413>

under the terms of the  license.

Aceleración del software de diseño de filtros de microondas mediante GPUs y diferenciación automática

Javier del Río Toledano

30 de junio de 2026

Resumen– Este trabajo presenta la aceleración de un motor de síntesis y optimización de filtros de microondas mediante vectorización matricial y diferenciación automática (AD). Frente a los enfoques secuenciales clásicos que emplean aproximaciones por diferencias finitas, la herramienta propuesta agrupa los datos en tensores y utiliza retropropagación analítica en MATLAB, permitiendo la evaluación simultánea de lotes de filtros. Los resultados demuestran una concordancia electromagnética exacta con el simulador original, logrando una aceleración en CPU a partir de 100 filtros. Asimismo, la implementación en GPU para optimización por gradiente obtiene un rendimiento superior en un factor de $3\times$ respecto a la CPU, alcanzando un *speedup* de más de $120\times$ frente al método secuencial tradicional con diferencias finitas. Esto valida la viabilidad del motor desarrollado para tareas de optimización global masiva y diseño en tiempo real.

Palabras clave– Síntesis de filtros, diferenciación automática, vectorización por lotes, aceleración por GPU, MATLAB.

Abstract– This project presents the acceleration of a microwave filter synthesis and optimization engine using matrix vectorization and automatic differentiation (AD). In contrast to classical sequential approaches utilizing finite difference approximations, the proposed tool structures filter data into tensors and employs analytical backpropagation in MATLAB to evaluate batches of filters simultaneously. The results show an exact electromagnetic agreement with the original simulator, achieving a CPU *speedup* for batches as small as 100 filters. Furthermore, the GPU implementation for gradient-based optimization achieves a $3\times$ *speedup* over the CPU, reaching a total *speedup* of over $120\times$ compared to the traditional sequential finite difference method. These findings validate the developed engine's capability for massive global optimization tasks and real-time microwave design.

Keywords– Filter synthesis, automatic differentiation, batch vectorization, GPU acceleration, MATLAB.



1 INTRODUCCIÓN Y CONTEXTO

EN EL DISEÑO de sistemas de telecomunicaciones de cualquier tipo y mediante cualquier medio, los filtros juegan un papel fundamental. Son los dispositivos encargados de seleccionar la parte del espectro radioeléctrico con el que queremos trabajar o, lo que es lo mismo, de eliminar las frecuencias no deseadas. Estos elementos se encuentran en todo tipo de dispositivos y aplicaciones, desde los teléfonos móviles hasta los sistemas de radar. Su di-

seño y fabricación forman parte fundamental de la industria como de la investigación de sistemas de telecomunicación.

Dentro de todos los tipos de filtros, nos enfocamos en los filtros acústicos [1]. Su principio de funcionamiento se basa en la transformación de la onda electromagnética en una onda mecánica que se realiza mediante materiales piezoeléctricos que permiten la conversión en ambos sentidos. La conveniencia de este cambio reside en que, al transferirse la onda del aire a un medio sólido, la velocidad de propagación en este nuevo medio es mucho menor, pero su frecuencia se mantiene constante, por lo tanto, para satisfacer la relación $v = f \cdot \lambda$ la longitud de la onda también se reduce, pudiendo fabricar filtros significativamente más pequeños que la longitud de onda de la señal original. Este tipo de filtros son de especial interés en dispositivos portátiles y ligeros para los cuales un filtro electromagnético de cavidades no es una opción viable.

• E-mail de contacte: javier.delrio@uab.cat
 • Menció: Tecnologies de la Informació
 • Treball tutoritzat per: Pedro Antonio de Paco Sánchez (Departament d'Enginyeria de Telecomunicacions)
 • Curs 2025/26

En este trabajo se aborda el problema del diseño y síntesis de filtros desde un punto de vista computacional y de arquitectura de *software*. El diseño de filtros acústicos admite gran variedad de acercamientos, en nuestro caso, nos centraremos en una secuencia de pasos completamente analítica a partir de las especificaciones del mismo. A partir de las especificaciones se genera un filtro prototipo paso bajo (LPP, del inglés, *Low-Pass Prototype*) que luego se puede trasladar a la frecuencia de interés y al ancho de banda deseado.

El objetivo original consistía en optimizar un *software* de síntesis de multiplexores. Un multiplexor en radiofrecuencia es un circuito que separa y/o combina señales de distintas frecuencias, es decir, un mismo dispositivo filtra múltiples frecuencias de interés. Al considerarse un objetivo demasiado ambicioso, se ha preferido trabajar con filtros de dos puertos, una entrada y una salida, ya que son la base de los multiplexores. Si los resultados de este trabajo son prometedores, parte del esfuerzo y del conocimiento se podrá trasladar a ese problema.

El *software* original (Baldr de ahora en adelante) [2] es funcional y completo y ha sido nuestra referencia en cuanto a rendimiento y resultados se refiere. Para tratar de mejorar el rendimiento de este programa se plantean diversas estrategias, entre ellas, el uso de *hardware* de cómputo paralelo como son las Unidades de Procesamiento Gráfico (GPU). Las GPUs presentan un elevado número de núcleos de procesamiento, pero más sencillos que si los comparamos con una Unidad de Procesamiento Central (CPU). Este elevado número de núcleos permite la ejecución en paralelo de una operación sobre un conjunto de datos, teniendo el potencial para acelerar significativamente cualquier programa cuyos datos se encuentren convenientemente estructurados. De manera natural, este no es el caso de las operaciones y datos del problema a tratar lo que requiere un replanteamiento del *software*.

El resto del documento se estructura detallando los objetivos y el alcance en la Sección 2, el estado del arte en la Sección 3, la metodología y técnicas utilizadas en la Sección 4, y la evaluación de resultados en la Sección 5. Finalmente, las conclusiones se presentan en la Sección 6.

2 OBJETIVOS Y ALCANCE

2.1. Objetivos

El objetivo principal de este trabajo es reducir de forma sustancial el tiempo de ejecución del *software* original Baldr garantizando la precisión y validez de los resultados obtenidos. Se estableció un objetivo cuantitativo inicial de aceleración (*speedup*) mínimo de 2.

También existen una serie de objetivos secundarios específicos con respecto al código y el trabajo realizado:

1. **Orientación a objetos:** Crear un nuevo código fuente orientado a una arquitectura orientada a objetos (OOP) que sea extensible y mantenible.
2. **Paralelización:** Implementar la representación adecuada de los datos, matrices y tensores, que permita la paralelización mediante el uso de GPUs.

3. **Diferenciación Automática:** Sustituir el uso de diferencias finitas por la diferenciación automática para el cálculo de gradientes en la fase de optimización.

2.2. Alcance

El trabajo se centra en el *software* Baldr [2] que se utilizará como métrica para el análisis de resultados en cuanto a rendimiento, completitud y precisión se refiere. Se ha tratado de implementar el ciclo completo de síntesis, pero no se ha alcanzado, faltando la conversión a la tecnología acústica y la conversión a paso banda del filtro LPP. La razón de no alcanzar a implementar la síntesis completa se ha debido a limitaciones temporales y a subestimar el esfuerzo requerido para este proyecto.

3 ESTADO DEL ARTE

En el estado del arte tenemos el estudio de la síntesis de filtros de manera analítica, así como el uso de técnicas de optimización basadas en el descenso de gradiente como diferenciación automática.

3.1. Síntesis de filtros

El proceso de síntesis utilizado se puede dividir en grandes bloques conceptuales que pueden ser intercambiados por otros que cumplan la misma función. El marco conceptual e histórico de la síntesis de matrices de acoplamiento de filtros está extensamente documentado en la literatura clásica [3]. Por ejemplo, el primer paso es la construcción de los polinomios característicos que son los que definen su comportamiento. En nuestro caso se han utilizado los polinomios de Chebyshev [4], pero se podrían haber construido otros como los elípticos. Aquí se presenta una selección para cada bloque que se puede observar en la Figura 1.

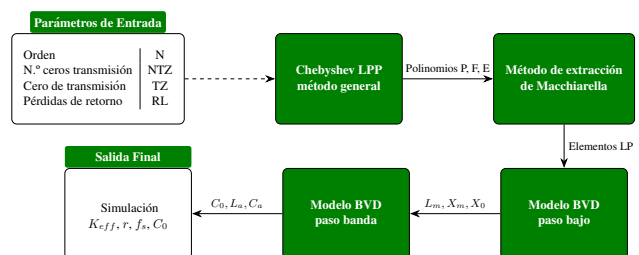


Fig. 1: Diagrama de bloques del proceso de síntesis de filtros acústicos

3.1.1. Polinomios característicos

El primer bloque del proceso de síntesis toma como entrada las especificaciones del filtro y produce como salida un conjunto de tres polinomios y factores de escala que describen matemáticamente su comportamiento frecuencial. Estos parámetros son:

- N : orden del filtro (número de polos), que determina el número de elementos y, por ende, el tamaño del filtro.
- N_{TZ} : número de ceros de transmisión, que son frecuencias a las que el filtro atenúa completamente la señal.

- $\{TZ\}$: las posiciones de dichos ceros en el eje de frecuencias.
- RL : pérdida de retorno (*Return Loss*), que cuantifica cuánta señal se refleja en la banda de paso y determina el nivel de rizado permitido.

El motivo y la manera de contruir estos polinomios se explican en [4]. Su construcción es algo extensa y no es el objetivo de este trabajo. El proceso construye tres polinomios en la variable compleja s , que es la variable del espacio de frecuencias complejas. El resultado es el siguiente:

- $P(s)$: polinomio de transmisión, cuyos **raíces** son exactamente los ceros de transmisión $\{TZ_i\}$. Determina en qué frecuencias el filtro bloquea completamente la señal.
- $F(s)$: polinomio de reflexión, construido para garantizar el comportamiento de tipo Chebyshev en la banda de paso. Sus coeficientes se calculan de forma que la función $|F(j\omega)/E(j\omega)|^2$ oscila de forma equirrizada entre 0 y un valor máximo fijado por RL .
- $E(s)$: polinomio de Hurwitz, denominador común. Se obtiene a partir del producto $\varepsilon_R P(s) + \varepsilon F$, pasando todas las raíces al semiplano izquierdo del plano complejo, lo que garantiza la estabilidad del sistema (equivalente a la condición de causalidad en sistemas digitales).

Además se calculan los escalares ε y ε_R de normalización. Con estos elementos, los parámetros de dispersión (*S-parameters*) que describen la respuesta del filtro se expresan como:

$$S_{11}(s) = \frac{F(s)}{\varepsilon_R \cdot E(s)}, \quad S_{21}(s) = \frac{P(s)}{\varepsilon \cdot E(s)} \quad (1)$$

donde S_{11} es la fracción de señal reflejada y S_{21} la fracción transmitida al otro extremo del filtro. El resto del proceso de síntesis toma $\{P, F, E, \varepsilon, \varepsilon_R\}$ como punto de partida para extraer los valores de los componentes físicos del circuito.

3.1.2. Extracción de elementos

El segundo bloque toma como entrada los polinomios $\{P, F, E, \varepsilon, \varepsilon_R\}$ obtenidos en el paso anterior y produce como salida los valores numéricos de cada componente del circuito. El circuito sigue una topología *ladder* [5], donde la red está compuesta por los elementos en secuencia como se puede ver en la Figura 3. En esta representación los nodos resonantes y no resonantes (NRN) [6] están compuestos por una serie de elementos circuitales: capacidades c_k , susceptancias b_k e inversores de admitancia J_k (acoplamientos entre nodos). Son los parámetros que se necesitan para la implementación.

El procedimiento [7] puede entenderse como un algoritmo iterativo de descomposición: en cada iteración se extrae un componente del circuito y se actualiza una representación matricial intermedia.

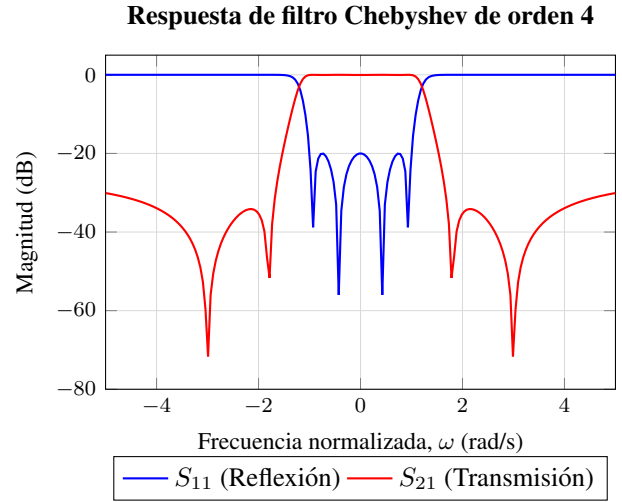


Fig. 2: Respuesta de un filtro de orden 4 con 4 ceros de transmisión y unas pérdidas de retorno de 20 dB

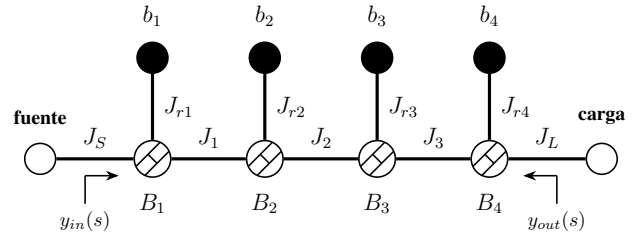


Fig. 3: Representación nodal de un filtro de orden 4 con 4 ceros de transmisión en topología *ladder*. Esta representación está compuesta por los nodos resonantes (círculos negros) y los no resonantes (círculos con H inclinada en su interior)

Formalmente, el estado del algoritmo en cada iteración k es una matriz polinomial $[ABCD]_k$, que encapsula la respuesta frecuencial del subcircuito que aún no ha sido descompuesto. Esta representación es habitual en teoría de redes: del mismo modo que una función de transferencia describe completamente un sistema lineal, la matriz $ABCD$ describe la relación entre voltaje y corriente en los dos puertos del subcircuito restante. El estado inicial se construye directamente a partir de los polinomios de entrada:

$$[ABCD]_1 = \frac{1}{jP(s)} \begin{bmatrix} A_1(s) & B_1(s) \\ C_1(s) & D_1(s) \end{bmatrix} \quad (2)$$

donde A_1, B_1, C_1, D_1 son combinaciones lineales de los polinomios $E(s)$ y $F(s)$.

En cada iteración $k = 1, \dots, N$ se extraen en secuencia:

1. **Un acoplamiento cruzado J_k** (inversor de admitancia): captura el efecto de los ceros de transmisión. Se calcula como el límite de un cociente de polinomios y se sustrae de la matriz, reduciendo el grado de $P_k(s)$ en una unidad.
2. **Una susceptancia b_k** (componente de desfase): se obtiene igualmente como cociente de polinomios $D_k(s)/B_k(s)$ y se sustrae de la matriz $ABCD_k$.
3. **Un resonador o nodo no resonante**, descrito por su parámetro c_k o $b_{0,k}$: se extrae mediante la misma lógica.

ca de evaluación en un punto específico del eje de frecuencias.

Tras cada extracción, los polinomios de la matriz se actualizan de forma cerrada (no hay minimización ni búsqueda numérica), lo que hace el proceso completamente analítico. Al finalizar las N iteraciones, la matriz $[ABCD]$ queda reducida a un inversor residual trivial, verificando que todos los componentes han sido correctamente extraídos.

El resultado final del bloque es una lista de elementos, entre los que se encuentran: $\{J_k, b_k, c_k\}_{k=1}^N$ que nos permiten construir la matriz de acoplamientos del filtro LPP. Esta estructura nos es conveniente para realizar transformaciones de similitud o escalados de los elementos.

3.1.3. Transformación a modelo acústico y paso banda

Una vez sintetizados los elementos equivalentes en el dominio de paso bajo normalizado, el proceso de diseño requiere dos etapas para obtener una red física realizable: la conversión de paso bajo a paso banda (LP2BP) y la conversión de los componentes circuitales a parámetros tecnológicos del modelo de ondas acústicas.

1. Conversión de Paso Bajo a Paso Banda Esta etapa transforma la respuesta frecuencial normalizada a la banda de interés y modela la estructura física de la red:

- **Adaptación de fuente y carga:** La naturaleza de los elementos de entrada (fuente) y salida (carga) del filtro se determina en función de la configuración (serie o en derivación) de los resonadores adyacentes a los extremos. Por ejemplo, si el primer resonador de la red está configurado en serie, se requiere un elemento de fuente en derivación (*shunt*), el cual se transforma en una capacidad o inductancia en el dominio paso banda [2].
- **Modelo equivalente de resonadores:** Cada resonador se representa mediante el modelo equivalente de Butterworth-Van Dyke (BVD), compuesto por una rama motriz serie (reactancia X_m e inductancia L_m) en paralelo con una reactancia estática (X_0).
- **Escalado frecuencial:** Las reactancias del dominio paso bajo se escalan en frecuencia a la banda de operación real a partir de la frecuencia central ω_0 y el ancho de banda fraccional del filtro, calculando los elementos finales del circuito paso banda: la inductancia motriz L_a , la capacidad motriz C_a y la capacidad estática C_0 .

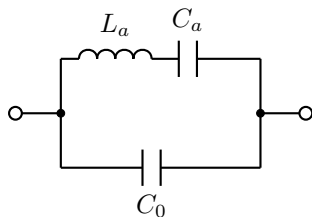


Fig. 4: Modelo equivalente de Butterworth-Van Dyke (BVD) de un resonador acústico en paso banda.

2. Conversión al Modelo Acústico y Acomodación Tecnológica Los valores del circuito BVD paso banda se traducen a los parámetros físicos que describen el resonador piezoeléctrico:

$$f_s = \frac{1}{2\pi\sqrt{L_a C_a}} \quad (\text{Frecuencia de resonancia serie}) \quad (3)$$

$$f_p = \sqrt{\frac{(C_a + C_0)f_s^2}{C_0}} \quad (\text{Frecuencia de resonancia paralelo}) \quad (4)$$

$$k_{\text{eff}}^2 = \frac{\pi^2 f_s (f_p - f_s)}{4 f_p^2} \quad (\text{Acoplamiento electromecánico}) \quad (5)$$

Un factor crítico en la realización física es la relación de capacidades $R = C_0/C_a$. En la síntesis matemática, esta relación puede tomar valores arbitrarios, pero en la práctica está rígidamente limitada por las propiedades piezoeléctricas del material utilizado (por ejemplo, nitrato de aluminio), el cual impone una relación objetivo R_{target} .

3.2. Métodos de diferenciación

Aún existiendo expresiones cerradas o procesos iterativos para la obtención de un filtro, no existe un método cerrado para encontrar uno que cumpla con unas restricciones de diseño y unas máscaras de frecuencias. Por ello se opta por utilizar métodos basados en descenso de gradiente. Los métodos de descenso de gradiente requieren una función de coste, que calcula la penalización de un resultado según se aleja o no cumple con unos requisitos. Estos métodos permiten un proceso iterativo generalizable para obtener soluciones subóptimas (mínimos locales), para problemas de los que no podemos o no sabemos obtener una solución cerrada. Existen 3 métodos principales:

3.2.1. Matemáticas simbólicas

Este método consiste en tratar las expresiones matemáticas de forma analítica y algebraica, representando las funciones como árboles de expresión cuyos nodos son operaciones matemáticas y cuyas hojas son constantes y variables simbólicas. Para calcular la derivada, se aplican reglas algebraicas de derivación (derivada del producto, suma, etc.) de manera recursiva sobre el árbol, obteniendo una nueva expresión simbólica que representa la derivada exacta.

Ventajas:

- **Precisión analítica absoluta:** La derivada resultante es exacta y no sufre de ningún tipo de error numérico de aproximación o truncamiento.
- **Simplificación y optimización algebraica:** Permite realizar simplificaciones algebraicas previas sobre la expresión de la derivada, lo que puede resultar en fórmulas muy eficientes una vez simplificadas.

Desventajas:

- **Explosión de expresiones:** Al aplicar de manera sistemática reglas como la del producto o del cociente, el

tamaño de la expresión matemática resultante crece de forma exponencial con respecto a la complejidad de la función original. Esto satura rápidamente la memoria RAM y ralentiza drásticamente la evaluación numérica.

3.2.2. Diferencias finitas

Es el método clásico de aproximación numérica de derivadas. Se basa directamente en la definición matemática de límite de la derivada de una función unidimensional:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \quad (6)$$

donde h es un paso numérico pequeño. Este enfoque se conoce como diferencia hacia adelante (*forward difference*). Para mejorar la aproximación, se puede recurrir a la diferencia centrada, que requiere evaluar la función en $f(x+h)$ y $f(x-h)$, reduciendo el error de truncamiento a costa de duplicar el número de evaluaciones.

Ventajas:

- **Extrema sencillez de implementación:** No requiere conocer la estructura interna del código ni el algoritmo de la función. Trata al programa como una “caja negra”, requiriendo únicamente evaluar la función con diferentes valores de entrada.
- **Universalidad:** Es aplicable a cualquier función ejecutable, independientemente de que contenga bucles, condicionales o código no diferenciable analíticamente.

Desventajas:

- **Inestabilidad numérica e imprecisión:** La elección del paso h presenta un dilema irresoluble. Si h es demasiado grande, se introduce un error de truncamiento. Si h es demasiado pequeño (para mitigar el truncamiento), se incurre en errores de cancelación sustractiva en aritmética de coma flotante, perdiendo precisión de manera catastrófica.
- **Elevado coste computacional:** Para calcular el gradiente de una función respecto a N variables de diseño, se requieren al menos $N + 1$ evaluaciones de la función completa (o $2N$ si se emplea diferencia centrada). En problemas de optimización con cientos o miles de variables, este coste resulta prohibitivo.

Este es el método utilizado en Baldr. Este método se utiliza mediante la función `fmincon` que realiza la minimización de una función de coste con restricciones.

3.2.3. Diferenciación automática

La diferenciación automática (AD) es un conjunto de técnicas para evaluar la derivada de una función implementada en un programa informático. A diferencia de la diferenciación simbólica (que trabaja con expresiones) y de las diferencias finitas (que trabaja con aproximaciones), AD

explota el hecho de que cualquier programa, por complejo que sea, ejecuta una secuencia de operaciones aritméticas elementales (sumas, multiplicaciones, funciones trigonométricas, etc.). Aplicando de forma sistemática la regla de la cadena sobre estas operaciones, se obtienen las derivadas exactas con precisión de máquina.

El cálculo se organiza representando el flujo del programa como un Grafo Acíclico Dirigido (DAG), donde los nodos representan las operaciones intermedias. Existen dos modos principales de ejecución:

- **Modo directo (*Forward Mode*):** Se calcula la derivada de todas las variables intermedias con respecto a una única variable de entrada de forma paralela a la evaluación de la función. Es eficiente cuando el número de entradas es muy pequeño frente al número de salidas.
- **Modo inverso (*Reverse Mode* o *Backpropagation*):** Consta de una fase hacia adelante (*forward pass*) que calcula y almacena los valores de las variables intermedias, seguida de una fase hacia atrás (*backward pass*) que propaga los gradientes desde la salida (normalmente una función de coste escalar) hacia todas las variables de entrada. Es idóneo para problemas de optimización donde el número de variables de entrada N es muy superior al número de salidas (usualmente una única métrica de pérdida).

Ventajas:

- **Precisión de máquina:** Al no utilizar aproximaciones de paso h , las derivadas obtenidas son exactas y libres de errores de truncamiento o cancelación sustractiva.
- **Eficiencia computacional en optimización:** En el modo inverso, el coste temporal de calcular el gradiente completo con respecto a N variables es independiente de N , siendo típicamente equivalente a unas pocas evaluaciones (entre 2 y 5 veces) de la función original.
- **Integración algorítmica:** Es capaz de diferenciar a través de flujos de control complejos (bucles, bifurcaciones) siempre que las operaciones locales sean diferenciables.

Desventajas:

- **Consumo elevado de memoria:** El modo inverso requiere almacenar en memoria todo el grafo de operaciones y los valores intermedios durante el paso hacia adelante para poder calcular los gradientes en el paso hacia atrás. Esto puede limitar el tamaño de los lotes de datos procesados.
- **Dependencia de infraestructura y tipos de datos:** Exige el uso de tipos de datos especializados (como `dldouble` en MATLAB) y que todas las funciones del algoritmo estén soportadas por el motor de diferenciación.

Para una comparación detallada de las características de cada método de diferenciación, véase el Apéndice A.1.

4 METODOLOGIA

El flujo de trabajo seguido ha sido una fase de estudio, seguida de un análisis de la solución original para identificar oportunidades y opciones. Una vez identificadas se ha procedido a implementar cada bloque de manera iterativa, asegurándose de que funciona y de su mejoría previa a continuar con el siguiente.

4.1. Análisis del software

Tras analizar en detalle el proceso de síntesis 3.1 y su implementación original en Baldr, se han identificado tres limitaciones y cuellos de botella principales en términos de rendimiento y arquitectura de software:

1. **Ejecución secuencial de filtros individuales:** El software original procesa únicamente un filtro por ejecución. En flujos de optimización masivos o durante la exploración del espacio de diseño (donde es necesario evaluar miles de alternativas), esta restricción secuencial desaprovecha los recursos de hardware moderno y limita la velocidad de cómputo.
2. **Acoplamiento de la lógica condicional y el cálculo numérico:** En el bucle principal de extracción, se evalúan constantemente condicionales dinámicos (*if-else*) para dar soporte a cualquier topología física de filtro. Dado que para un filtro concreto la topología es estática, estas evaluaciones repetitivas suponemos que causan fallos de predicción de saltos (*branch misprediction*) en la CPU, lo que tiene un coste en tiempos de ejecución.
3. **Evaluación aproximada e inestabilidades:** El proceso de optimización del filtro se basa en estimar numéricamente el gradiente de la función de coste mediante diferencias finitas. Para un espacio de N parámetros, esto requiere al menos $N + 1$ evaluaciones completas de la síntesis del filtro, además de introducir inestabilidades numéricas (errores de truncamiento).

Este último es el método utilizado en Baldr. Este método se utiliza mediante la función `fmincon` que realiza la minimización de una función de coste con restricciones.

4.2. Mejoras propuestas

Para superar estas limitaciones, se ha desarrollado un nuevo *software* aplicando principios de diseño orientado a objetos y computación en tarjeta gráfica (GPU).

4.2.1. Representación matricial y vectorización

Para solucionar el cuello de botella de la ejecución secuencial, se ha modificado la estructura de datos para admitir el procesamiento en lotes (*batches*). Mediante la implementación de clases específicas como `PolynomialMatrix` y `AbcdBatch`, se almacenan los polinomios y matrices de transferencia de múltiples filtros de forma simultánea en matrices y tensores. Esto permite que las operaciones aritméticas de síntesis y evaluación de parámetros S se ejecuten de manera vectorizada, aprovechando la capacidad SIMD de los procesadores actuales y la arquitectura masivamente paralela de las GPU.

La representación numérica de un polinomio se realiza mediante un vector de coeficientes de la forma $P = [1, 0, 2, 0, 3, 0]$ que representaría el polinomio $P(s) = s^2 + 2s + 3$. Dado que se trabaja con polinomios de orden de unidades o decena como máximo no tiene sentido el uso de GPUs y la paralelización está muy limitada. Ahora bien, si los agrupamos en matrices donde cada columna representa el polinomio de un filtro distinto podemos tener del orden de miles y convertir las operaciones en productos matriciales introduciendo el uso de GPUs.

De este modo, si definimos un lote de K filtros, donde cada filtro $k \in \{1, \dots, K\}$ tiene un polinomio característico $P_k(s)$ de grado M , la estructura de datos se modela mediante la matriz de coeficientes $\mathbf{P} \in \mathbb{R}^{(M+1) \times K}$:

$$\mathbf{P} = \begin{bmatrix} p_{0,1} & p_{0,2} & \cdots & p_{0,K} \\ p_{1,1} & p_{1,2} & \cdots & p_{1,K} \\ \vdots & \vdots & \ddots & \vdots \\ p_{M,1} & p_{M,2} & \cdots & p_{M,K} \end{bmatrix} \quad (7)$$

donde cada columna k contiene los coeficientes ordenados del polinomio $P_k(s) = \sum_{i=0}^M p_{i,k} s^i$. Las operaciones algebraicas entre polinomios de diferentes filtros (como la suma o la convolución) se pueden formular como operaciones matriciales globales sobre la matriz $\mathbf{P}$. Esto permite procesar todo el lote de filtros de manera concurrente en la GPU utilizando operaciones vectoriales altamente optimizadas.

4.2.2. Separación de lógica y pre-compilación topológica

Para eliminar las evaluaciones condicionales redundantes en el bucle de cálculo, se ha rediseñado la arquitectura del motor de extracción aplicando patrones de diseño de ingeniería de software:

- **Patrón Composite:** Se ha modelado la topología del filtro como una jerarquía de clases. Los elementos circuitales (capacidades, inversores, susceptancias) actúan como hojas de un árbol que implementan una interfaz común de extracción. Estos se agrupan en nodos de orden superior (nodos resonantes y no resonantes) de manera polimórfica. La construcción se realiza una vez y se reutiliza para todas las ejecuciones del ciclo de optimización. El diagrama de clases completo de esta jerarquía se detalla en el Apéndice A.2.
- **Patrones Builder y Command:** El motor de extracción (`TamiazzoEngine`) funciona bajo una filosofía de pre-compilación topológica (*Ahead-of-Time* o AOT). En su constructor (fase de planificación), actúa como un *Builder* leyendo el plano topológico del filtro y construyendo una secuencia ordenada de comandos de extracción. En la fase de ejecución, el bucle principal recorre este vector de objetos de forma ciega, eliminando por completo cualquier instrucción condicional *if/else* durante el bloque de cómputo más pesado.

La estructura y lógica detallada de ambos esquemas de ejecución se presenta mediante pseudocódigo en el Apéndice A.3.

4.2.3. Integración de diferenciación automática

Se ha sustituido el cálculo numérico de gradientes por Diferenciación Automática (AD) en modo inverso. Utilizando la estructura de datos `dlarray` y el método de evaluación `dlgradient` de MATLAB, se registran las operaciones matemáticas durante el paso hacia adelante (*forward pass*) en un grafo de computación, véase el ejemplo en la Figura 5. Posteriormente, durante el paso hacia atrás (*backward pass*), se propagan los gradientes desde la función de coste escalar hacia todas las variables de diseño con precisión analítica de máquina. Esto reduce el coste temporal del cálculo del gradiente completo a un coste constante $O(1)$, independiente del número de variables de optimización. Para asegurar la diferenciabilidad de todo el flujo, se ha integrado además el método de Aberth-Ehrlich [8] como alternativa diferenciable para el cálculo de raíces de polinomios en lugar de la función nativa `roots`.

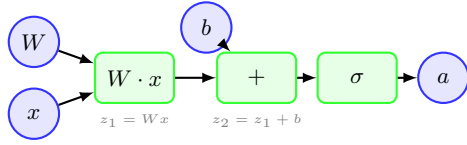


Fig. 5: Grafo Acíclico Dirigido (DAG) de cómputo para una capa elemental de red neuronal. Los círculos azules representan las variables de entrada, pesos y salida (x , W , b , a), mientras que los rectángulos verdes representan las operaciones elementales del paso hacia adelante (*forward pass*).

5 RESULTADOS

Las pruebas se han realizado en un único ordenador con las siguientes características:

- **Procesador (CPU):** AMD Ryzen 7 5700X.
- **Memoria RAM:** 32 GB DDR4.
- **Tarjeta gráfica (GPU):** NVIDIA GeForce RTX 3070 (8 GB de VRAM).
- **Entorno de software:** MATLAB R2025a con *Deep Learning Toolbox*.
- **Sistema Operativo:** Windows 11

5.1. Validez resultados

Para validar la precisión del nuevo motor de síntesis y verificar que no se han introducido desviaciones numéricas, se ha comparado la respuesta electromagnética del filtro obtenida con la nueva herramienta frente a la original.

Se ha sintetizado un filtro Chebyshev de orden 6 con 6 ceros de transmisión colocados en $\pm 1,8j$, $\pm 3j$ y $\pm 4j$ en el plano s , con unas pérdidas de retorno de 20 dB. A partir de las matrices de acoplamiento obtenidas por ambos métodos, se han calculado los parámetros de dispersión (S_{11} y S_{21}) en el rango de frecuencias de -5 a 5 .

En la Figura 6 se muestra la superposición de los resultados de ambos simuladores. Para permitir la visualización de las curvas debido a su solapamiento, se han representado los resultados de Baldr mediante líneas continuas y los del

nuevo motor mediante líneas discontinuas. Como se puede observar, las respuestas coinciden de forma exacta, lo que valida el correcto funcionamiento del nuevo código.

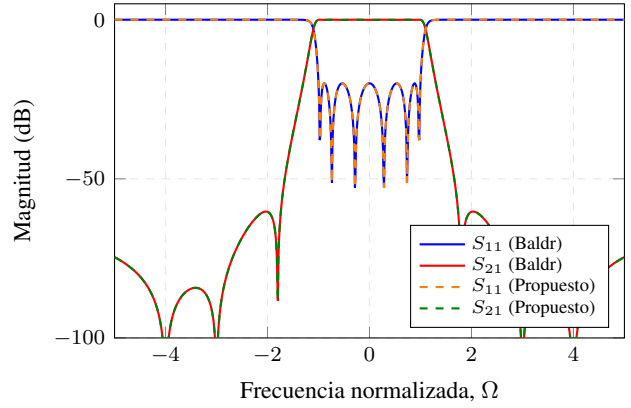


Fig. 6: Comparación de los parámetros de dispersión (S_{11} y S_{21}) obtenidos mediante el software Baldr original y el nuevo motor de síntesis propuesto utilizando la matriz de acoplamiento normalizada.

5.2. Rendimiento computacional

Para evaluar el rendimiento computacional de la arquitectura propuesta frente al software de referencia Baldr, se ha realizado una batería de pruebas de síntesis completa variando el tamaño del lote de filtros desde $K = 1$ hasta $K = 100\,000$. En cada ejecución, los parámetros de diseño (ceros de transmisión y pérdidas de retorno) de cada filtro se han perturbado aleatoriamente para simular un escenario de optimización real.

Se han comparado cuatro configuraciones de ejecución:

1. **Baldr (CPU Secuencial):** Ejecución secuencial tradicional utilizando un único hilo en el procesador.
2. **Baldr (CPU Paralelo):** Ejecución paralelizada mediante la directiva `parfor` de MATLAB distribuida en un pool de 8 hilos de procesamiento (workers).
3. **Propuesto (CPU Vectorizado):** Ejecución en el nuevo software orientado a objetos utilizando la vectorización nativa y la pre-computación topológica en CPU.
4. **Propuesto (GPU Vectorizado):** Ejecución en el nuevo software orientado a objetos sobre GPU empleando tipos de datos `gpuArray` para evaluar todo el lote en paralelo.

En la Figura 7 se muestra el tiempo promedio de ejecución requerido para sintetizar un único filtro en función del tamaño del lote (K) en escala logarítmica.

Como puede observarse, para lotes pequeños ($K < 10$), los métodos secuenciales tradicionales presentan menor sobrecarga inicial. Sin embargo, a medida que K aumenta, la vectorización nativa del método propuesto reduce drásticamente el tiempo promedio por filtro. Para un lote de 100 000 filtros:

La ejecución secuencial en Baldr sigue requiriendo el mismo tiempo. La paralelización tradicional (`parfor`) con 8 workers logra reducir el tiempo a medida que aumenta el

número de filtros. El método propuesto vectorizado en CPU empieza a sobrepasar al resto con tan solo 100 filtros. El método propuesto en GPU logra un mejor rendimiento en el caso más extremo de 100.000 filtros.

Estos resultados demuestran que la vectorización basada en la acumulación de datos en matrices y tensores permite un incremento de velocidad de más de $10\times$ frente a la paralelización con múltiples núcleos de CPU, sin incurrir en la sobrecarga de gestión de hilos y sincronización. Además, el comportamiento de la GPU demuestra una excelente escalabilidad para lotes masivos, aunque el resultado solo se puede apreciar en el extremos.

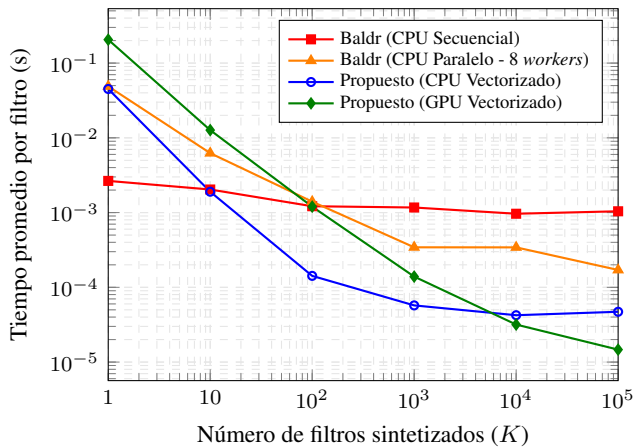


Fig. 7: Comparativa del tiempo promedio de ejecución por filtro en función del tamaño del lote (K) para los diferentes esquemas evaluados.

Con el fin de analizar en detalle la eficiencia de la paralelización en la GPU y detectar posibles cuellos de botella en la ejecución vectorizada, se ha registrado el tiempo de ejecución promedio por filtro desglosado para cada una de las cinco fases principales de la síntesis. Este análisis se ha realizado sobre un lote masivo de $K = 100\,000$ filtros, comparando el rendimiento de la CPU frente a la GPU. Los resultados de esta comparativa se presentan en la Figura 8.

Como puede observarse en la gráfica de barras, la mayor parte del tiempo de cómputo en la CPU se concentra en la Fase 1 (cálculo de polinomios y corrección de fase), requiriendo $33,6\,\mu\text{s}$ por filtro, mientras que en la GPU esta etapa se reduce drásticamente a solo $6,4\,\mu\text{s}$ por filtro (lo que representa una aceleración de $5,3\times$). Esto demuestra la gran efectividad de la paralelización masiva en la GPU para la resolución simultánea de polinomios sobre lotes grandes.

Sin embargo, también en las fases puramente matriciales se aprecia la superioridad de la aceleración por hardware de la GPU. De igual manera, tenemos mejoras sustanciales en las fases de construcción de la matriz ABCD como en la extracción de elementos. Aun así, observamos una menor mejora en la fase 4, construcción y normalización de la matriz de acoplamientos, pero, aún así, no es despreciable. Por último, en la Fase 5 (reconstrucción y transformación tecnológica), ambos dispositivos requieren prácticamente el mismo tiempo, lo que nos indica una limitación por el procesado secuencial de objetos circuitales en la CPU.

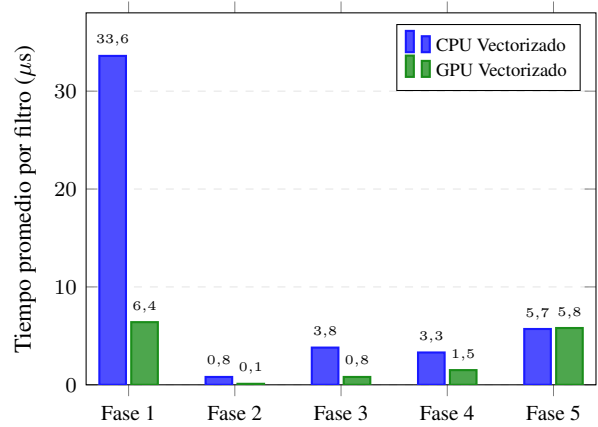


Fig. 8: Comparativa del tiempo promedio de ejecución por filtro desglosado por fases entre la CPU y la GPU para un lote de $K = 100\,000$ filtros.

5.3. Análisis del esquema en dos pasos

La separación del proceso de síntesis en dos fases de ejecución diferenciadas —Fase de Planificación (construcción del grafo y planificación topológica Ahead-of-Time) y Fase de Ejecución (evaluación matemática pura)— se diseñó bajo la hipótesis de que evitar la toma de decisiones lógicas en cada iteración secuencial de un bucle de optimización reduciría significativamente el tiempo de cómputo.

Para validar esta hipótesis, se comparó el tiempo de ejecución promedio por filtro en función del número de iteraciones (N) bajo cuatro estrategias distintas:

1. **Seq 1-Step (Tradicional):** Planificación y extracción acopladas dentro del bucle secuencial (reinstanciación en cada iteración).
2. **Seq 2-Step (Separado secuencial):** Planificación realizada una única vez fuera del bucle, y ejecución secuencial reutilizando el planificador.
3. **Batch 1-Step (Lote con planificación):** Ejecución de todo el lote en una sola llamada vectorizada, incluyendo el tiempo de planificación y creación de motores en la medición.
4. **Batch 2-Step (Lote sin planificación):** Ejecución del lote vectorizado asumiendo que la planificación ya se ha realizado previamente (solo se mide la ejecución).

Los resultados obtenidos se muestran en la Figura 9. Al analizar el rendimiento secuencial (comparando *Seq 1-Step* frente a *Seq 2-Step*), no se obtuvo el resultado esperado. Se preveía que evitar la fase de planificación en cada iteración secuencial generaría una mejora de velocidad notable. Sin embargo, para $N \geq 100$, ambas curvas convergen a un valor prácticamente idéntico, siendo incluso la opción en dos pasos ligeramente más lenta en algunos puntos debido a la sobrecarga de gestión y acceso a propiedades de objetos persistentes en el entorno de MATLAB. Esto demuestra que en un flujo puramente secuencial, el coste computacional de la toma de decisiones lógicas y de la instanciación de los objetos planificadores es despreciable en comparación con el coste del cálculo numérico de la extracción circuital, haciendo que la separación en dos pasos carezca de justificación práctica para ejecuciones monohilo tradicionales.

Aún así, el resultado nos da la seguridad de que podemos seguir trabajando con el código en su forma tradicional sin tener penalización de rendimiento.

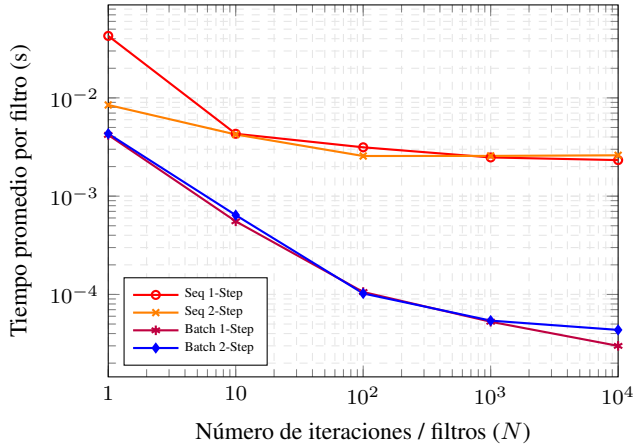


Fig. 9: Comparativa del tiempo promedio de ejecución por filtro en función del número de iteraciones (N) para evaluar el impacto de la separación en dos pasos (planificación y ejecución).

5.4. Rendimiento en la optimización por gradiente

Para evaluar el impacto de la diferenciación automática en el contexto de optimización, se ha analizado cómo escala el tiempo promedio requerido para calcular el gradiente (o realizar una iteración completa del optimizador) a medida que aumenta la complejidad del problema. En el diseño de filtros, la complejidad viene determinada por el orden del filtro y, por ende, por el número de ceros de transmisión (P) que actúan como variables de optimización.

Se han comparado los siguientes cuatro escenarios:

1. **Single fmincon (Diferencias Finitas):** Optimización de un único filtro utilizando el solver estándar `fmincon` de MATLAB con gradientes numéricos calculados por diferencias finitas en CPU.
2. **Single AD (CPU):** Optimización de un único filtro en CPU utilizando diferenciación automática (AD) mediante `dlarray`.
3. **Batch AD (CPU - 1K filtros):** Optimización simultánea de un lote (enjambre) de 1000 filtros en CPU calculando el gradiente mediante AD vectorizada.
4. **Batch AD (GPU - 1K filtros):** Optimización simultánea de un lote de 1000 filtros en GPU calculando el gradiente mediante AD vectorizada sobre `gpuArray`.

En la Figura 10 se presenta el tiempo promedio de ejecución por iteración y por filtro en función del número de ceros de transmisión (P) en escala logarítmica.

Como puede observarse, la estrategia convencional de diferencias finitas (*Single fmincon*) presenta un comportamiento algo errático e igual de desfavorable que el caso de un solo filtro utilizando AD. Esto hace que para $P = 10$ variables, el tiempo por iteración se acerque al segundo. Por

su parte, la diferenciación automática monohilo (*Single AD*) mitiga este efecto al calcular el gradiente analítico exacto en un único paso hacia atrás (paso de retropropagación), dando un rendimiento más consistente, siendo mejor o peor en algunos casos.

No obstante, la gran revolución en rendimiento ocurre cuando combinamos la diferenciación automática con el procesamiento en lote. La optimización vectorizada en CPU (*Batch AD CPU*) logra amortizar el coste de evaluación dividiendo la carga entre el lote de filtros, reduciendo el coste por filtro en un factor de 25× frente a la ejecución individual. Finalmente, la implementación en GPU (*Batch AD GPU*) exprime al máximo la paralelización tensorial y la paralelización por hardware: para $P = 10$, completa la iteración de los 1000 filtros en tan solo 7,7 ms por filtro, lo que representa un *speedup* de más de 3× en comparación con la misma optimización en CPU. Esto demuestra el gran potencial de la combinación de la vectorización en lotes y la diferenciación automática para optimización global en casos donde podemos darle una forma conveniente a los datos.

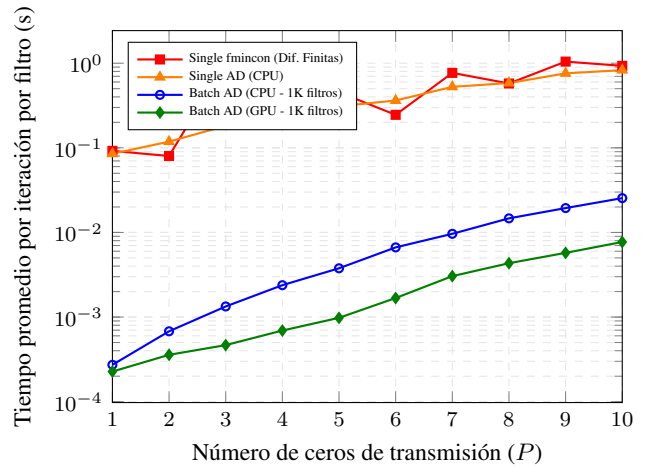


Fig. 10: Comparativa del tiempo promedio de ejecución por iteración y por filtro en función del número de dimensiones (ceros de transmisión) para optimización individual y por lotes (enjambre).

6 CONCLUSIONES

En este proyecto se ha desarrollado e implementado con éxito un nuevo software de síntesis de filtros de microondas optimizado mediante la vectorización matricial y la diferenciación automática. Hemos garantizado que el motor propuesto no introduce desviación numérica significativa respecto al código original.

El análisis de rendimiento computacional revela que tanto la paralelización por hardware como la correcta estructuración de los datos aumentan la escalabilidad del problema de síntesis. Este acercamiento nos ha permitido multiplicar hasta por 10 el rendimiento en ciertas situaciones, una vez se supera el umbral del *overhead* introducido.

Para el caso de optimización, la primera mejora (secuencial vs lotes) ya permite explorar una mayor parte del espacio de posibles soluciones, pero al incluir una GPU podemos reducir los tiempos por un factor 3 sin modificar ninguna parte del código interno de la síntesis.

Por último, separar la ejecución en dos partes mediante un patrón *Builder* no ha resultado en ninguna mejora y vuelve más obtuso el código fuente, por lo que es una línea de trabajo que descartamos por el momento.

En cuanto al trabajo futuro derivado de este proyecto se incluyen: completar el flujo de síntesis introduciendo la conversión al modelo acústico y el desplazamiento a la frecuencia de interés. Por último, el enfoque modular del trabajo permite la sustitución y la incorporación de partes del flujo de síntesis, lo que dotaría a la herramienta de más capacidad y utilidad en diversidad de contextos.

Tener una herramienta que logra realizar el proceso de síntesis a gran velocidad no solo permite utilizar mejor el tiempo del diseñador, sino que también reduce el consumo eléctrico y la huella de carbono de estos procesos. También abre la puerta a explorar espacios de soluciones más amplios cuyo análisis resulta prohibitivo con los tiempos de cómputo actuales.

ASPECTOS ÉTICOS Y LICENCIAS

El *software* utilizado para el desarrollo de este trabajo es libre, como el caso de $\text{\LaTeX}$ ha sido debidamente licenciado en el caso de MATLAB. El *software* desarrollado durante este proyecto se redistribuye con licencia abierta MIT.

USO DE LA IA GENERATIVA

En cumplimiento de las directrices éticas y académicas de la Universitat Autònoma de Barcelona (UAB) para el desarrollo del Trabajo de Fin de Grado (TFG), se declara el uso de herramientas de Inteligencia Artificial Generativa (principalmente los modelos Gemini de Google) con la intención de acelerar el desarrollo.

Se ha empleado la IA para corregir la gramática, sintaxis y cohesión textual de la memoria en castellano, garantizando un tono formal y adecuado para el ámbito de la ingeniería informática. También se ha utilizado como herramienta de depuración de código y para asegurar su corrección. Las ideas propuestas en este trabajo son originales. Los diagramas `tikz` y gráficas se han solicitado a la IA puesto que no es un trabajo esencial.

Se declara que se ha hecho un uso responsable, que toda sugerencia, fragmento de código o estructura generada por la IA ha sido analizada críticamente, depurada y verificada manualmente por el autor mediante la ejecución de pruebas. Las decisiones metodológicas, la formulación matemática de las matrices de transferencia, los algoritmos de síntesis y la interpretación de los resultados son de autoría exclusiva del estudiante, quien asume la responsabilidad total del contenido de este trabajo.

AGRADECIMIENTOS

Agradecimientos a mi tutor por su ayuda y por poder contar con él.

REFERENCIAS

[1] A. Gimenez, J. Verdu, and P. De Paco Sanchez, "General Synthesis Methodology for the Design of Acoustic

Wave Ladder Filters and Duplexers," *IEEE Access*, vol. 6, pp. 47 969–47 979, 2018. [Online]. Available: <https://ieeexplore.ieee.org/document/8438474/>

- [2] E. Guerrero, "Synthesis considerations for acoustic wave filters and duplexers starting with shunt resonator," Treball de fi de postgrau, Universitat Autònoma de Barcelona, Escola d'Enginyeria, 2019. [Online]. Available: https://ddd.uab.cat/pub/trerecpro/2019/hdl_2072_374637/TFM_EloiGuerreroMenendez.pdf
- [3] R. J. Cameron, C. M. Kudsia, and R. R. Mansour, *Microwave Filters for Communication Systems: Fundamentals, Design and Applications*, 1st ed. Wiley, Mar. 2018. [Online]. Available: <https://onlinelibrary.wiley.com/doi/book/10.1002/9781119292371>
- [4] R. Cameron, "Advanced coupling matrix synthesis techniques for microwave filters," *IEEE Transactions on Microwave Theory and Techniques*, vol. 51, no. 1, pp. 1–10, Jan. 2003. [Online]. Available: <http://ieeexplore.ieee.org/document/1159661/>
- [5] S. Amari and U. Rosenberg, "Synthesis and Design of Novel In-Line Filters With One or Two Real Transmission Zeros," *IEEE Transactions on Microwave Theory and Techniques*, vol. 52, no. 5, pp. 1464–1478, May 2004. [Online]. Available: <http://ieeexplore.ieee.org/document/1295146/>
- [6] S. Amari and G. Macchiarella, "Synthesis of inline filters with arbitrarily placed attenuation poles by using nonresonating nodes," *IEEE Transactions on Microwave Theory and Techniques*, vol. 53, no. 10, pp. 3075–3081, Oct. 2005. [Online]. Available: <http://ieeexplore.ieee.org/document/1516308/>
- [7] S. Tamiazzo and G. Macchiarella, "Synthesis of Cross-Coupled Prototype Filters Including Resonant and Non-Resonant Nodes," *IEEE Transactions on Microwave Theory and Techniques*, vol. 63, no. 10, pp. 3408–3415, Oct. 2015. [Online]. Available: <http://ieeexplore.ieee.org/document/7174561/>
- [8] M. Petković, *Iterative methods for simultaneous inclusion of polynomial zeros*. Berlin: Springer, 1989.

APÉNDICE

A.1. Comparativa de métodos de diferenciación

En la Tabla 1 se presenta una comparación detallada de las características críticas de los tres métodos de diferenciación descritos para el contexto de optimización de filtros de microondas.

A.2. Diagrama de clases de la topología circuital

Para complementar la explicación de la separación de lógica y la pre-compilación topológica del motor de extracción, en la Fig. 11 se muestra el diagrama de clases detallado que implementa el patrón de diseño *Composite*.

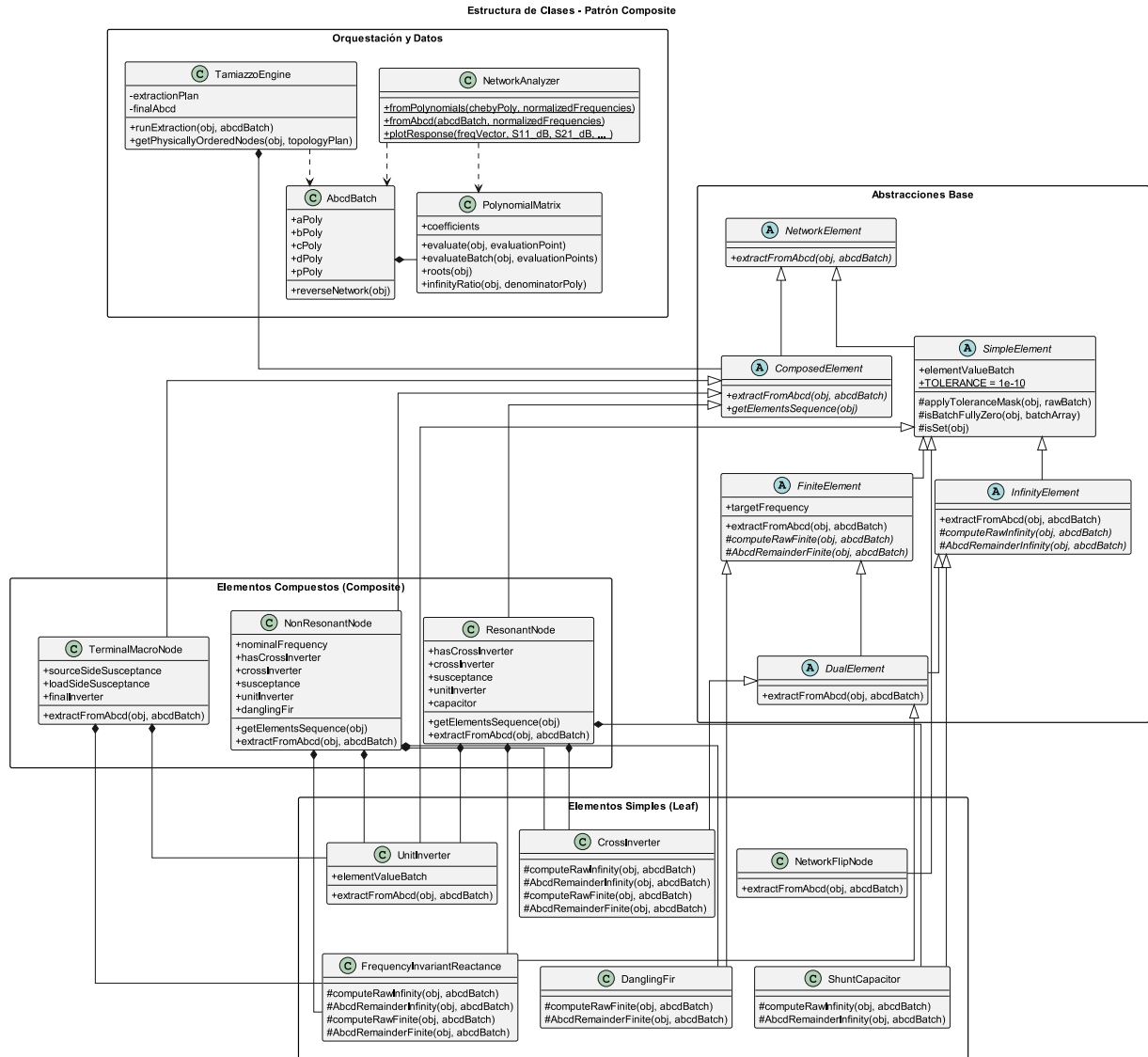


Fig. 11: Diagrama de clases detallado de la jerarquía del patrón *Composite* para la extracción de elementos circuitales del filtro.

TAULA 1: COMPARATIVA DE LOS MÉTODOS DE DIFERENCIACIÓN PARA OPTIMIZACIÓN DE FILTROS.

Característica	Diferenciación Simbólica	Diferencias Finitas	Diferenciación Automática
Principio	Reglas algebraicas sobre árboles de expresiones.	Aproximación numérica local: $\frac{f(x+h)-f(x)}{h}$.	Regla de la cadena sobre el grafo de ejecución (DAG).
Precisión	Exacta (analítica).	Baja (sujeta a redondeo y truncamiento).	Exacta (precisión de máquina).
Complejidad temporal (Gradiente con N vars.)	No computable en problemas complejos (explosión algebraica).	$O(N)$ evaluaciones de la función completa.	$O(1)$ evaluaciones de la función completa (modo inverso).
Consumo de memoria	Muy alto durante la generación de expresiones.	Extremadamente bajo (solo almacena variables locales).	Alto en modo inverso (requiere almacenar el grafo y variables).
Soporte de control de flujo	Nulo (no admite bucles dinámicos ni condicionales).	Total (trata al algoritmo como una caja negra).	Alto (sigue el flujo de ejecución del código ejecutable).
Facilidad de uso en código heredado	Muy compleja (requiere reescribir la lógica en formato simbólico).	Trivial (no requiere modificar el código interno).	Media-alta (requiere usar tipos de datos tensoriales/gradientes).

A.3. Esquemas de ejecución en la extracción de componentes

El Código 1 expone la diferencia de flujo lógico entre el esquema de una pasada tradicional (bucle con condicionales evaluados en tiempo de ejecución) y el propuesto de dos pasadas (planificación de topología Ahead-of-Time y posterior ejecución limpia de comandos).

```
% 1. ESQUEMA EN UNA PASADA (TRADICIONAL)
Bucle de optimizacion (M iteraciones):
  Para cada nodo k de 1 a N:
    % Evaluacion de la topologia en
    cada iteracion
    Si es_resonante(k) y tipo_ladder:
      Extraer_Capacitor(k)
    Sino:
      Extraer_Inversor(k)

% 2. ESQUEMA PROPUESTO (2 PASADAS)
Fase 1: Planificacion (Se ejecuta UNA vez)
  Para cada nodo k de 1 a N:
    Si es_resonante(k) y tipo_ladder:
      plan.agregar(Extraer_Capacitor)
    Sino:
      plan.agregar(Extraer_Inversor)

Fase 2: Ejecucion (Bucle de optimizacion, M
iteraciones)
  Bucle de optimizacion:
    % Bucle ciego sin condicionales de
    topologia
    Para cada accion en plan:
      accion.ejecutar()
```

Listing 1: Comparativa de flujo de ejecución: esquema de una pasada frente a esquema de dos pasadas (planificación y ejecución).