Trabajo Fin de Grado
Ingeniería Aeroespacial

Software de modelado de filtros EMI mediante algoritmos de búsqueda avanzados

Autor: Álvaro Romero Muñiz

Tutor: Joaquín Bernal Méndez

Dpto. Física Aplicada III
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado
Ingeniería Aeroespacial

Software de modelado de filtros EMI mediante algoritmos de búsqueda avanzados

Autor:
Álvaro Romero Muñiz

Tutor:
Joaquín Bernal Méndez
Profesor Titular

Dpto. Física Aplicada III
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado: Software de modelado de filtros EMI
mediante algoritmos de búsqueda avanzados

Autor: Álvaro Romero Muñiz
Tutor: Joaquín Bernal Méndez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Resumen

En este trabajo se muestra el desarrollo de una interfaz gráfica de usuario (GUI) para modelar el comportamiento de filtros EMI en un amplio rango de frecuencias. Se parte de un modelo de circuitos de parámetros localizados del filtro EMI validado en el rango de frecuencias en el que la mayor parte de las normas de compatibilidad electromagnética exigen la medida de emisiones conducidas de los equipos. Este modelo incorpora efectos parásitos como las inducciones parásitas de condensadores, las capacidades parásitas de los elementos inductivos y el acoplo magnético entre componentes, que se ha demostrado que determinan en gran medida la respuesta real del filtro a frecuencias altas. El software desarrollado permite comparar las medidas de atenuación del filtro en modo común y en modo diferencial con la respuesta predicha por el modelo, y utilizar un algoritmo de búsqueda avanzada (*Particle Swarm Optimization*) para hallar los parámetros del modelo del filtro. Para comprobar el correcto funcionamiento de la herramienta software desarrollada se han utilizado medidas experimentales de diferentes filtros.

El desarrollo de esta herramienta se enmarca en el contexto más amplio de un esfuerzo por desarrollar una herramienta software útil para el diseño y optimización de filtros EMI. La idea es unificar las herramientas de modelado y diseño desarrolladas por el grupo de investigación en un único software, que permita generar un modelo preciso del filtro en un rango amplio de frecuencias e investigar el efecto en cada rango de frecuencias de los diferentes efectos parásitos que afectan al rendimiento del filtro. Este es un paso previo fundamental para el desarrollo de técnicas avanzadas de mitigación de efectos parásitos que permitan incrementar el rendimiento del filtro sin aumentar su peso y volumen ni incrementar su coste.

Abstract

This work focus on the development of a graphical user interface (GUI) to model the behavior of EMI filters in a wide frequency range. It makes use of a circuit model of the EMI filter, validated in the frequency range in which most of the electromagnetic compatibility standards require the measurement of conducted emissions from the equipment. This model incorporates parasitic effects such as parasitic inductances of capacitors, capacitances of inductive elements, and magnetic coupling between components, which have been shown to largely determine the actual filter response at high frequencies. The software developed allows to compare the measured attenuation of the filter (both in common mode and in differential mode) with the response predicted by the model, and to use an advanced search algorithm (*Particle Swarm Optimization*) to find the parameters of the filter model. To check the correct operation of the developed software tool, experimental measurements of different filters have been used.

The development of this tool is framed in the broader context of an effort to develop a useful software tool for the design and optimization of EMI filters. The idea is to unify the modeling and design tools developed by the research group in a single software, which allows generating an accurate filter model in a wide range of frequencies and investigating the effect in each frequency range of the different parasitic effects that affect filter performance. This is a fundamental previous step for the development of advanced techniques for mitigation of parasitic effects that allow increasing the performance of the filter without increasing its weight, volume and cost.

Índice

1 Introducción

La interferencia electromagnética (EMI) consiste en cualquier señal no deseada radiada o conducida que provoque la degradación del desempeño de un equipo [1]. Este fenómeno puede observarse cuando la señal Wi-Fi empeora al colocar dispositivos electrónicos cerca de un *router* o saltan los interruptores automáticos de una vivienda al caer un rayo. Con el tiempo la electrónica se ha miniaturizado y se ha elevado la frecuencia de reloj de los equipos, lo que ha provocado la creación de normativa para controlar los efectos de EMI y asegurar el funcionamiento de los dispositivos comercializados [2]. La compatibilidad electromagnética (EMC) es la capacidad de un sistema electrónico para funcionar correctamente sin alterar el funcionamiento del resto de dispositivos del entorno. Es decir, no debe causar interferencia a otros equipos ni a sí mismo, ni tampoco ser susceptible a las emisiones de los demás equipos [3]

La tendencia actual en el sector aeronáutico por sustituir sistemas neumáticos e hidráulicos por actuadores eléctricos (*More Electric Aircraft*) [4], el uso extensivo de la electrónica y los requerimientos de peso y volumen de los equipos embarcados suponen un reto de diseño desde el punto de vista de la compatibilidad electromagnética. Las emisiones electromagnéticas, tanto radiadas como conducidas, de los equipos deben controlarse para cumplir con la estricta normativa del sector y asegurar su fiabilidad [5] [6]. Un dispositivo ampliamente utilizado para atenuar emisiones conducidas son los filtros EMI, que reducen el ruido de alta frecuencia producido por dispositivos electrónicos [7]. Es de especial interés su uso en las fuentes conmutadas que, a pesar de su elevada eficiencia y reducido tamaño, presentan la desventaja de generar una gran cantidad de ruido electromagnético [8].

Los filtros EMI son filtros paso-bajo compuestos por condensadores que derivan el ruido de alta frecuencia e inductores que representan un camino de alta impedancia para este ruido. Para reducir la corriente en modo común se utilizan chokes, parejas de inductores contra-acoplados que filtran las señales en modo común afectando en menor medida a la corriente en modo diferencial.

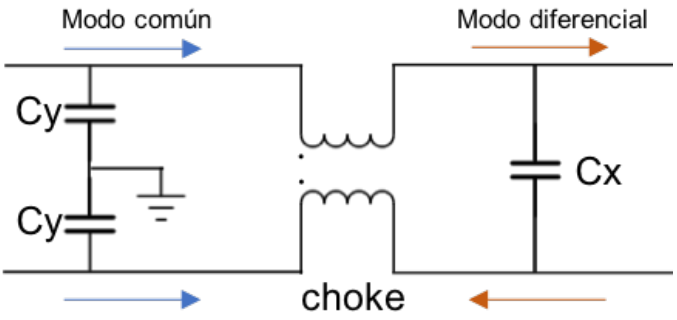


Figura 1.1 Esquemático de un filtro EMI.

Al modelo básico del filtro se pueden añadir los efectos parásitos de los diferentes componentes integrados en el filtro y el acoplo entre las inductancias, creándose de este modo un modelo de parámetros localizados capaz de reproducir con precisión la respuesta del filtro en el rango de frecuencias especificado por la normativa.

En el grupo de EMC formado por investigadores de los departamentos de Física Aplicada III e Ingeniería Electrónica de la Escuela Técnica Superior de Ingeniería de Sevilla se trabaja de forma recurrente con filtros EMI. Esto requiere el uso de diferentes programas para la resolución de circuitos, búsqueda de parámetros mediante algoritmos genéticos y representación de gráficas, teniendo que desarrollar los script necesarios para ello.

En este trabajo se expone el desarrollo de una interfaz gráfica de usuario (GUI) para modelar filtros EMI de forma simplificada realizado con el paquete *tkinter* [9] de Python. Permite la resolución de las ecuaciones del circuito de un filtro EMI a partir de los parámetros que lo modelan, la obtención de parámetros a través del algoritmo *pso* de *pyswarm* (*Particle Swarm Optimization*) [10] y la representación de las curvas obtenidas.

El algoritmo PSO se basa en la optimización de una función de coste mediante la evaluación iterativa de puntos (partículas) distribuidos en el espacio de búsqueda. El movimiento de cada partícula viene dado por su posición relativa a la mejor solución encontrada por la misma, la mejor solución encontrada por el conjunto entero (enjambre) y el movimiento de la anterior iteración. De este modo se consigue que las partículas converjan hacia un mínimo (o máximo) de la función de coste [11].

Para verificar el correcto funcionamiento del software se ha probado introduciendo medidas simuladas de un filtro previamente modelado [12] y los datos obtenidos en una experiencia previa para comprobar atenuación producida por láminas de ferrita y cobre en los condensadores [13].

2 Descripción del software

El desarrollo del software parte de la obtención de las ecuaciones del circuito, que se resolverán para obtener la respuesta en frecuencia del modelo. Para cumplir las especificaciones solicitadas la función que resuelve el circuito se integrará con otras funciones que permitan calcular parámetros mediante el ajuste de la respuesta en frecuencia del modelo a los datos experimentales y su posterior representación, así como una serie de opciones para facilitar el uso de la aplicación. Todas las funcionalidades del programa se detallan en la guía de uso, donde se explica paso a paso la correcta operación del software.

2.1 Modelado del filtro

Al tener en cuenta los efectos parásitos en el filtro deben añadirse las inductancias, capacidades y resistencias presentes en los distintos componentes, siendo estos valores diferentes según se analice para la corriente en modo común o diferencial.

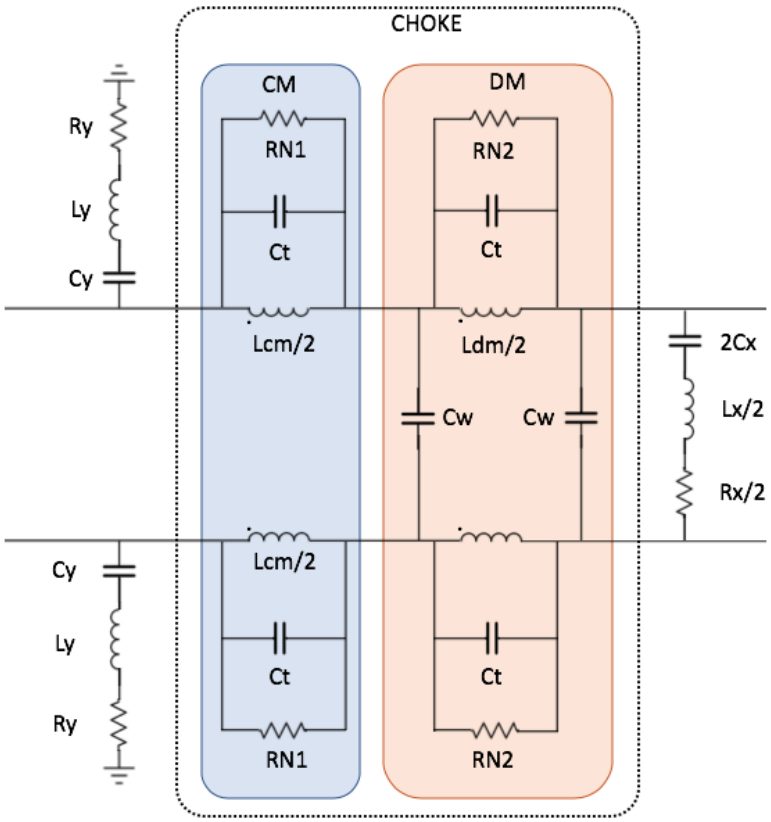


Figura 2.1 Esquema del filtro con efectos parásitos.

Los diferentes parámetros que se utilizarán para modelar la respuesta del filtro EMI serán por tanto:

- Cy: capacidad del condensador Y.
- Ly: inductancia parásita del condensador Y.
- Ry: resistencia del condensador Y.
- Cx: capacidad del condensador X.
- Lx: inductancia parásita del condensador X.
- Rx: resistencia del condensador X.
- Lcm: inductancia en modo común (CM) del choke.
- Ldm: inductancia en modo diferencial (DM) del choke.
- Cw: capacitancia parásita entre devanados.
- Ct: capacitancia parásita del choke.
- RN1: resistencia parásita del choke en CM.
- RN2: resistencia parásita del choke en DM.
- Kycm: entre Cy y el choke en CM.
- Kydm: entre Cy y el choke en DM.
- Kxdm: entre Cx y el choke en DM.
- Kxy: entre Cx y Cy.
- Rs: impedancia de la fuente.
- Rl: impedancia de la carga.

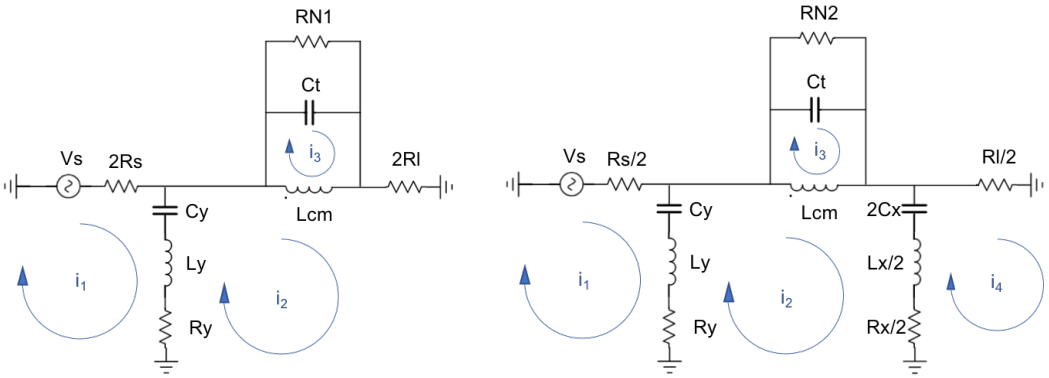
A partir de este modelo pueden obtenerse dos circuitos diferentes, uno para la respuesta en modo común y otro para la respuesta en modo diferencial mostrados en la Figura 2.2. Los acoplos entre las distintas inductancias presentes en estos circuitos pueden representarse como fuentes dependientes de tensión. Las capacidades C_w no se han tenido en cuenta en el modelado ya que tienen poca repercusión en la respuesta en frecuencia y simplifican el circuito, aún así se ha dejado como entrada en el entorno gráfico para facilitar su posible adición en futuros modelos [12].

Para resolver el circuito por el método de las mallas deben obtenerse las expresiones de las matrices de impedancia de ambos circuitos y tomar tensión unidad en la fuente de tensión con distintos valores de frecuencia. De esta forma puede obtenerse la transmisión de potencia mediante un barrido en frecuencia. La matriz de impedancia del circuito equivalente en modo común viene dada por los elementos:

$$Z_{11} = 2R_s + Z_y \tag{2.1}$$
$$Z_{12} = -Z_y + j\omega M_{ycm} \tag{2.2}$$
$$Z_{13} = -j\omega M_{ycm} \tag{2.3}$$
$$Z_{22} = Z_y + j\omega(L_{cm} - 2M_{ycm}) \tag{2.4}$$
$$Z_{23} = -j\omega(L_{cm} - M_{ycm}) \tag{2.5}$$
$$Z_{33} = Z_p + j\omega L_{cm} \tag{2.6}$$

Siendo:

$$Z_p = \frac{1}{j\omega C_t + \frac{1}{RN1}} \qquad Z_y = R_y + j\omega L_y + \frac{1}{j\omega C_y} \tag{2.7}$$



(a) Esquemático del modo común.

(b) Esquemático del modo diferencial.

Figura 2.2 Circuitos equivalentes del filtro.

Y los elementos de la matriz de impedancia en modo diferencial:

$$Z_{11} = \frac{R_s}{2} + z_y \tag{2.8}$$

$$Z_{12} = j\omega(M_{ydm} + M_{yx}) - Zy \tag{2.9}$$

$$Z_{13} = -j\omega M_{ydm} \tag{2.10}$$

$$Z_{14} = -j\omega M_{yx} \tag{2.11}$$

$$Z_{22} = Z_x + Z_y + j\omega(L_{dm} + 2(M_{xdm} - M_{ydm} - M_{yx})) \tag{2.12}$$

$$Z_{23} = j\omega(M_{ydm} - M_{xdm} - L_{dm}) \tag{2.13}$$

$$Z_{24} = j\omega(M_{yx} - M_{xdm}) - Z_x \tag{2.14}$$

$$Z_{33} = Z_p + j\omega L_{dm} \tag{2.15}$$

$$Z_{34} = j\omega M_{xdm} \tag{2.16}$$

$$Z_{44} = Z_x + \frac{R_l}{2} \tag{2.17}$$

Siendo:

$$Z_p = \frac{1}{j\omega C_t + \frac{1}{RN2}} \qquad Z_y = R_y + j\omega L_y + \frac{1}{j\omega C_y} \qquad Z_x = \frac{R_x}{2} + j\omega\frac{L_x}{2} + \frac{1}{j\omega 2C_x} \tag{2.18}$$

Una vez obtenidas las matrices pueden calcularse las intensidades a partir de la tensión en cada bucle. El único valor no nulo del vector de tensiones corresponde en ambos circuitos a V_s.

$$\mathbf{V} = \overline{Z}\,\mathbf{I} \tag{2.19}$$

Al resolver el sistema de ecuaciones queda definido el valor de la intensidad que pasa por la carga R_l (i_2 en el circuito equivalente del modo común o i_4 en el del modo diferencial), y por tanto el valor de la tensión en la carga V_l.

La atenuación de un filtro puede medirse a través de su respuesta en frecuencia, el cociente entre la potencia de salida y entrada del filtro. Cuando la impedancia de la fuente y de la carga coinciden, su expresión en valor absoluto viene dada por:

$$|S_{21}| = 20\log\left(2\,\frac{V_l}{V_s}\right) \tag{2.20}$$

2.2 Especificaciones

A continuación se presenta la lista completa de especificaciones acordada con el Grupo de Investigación. En cada una de las especificaciones se añade un comentario para aclarar el funcionamiento del programa y la manera en que cumple con esa especificación concreta.

1. Debe partir de una gráfica de S21 medida (curva medida) que se lea de un fichero que se pueda especificar/buscar en el PC.

   Para ello con el botón "Insertar datos" se llama al explorador de archivos del ordenador que permite seleccionar el archivo .txt en el que se encuentran los datos. Una vez seleccionado una nueva ventana solicita el número de columna en el que se encuentran los datos.

2. Debe mostrar permanentemente o bajo demanda el esquemático del circuito equivalente (para identificar los parámetros que se incluyen en la ventana).

   El esquemático que se resuelve aparece en todo momento en la aplicación junto a los parámetros del mismo (Figura 2.3).

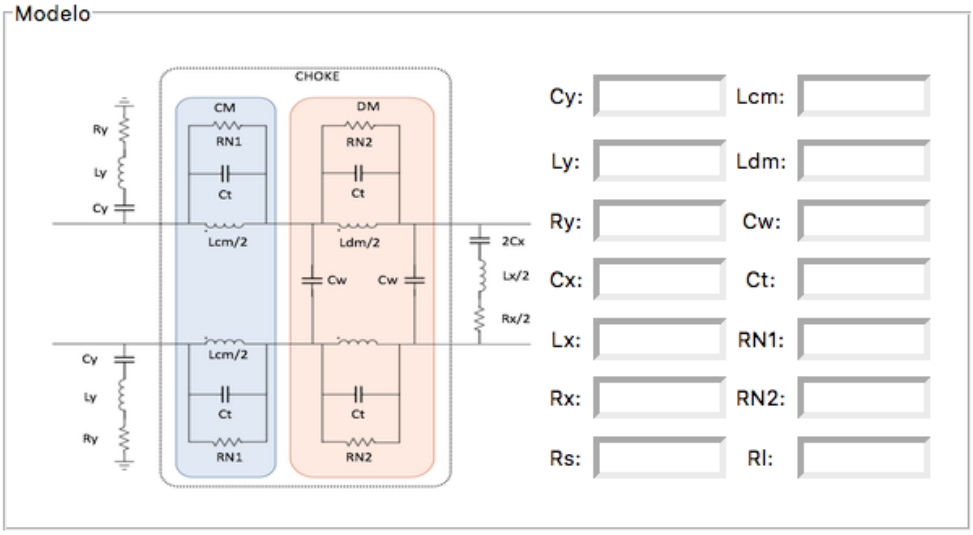


Figura 2.3 Sección del modelo.

3. También debe mostrar la curva teórica S21 que le corresponde con el valor especificado para los parámetros en las casillas/sliders: esta sería la “curva activa”.

   La curva generada por los parámetros introducidos por pantalla es mostrada por defecto al resolver el circuito.

4. Se deben poder modificar los parámetros del filtro con la consiguiente actualización de la curva activa.

   Cada vez que el botón "Calcula" es accionado se resuelve el circuito y se refrescan las gráficas.

5. Para estudiar sensibilidad y posibles soluciones múltiples: opción de guardar en memoria varias curvas correspondientes a diferentes conjuntos de parámetros, y que se pueda volver a ellas (recuperar sus parámetros): es decir, guardar la curva activa en una memoria y poder pasar una curva guardada a curva activa. Las curvas en memoria pueden hacerse visibles en la gráfica o no.

   La memoria consiste en un archivo de texto tabulado con diez columnas (de 0 a 9) de forma que los parámetros pueden cargarse y guardarse en las posiciones que se necesiten (Figura 2.4).

6. Posibilidad de guardar en un fichero de texto el conjunto de parámetros actual y la curva activa (para las curvas en memoria bastaría con pasarlas a activa y guardarlas, o bien incluir una opción de guardarlas todas/o solo activa).

   Las gráficas generadas puden visualizarse en un tamaño mayor al pulsar el botón "Ver". En la nueva ventana generada hay una casilla en la que debe introducirse el nombre del archivo que se quiera guardar:

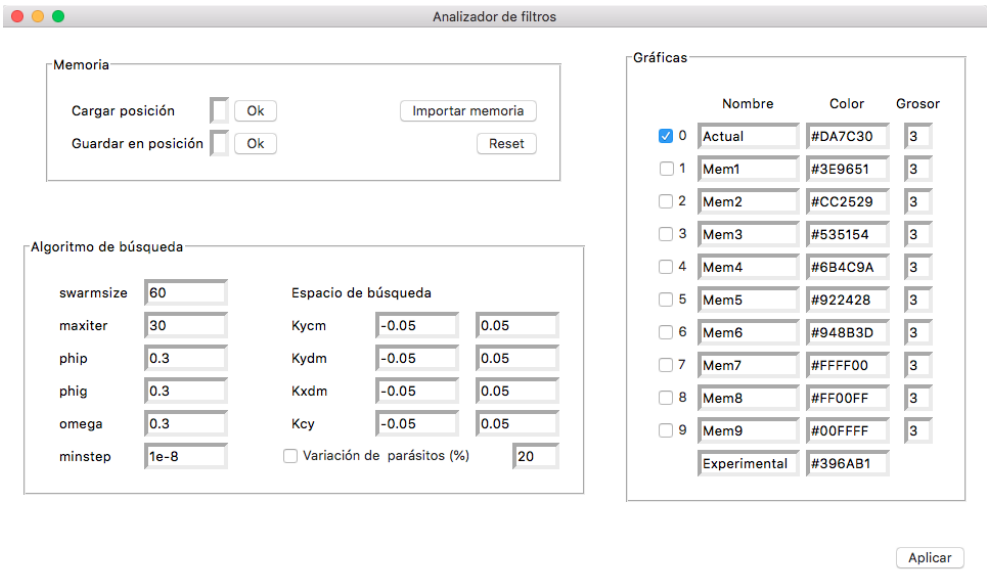


Figura 2.4 Ventana de opciones.

la memoria que contiene las diez posiciones con los parámetros guardados o las curvas en .txt o .svg (Figura 2.5).

7. Posibilidad de recuperar de un fichero con ese mismo formato una curva activa a partir del conjunto de parámetros guardados en ese fichero.

   El programa permite importar las memorias que previamente se han guardado. En caso de fallo también puede restaurarse la memoria escribiendo ceros en todas las posiciones al pulsar "Reset" (Figura 2.4).

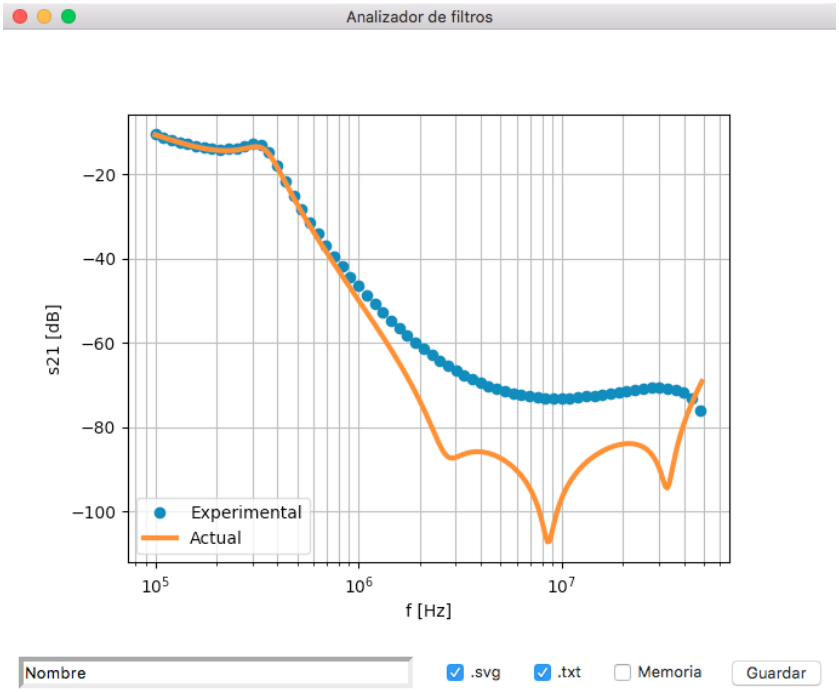


Figura 2.5 Gráfica aumentada.

8. Botón “imprimir” o “save graph” que genere un fichero de imagen (PNG o SVG) con la curva de S21 medida frente a la curva activa y/o las curvas en memoria. Bastaría con poder controlar cuáles se muestran en la gráfica y que el botón de “imprimir” saque la gráfica que esté actualmente en pantalla.

Puede guardarse de igual forma que en el punto 6 la imagen en .svg que contiene las gráficas generadas por las posiciones de memoria elegidas en las opciones (Figura 2.4).

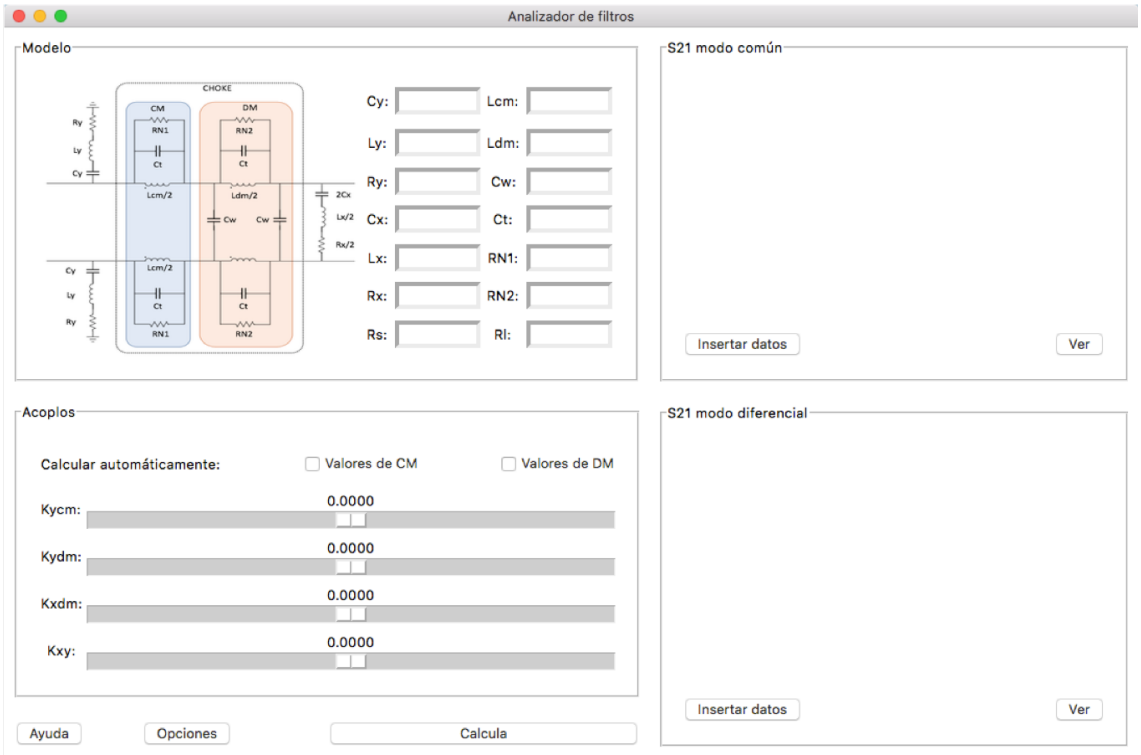


Figura 2.6 Ventana principal.

9. Opción de búsqueda de parámetros óptimos con algoritmo de búsqueda:

Todas las opciones de búsqueda son accesibles desde el menú de opciones (Figura 2.4).

a) Parámetros de partida para búsqueda serían los actuales en ventana (curva activa).

El espacio de búsqueda es por defecto los valores de los acoplos ±0.05 y en el caso de incluir los efectos parásitos en la búsqueda, su variación está limitada a un ±20 % (valor por defecto) de los valores actuales.

b) Posibilidad de especificar rango de búsqueda para cada parámetro. Se trata de evitar valores sin sentido físico y de acelerar el cálculo cuando se tenga una idea de por donde anda el valor que se busca. Congelar/descongelar parámetros: que sean o no variables para el proceso de búsqueda según interese.

El rango de búsqueda se puede especificar en las opciones, pudiendo incluirse o no la variación de los efectos parásitos de los condensadores (Figura 2.4).

c) Ventana con valores ajustables del algoritmo de búsqueda, como número máximo de iteraciones, error mínimo etc. En todo caso oculta en opciones avanzadas.

Se pueden especificar todos los parámetros del algoritmo PSO en las opciones (Figura 2.4).

d) Si un ajuste sale bien, se podría guardar en una memoria para poder correr varias veces el algoritmo de búsqueda y ver si los parámetros que da son estables, pudiendo comparar una curva (y conjunto de parámetros) con otra.

En el menú de opciones pueden guardarse los parámetros actuales en cualquier posición de memoria (Figura 2.4).

10. Opciones avanzadas: definición de colores/grosores y tipo de línea para curva medida, curva activa y curvas en memoria.

El aspecto de las curvas pueden cambiarse dentro de las opciones, variando su etiqueta, color, grosor y que curvas representar (Figura 2.4).

11. Posibilidad de ajuste con dos modos: común y diferencial (dos modelos de circuitos y dos conjuntos de parámetros). Podrían ser dos scripts diferentes, pero resulta cómodo ajustar los valores de CM primero y luego usar esos parámetros como punto de partida para el ajuste DM, que incorpora nuevos parámetros como los acoplos entre componentes.

El programa calcula las curvas $|S_{21}|$ en modo común y diferencial, pudiéndose realizar la búsqueda de parámetros en cada modo de forma independiente (Figura 2.7).

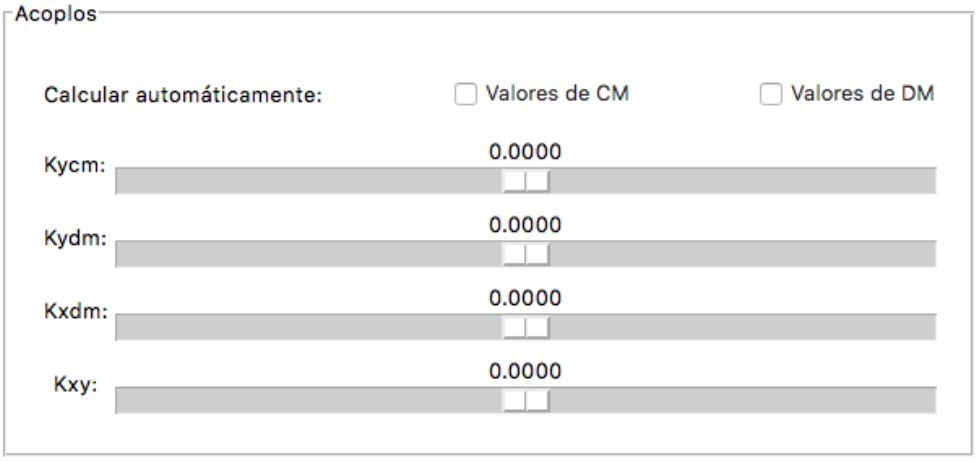


Figura 2.7 Sección de acoplos.

2.3 Algoritmo básico

El programa integra todas las especificaciones de forma que facilite el uso del mismo, minimizando el número de ventanas para hacer más accesibles los diferentes campos. Al usar la aplicación solo es necesario navegar por tres ventanas: principal, de opciones y la gráfica aumentada.

La operación básica del mismo consiste en introducir los datos necesarios en la ventana principal, modificar las opciones que se deseen para resolver el circuito y, una vez obtenidos los resultados, exportar los documentos desde la vista ampliada de la gráfica.

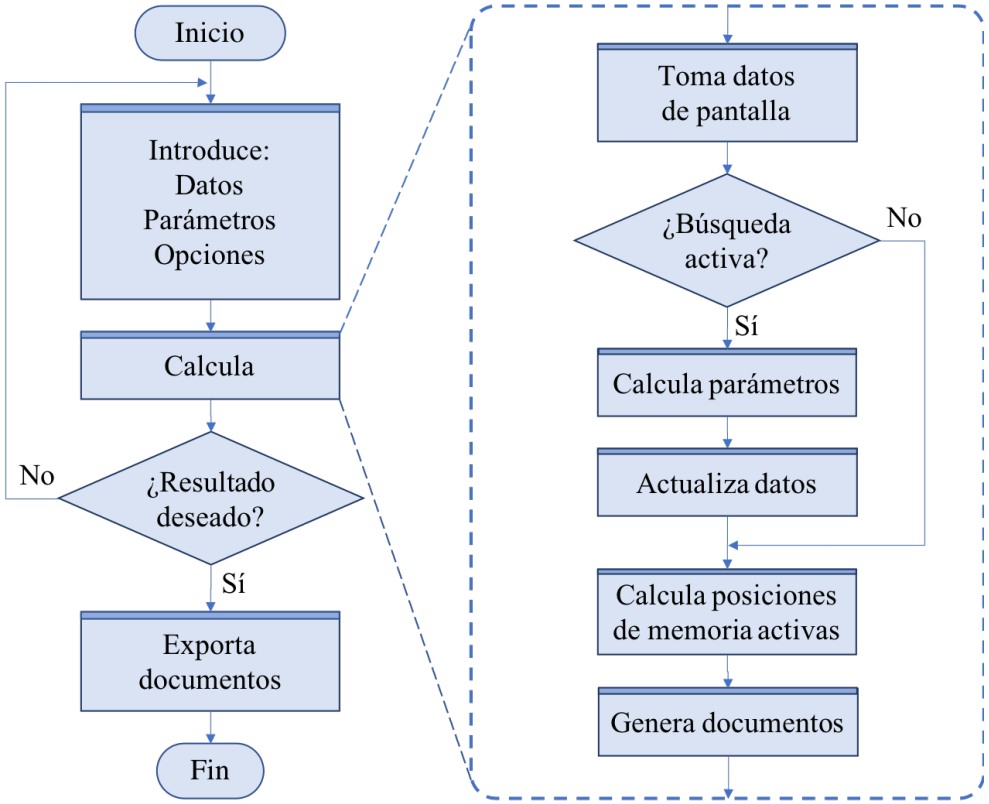


Figura 2.8 Diagrama de flujo con los procesos básicos de la aplicación.

La búsqueda de parámetros se realiza mediante la optimización de una función de coste por *Particle Swarm Optimization* (PSO). Dicha función es el error cuadrático medio que presentan los puntos de la curva $|S21|$ respecto a los datos experimentales introducidos en la aplicación, tomando como variables los acoplos y, si se desea, los efectos parásitos de los condensadores. La PSO se basa en la búsqueda de un mínimo en la función de coste mediante la distribución de puntos en el espacio de búsqueda (partículas) que evalúan la función de coste en su posición. En las distintas iteraciones cada partícula se moverá según su posición relativa a su mejor solución obtenida y la mejor solución del conjunto (enjambre). Esto hace que las partículas converjan eventualmente a un mínimo de la función de coste [11].

2.4 Guía de uso del programa

2.4.1 Antes de iniciar el programa

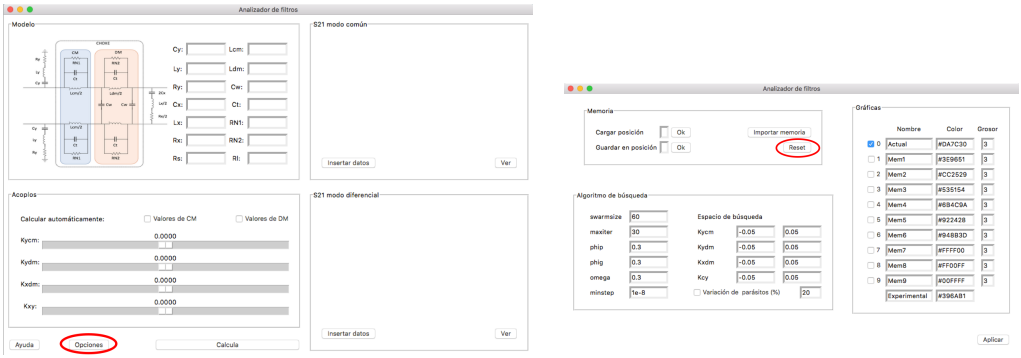
El ordenador donde quiera utilizarse la aplicación debe tener instalado Python 3.0 o una versión más reciente. También requiere las siguientes librerías:

- math
- numpy
- matplotlib
- shutil
- tkinter
- pillow
- pyswarm

Para iniciar el programa debe crearse una carpeta que contenga el script y la carpeta "imagenes". En el interior de esta carpeta se encuentra un esquema del filtro y una imagen blanca, "model.png" y "spacer.png" respectivamente. La imagen en blanco se utilizará cuando aún no se hayan creado gráficas al iniciar el programa. En la misma carpeta del script se crearán más archivos a medida que se use el programa, como son la memoria, las imágenes de las gráficas o los archivos de texto con sus puntos.

2.4.2 Inicio del programa

Al ejecutar el código se genera una ventana que contiene los bloques de la aplicación que permiten introducir parámetros y visualizar las gráficas, todos ellos en blanco. Si es la primera vez que se utiliza el programa debe crearse la memoria del mismo: un archivo de texto tabulado de 18 filas y 10 columnas con el nombre "memory.txt". Para ello dentro del apartado "Opciones" debe pulsarse el botón "Reset" como muestra la figura Figura 2.9. De esta forma se crea el archivo con ceros en todas sus posiciones.



(a) Ventana principal.

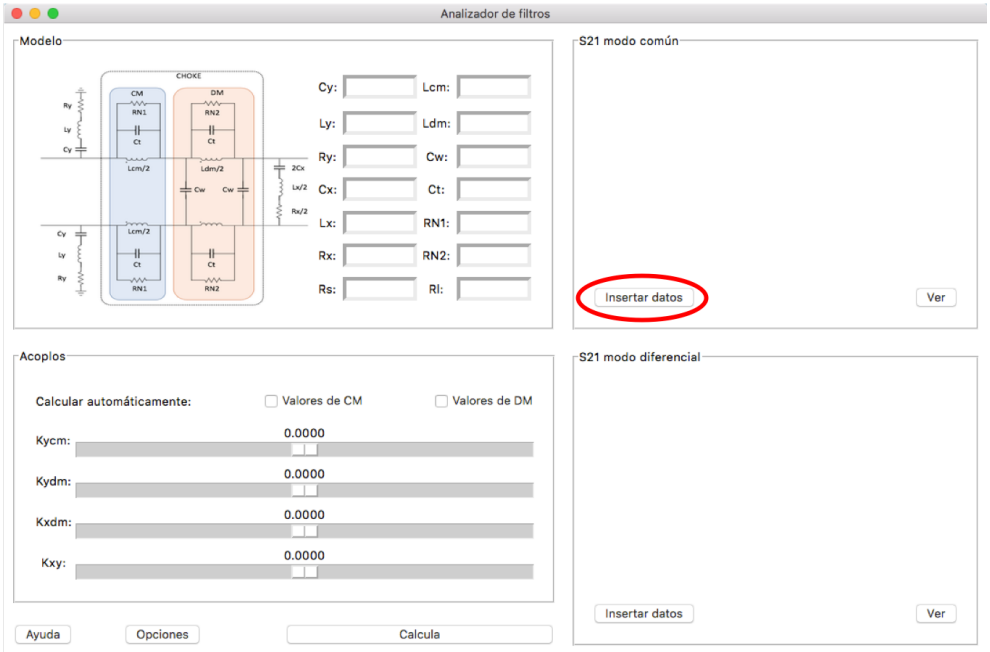
(b) Ventana de opciones.

Figura 2.9 Creación o reinicio de la memoria.

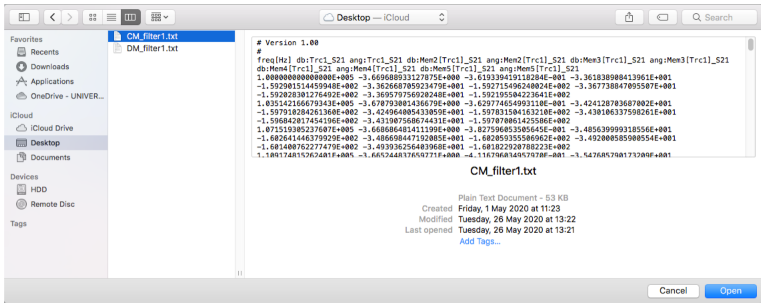
Para introducir los datos experimentales de las curvas $|S_{21}|$ en modo común y diferencial estos deben estar en un fichero de texto distribuido en columnas. Al pulsar el botón "Insertar datos" se abre el explorador de archivos del ordenador y al elegir un archivo aparece una ventana para seleccionar las columnas de los datos (Figura 2.10). Los parámetros del circuito pueden introducirse escribiéndolos en las cajas disponibles del modelo. Acepta los sufijos SPICE (p, n, u, m, K, etc).

2.4.3 Opciones de memoria

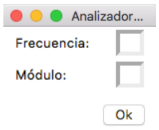
Es el primero de los bloques de la ventana de opciones. La memoria consta de diez posiciones en las que pueden guardarse los parámetros del modelo y los acoplos. Dentro de las opciones se pueden guardar y cargar en la pantalla posiciones de la memoria escribiendo su número $(0-9)$ y haciendo click en "Ok" (Figura 2.11 - I). Una vez abierta la ventana de opciones pueden modificarse tantas posiciones de memoria como se quiera sin necesidad de cerrarla accionado "Aplicar".



(a) Ventana principal.



(b) Ventana explorador de archivos.



(c) Elección de columnas.

Figura 2.10 Insercción de datos experimentales.

Si ya se dispone de un fichero de texto con las posiciones de memoria este puede importarse con el botón "Importar memoria". Junto a él se encuentra el botón "Reset", que sirve para inicializar la memoria y restaurarla en caso de fallo (Figura 2.11 - II).

2.4.4 Opciones de búsqueda

Una vez introducidos los datos deben especificarse los parámetros de la búsqueda, que se dividen en tres bloques:

1. Parámetros del algoritmo PSO (*Particle Swarm Optimization*): son los parámetros propios de este algoritmo que se introducen por pantalla como se muestra en Figura 2.12 - I:
   - **a)** *swarmsize*: número de partículas.
   - **b)** *maxiter*: número máximo de iteraciones.
   - **c)** *phip*: peso de la mejor solución de la partícula en la búsqueda.
   - **d)** *phig*: peso de la mejor solución global.
   - **e)** *omega*: inercia de la partícula.
   - **f)** *minstep*: diferencia entre aproximaciones sucesivas a partir del cual la búsqueda se detiene.
2. Espacio de búsqueda: son los valores que pueden tomar los parámetros. Los valores de los acoplos son por defecto un intervalo de 0.1 centrado en el valor que tenga el slider, y pueden modificarse escribiendo

el rango deseado. Además se pueden varias los efectos parásitos de los condensadores durante el ajuste en un tanto por ciento del valor original (Figura 2.12 - II).

3. El modo de búsqueda: en la ventana principal se selecciona el modo común o diferencial.

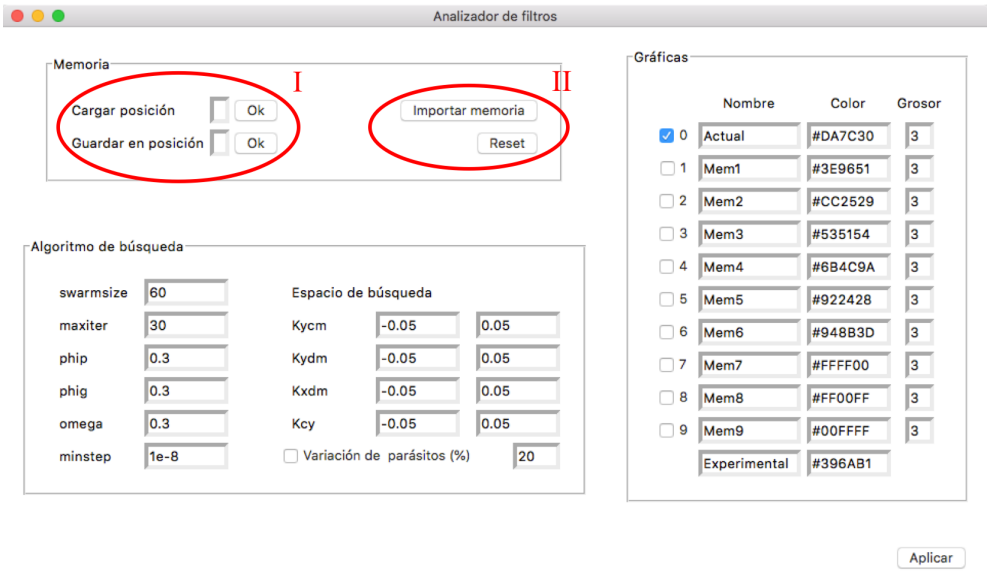


Figura 2.11 Opciones de memoria.

2.4.5 Opciones de representación

En este bloque se puede elegir qué curvas representar de los diez conjuntos de parámetros pudiendo alterarse de cada una la etiqueta en la leyenda, el color y el grosor. Siempre que se abra el menú de opciones solo estará marcada la curva actual, representándose por defecto junto a la curva experimental.

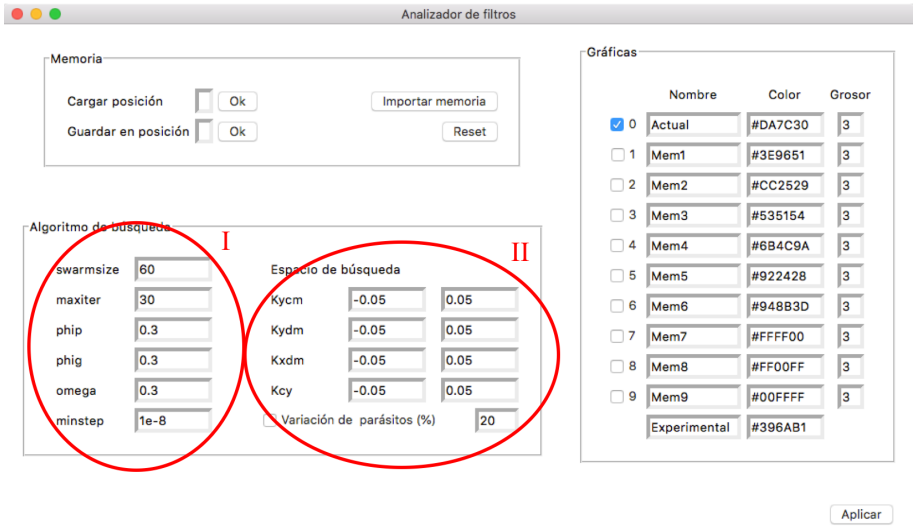


Figura 2.12 Opciones de búsqueda.

2.4.6 Obtención de datos

Desde la ventana principal se puede acceder a una vista ampliada de cada una de las gráficas. En esta nueva ventana están las opciones de guardado, pudiéndose obtener una imagen de la gráfica en formato vectorial y un archivo de texto con los puntos de las curvas que aparecen en la gráfica. La opción "Memoria" permite exportar el fichero con todas las posiciones de la memoria, de esta forma se puede trabajar con varios proyectos

importando y exportando los archivos de memoria y comparar fácilmente los parámetros obtenidos abriendo estos archivos. Para exportar las imágenes y los archivos de texto debe marcarse la casilla correspondiente, escribir el nombre con el que desea guardar el documento y pulsar "Guardar" (Figura 2.13).

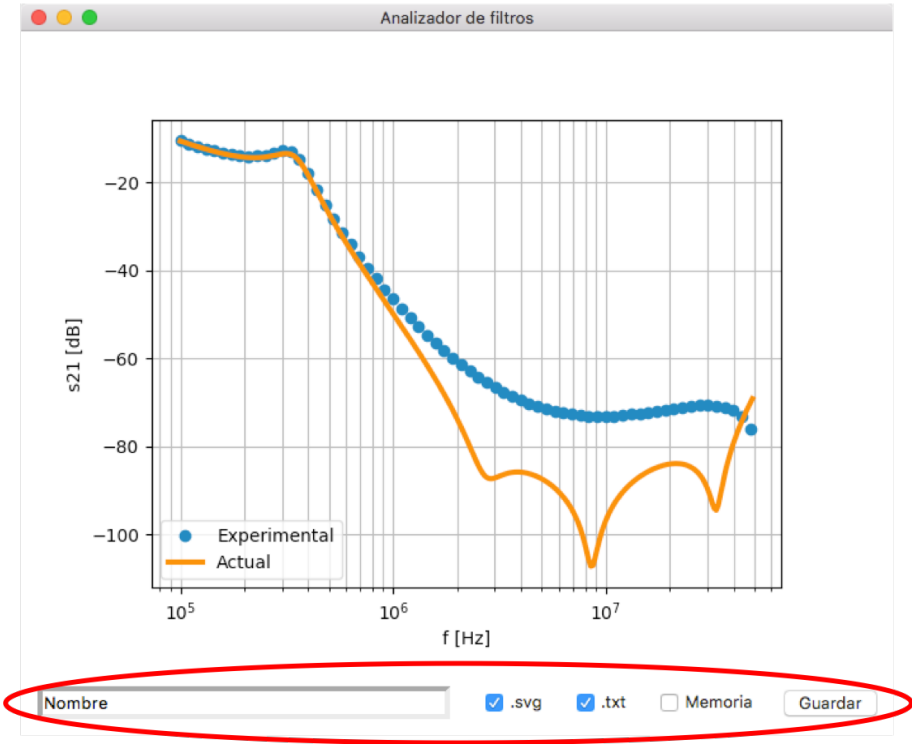


Figura 2.13 Opciones de guardado.

2.5 Conclusiones del capítulo

En este capítulo se ha expuesto el funcionamiento del programa, empezando por la obtención del modelo del filtro EMI y la descripción de los parámetros del mismo. El software se ha desarrollado a partir de las especificaciones mostradas en el segundo apartado, donde también se muestra cómo se ha cumplido con ellas. La operación de la aplicación se ha explicado en dos apartados distintos: en el primero se exponen los pasos de forma general junto a una breve explicación del algoritmo de búsqueda utilizado y el segundo es una guía de uso donde se explican detalladamente las funcionalidades del programa.

3 Resultados

Para verificar el funcionamiento del software se han realizado dos pruebas diferentes. En primer lugar se toman los datos del modelo SPICE de un filtro para comprobar la correcta resolución del circuito y la eficacia de la búsqueda de parámetros [12]. También se han usado las medidas experimentales de otro filtro al que se le han atenuado los acoplos entre sus componentes mediante el uso de caperuzas de cobre en los condensadores [13]. De este modo se puede comprobar la utilidad de la herramienta para modelar un filtro y estimar los acoplos después de aplicar distintas técnicas de atenuación.

3.1 Datos simulados con SPICE

Para obtener la respuesta en frecuencia del circuito se crean los modelos equivalentes para la excitaciones en modo común y en modo diferencial utilizando un programa de resolución de circuitos. Se realiza un barrido en frecuencia ente 100 kHz y 50 MHz dando a la fuente de tensión una amplitud de 1 V. De esta forma la expresión de la ganancia $|S_{21}|$ depende únicamente de V_l: la tensión a lo largo de la carga R_l.

$$|S_{21}| = 20\log(2V_l) \tag{3.1}$$

Código 3.1 Código del circuito equivalente en modo común.

```
Comon mode

*Netlist
Vs      1       0     DC 0 AC 1
Rs      1       2     100
Cy      2       3     44n
Ly      3       4     13.689n
Ry      4       0     46.361m
Lcm     2       5     4.62m
Ct      2       5     4.67p
RN1     2       5     24670
Rl      5       0     100
Kycm    Ly      Lcm   0

.control
    ac dec 100 100k 50MEG
    set filetype=ascii
    plot db(2*V(5))
    write datos_cm.txt db(2*V(5))
.endc
.end
```

Estos códigos guardan las curvas en archivos .txt que pueden ser leídos por el programa como si fueran datos experimentales. Se han añadido los acoplamientos y los efectos parásitos calculados para modelar el circuito, de forma que la aplicación tendrá que partir de unos valores aproximados de los mismos antes de realizar el ajuste.

Código 3.2 Código del circuito equivalente en modo diferencial.

```
Differential Mode

*Netlist
Vs      1       0       DC 0 AC 1
Rs      1       2       25
Cy      2       3       44n
Ly      3       4       13.689n
Ry      4       0       46.361m
Ldm     2       5       4.95u
Ct      2       5       4.67p
RN2     2       5       10310
Rl      5       0       25
Cx      5       6       880n
Lx      6       7       5.6408n
Rx      7       0       20m
Kydm    Ly      Ldm     -0.4703
Kxdm    Lx      Ldm     -0.1240
Kyx     Ly      Lx      0.0504

.control
    ac dec 100 100k 50MEG
    set filetype=ascii
    plot db(2*V(5))
    write datos_dm.txt db(2*V(5))
.endc
.end
```

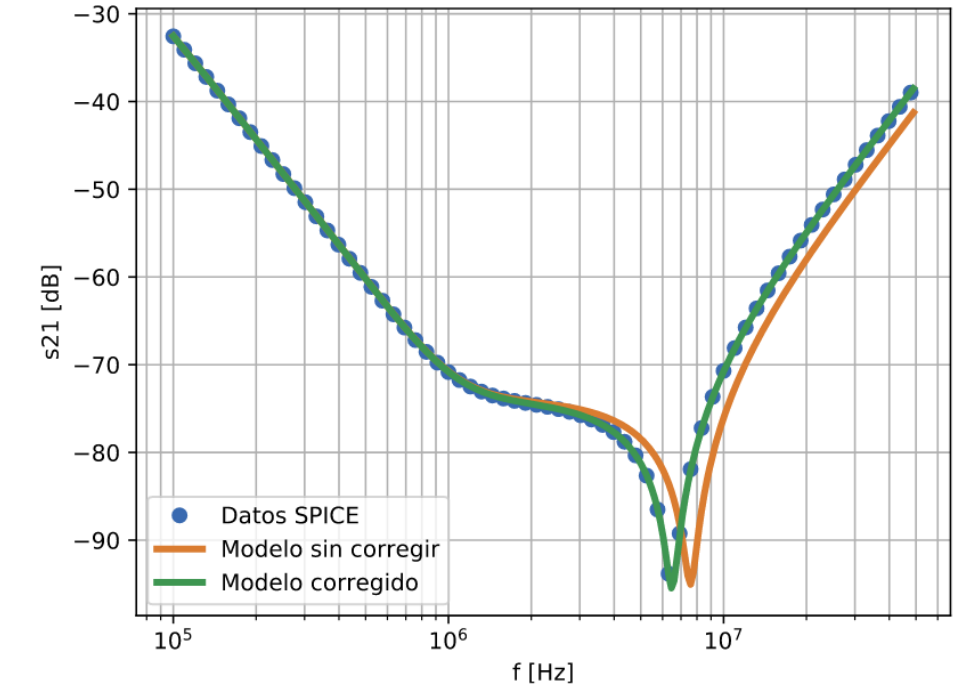
Tanto en los condensadores C_y como en el C_x se toman como valores parásitos iniciales una inductancia de 10 nH y una resistencia de 40 mΩ. Al realizar aproximaciones sucesivas estos parámetros varían dentro de su espacio de búsqueda hasta encontrar un error mínimo entre las curvas introducidas como dato y las modeladas.

Dependiendo del espacio de búsqueda elegido el programa puede encontrar distintos conjuntos de parámetros hacia los que converge. Haciendo uso de la memoria es posible distinguir la mejor solución de las obtenidas por el programa. Tras realizar la búsqueda de parámetros se obtienen las aproximaciones de ambas curvas con un error cuadrático medio inferior a 1 dB como muestran la Figura 3.1.

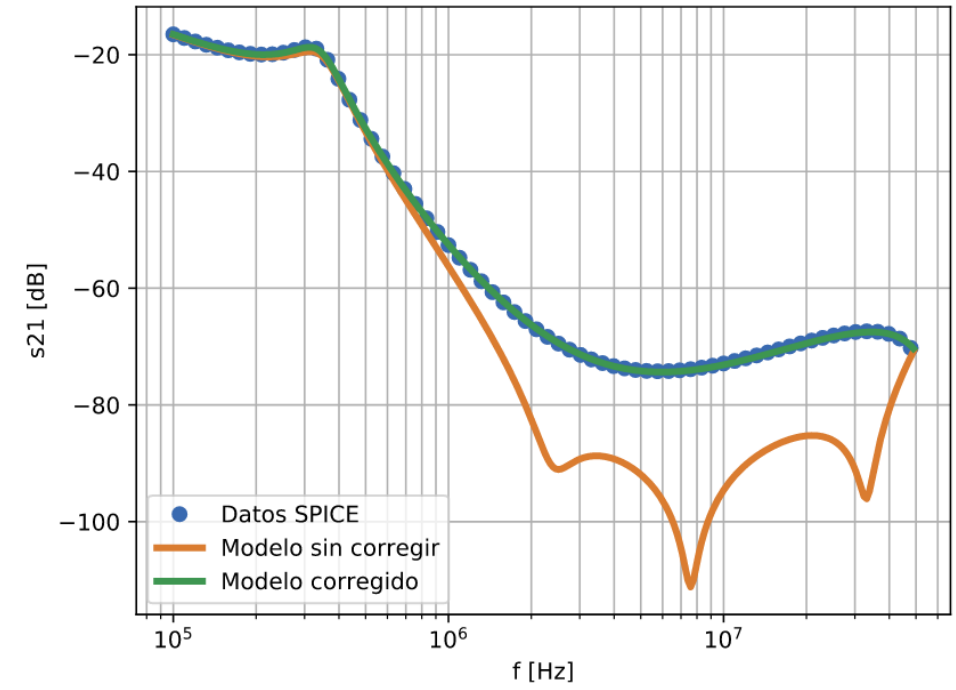
Tabla 3.1 Valores de parámetros del modelo SPICE y su aproximación.

	L_y (nH)	R_y (mΩ)	L_x (nH)	R_x (mΩ)	K_{ycm}	K_{ydm}	K_{xdm}	K_{xy}
Modelo SPICE	13.689	46.361	11.282	40.000	0.0000	−0.4703	−0.1240	0.0504
Aproximación	13.6	44.7	10.0	31.3	0.0002	−0.4145	−0.1243	0.0469

Los parámetros obtenidos con la mejor aproximación se muestran en la Tabla 3.1. Las diferencias existentes entre los valores de la simulación y la aproximación muestran la sensibilidad de los parámetros, siendo en cualquier caso un modelo aceptable al ajustar las gráficas con poco error.



(a) Respuestas en frecuencia en modo común.



(b) Respuestas en frecuencia en modo diferencial.

Figura 3.1 Respuesta en frecuencia del modelo sin acoplos y ajustado a los datos SPICE.

3.2 Datos experimentales

Para comprobar que la aplicación es capaz de ajustar curvas con distintos valores de acoplos en un mismo filtro se parte de dos medidas:

- Respuesta en frecuencia de un filtro en modo común y diferencial. Está compuesto por un C_x de 470 nF, dos C_y de 47 nF y el choke caracterizado con anterioridad.
- Respuesta en frecuencia del mismo filtro en modo común y diferencial tras colocar caperuzas de cobre en sus condensadores. Esto hará que los acoplos entre los componentes del filtro sean menores y por tanto el filtro tenga una mayor atenuación [14].

Del mismo modo que en el apartado anterior se estiman los efectos parásitos de los componentes del filtro y al introducirlos en el algoritmo se modificarán para obtener un mejor ajuste. En este caso se han tomado unos límites más amplios en la búsqueda: impedancias de 5 a 20 nH y resistencias de 10 a 40 mΩ.

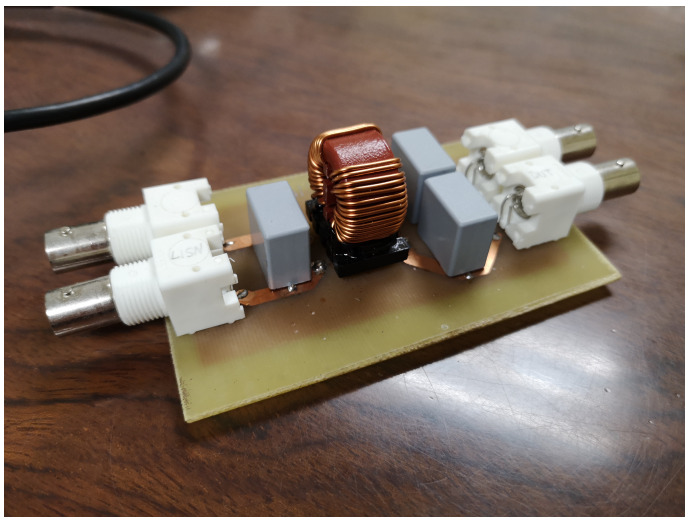


Figura 3.2 Filtro del que se han obtenido las medidas (Adaptado de [13]).

Se realiza la búsqueda de parámetros para ajustar las curvas y se repite cambiando el espacio de búsqueda de los acoplos para obtener distintas soluciones. Estas se guardan en diferentes posiciones de la memoria y se comparan para encontrar la mejor aproximación.

Para tomar los dos conjuntos de parámetros que modelan los circuitos equivalentes se ha tenido en cuenta que el orden de magnitud de los acoplos debe ser coherente con el filtro previamente modelado y los efectos parásitos de los componentes del filtro con y sin caperuzas deben ser coherentes.

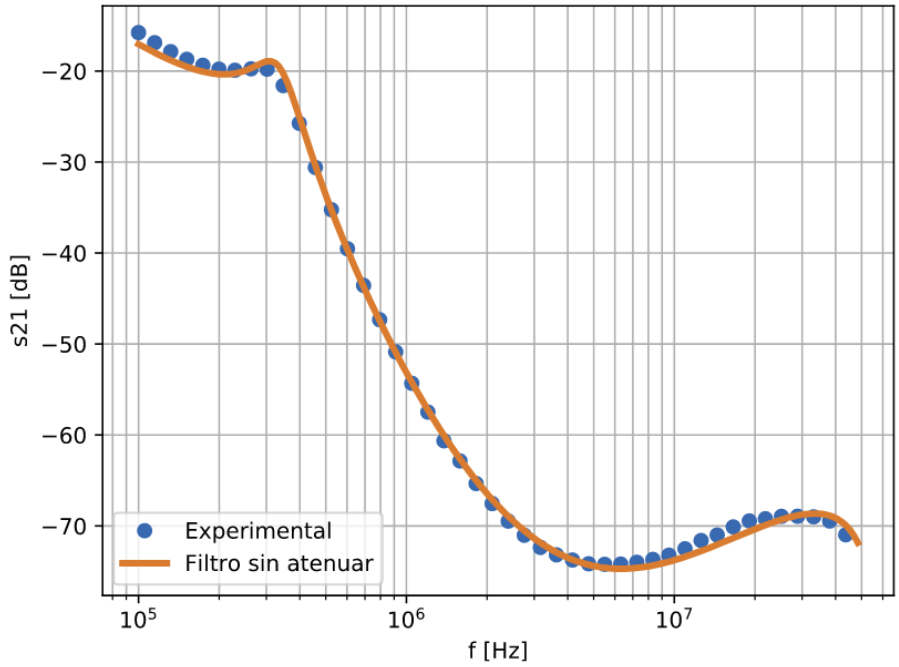
3.2.1 Modo diferencial

Para el ajuste en modo diferencial se puede apreciar en la Tabla 3.2 que los valores de los acoplos en el caso del filtro atenuado son menores en valor absoluto mientras el resto de los valores no varían en gran medida. Debe de tenerse en cuenta que la sensibilidad de las resistencias es menor que la de las inductancias parásitas y las constantes de acoplamiento, por lo que la precisión con la que son calculadas es menor.

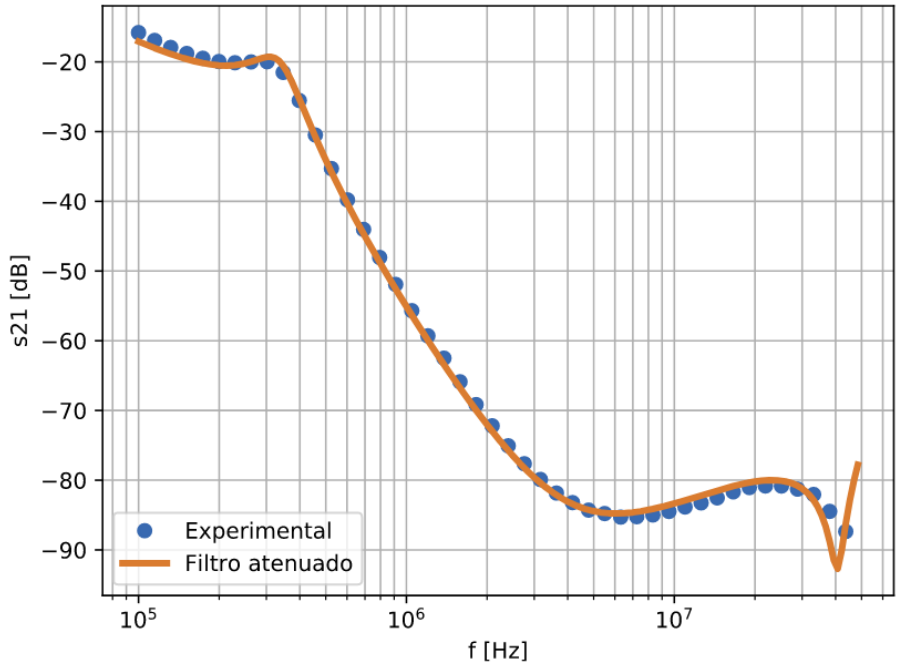
Tabla 3.2 Modelo en modo diferencial a partir de datos experimentales.

	L_x (nH)	R_x (mΩ)	K_{ydm}	K_{xdm}	K_{xy}
Filtro sin atenuar	6.49	44.7	−0.1417	−0.1206	0.0231
Filtro atenuado	6.88	40.0	−0.0087	−0.0444	0.0029

En la Figura 3.3 se muestran los ajustes de ambas curvas, siendo algo más pobre en los extremos del intervalo. El error cuadrático medio de las aproximaciones es menor de 2 dB.



(a) Filtro sin atenuar.



(b) Filtro atenuado con caperuzas de cobre.

Figura 3.3 Ajustes en modo diferencial.

3.2.2 Modo común

El ajuste en modo común, al tener menos variables presenta un mayor número de posibles soluciones. Esto hace que sea más difícil encontrar una pareja de soluciones coherentes. Los valores mostrados en la Tabla 3.3 resultan en un buen ajuste de las curvas (Figura 3.3).

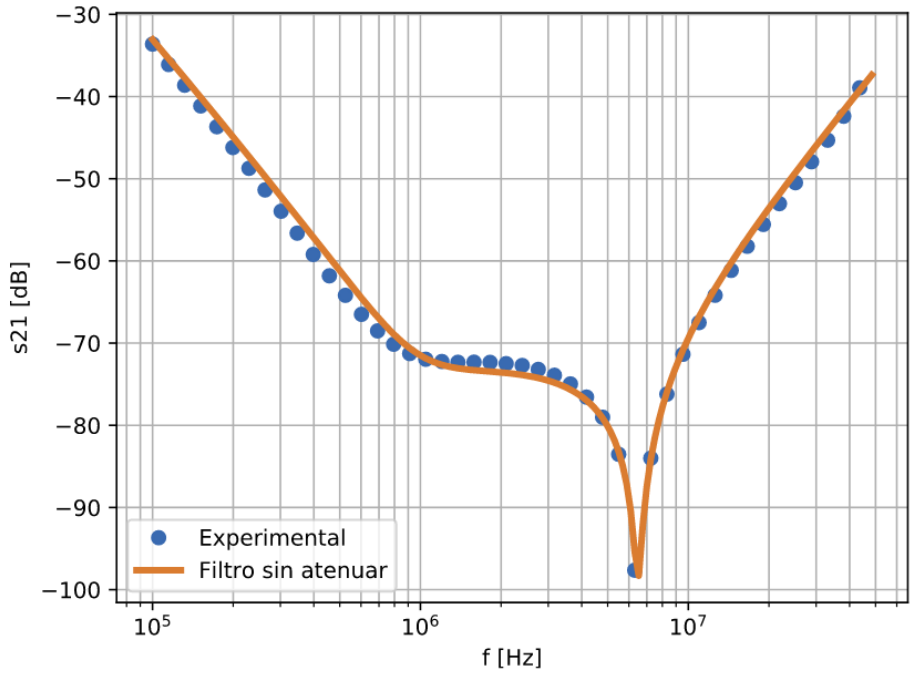
Tabla 3.3 Modelo modo común a partir de datos experimentales.

	L_y (nH)	R_y (mΩ)	K_{ycm}
Filtro sin atenuar	15.7	16.1	−0.0123
Filtro atenuado	11.8	17.5	−0.0138

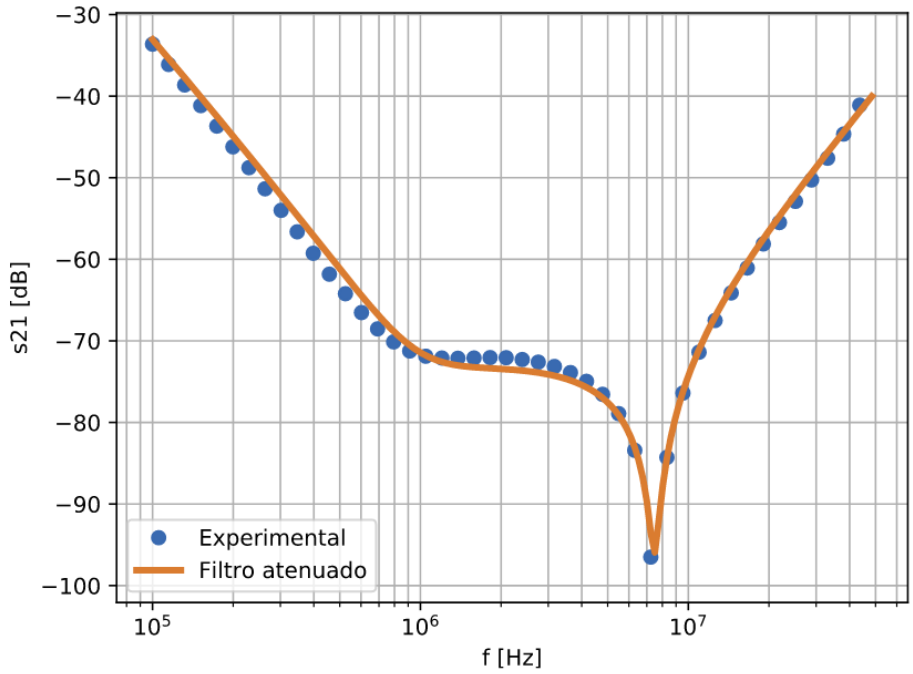
Para solucionar la incertidumbre se podrían caracterizar los componentes del filtro antes de realizar el análisis, dejando por determinar un solo grado de libertad en modo común: el acoplo entre el condensador C_y y el choke. También ayudaría la caracterización de los componentes al ser modificados, como la variación de la inductancia parásita de los condensadores al colocar el recubrimiento de cobre.

3.3 Conclusiones del capítulo

El ajuste de los datos simulados en SPICE demuestra que las ecuaciones implementadas en el programa resuelven correctamente el circuito, y el algoritmo de búsqueda es capaz de encontrar los parámetros del circuito minimizando el error cuadrático medio. Los datos experimentales también se han ajustado de manera satisfactoria por el mismo procedimiento, siendo el ajuste en modo común más difícil por tener menos parámetros y ser el rango de soluciones aceptables más amplio.



(a) Filtro sin atenuar.



(b) Filtro atenuado con caperuzas de cobre.

Figura 3.4 Ajustes en modo común.

4 Conclusiones

A lo largo de este trabajo se ha explicado el desarrollo del software de modelado de filtros EMI y se ha probado su eficacia con datos simulados y experimentales.

Al introducir los datos simulados en SPICE se prueba que la aplicación resuelve correctamente las ecuaciones de los circuitos equivalentes, ajustando los valores de los efectos parásitos y los acoplos para dar la solución óptima (menor error cuadrático medio), que coincide con los valores dados en la simulación.

El programa es capaz de ajustar las curvas experimentales introducidas con un error cuadrático medio menor a 2 dB, creando un modelo del filtro que reproduce el comportamiento del filtro medido en todo el rango de frecuencias.

En el modelado del circuito equivalente en modo diferencial es capaz de encontrar la solución óptima a partir de los valores estimados de los efectos parásitos de los condensadores. En el caso del circuito equivalente en modo común es necesario calcular estos valores previamente para poder distinguir las posibles soluciones con mayos precisión.

Debido a la gran variación de sensibilidad entre los diferentes parámetros que modelan el filtro sería de interés realizar un estudio de sensibilidad de los parámetros de los filtros EMI.

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] N. Violette, *Electromagnetic Compatibility Handbook*, 1st ed. Springer, 1987, ch. 1.

[2] "Radio frequency devices," Federal Communications Commission, Standard 47 CFR Part 15, 1975.

[3] C. R. Paul, *Introduction to Electromagnetic Compatibility*, 2nd ed. John Wiley & Sons, Inc, 2006, ch. 1.

[4] J. A. Rosero, J. A. Ortega, E. Aldabas and L. Romeral, "Moving towards a more electric aircraft," *IEEE Aerospace and Electronic Systems Magazine*, vol. 22, no. 3, pp. 3–9, 2007.

[5] "Environmental Conditions and Test Procedures for Airborne Equipment," Radio Technical Commission for Aeronautics, Standard DO-160G, dec 2010.

[6] "Airworthiness Certification Criteria," Department of Defense, U.S., Handbook MIL-HDBK-561C, dec 2014.

[7] C. R. Paul, *Introduction to Electromagnetic Compatibility*, 2nd ed. John Wiley & Sons, Inc, 2006, ch. 6.2.

[8] H. W. Ott, *Electromagnetic Compatibility Engineering*. John Wiley & Sons, Inc, 2009, ch. 13.2.

[9] tkinter — python interface to tcl/tk. [Online]. Available: https://docs.python.org/3/library/tkinter.html#module-tkinter

[10] Particle swarm optimization (pso) with constraint support. [Online]. Available: https://pythonhosted.org/pyswarm/

[11] R. Poli, J. Kennedy and T. Blackwell , "Particle swarm optimization," *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[12] N. Navea, "Caracterizacion de efectos de alta frecuencia en filtros EMI para convertidores de potencia," TFM, Universidad de Sevilla, 2018.

[13] J. Bernal, "Efecto de laminas de ferrita y de caperuzas de cobre en filtros EMI," (sin publicar).

[14] J. Bernal and M. J. Freire, "On-site, quick and cost-effective techniques for improving the performance of EMI filters by using conducting bands." 2016 IEEE International Symposium on Electromagnetic Compatibility (EMC), 2016, pp. 390–395.

Anexo: Código completo

```
1  #! usr/bin/python
2
3  from tkinter import *
4  from tkinter import messagebox
5  from tkinter import filedialog
6  from PIL import ImageTk,Image
7  from pyswarm import pso
8  import numpy as np
9  import math as m
10 import matplotlib.pyplot as plt
11 import shutil
12
13
14
15
16
17 # Definición de las funciones
       ########################################################################
18
19 def show_help():
20 # Abre el documento con la info detallada
21  messagebox.showinfo(title="Ayuda",  message='''Este programa permite
       representar las respuestas\
22  en modo común y diferencial de un filtro teniendo en cuenta los efectos pará
       sitos de sus compo\
23 nentes y sus acoplamientos, los cuales pueden calcularse o intoducirse
       manualmente. Los paráme\
24 tros que modelan el filtro son:
25
26 Condensadores:
27 -Cy: capacidad de Y
28 -Ly: inductancia parásita de Y
29 -Ry: resistencia de Y
30 -Cx: capacidad de X
31 -Lx: inductancia parásita X
32 -Rx: resistencia del X
33
34 CMC (choke de  común):
35 -Lcm: inductancia en modo común (cm)
36 -Ldm: inductancia en modo diferencial (dm)
37 -Cw: capacitancia parásita entre devanados
```

```
-Ct: capacitancia parásita
-RN1: resistencia parásita en cm
-RN2: resistencia parásita en dm

Acoplamientos:
-Kycm: entre Cy y CMC en cm
-Kydm: entre Cy y CMC en dm
-Kxdm: entre Cx y CMC en dm-Kxy: entre Cx y Cy

Equipo de medida
-Rs: impedancia de la fuente
-Rl: impedancia de la carga''')




def solve_circuit(K=None, search_mode=None):
# Resuleve el circuito CM o DM
 global s21_cm
 global s21_dm
 global s21x_cm
 global s21x_dm
 global parasite

 # Define las variables del algoritmo de búsqueda
 if search_mode is "cm":
  data[14] = str(K[0])
  if search_parasite.get():
   data[1] = str(K[1])
   data[2] = str(K[2])

 if search_mode is "dm":
  data[15] = str(K[0])
  data[16] = str(K[1])
  data[17] = str(K[2])
  if search_parasite.get():
   data[4] = str(K[3])
   data[5] = str(K[4])

 # Convierte los str a float con los sufijos de SPICE
 Cy, Ly, Ry, Cx, Lx, Rx, Rs, Lcm, Ldm, Cw, Ct, RN1, RN2, Rl, Kycm, Kydm, Kxdm,
    Kyx = [float(i.replace("T", "e12").replace("G", "e9").replace("MEG", "e6").
    replace("K", "e3").replace("m", "e-3").replace("u", "e-6").replace("n", "e
    -9").replace("p", "e-12").replace(" ", "")) for i in data]

 # Valores parásitos a optimizar
 parasite = [Ly, Ry, Lx, Rx]

 # Resolución del modo común
 s21_cm = np.zeros(len(freqx_cm))
 cont = 0

 for f in freqx_cm:
  # Frecuencia
  jw = 2*m.pi*f*1j

```

```
92    # Acoplo
93    M = Kycm*m.sqrt(Ly*Lcm)
94
95    # Impedancias agrupadas por elemento
96    zp = 1/(jw*Ct + 1/RN1)
97    zy = Ry + 1/(jw*Cy) + jw*Ly
98
99    # Matriz de impedancias
100   z11 = 2*Rs + zy
101   z12 = -zy + jw*(M)
102   z13 = -jw*M
103   z22 = zy + jw*(Lcm-2*M) + Rl/2
104   z23 = -jw*(Lcm-M)
105   z33 = zp + jw*Lcm
106
107   z_matrix_cm = np.mat([[z11, z12, z13],
108                         [z12, z22, z23],
109                         [z13, z23, z33]])
110
111   # Vectores de tensión e intenssidad
112   v_cm = np.array([1, 0, 0])
113
114   # Resolución
115   i_cm = np.dot(np.linalg.inv(z_matrix_cm), v_cm)
116   vlcm = np.absolute(np.dot([0, 2*Rl, 0], i_cm.transpose()))
117
118   # s21 en CM
119   s21_cm[cont] = 20*m.log10(2*vlcm)
120
121   cont = cont + 1
122
123  # Resolución del modo diferencial
124  s21_dm = np.zeros(len(freqx_dm))
125  cont = 0
126
127  for f in freqx_dm:
128   # Frecuencia
129   jw = 2*m.pi*f*1j
130
131   # Acoplos
132   Mydm = Kydm*m.sqrt(Ly*Ldm)
133   Mxdm = Kxdm*m.sqrt(Lx*Ldm/2)
134   Myx = Kyx*m.sqrt(Ly*Lx/2)
135
136   # Impedancias agrupadas por elemento
137   Zp = 1/(jw*Ct + 1/(RN2))
138   Zy = Ry + jw*Ly + 1/(jw*Cy)
139   Zx = Rx/2 + jw*Lx/2 + 1/(jw*2*Cx)
140
141   # Matriz de impedancias
142   z11 = Rs/2 + Zy
143   z12 = jw*(Mydm + Myx) - Zy
144   z13 = -jw*Mydm
145   z14 = -jw*Myx
146   z22 = Zx + Zy + jw*(Ldm + 2*(Mxdm - Mydm - Myx))
147   z23 = jw*(Mydm - Mxdm - Ldm)
148   z24 = jw*(Myx - Mxdm) - Zx
```

```
149   z33 = Zp + jw*Ldm
150   z34 = jw*Mxdm
151   z44 = Zx + Rl/2
152
153   z_matrix_dm = np.mat([[z11, z12, z13, z14],
154                         [z12, z22, z23, z24],
155                         [z13, z23, z33, z34],
156                         [z14, z24, z34, z44]])
157
158   # Vector tensión normalizado
159   v_dm = np.array([1, 0, 0, 0])
160
161   # Resolución
162   i_dm = np.dot(np.linalg.inv(z_matrix_dm), v_dm)
163   vldm = np.absolute(np.dot([0, 0, 0, Rl/2], i_dm.transpose()))
164
165   # s21 en DM
166   s21_dm[cont] = 20*m.log10(2*vldm)
167
168   cont = cont + 1
169
170
171
172
173
174 def cost_function_cm(K):
175 # Función a minimizar para buscar Kycm
176  solve_circuit(K, search_mode="cm")
177
178  # Error cuadrático medio de los puntos de las curvas
179  aux = np.array(s21x_cm) - np.array(s21_cm)
180  return np.mean(np.multiply(aux, aux))
181
182
183
184
185
186 def cost_function_dm(K):
187 # Función a minimizar para buscar Kydm, Kxdm, Kyx
188  solve_circuit(K, search_mode="dm")
189
190  # Error cuadrático medio de los puntos de las curvas
191  aux_1 = np.array(s21x_dm) - np.array(s21_dm)
192  return np.mean(np.multiply(aux_1, aux_1))
193
194
195
196
197
198 def solve_main():
199 # Funcón resolución (lee opciones en pantalla y llama a solve_cicuit)
200  global autoslot
201
202  # Toma los datos de la pantalla
203  data  = [box_1.get(), box_2.get(), box_3.get(), box_4.get(), box_5.get(),
     box_6.get(), box_7.get(), box_8.get(), box_9.get(), box_10.get(), box_11.
```

```
    get(), box_12.get(), box_13.get(), box_14.get(), str(slider_1.get()), str(
    slider_2.get()), str(slider_3.get()), str(slider_4.get())]
204
205 # Inicia la búsqueda de parámetros CM
206 if search_cm.get():
207
208  # Creación del espacio de búsqueda
209  if search_parasite.get():
210   lb = [lower_bound[0], (1-delta)*parasite[0], (1-delta)*parasite[1]]
211   ub = [upper_bound[0], (1+delta)*parasite[0], (1+delta)*parasite[1]]
212  else:
213   lb = [lower_bound[0]]
214   ub = [upper_bound[0]]
215
216  # Optimización
217  Kopt, fopt = pso(cost_function_cm, lb, ub, swarmsize=int(data_opt[0]),
      maxiter=int(data_opt[1]), phip=float(data_opt[2]), phig=float(data_opt[3]),
       omega=float(data_opt[4]), minstep=float(data_opt[5]), debug=True)
218
219  # Actualiza el valor óptimo
220  slider_1.set(Kopt[0])
221  data[14] = str(Kopt[0])
222
223  if search_parasite.get():
224   data[1] = str("%.3g" % Kopt[1])
225   data[2] = str("%.3g" % Kopt[2])
226   box_2.delete(0, END)
227   box_3.delete(0, END)
228   box_2.insert(0, data[1])
229   box_3.insert(0, data[2])
230
231 # Inicia la búsqueda de parámetros DM
232 if search_dm.get():
233  # Creación del espacio de búsqueda (Se divide en dos para evitar la
      convergencia a un mínimo relativo)
234  if search_parasite.get():
235   lb = [lower_bound[1], lower_bound[2], lower_bound[3], (1-delta)*parasite[2],
       (1-delta)*parasite[3]]
236   ub = [upper_bound[1], upper_bound[2], upper_bound[3], (1+delta)*parasite[2],
       (1+delta)*parasite[3]]
237  else:
238   lb = [lower_bound[1], lower_bound[2], lower_bound[3]]
239   ub = [upper_bound[1], upper_bound[2], upper_bound[3]]
240
241  # Optimización
242  Kopt, fopt = pso(cost_function_dm, lb, ub, swarmsize=int(data_opt[0]),
      maxiter=int(data_opt[1]), phip=float(data_opt[2]), phig=float(data_opt[3]),
       omega=float(data_opt[4]), minstep=float(data_opt[5]), debug=True)
243
244  # Actualiza los valores
245  slider_2.set(Kopt[0])
246  slider_3.set(Kopt[1])
247  slider_4.set(Kopt[2])
248  data[15] = str(Kopt[0])
249  data[16] = str(Kopt[1])
250  data[17] = str(Kopt[2])
251
```

```
 if search_parasite.get():
  data[4] = str("%.3g" % Kopt[3])
  data[5] = str("%.3g" % Kopt[4])
  box_5.delete(0, END)
  box_6.delete(0, END)
  box_5.insert(0, data[4])
  box_6.insert(0, data[5])

# Guarda los parametros actuales en la ranura 0
autoslot = 0
save_data()

# Inicializa las figuras y representa la curva experimental
fig_cm, ax_cm = plt.subplots()
ax_cm.plot(freqx_cm, s21x_cm, 'o', markevery=4, label=labels[11], color=colors
   [10])
fig_dm, ax_dm = plt.subplots()
ax_dm.plot(freqx_dm, s21x_dm, 'o', markevery=4, label=labels[11], color=colors
   [10])

# Inicializa la matriz con los puntos
txt_cm = [np.transpose(freqx_cm),0,0,0,0,0,0,0,0,0,0]
txt_dm = [np.transpose(freqx_dm),0,0,0,0,0,0,0,0,0,0]

# Resuleve el circuito para las ranuras de memoria selecionadas
for i in plot_index:
 if i.get():
  load_data(refresh=0)   # Se carga en la posición 0 (Actual)
  solve_circuit()    # Se llama a la función que resuelve el circuito y se
   representa
  ax_cm.plot(freqx_cm, s21_cm, linewidth=widths[autoslot], color=colors[
   autoslot], label=labels[autoslot+1])
  ax_dm.plot(freqx_dm, s21_dm, linewidth=widths[autoslot], color=colors[
   autoslot], label=labels[autoslot+1])
  txt_cm[autoslot+1] = np.transpose(s21_cm)
  txt_dm[autoslot+1] = np.transpose(s21_dm)
 else:
  txt_cm[autoslot+1] = np.transpose(np.zeros(len(s21_cm)))
  txt_dm[autoslot+1] = np.transpose(np.zeros(len(s21_dm)))
 autoslot = autoslot + 1
del autoslot
txt_cm = np.transpose(txt_cm)
txt_dm = np.transpose(txt_dm)

# Escribe los puntos en un txt tabulado
txt_index = [1]
for i in plot_index:
 txt_index.append(i.get())

# Puntos CM
with open('curve_cm.txt', 'w') as curve:
 for i in list(range(11)):
  if txt_index[i]:
   curve.write("%s" % labels[i])
  if i is 10:
   curve.write("\n")
  elif txt_index[i]:
```

```
    curve.write("\t")
  for line in txt_cm:
   for i in list(range(11)):
    if txt_index[i]:
     curve.write("%s" % line[i])
    if i is 10:
     curve.write("\n")
    elif txt_index[i]:
     curve.write("\t")

# Puntos DM
with open('curve_dm.txt', 'w') as curve:
 for i in list(range(11)):
  if txt_index[i]:
   curve.write("%s" % labels[i])
  if i is 10:
   curve.write("\n")
  elif txt_index[i]:
   curve.write("\t")
 for line in txt_dm:
  for i in list(range(11)):
   if txt_index[i]:
    curve.write("%s" % line[i])
   if i is 10:
    curve.write("\n")
   elif txt_index[i]:
    curve.write("\t")

# Opciones de las curvas CM
ax_cm.semilogx()
ax_cm.grid(which='both')
ax_cm.legend(loc='lower left')
ax_cm.set_ylabel('s21 [dB]')
ax_cm.set_xlabel('f [Hz]')
fig_cm.savefig("./imagenes/s21cm.png")
fig_cm.savefig("./imagenes/s21cm.svg")

# Opciones de las curvas DM
ax_dm.semilogx()
ax_dm.grid(which="both")
ax_dm.legend(loc='lower left')
ax_dm.set_ylabel('s21 [dB]')
ax_dm.set_xlabel('f [Hz]')
fig_dm.savefig("./imagenes/s21dm.png")
fig_dm.savefig("./imagenes/s21dm.svg")
plt.close('all')

# Actualizan la gráfica CM
global img_2
global figure_2
figure_2.grid_forget()
img_2 = Image.open("./imagenes/s21cm.png")
img_2 = img_2.resize((400, 236), Image.ANTIALIAS)
img_2 = ImageTk.PhotoImage(img_2)
figure_2 = Label(frame_3, image=img_2)
figure_2.grid(row=0, column=0, columnspan=2)
```

```
361  # Actualiza la gráfica DM
362  global img_3
363  global figure_3
364  figure_3.grid_forget()
365  img_3 = Image.open("./imagenes/s21dm.png")
366  img_3 = img_3.resize((400, 236), Image.ANTIALIAS)
367  img_3 = ImageTk.PhotoImage(img_3)
368  figure_3 = Label(frame_4, image=img_3)
369  figure_3.grid(row=0, column=0, columnspan=2)
370
371
372
373
374
375 def ask_file(mode):
376 # Obtiene el fichero donde con los datos experimentales
377  try:
378   filename = filedialog.askopenfilename(filetypes=[("Text files", "*.txt")])
379  except:
380   filename = filedialog.askopenfilename()
381  if filename is "":
382   return
383  ask = Toplevel()
384  field_freq = Label(ask, text="Frecuencia:")
385  field_mod = Label(ask, text="Módulo:")
386  box_freq = Entry(ask, width=2, borderwidth=4)
387  box_mod = Entry(ask, width=2, borderwidth=4)
388  ok_button = Button(ask, text="Ok", command=lambda: get_index(ask, box_freq,
        box_mod, filename, mode))
389  field_freq.grid(row=0, column=0, padx=10, sticky=W)
390  field_mod.grid(row=1, column=0, padx=10, sticky=W)
391  box_freq.grid(row=0, column=1, padx=10)
392  box_mod.grid(row=1, column=1, padx=10)
393  ok_button.grid(row=2, column=0, columnspan=2, padx=10, pady=10, sticky=E)
394
395
396
397
398
399 def get_index(ask, box_freq, box_mod, filename, mode):
400 # Obtiene el índice de las columnas con datos
401  global index_freq
402  global index_mod
403
404  index_freq = int(box_freq.get())
405  index_mod = int(box_mod.get())
406  ask.destroy()
407  load_file(filename, mode)
408
409
410
411
412
413 def load_file(filename, mode):
414 # Lee los datos experimentales
415  global s21x_cm
416  global s21x_dm
```

```
 global freqx_cm
 global freqx_dm

 x=[]
 y=[]
 fdata = open(filename,'r')
 for linea in fdata:
  line = linea.split()
  if line: # Si no es linea vacia
   try:
    xfloat=float(line[index_freq])
    x.append(xfloat)
    y.append(float(line[index_mod]))
   except:
    pass
 fdata.close()
 if mode is "cm":
  s21x_cm = np.array(y)
  freqx_cm = np.array(x)
 if mode is "dm":
  s21x_dm = np.array(y)
  freqx_dm = np.array(x)




def save_image(mode, name, top, svg_enable, txt_enable, mem_enable):
# Guarda la gráfica en .svg
 new_path = filedialog.askdirectory()
 if new_path is "":
  return
 if mode is "cm":
  if svg_enable.get():
   shutil.copy("./imagenes/s21cm.svg", new_path+"/"+name.get()+".svg")
  if txt_enable.get():
   shutil.copy("./curve_cm.txt", new_path+"/"+name.get()+".txt")
 if mode is "dm":
  if svg_enable.get():
   shutil.copy("./imagenes/s21dm.svg", new_path+"/"+name.get()+".svg")
  if txt_enable.get():
   shutil.copy("./curve_dm.txt", new_path+"/"+name.get()+".txt")
 if mem_enable.get():
  shutil.copy("./memory.txt", new_path+"/"+name.get()+"_memory.txt")
 top.destroy()




def view_figure(mode):
# Amplia la figura y muestra opciones de guardado
 global large_img

 # Abre la imagen en una ventana nueva
 top = Toplevel()
 if mode is "cm":
```

```
474     large_img = ImageTk.PhotoImage(Image.open("./imagenes/s21cm.png"))
475     large_figure = Label(top, image=large_img)
476   if mode is "dm":
477     large_img = ImageTk.PhotoImage(Image.open("./imagenes/s21dm.png"))
478     large_figure = Label(top, image=large_img)
479
480   # Botonera con opciones de guardado
481   svg_enable = IntVar()
482   svg_button = Checkbutton(top, text=".svg", variable=svg_enable)
483   svg_button.select()
484   txt_enable = IntVar()
485   txt_button = Checkbutton(top, text=".txt", variable=txt_enable)
486   txt_button.select()
487   mem_enable = IntVar()
488   mem_button = Checkbutton(top, text="Memoria", variable=mem_enable)
489   name = Entry(top, width=35, borderwidth=4)
490   name.insert(0, "Nombre")
491   save_button = Button(top, text="Guardar", command=lambda: save_image(mode,name
        ,top,svg_enable,txt_enable,mem_enable))
492
493   # Posicionamiento en pantalla
494   large_figure.grid(row=0, column=0, columnspan=5, padx=10, pady=10)
495   svg_button.grid(row=1, column=1, padx=10, sticky=W)
496   txt_button.grid(row=1, column=2, padx=10, sticky=W)
497   mem_button.grid(row=1, column=3, padx=10, sticky=W)
498   name.grid(row=1, column=0, padx=10, pady=10, sticky=W)
499   save_button.grid(row=1, column=4, padx=10, pady=10, sticky=E)
500
501
502
503
504
505 def show_options():
506  # Abre el panel con las opciones
507   global opt
508   global box_memory_1
509   global box_memory_2
510   global box_search_1
511   global box_search_2
512   global box_search_3
513   global box_search_4
514   global box_search_5
515   global box_search_6
516   global plot_index
517   global plot_name_0
518   global plot_name_1
519   global plot_name_2
520   global plot_name_3
521   global plot_name_4
522   global plot_name_5
523   global plot_name_6
524   global plot_name_7
525   global plot_name_8
526   global plot_name_9
527   global plot_name_exp
528   global plot_color_0
529   global plot_color_1
```

```
530   global plot_color_2
531   global plot_color_3
532   global plot_color_4
533   global plot_color_5
534   global plot_color_6
535   global plot_color_7
536   global plot_color_8
537   global plot_color_9
538   global plot_color_exp
539   global plot_width_0
540   global plot_width_1
541   global plot_width_2
542   global plot_width_3
543   global plot_width_4
544   global plot_width_5
545   global plot_width_6
546   global plot_width_7
547   global plot_width_8
548   global plot_width_9
549   global bound_1
550   global bound_2
551   global bound_3
552   global bound_4
553   global bound_5
554   global bound_6
555   global bound_7
556   global bound_8
557   global search_parasite
558   global auto_parasite_delta
559
560   # Abre una nueva ventana
561   opt = Toplevel()
562
563   # Pestaña de opciones de memoria
564   frame_memory = LabelFrame(opt, text="Memoria", padx=20, pady=20)
565   memory_1 = Label(frame_memory, text="Cargar posición")
566   memory_2 = Label(frame_memory, text="Guardar en posición")
567   box_memory_1 = Entry(frame_memory, width=1, borderwidth=4)
568   box_memory_2 = Entry(frame_memory, width=1, borderwidth=4)
569   load_button = Button(frame_memory, text="Ok", command=lambda:load_data(refresh
        =1))
570   save_button = Button(frame_memory, text="Ok", command=save_data)
571   spacer_memory = Label(frame_memory, text="")
572   import_button = Button(frame_memory, text="Importar memoria", command=
        import_memory)
573   reset_button = Button(frame_memory, text="Reset", command=reset_memory)
574
575   # Pestaña de opciones de búsqueda
576   frame_search = LabelFrame(opt, text="Algoritmo de búsqueda", padx=20, pady=20)
577   search_1 = Label(frame_search, text="swarmsize")
578   search_2 = Label(frame_search, text="maxiter")
579   search_3 = Label(frame_search, text="phip")
580   search_4 = Label(frame_search, text="phig")
581   search_5 = Label(frame_search, text="omega")
582   search_6 = Label(frame_search, text="minstep")
583   search_7 = Label(frame_search, text="Espacio de búsqueda")
584   search_8 = Label(frame_search, text="Kycm")
```

```
585  search_9 = Label(frame_search, text="Kydm")
586  search_10 = Label(frame_search, text="Kxdm")
587  search_11 = Label(frame_search, text="Kcy")
588
589  if 'data_opt' in globals():  # Toma valores existentes o los genera
590   default_data = data_opt
591  else:
592   default_data = ['60','30','0.3','0.3','0.3','1e-8']
593
594  box_search_1 = Entry(frame_search, width=8, borderwidth=4)
595  box_search_2 = Entry(frame_search, width=8, borderwidth=4)
596  box_search_3 = Entry(frame_search, width=8, borderwidth=4)
597  box_search_4 = Entry(frame_search, width=8, borderwidth=4)
598  box_search_5 = Entry(frame_search, width=8, borderwidth=4)
599  box_search_6 = Entry(frame_search, width=8, borderwidth=4)
600  box_search_1.insert(0,default_data[0])
601  box_search_2.insert(0,default_data[1])
602  box_search_3.insert(0,default_data[2])
603  box_search_4.insert(0,default_data[3])
604  box_search_5.insert(0,default_data[4])
605  box_search_6.insert(0,default_data[5])
606
607  spacer_search = Label(frame_search, text="")
608  bound_1 = Entry(frame_search, width=8, borderwidth=4)
609  bound_2 = Entry(frame_search, width=8, borderwidth=4)
610  bound_3 = Entry(frame_search, width=8, borderwidth=4)
611  bound_4 = Entry(frame_search, width=8, borderwidth=4)
612  bound_5 = Entry(frame_search, width=8, borderwidth=4)
613  bound_6 = Entry(frame_search, width=8, borderwidth=4)
614  bound_7 = Entry(frame_search, width=8, borderwidth=4)
615  bound_8 = Entry(frame_search, width=8, borderwidth=4)
616
617  bound_1.insert(0, str(slider_1.get()-0.05))    # Toma los valores de pantalla
        con un margen
618  bound_2.insert(0, str(slider_2.get()-0.05))
619  bound_3.insert(0, str(slider_3.get()-0.05))
620  bound_4.insert(0, str(slider_4.get()-0.05))
621  bound_5.insert(0, str(slider_1.get()+0.05))
622  bound_6.insert(0, str(slider_2.get()+0.05))
623  bound_7.insert(0, str(slider_3.get()+0.05))
624  bound_8.insert(0, str(slider_4.get()+0.05))
625
626  # Variación de los efectos parásitos en %
627  if 'delta' in globals():   #Toma el valor existente o lo genera
628   default_delta = str(delta*100)
629  else:
630   default_delta = '20'
631  search_parasite = IntVar()
632  auto_parasite_button = Checkbutton(frame_search, text="Variación de  parásitos
        (%)", variable=search_parasite)
633  auto_parasite_delta = Entry(frame_search, width=4, borderwidth=4)
634  auto_parasite_delta.insert(0, default_delta)
635
636  # Pestaña de opciones de representación
637  frame_plot = LabelFrame(opt, text="Gráficas", padx=20, pady=20)
638  plot_index = []  # Crea variable para seleccionar/deseleccionar gráficas
639  for i in list(range(10)):
```

```
 plot_index.append(IntVar())
plot_1 = Label(frame_plot, text="")
plot_index_0 = Checkbutton(frame_plot, text="0",variable=plot_index[0])
plot_index_1 = Checkbutton(frame_plot, text="1",variable=plot_index[1])
plot_index_2 = Checkbutton(frame_plot, text="2",variable=plot_index[2])
plot_index_3 = Checkbutton(frame_plot, text="3",variable=plot_index[3])
plot_index_4 = Checkbutton(frame_plot, text="4",variable=plot_index[4])
plot_index_5 = Checkbutton(frame_plot, text="5",variable=plot_index[5])
plot_index_6 = Checkbutton(frame_plot, text="6",variable=plot_index[6])
plot_index_7 = Checkbutton(frame_plot, text="7",variable=plot_index[7])
plot_index_8 = Checkbutton(frame_plot, text="8",variable=plot_index[8])
plot_index_9 = Checkbutton(frame_plot, text="9",variable=plot_index[9])
plot_index_0.select()

if 'labels' in globals():  # Toma valores existentes o los genera
 default_name = labels
else:
 default_name = ['Freq','Actual','Mem1','Mem2','Mem3','Mem4','Mem5','Mem6','
   Mem7','Mem8','Mem9','Experimental']

plot_2 = Label(frame_plot, text="Nombre")
plot_name_0 = Entry(frame_plot, width=10, borderwidth=4)
plot_name_1 = Entry(frame_plot, width=10, borderwidth=4)
plot_name_2 = Entry(frame_plot, width=10, borderwidth=4)
plot_name_3 = Entry(frame_plot, width=10, borderwidth=4)
plot_name_4 = Entry(frame_plot, width=10, borderwidth=4)
plot_name_5 = Entry(frame_plot, width=10, borderwidth=4)
plot_name_6 = Entry(frame_plot, width=10, borderwidth=4)
plot_name_7 = Entry(frame_plot, width=10, borderwidth=4)
plot_name_8 = Entry(frame_plot, width=10, borderwidth=4)
plot_name_9 = Entry(frame_plot, width=10, borderwidth=4)
plot_name_exp = Entry(frame_plot, width=10, borderwidth=4)
plot_name_0.insert(0,default_name[1])
plot_name_1.insert(0,default_name[2])
plot_name_2.insert(0,default_name[3])
plot_name_3.insert(0,default_name[4])
plot_name_4.insert(0,default_name[5])
plot_name_5.insert(0,default_name[6])
plot_name_6.insert(0,default_name[7])
plot_name_7.insert(0,default_name[8])
plot_name_8.insert(0,default_name[9])
plot_name_9.insert(0,default_name[10])
plot_name_exp.insert(0,default_name[11])

if 'colors' in globals():  # Toma valores existentes o los genera
 default_color = colors
else:
 default_color = ['#DA7C30','#3E9651','#CC2529','#535154','#6B4C9A','#922428',
   '#948B3D','#FFFF00','#FF00FF','#00FFFF','#396AB1']

plot_3 = Label(frame_plot, text="Color")
plot_color_0 = Entry(frame_plot, width=8, borderwidth=4)
plot_color_1 = Entry(frame_plot, width=8, borderwidth=4)
plot_color_2 = Entry(frame_plot, width=8, borderwidth=4)
plot_color_3 = Entry(frame_plot, width=8, borderwidth=4)
plot_color_4 = Entry(frame_plot, width=8, borderwidth=4)
plot_color_5 = Entry(frame_plot, width=8, borderwidth=4)
```

```
695  plot_color_6 = Entry(frame_plot, width=8, borderwidth=4)
696  plot_color_7 = Entry(frame_plot, width=8, borderwidth=4)
697  plot_color_8 = Entry(frame_plot, width=8, borderwidth=4)
698  plot_color_9 = Entry(frame_plot, width=8, borderwidth=4)
699  plot_color_exp = Entry(frame_plot, width=8, borderwidth=4)
700  plot_color_0.insert(0,default_color[0])
701  plot_color_1.insert(0,default_color[1])
702  plot_color_2.insert(0,default_color[2])
703  plot_color_3.insert(0,default_color[3])
704  plot_color_4.insert(0,default_color[4])
705  plot_color_5.insert(0,default_color[5])
706  plot_color_6.insert(0,default_color[6])
707  plot_color_7.insert(0,default_color[7])
708  plot_color_8.insert(0,default_color[8])
709  plot_color_9.insert(0,default_color[9])
710  plot_color_exp.insert(0,default_color[10])
711
712  if 'widths' in globals():  # Toma valores existentes o los genera
713   default_width = widths
714  else:
715
716   default_width = [3,3,3,3,3,3,3,3,3,3]
717  plot_4 = Label(frame_plot, text="Grosor")
718  plot_width_0 = Entry(frame_plot, width=2, borderwidth=4)
719  plot_width_1 = Entry(frame_plot, width=2, borderwidth=4)
720  plot_width_2 = Entry(frame_plot, width=2, borderwidth=4)
721  plot_width_3 = Entry(frame_plot, width=2, borderwidth=4)
722  plot_width_4 = Entry(frame_plot, width=2, borderwidth=4)
723  plot_width_5 = Entry(frame_plot, width=2, borderwidth=4)
724  plot_width_6 = Entry(frame_plot, width=2, borderwidth=4)
725  plot_width_7 = Entry(frame_plot, width=2, borderwidth=4)
726  plot_width_8 = Entry(frame_plot, width=2, borderwidth=4)
727  plot_width_9 = Entry(frame_plot, width=2, borderwidth=4)
728  plot_width_0.insert(0,default_width[0])
729  plot_width_1.insert(0,default_width[1])
730  plot_width_2.insert(0,default_width[2])
731  plot_width_3.insert(0,default_width[3])
732  plot_width_4.insert(0,default_width[4])
733  plot_width_5.insert(0,default_width[5])
734  plot_width_6.insert(0,default_width[6])
735  plot_width_7.insert(0,default_width[7])
736  plot_width_8.insert(0,default_width[8])
737  plot_width_9.insert(0,default_width[9])
738
739  # Botón aplicar para guardar opciones
740  aply_button = Button(opt, text="Aplicar", command=save_options)
741
742  # Posicionamiento en pantalla
743
744  # Pestaña de opciones de memoria
745  frame_memory.grid(row=0, column=0, padx=20, pady=20)
746  memory_1.grid(row=0, column=0, sticky=W)
747  memory_2.grid(row=1, column=0, sticky=W)
748  box_memory_1.grid(row=0, column=1, sticky=W)
749  box_memory_2.grid(row=1, column=1, sticky=W)
750  load_button.grid(row=0, column=2, sticky=W)
751  save_button.grid(row=1, column=2, sticky=W)
```

```
752   spacer_memory.grid(row=0, column=3, padx=55)
753   import_button.grid(row=0, column=5, sticky=E)
754   reset_button.grid(row=1, column=5, sticky=E)
755
756   # Pestaña de opciones de búsqueda
757   frame_search.grid(row=1, column=0, padx=20, pady=20)
758   search_1.grid(row=0, column=0, padx=10, sticky=W)
759   search_2.grid(row=1, column=0, padx=10, sticky=W)
760   search_3.grid(row=2, column=0, padx=10, sticky=W)
761   search_4.grid(row=3, column=0, padx=10, sticky=W)
762   search_5.grid(row=4, column=0, padx=10, sticky=W)
763   search_6.grid(row=5, column=0, padx=10, sticky=W)
764   search_7.grid(row=0, column=3, columnspan=3, padx=10, sticky=W)
765   search_8.grid(row=1, column=3, padx=10, sticky=W)
766   search_9.grid(row=2, column=3, padx=10, sticky=W)
767   search_10.grid(row=3, column=3, padx=10, sticky=W)
768   search_11.grid(row=4, column=3, padx=10, sticky=W)
769
770   box_search_1.grid(row=0, column=1)
771   box_search_2.grid(row=1, column=1)
772   box_search_3.grid(row=2, column=1)
773   box_search_4.grid(row=3, column=1)
774   box_search_5.grid(row=4, column=1)
775   box_search_6.grid(row=5, column=1)
776
777   spacer_search.grid(row=5, column=2, rowspan=6, padx=20)
778   bound_1.grid(row=1, column=4)
779   bound_2.grid(row=2, column=4)
780   bound_3.grid(row=3, column=4)
781   bound_4.grid(row=4, column=4)
782   bound_5.grid(row=1, column=5)
783   bound_6.grid(row=2, column=5)
784   bound_7.grid(row=3, column=5)
785   bound_8.grid(row=4, column=5)
786
787   auto_parasite_button.grid(row=5, column=3, columnspan=2, sticky=E)
788   auto_parasite_delta.grid(row=5,column=5,sticky=E)
789
790   # Pestaña de opciones de representación
791   frame_plot.grid(row=0, column=1, padx=20, pady=20, rowspan=2)
792   plot_1.grid(row=0, column=0)
793   plot_index_0.grid(row=1, column=0, padx=5)
794   plot_index_1.grid(row=2, column=0)
795   plot_index_2.grid(row=3, column=0)
796   plot_index_3.grid(row=4, column=0)
797   plot_index_4.grid(row=5, column=0)
798   plot_index_5.grid(row=6, column=0)
799   plot_index_6.grid(row=7, column=0)
800   plot_index_7.grid(row=8, column=0)
801   plot_index_8.grid(row=9, column=0)
802   plot_index_9.grid(row=10, column=0)
803
804   plot_2.grid(row=0, column=1, pady=5)
805   plot_name_0.grid(row=1, column=1)
806   plot_name_1.grid(row=2, column=1)
807   plot_name_2.grid(row=3, column=1)
808   plot_name_3.grid(row=4, column=1)
```

```
809  plot_name_4.grid(row=5, column=1)
810  plot_name_5.grid(row=6, column=1)
811  plot_name_6.grid(row=7, column=1)
812  plot_name_7.grid(row=8, column=1)
813  plot_name_8.grid(row=9, column=1)
814  plot_name_9.grid(row=10, column=1)
815  plot_name_exp.grid(row=11, column=1)
816
817  plot_3.grid(row=0, column=2)
818  plot_color_0.grid(row=1, column=2)
819  plot_color_1.grid(row=2, column=2)
820  plot_color_2.grid(row=3, column=2)
821  plot_color_3.grid(row=4, column=2)
822  plot_color_4.grid(row=5, column=2)
823  plot_color_5.grid(row=6, column=2)
824  plot_color_6.grid(row=7, column=2)
825  plot_color_7.grid(row=8, column=2)
826  plot_color_8.grid(row=9, column=2)
827  plot_color_9.grid(row=10, column=2)
828  plot_color_exp.grid(row=11, column=2)
829
830  plot_4.grid(row=0, column=3)
831  plot_width_0.grid(row=1, column=3)
832  plot_width_1.grid(row=2, column=3)
833  plot_width_2.grid(row=3, column=3)
834  plot_width_3.grid(row=4, column=3)
835  plot_width_4.grid(row=5, column=3)
836  plot_width_5.grid(row=6, column=3)
837  plot_width_6.grid(row=7, column=3)
838  plot_width_7.grid(row=8, column=3)
839  plot_width_8.grid(row=9, column=3)
840  plot_width_9.grid(row=10, column=3)
841
842  # Botón aplicar
843  aply_button.grid(row=2, column=1, padx=20, pady=20, sticky=E)
844
845
846
847
848
849 def save_data():
850 # Guarda el valor de data en una posición de la memoria
851  global data
852
853  data  = [box_1.get(), box_2.get(), box_3.get(), box_4.get(), box_5.get(),
        box_6.get(), box_7.get(), box_8.get(), box_9.get(), box_10.get(), box_11.
        get(), box_12.get(), box_13.get(), box_14.get(), str(slider_1.get()), str(
        slider_2.get()), str(slider_3.get()), str(slider_4.get())]
854
855  memory = open('memory.txt', 'r') # Primero lee la memoria
856  if 'autoslot' in globals():
857   slot = autoslot
858  else:
859   slot = int(box_memory_2.get())
860  new_memory = []
861  count = 0
862  for line in memory:
```

```
    element = line.split('\t')
    element[slot] = str(data[count])
    new_memory.append(element)    # Genera una nueva lista de parámetros
    count = count + 1
  memory.close()

  with open('memory.txt', 'w') as mem:
    for line in new_memory:
      for i in list(range(10)):
        mem.write("%s\t" % line[i])   # Sobreescribe la nueva memoria
      mem.write("\n")




def load_data(refresh=1):
# Lee el valor de data de una posición de la memoria
  global data

  data = []
  memory = open('memory.txt', 'r')
  if 'autoslot' in globals():
    slot = autoslot
  else:
    slot = int(box_memory_1.get())
  for line in memory:
    elements = line.split('\t')
    data.append(elements[slot])

  if refresh:        # Para cargar datos en pantala refresh = 1
    box_1.delete(0, END)
    box_2.delete(0, END)
    box_3.delete(0, END)
    box_4.delete(0, END)
    box_5.delete(0, END)
    box_6.delete(0, END)
    box_7.delete(0, END)
    box_8.delete(0, END)
    box_9.delete(0, END)
    box_10.delete(0, END)
    box_11.delete(0, END)
    box_12.delete(0, END)
    box_13.delete(0, END)
    box_14.delete(0, END)

    box_1.insert(0, data[0])
    box_2.insert(0, data[1])
    box_3.insert(0, data[2])
    box_4.insert(0, data[3])
    box_5.insert(0, data[4])
    box_6.insert(0, data[5])
    box_7.insert(0, data[6])
    box_8.insert(0, data[7])
    box_9.insert(0, data[8])
    box_10.insert(0, data[9])
    box_11.insert(0, data[10])
```

```
920    box_12.insert(0, data[11])
921    box_13.insert(0, data[12])
922    box_14.insert(0, data[13])
923
924    slider_1.set(data[14])
925    slider_2.set(data[15])
926    slider_3.set(data[16])
927    slider_4.set(data[17])
928
929
930
931
932
933  def import_memory():
934  # Cambia la memoria por el txt elegido (una memoria exportada previamente)
935
936   try:
937    filename = filedialog.askopenfilename(filetypes=[("Text files", "*.txt")])
938   except:
939    filename = filedialog.askopenfilename()
940   if filename is "":
941    return
942   shutil.copy(filename, "./memory.txt")
943
944
945
946
947
948  def reset_memory():
949  # Sobreescribe la memoria con zeros en todas sus posiciones
950
951   with open('memory.txt', 'w') as mem:
952    for i in list(range(18)):
953     for j in list(range(10)):
954      mem.write("0\t")    # Sobreescribe la nueva memoria
955     mem.write("\n")
956
957
958
959
960
961  def save_options():
962  # Guarda las opciones y cierra el panel
963   global data_opt
964   global labels
965   global colors
966   global widths
967   global upper_bound
968   global lower_bound
969   global delta
970
971   data_opt = [box_search_1.get(), box_search_2.get(), box_search_3.get(),
        box_search_4.get(), box_search_5.get(), box_search_6.get()]
972
973   labels = ['Freq', plot_name_0.get(), plot_name_1.get(), plot_name_2.get(),
        plot_name_3.get(), plot_name_4.get(), plot_name_5.get(), plot_name_6.get(),
```

```
     plot_name_7.get(), plot_name_8.get(), plot_name_9.get(), plot_name_exp.get
    ()]

 colors = [plot_color_0.get(), plot_color_1.get(), plot_color_2.get(),
    plot_color_3.get(), plot_color_4.get(), plot_color_5.get(), plot_color_6.
    get(), plot_color_7.get(), plot_color_8.get(), plot_color_9.get(),
    plot_color_exp.get()]

 widths = [plot_width_0.get(), plot_width_1.get(), plot_width_2.get(),
    plot_width_3.get(), plot_width_4.get(), plot_width_5.get(), plot_width_6.
    get(), plot_width_7.get(), plot_width_8.get(), plot_width_9.get()]

 lower_bound = [float(bound_1.get()), float(bound_2.get()), float(bound_3.get()
    ), float(bound_4.get())]
 upper_bound = [float(bound_5.get()), float(bound_6.get()), float(bound_7.get()
    ), float(bound_8.get())]

 delta = float(auto_parasite_delta.get())/100

 opt.destroy()




# Inicio de la ventana principal
    ##########################################################

root = Tk()
root.title("Analizador de filtros")

# Botones de ayuda y resolución
help_button = Button(root,text="Ayuda", command=show_help)
option_button = Button(root, text="Opciones", command=show_options)
solve_button = Button(root, text="Calcula", width=30, command=solve_main)


## Frame 1: Modelo
frame_1 = LabelFrame(root, text="Modelo", padx=20, pady=20)

# Imagen del crcuito
img_1 = Image.open("./imagenes/model.png")
img_1 = img_1.resize((308, 264), Image.ANTIALIAS)
img_1 = ImageTk.PhotoImage(img_1)
figure_1 = Label(frame_1, image=img_1)

# Parametros del modelo (conocidos a priori)
field_1  = Label(frame_1, text="Cy:")
field_2  = Label(frame_1, text="Ly:")
field_3  = Label(frame_1, text="Ry:")
field_4  = Label(frame_1, text="Cx:")
field_5  = Label(frame_1, text="Lx:")
field_6  = Label(frame_1, text="Rx:")
field_7  = Label(frame_1, text="Rs:")
field_8  = Label(frame_1, text="Lcm:")
field_9  = Label(frame_1, text="Ldm:")
field_10 = Label(frame_1, text="Cw:")
```

```
1021 field_11 = Label(frame_1, text="Ct:")
1022 field_12 = Label(frame_1, text="RN1:")
1023 field_13 = Label(frame_1, text="RN2:")
1024 field_14 = Label(frame_1, text="Rl:")
1025
1026 # Entradas de los parámetros
1027 box_1  = Entry(frame_1, width=8, borderwidth=4)
1028 box_2  = Entry(frame_1, width=8, borderwidth=4)
1029 box_3  = Entry(frame_1, width=8, borderwidth=4)
1030 box_4  = Entry(frame_1, width=8, borderwidth=4)
1031 box_5  = Entry(frame_1, width=8, borderwidth=4)
1032 box_6  = Entry(frame_1, width=8, borderwidth=4)
1033 box_7  = Entry(frame_1, width=8, borderwidth=4)
1034 box_8  = Entry(frame_1, width=8, borderwidth=4)
1035 box_9  = Entry(frame_1, width=8, borderwidth=4)
1036 box_10 = Entry(frame_1, width=8, borderwidth=4)
1037 box_11 = Entry(frame_1, width=8, borderwidth=4)
1038 box_12 = Entry(frame_1, width=8, borderwidth=4)
1039 box_13 = Entry(frame_1, width=8, borderwidth=4)
1040 box_14 = Entry(frame_1, width=8, borderwidth=4)
1041
1042
1043 ## Frame 2: Acoplamientos
1044 frame_2 = LabelFrame(root, text="Acoplos", padx=20, pady=20)
1045
1046 # Botón auto: para calcular o no automáticamente los valores
1047 search_cm = IntVar()
1048 auto_label = Label(frame_2, text="Calcular automáticamente:")
1049 auto_cm_button = Checkbutton(frame_2, text="Valores de CM", variable=search_cm)
1050 search_dm = IntVar()
1051 auto_dm_button = Checkbutton(frame_2, text="Valores de DM", variable=search_dm)
1052
1053 # Acoplamientos
1054 field_15 = Label(frame_2, text="Kycm:")
1055 field_16 = Label(frame_2, text="Kydm:")
1056 field_17 = Label(frame_2, text="Kxdm:")
1057 field_18 = Label(frame_2, text="Kxy:")
1058
1059 # Valores de los acoplamientos
1060 slider_1 = Scale(frame_2, from_=-1, to=1, resolution=0.0001, orient=HORIZONTAL,
         length=506)
1061 slider_2 = Scale(frame_2, from_=-1, to=1, resolution=0.0001, orient=HORIZONTAL,
         length=506)
1062 slider_3 = Scale(frame_2, from_=-1, to=1, resolution=0.0001, orient=HORIZONTAL,
         length=506)
1063 slider_4 = Scale(frame_2, from_=-1, to=1, resolution=0.0001, orient=HORIZONTAL,
         length=506)
1064
1065
1066
1067 ## Frame 3: Gáfica de la respuesta en modo común
1068 frame_3 = LabelFrame(root, text="S21 modo común",padx=20, pady=20)
1069 img_2 = Image.open("./imagenes/spacer.png")
1070 img_2 = img_2.resize((400, 236), Image.ANTIALIAS)
1071 img_2 = ImageTk.PhotoImage(img_2)
1072 figure_2 = Label(frame_3, image=img_2)
```

```
file_button_1 = Button(frame_3, text="Insertar datos", command=lambda: ask_file("cm"))
view_button_1 = Button(frame_3, text="Ver", command=lambda: view_figure("cm"))



## Frame 4: Gráfica de la respuesta en modo diferencial
frame_4 = LabelFrame(root, text="S21 modo diferencial",padx=20, pady=20)
img_3 = Image.open("./imagenes/spacer.png")
img_3 = img_3.resize((400, 236), Image.ANTIALIAS)
img_3 = ImageTk.PhotoImage(img_3)
figure_3 = Label(frame_4, image=img_3)
file_button_2 = Button(frame_4, text="Insertar datos", command=lambda: ask_file("dm"))
view_button_2 = Button(frame_4, text="Ver", command=lambda: view_figure("dm"))







# Posicionamiento en la ventana
    ############################################################3

# Botones
help_button.grid(row=2, column=0, padx=10, pady=10, sticky=W)
option_button.grid(row=2, column=1, padx=10, pady=10,sticky=W)
solve_button.grid(row=2, column=2, padx=10, pady=10, sticky=E)

# Frame 1
frame_1.grid(row=0, column=0, padx=10, pady=10, columnspan=3, sticky=N)

figure_1.grid(row=0, column=0, rowspan=8, columnspan=2)

field_1.grid(row=0, column=2)
field_2.grid(row=1, column=2)
field_3.grid(row=2, column=2)
field_4.grid(row=3, column=2)
field_5.grid(row=4, column=2)
field_6.grid(row=5, column=2)
field_7.grid(row=6, column=2)
field_8.grid(row=0, column=4)
field_9.grid(row=1, column=4)
field_10.grid(row=2, column=4)
field_11.grid(row=3, column=4)
field_12.grid(row=4, column=4)
field_13.grid(row=5, column=4)
field_14.grid(row=6, column=4)

box_1.grid(row=0, column=3, pady=4)
box_2.grid(row=1, column=3)
box_3.grid(row=2, column=3)
box_4.grid(row=3, column=3)
box_5.grid(row=4, column=3)
box_6.grid(row=5, column=3)
box_7.grid(row=6, column=3)
box_8.grid(row=0, column=5)
box_9.grid(row=1, column=5)
```

```
1127 box_10.grid(row=2, column=5)
1128 box_11.grid(row=3, column=5)
1129 box_12.grid(row=4, column=5)
1130 box_13.grid(row=5, column=5)
1131 box_14.grid(row=6, column=5)
1132
1133 # Frame 2
1134 frame_2.grid(row=1, column=0, padx=10, pady=10, columnspan=3, sticky=N)
1135
1136 auto_label.grid(row=0, column=0, columnspan=2, pady=10, sticky=W)
1137 auto_cm_button.grid(row=0, column=2, sticky=E)
1138 auto_dm_button.grid(row=0, column=3, sticky=E)
1139
1140 field_15.grid(row=1, column=0, pady=3)
1141 field_16.grid(row=2, column=0, pady=3)
1142 field_17.grid(row=3, column=0, pady=3)
1143 field_18.grid(row=4, column=0, pady=3)
1144
1145 slider_1.grid(row=1, column=1, columnspan=3, pady=3)
1146 slider_2.grid(row=2, column=1, columnspan=3, pady=3)
1147 slider_3.grid(row=3, column=1, columnspan=3, pady=3)
1148 slider_4.grid(row=4, column=1, columnspan=3, pady=3)
1149
1150 # Frame 3
1151 frame_3.grid(row=0, column=3, padx=10, pady=10, columnspan=2)
1152 figure_2.grid(row=0, column=0, columnspan=2)
1153 file_button_1.grid(row=1, column=0, sticky=W)
1154 view_button_1.grid(row=1, column=1, sticky=E)
1155
1156 # frame 4
1157 frame_4.grid(row=1, column=3, padx=10, pady=10, columnspan=2, rowspan=2)
1158 figure_3.grid(row=0, column=0, columnspan=2)
1159 file_button_2.grid(row=1, column=0, sticky=W)
1160 view_button_2.grid(row=1, column=1, sticky=E)
1161
1162 root.mainloop()   # Mantiene la ventana abierta
```