Extracción de parámetros de un resonador FBAR por optimización

Memoria del Trabajo Final de Carrera
de Ingeniería Técnica de Telecomunicaciones,
especialidad Sistemas Electrónicos
realizado por
Jordi Marin Garcia
y dirigido por
Pedro de Paco Sánchez
Bellaterra, 15 de Junio de 2007

Escola Tècnica Superior d'Enginyeria

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Paradigma de la tecnología actual

En los últimos años hemos podido ver una expansión extraordinaria de las tecnologías inalámbricas. Desde la entrada de los teléfonos móviles a la vida cuotidiana, un constante goteo de nuevas tecnologías inalámbricas se han abierto paso hacia el mercado actual. El abanico de servicios que ofrecen estas tecnologías no solo se limita a las comunicaciones personales, sino que han surgido toda una serie de nuevas aplicaciones que tienen como fin simplificar la vida de los usuarios. De esta manera, tecnologías tales como *Bluetooth* o *ZigBee*, crean redes inalámbricas personales que mantienen comunicados todos los dispositivos electrónicos que un mismo usuario posee, permitiendo por ejemplo, intercambiar datos de un ordenador a un móvil, o de una PDA[1] a una impresora. Otras tecnologías también se han expandido rápidamente como las redes de ordenadores inalámbricas o *WLANs*[2], así como sistemas de posicionamiento global tales como el americano *GPS*[3], o como el europeo Galileo, aún en fase de desarrollo.

Todo este conjunto de nuevas tecnologías, así como muchas otras aplicaciones científicas y militares, operan en rangos de frecuencia similares que comprenden desde los 100MHz a los 10GHz, rangos de radiofrecuencia (RF) y microondas. Para evitar interferencias entre todo el conjunto de tecnologías, se necesitan dispositivos con grandes prestaciones y que además sean cada vez más pequeños con el propósito generalizado de miniaturizar los diferentes aparatos, así como faciles de implementar en un proceso de fabracación a gran escala.

[1] Asistente personal de datos
[2] Red de área local inalámbrica.
[3] Global Positioning System

1.2. Estado del arte

Actualmente, los dispositivos utilizados para implementar sistemas inalámbricos, usan mayoritariamente tecnología SAW (*Surface Acoustic Wave*). Esto se debe a que esta tecnología es fácil de fabricar y su implementación tan solo requiere de una pequeña cantidad de pasos. Se consiguen de esta manera unos dispositivos relativamente pequeños y baratos.

Sin embargo la tecnología SAW tiene grandes limitaciones que empeoran sus prestaciones. A partir de 2 – 2,5 GHz, la fabricación de dichos dispositivos empieza a ser crítica debido a la resolución que necesitan las mascaras con que se llevan a cabo los procesos litográficos. Además de los problemas de fabricación, también tiene problemas de potencia debido a electromigración y sobrecalentamiento, así como una gran sensibilidad a descargas electrostáticas (ESD). Otro gran problema que presenta la tecnología SAW es la incompatibilidad con los substratos de silicio, que limita tanto la miniaturización como su coste [1].

El hecho de no contar con una tecnología fiable para rangos de microondas, ha provocado un fuerte incremento en la investigación y desarrollo de otras tecnologías. En concreto la tecnología FBAR (*Film Bulk Acoustic Resonator*). Dicha tecnología nace como una extensión directa de los resonadores de cristal de cuarzo, ampliamente utilizados desde hace más de seis décadas.

Los cristales de cuarzo, así como otros piezoeléctricos, son de un gran interés electromecánico puesto nos proporcionan dispositivos con unas resonancias mecánicas de gran calidad, con factores Q por encima de los 10000 [2].

Sin embargo, al igual que pasaba con los dispositivos SAW, es viable fabricar resonadores de cristal de cuarzo solo hasta ciertas frecuencias. En este caso la máxima frecuencia es de 250 MHz. Esta limitación se debe a que al incrementar la frecuencia, la capa de material piezoeléctrico que forma el resonador debe ser cada vez más delgada, y en consecuencia para dichas frecuencias es inviable la fabricación a gran escala de estos dispositivos [1].

Durante cuarenta años, se han llevado a cabo numerosas investigaciones con el propósito de conseguir reducir más el grosor de las capas piezoeléctricas de los resonadores. Pronto se vio que hacer crecer las capas piezoeléctricas encima del substrato, podía ser una mejor solución que rebajar cristales hasta las medidas deseadas. A partir de este hecho, toda una serie de compuestos piezoeléctricos fueron desarrollados para poder ser depositados encima de substratos tales como el silicio.

Una de los motivos por el cual los resonadores FBAR han tenido un gran éxito, ha sido la mejora de las técnicas de procesado microelectrónico llevadas a cabo en el silicio, que han hecho posible la fabricación de resonadores a gran escala [2].

1.3. Resonadores FBAR en la actualidad

Actualmente, nos encontramos en el momento de pleno apogeo del desarrollo de dispositivos FBAR. La gran variedad de compuestos tales como el nitruro de aluminio (AlN) implementable en silicio, ha proporcionado el marco ideal para su utilización en multitud de aplicaciones con producción a gran escala. Podemos econtrar resonadores FBAR en las cabeceras de radiofrecuencia de aplicaciones tales como : telefonía móvil [3], redes inalámbricas WLANs [4], dispostivos GPS [5]...

Se han desarrollado resonadores que comprenden desde los 600MHz a los 12 GHz. En términos del factor de calidad, los primeros resonadores llevados a cabo, tenían tan solo Qs de 1000, mientras los últimos resonadores han logrado incluso factores de Q de 67000 [6].

1.4. Objetivos del trabajo

Este trabajo tiene como principal objetivo el desarrollo de un algoritmo de extracción de parámetros mediante una optimización, que permita obtener los valores del modelo teórico de un resonador a través de la medida de dispositivos fabricados en el CNM[4]. Para ello durante el trabajo se trataran los siguientes aspectos:

- Estudio del resonador FBAR, estructura, funcionamiento y modelado.
- Estudio de las diferentes técnicas de optimización.
- Elaboración de un algoritmo para la extracción de parámetros mediante optimización por ajuste de datos.

Basando nuestro trabajo en la optimación de parámetros de otros dispositivos de radiofrecuencia mediante la medida de los parámetros S de dispositivos tales como transistores FET [7] o filtros [8]. En la muchos de estos casos se ha adoptado una estrategia de optimización mediante mínimos cuadrados para la extracción de los parámetros del circuito equivalente.

1.5. Estructura de la memoria

La memoria de este trabajo, se divide en cuatro capítulos, a lo largo de los cuales se estudiaran los conceptos teóricos necesarios para la realización del trabajo y se propondrá un algoritmo objetivo principal de este trabajo. La estructura por capítulos es la siguiente:

[4]Centre Nacional de Microelectrónica.

- **Capítulo 1 :** En este capítulo, donde nos encontramos, se lleva a cabo la introducción del trabajo. Se expone el estado del arte de los resonadores FBAR así como una explicación del objetivo del trabajo y de la estructura de la memoria.
- **Capítulo 2 :** En el segundo capítulo se introduce la teoría de los resonadores FBAR: su estructura, su funcionamiento y finalmente su modelado, que nos servirá para desarrollar el algoritmo de optimización que tiene por objetivo este trabajo.
- **Capítulo 3 :** En el tercer capítulo, se desarrolla la teoría necesaria para comprender el funcionamiento de la optimización llevada a cabo en el trabajo. Para ello se hace una introducción al mundo de la programación matemática lineal y no lineal, y se presenta matemáticamente el método escogido en el trabajo.
- **Capítulo 4 :** En este capítulo se explica la estrategia utilizada para llevar a cabo el algoritmo, así como los resultados obtenidos mediante este.
- **Capítulo 5 :** El último capítulo tiene como fin presentar las conclusiones de este trabajo, así como las líneas futuras de trabajo.

Capítulo 2

Estructura y modelado del resonador FBAR

2.1. Introducción

En este capítulo se presenta la estructura, el funcionamiento y el modelado de un resonador FBAR. El objetivo es llegar a comprender la base teórica de estos dispositivos ya que forman el eje principal de este trabajo.

En el primer apartado de este capítulo se pretende dar una visón global de la estructura y funcionamiento de los resonadores FBAR. Además se describirán las dos estructuras más utilizadas en el desarrollo de resonadores: el resonador de membrana y el resonador con espejo de impedancias.

En el apartado de modelado, se introducirán los modelos teóricos más utilizados en el diseño, análisis y simulación de los resonadores, que posteriormente utilizaremos para el desarrollo del algoritmo que tiene como objetivo este trabajo.

2.2. Estructura y funcionamiento de un resonador FBAR

La estructura básica de un resonador FBAR consta de una lámina de material piezoeléctrico situada entre dos electrodos. Esta geometría es la misma que la de un condensador de placas paralelo con un material piezoeléctrico haciendo de dieléctrico.

La aplicación de un campo eléctrico en los electrodos, induce una deformación mecánica en la lamina debido al efecto piezoeléctrico inverso. Si en vez de aplicar un campo eléctrico constante, aplicamos una señal harmónica con una frecuencia cercana a la de resonancia del resonador, induciremos una onda acústica que viajará de forma perpendicular a los electrodos. En una primera aproximación, la frecuencia de resonancia, viene determinada por el grosor del piezoeléctrico,

que coincidirá con media longitud de la onda acústica propagada. Sin embargo se ha demostrado que los efectos de carga que producen los electrodos, no son despreciables en el cálculo de la frecuencia y por tanto deberán ser tomados en consideración en el diseño de los resonadores [9].

El resonador FBAR descansa tipicamente sobre un substrato de silicio. Para que el resonador este acústicamente aislado y podamos confinar la onda acústica entre los electrodos, se han ideado diferentes sistemas que llevan a cabo el aislamiento.

Una vez que tenemos la onda acústica confinada en nuestro resonador, del mismo modo que sucede con un resonador basado en cristal de cuarzo, una variación de la señal de entrada, tan solo produce una pequeña variación en la frecuencia del resonador; puesto la fase de la impedancia de nuestro resonador cercana a la frecuencia de resonancia tiene un comportamiento muy abrupto. De hecho se suele utilizar la derivada de la fase de la impedancia como evaluación de la calidad de los resonadores [10]:

$$Q = \frac{f}{2} \left| \frac{\partial \phi z}{\partial f} \right|_{\text{máx}} \tag{2.1}$$

El factor de calidad Q de un resonador FBAR puede ser muy elevado y suele tomar un valor entre 1000 y 67000 según su fabricación [6].

2.2.1. Concepto de Piezoelectricidad

Para comprender la piezoelectricidad, primero deberemos introducir el concepto de polarización. Cuando se aplica un campo eléctrico a un material dieléctrico, la carga eléctrica de las partículas hace que estas se distribuyan microscópicamente resultando en una polarización macroscópica del material.

Se ha demostrado, que para ciertos materiales, la polarización también ocurre como consecuencia de deformaciones o cargas. Un material es piezoeléctrico dependiendo de la distribución de sus cargas. En la figura 2.1 (a) podemos ver el esquema de la distribución de cargas de un material piezoeléctrico, y como en el apartado (b) se ha deformado la estructura generando una polarización.

Al aplicar una deformación a un material piezoeléctrico, la polarización de dicho material genera un campo eléctrico, a esto se le denomina *efecto piezoeléctrico*. De forma contraria, la aplicación de un campo eléctrico sobre un piezoeléctrico, genera una deformación, que da origen al *efecto inverso piezoeléctrico*[11].

A la hora de trabajar con los resonadores FBAR, es importante tener en cuenta los siguientes parámetros que caracterizan el comportamiento de los materiales piezoeléctricos [1]:

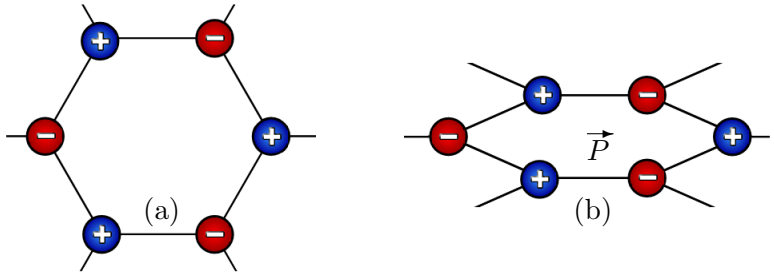

Figura 2.1: *Origen del efecto directo piezoeléctrico: (a) Material sin aplicar deformación. (b) Material con deformación aplicada.*

- **Coeficiente de acoplo electro-acústico:** Este parámetros es uno de los más importantes y utilizados. Nos informa de la cantidad de energía que se transforma del dominio mecánico al eléctrico y viceversa. A la hora de implementar resonadores, nos interesa que este parámetro sea lo mayor posible. Este parámetro es representado matemáticamente mediante el símbolo k_t^2.

- **Coeficiente de temperatura:** Dicho parámetro nos indica las variaciones de frecuencia introducida por las derivas resultantes de cambios de temperatura en nuestro material piezoeléctrico.

- **Constante dieléctrica:** La impedancia del resonador es determinada por su tamaño, el grosor del piezoeléctrico y la constante dieléctrica. Una constante dieléctrica grande permite hacer pequeño el resonador.

- **Pérdidas intrínsecas del material:** Este parámetro expresa las perdidas del material piezoeléctrico, debido a las variaciones internas del propio material, así como a las perdidas producidas por las vibraciones en forma de calor.

- **Velocidad de la onda acústica:** Simbolizado por v_p. Cada material tiene una velocidad diferente de propagación para la onda acústica. Cuanto menor sea el valor, mayor podra ser la miniaturización del dispositivo. Sin embargo debemos tener en cuenta que no se pueden conseguir todos los grosores deseados, por lo que este parámetro deberá ser convenientemente escogido para que el resonador diseñado pueda ser fabricado de forma industrial a gran escala.

Debido a las limitaciones que imponía el uso de materiales piezoeléctricos puros con bajos coeficientes de acoplo electro-acústico, se introdujeron en el diseño óxidos refractantes como el $LiNbO_3$ o el $LITaO_3$. En el panorama actual, encontramos una gran variedad de materiales piezoeléctricos: SiC, ZnO, AlN, $KNbO_3$, PZT, PVDF [14].

De todo el abanico actual de materiales disponibles, los más utilizados son el nitruro de aluminio (AlN), el óxido de zinc (ZnO) y el titanato de circonio de plomo (PZT). En la tabla 2.1 se puede ver una comparativa de dichos materiales con respecto a los coeficientes antes mencionados.

	Nitruro de Aluminio (AlN)	Óxido de Zinc (ZnO)	Titanato de circonio de plomo (PZT)
Coeficiente de acoplamiento k_t^2 (%)	6,5	7,5	$8-15$
Constante dieléctrica ε_r	9,5	9,2	$80-400$
Velocidad de onda acústica v_p	10400	6350	$4000-6000$
Perdidas intrínsecas	Muy pocas	Pocas	Grandes y aumentan con la frecuencia

Tabla 2.1: *Comparación de los materiales piezoeléctricos más utilizados*[1]

Cabe destacar que tan solo el nitruro de aluminio es actualmente compatible con la tecnología CMOS[1] [1].

2.2.2. Resonador FBAR tipo membrana

La forma más directa de implementar una extensión del principio de funcionamiento del cristal de cuarzo es construir la lámina piezoeléctrica y los electrodos encima de una membrana delgada, con aproximadamente 1μm de espesor. Dicha membrana puede ser construida mediante procesos de microfabricación que vacíen la parte inferior de la membrana o mediante una capa sacrificable, que se eliminara con procesos químicos después de haber creado la membrana. Esta construcción la podemos ver en la figura 2.2. Usualmente también se denomina a este tipo de resonador de *abertura de aire* o *air gap*.

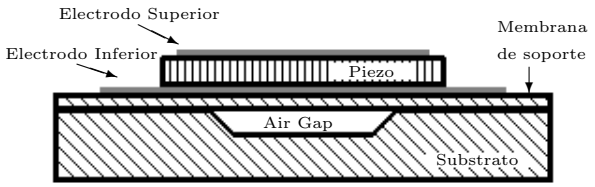

Figura 2.2: *Resonador FBAR tipo membrana.*

[1]Complementary Metal Oxide Semiconductor

Aplicando procesado microelectrónico a la estructura de membrana, se ha logrado eliminar la necesidad de construir nuestro resonador encima de la membrana y se ha conseguido una estructura más acorde con la realidad de los resonadores. Los resonadores utilizados actualmente solo estan formados por la lámina del piezoeléctrico, los dos electrodos y aire en ambas superficies, tal y como podemos ver en la figura 2.3.

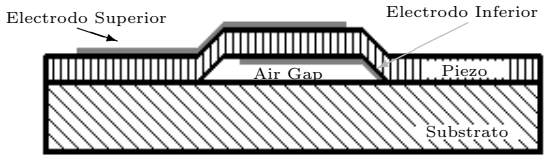

Figura 2.3: *Resonador FBAR con reflexión de onda mediante Air Gap.*

2.2.3. Resonador FBAR mediante espejo acústico

Una alternativa de diseño a la realización del resonador FBAR, tiene como objetivo aislar la onda acústica del substrato mediante una serie de capas de grosor $\lambda/4$, que alternan alta con baja impedancia. Este dispositivo es llamado *espejo de impedancias* o *Reflector de Bragg*. De la misma manera que sucede con la luz al cambiar de medios con índices de reflexión diferentes, podemos jugar con las impedancias de las diferentes capas del espejo para lograr que la onda acústica rebote casi por completo cada vez que cambia de medio. Podemos ver esta estructura en la figura 2.4.

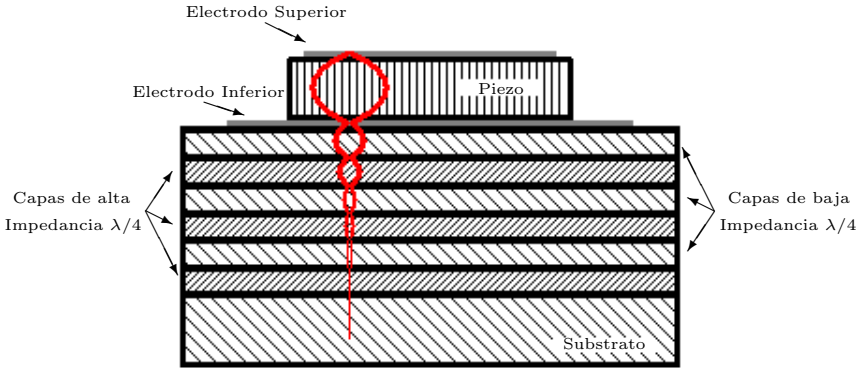

Figura 2.4: *Resonador FBAR con reflexión de onda mediante espejo acústico.*

2.3. Modelado del resonador FBAR

El modelado de resonadores FBAR se puede llevar a cabo a diferentes niveles de abstracción. El modelo más físico requiere de una simulación tridimensional del acoplamiento electro-acústico del resonador, que es prácticamente imposible de

formular y resolver mediante métodos analíticos. Pudiéndose resolver de manera numérica aunque dicho nivel de abstracción, sigue siendo poco práctico debido al elevado tiempo de computo que llevaría procesar incluso una simulación bidimensional del resonador.

El nivel físico con el que podemos trabajar de forma relativamente ágil y eficiente, utiliza ecuaciones acústicas y piezoeléctricas de una sola dimensión para describir la impedancia del resonador así como la carga y el estrés entre capas. El modelo unidimensional del resonador FBAR es llamado modelo de Mason. Mientras este tipo de modelado es más importante para etapas de desarrollo de los propios resonadores, resulta demasiado complejo para su utilización en el diseño de otros dispositivos que contengan resonadores como elemento básico. Por ello utilizamos un modelo más compacto, utilizando un circuito equivalente eléctrico conocido como el modelo de Butterworth Van-Dyke (BVD). Si queremos trabajar con un modelo más real que el de BVD, le añadiremos al modelo una serie de elementos que nos permitirán simular con mayor precisión el comportamiento real de un resonador FBAR. Este último modelo y el utilizado en este trabajo para hacer la extracción de parámetros es el llamado modelo de Butterworth Van-Dyke Modificado (MBVD).

2.3.1. Modelo de Mason

En un resonador FBAR, el análisis de la propagación de la onda acústica, en una sola dimensión, lleva a la obtención de un modelo eléctrico equivalente. El modelo de Mason describe el comportamiento unidimensional ideal del resonador, usando ecuaciones de acoplo electro-acústico y modela cada capa del resonador mediante secciones de elementos distribuidos o líneas de transmisión, imponiendo condiciones de contorno en sus extremos.

Para el modelado entero de las capas que constituyen un resonador, tanto capas de material piezoeléctrico como las no-piezoeléctrico, podemos encontrar el modelo de Mason concatenando las celdas de los circuitos equivalentes de cada capa.

2.3.1.1. Ecuación de onda acústica

Cuando aplicamos una fuerza a un cuerpo, además de los efectos descritos por la física newtoniana, obtenemos en el interior de dicho cuerpo, fuerzas internas, que denominamos tensiones mecánicas (*stresses*), y deformaciones (*strains*).

Podemos definir la tensión mecánica como el cociente entre la fuerza y el área de la superficie del cuerpo sobre la que se aplica.

$$T = \frac{dF}{dA} \tag{2.2}$$

Las deformaciones, se definen como el gradiente del desplazamiento de la partícula respecto a la posición.

$$S = \frac{\partial u}{\partial z} \tag{2.3}$$

La velocidad a la que se desplaza una partícula es la derivada del desplazamiento respecto el tiempo.

$$v = \frac{du}{dt} \tag{2.4}$$

Haciendo una analogía con la ley de *Hooke*, consideramos una relación lineal entre la tensión y la deformación mediante una constante que llamaremos constante de *stiffnes* (C).

$$T = CS \tag{2.5}$$

A partir de la tercera ley de Newton podemos establecer la siguiente relación entre la tensión mecánica y el desplazamiento de las partículas internas del cuerpo.

$$F = ma \rightarrow \frac{\partial T}{\partial z} dz dA = \rho A dz \frac{\partial^2 u}{\partial^2 t} \Rightarrow \frac{\partial T}{\partial z} = \rho \frac{\partial^2 u}{\partial^2 t} \tag{2.6}$$

Donde ρ es la densidad del cuerpo con unidades kg/m^2.

Si derivamos la tensión mecánica, ecuación (2.5), respecto el tiempo y substituyendo (2.4) obtenemos la ecuación de onda para el desplazamiento.

$$\frac{\partial}{\partial z} T = C \frac{\partial^2 u}{\partial^2 z} = \rho \frac{\partial^2 u}{\partial^2 t} \Rightarrow \frac{\partial^2 u}{\partial^2 z} = \frac{\rho}{C} \frac{\partial^2 u}{\partial^2 t} \tag{2.7}$$

Por simple analogía con la ecuación de onda electromagnética [15], obtenemos que la velocidad de fase de la onda acústica es:

$$v_p = \sqrt{\frac{C}{\rho}} \tag{2.8}$$

Si tomamos la consideración de estar en régimen permanente senoidal podemos asumir que ante cualquier excitación armónica, la ecuación de onda en el dominio fasorial toma la forma:

$$\frac{\partial^2 u}{\partial^2 z} + \frac{\rho}{C} \omega^2 u = 0 \tag{2.9}$$

Buscamos la solución de la ecuación, una ecuación diferencial homogénea de orden dos, y obtenemos:

$$u(z) = A^+ e^{-jkz} + A^- e^{+jkz} \tag{2.10}$$

Que representa dos ondas planas propagándose en z con sentidos opuestos. Y donde definimos la constante de fase como:

$$k = \sqrt{\omega^2 \frac{\rho}{C}} = \frac{\omega}{v_p} \tag{2.11}$$

Finalmente podemos definir la impedancia acústica de la onda como:

$$Z = \frac{-T}{v} = \sqrt{\rho C} = \rho v_p = \frac{Ck}{\omega} \tag{2.12}$$

Las unidades de dicha impedancia son $kg/(sm^2)$, y es una propiedad que depende del medio donde se esta propagando la onda.

2.3.1.2. Solución unideminsonal para una lámina de material no piezoeléctrico

Para el estudio de las diferentes capas que componen un resonador FBAR, deberemos establecer una analogía entre el modelo mecánico y el modelo eléctrico. Esto nos permitirá usar las herramientas que poseemos en el entorno electromagnético para el estudio de los resonadores. Hablar de fuerza y tensión es permisible puesto en el caso unidimensional son equivalentes.

Podemos establecer la siguiente analogía:

- La fuerza en el plano mecánico la podemos asociar a una tensión en el plano eléctrico.
- La velocidad de la partícula en el plano mecánico, la asociamos a la corriente en el plano eléctrico.

En la figura 2.5 podemos ver la geometría de la lámina que vamos a analizar. Una lámina de material no piezoeléctrico de dimensión vertical, d, y área de superficie superior e inferior A.

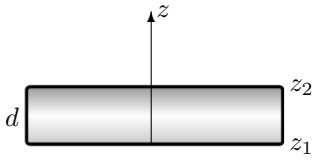

Figura 2.5: *Lámina de material no piezoeléctrico.*

Para analizar este problema, partiremos de la solución de la ecuación de onda para el desplazamiento de la partícula y aplicaremos las condiciones de contorno necesarias en los límites, z_1 y z_2, llegando a la siguiente solución:

$$\begin{aligned} F_1 &= \frac{Z}{j\sin(kd)}(v_1 - v_2) + jZ\tan\left(\tfrac{kd}{2}\right)v_1 \\ F_2 &= \frac{Z}{j\sin(kd)}(v_1 - v_2) - jZ\tan\left(\tfrac{kd}{2}\right)v_2 \end{aligned} \tag{2.13}$$

Con las soluciones de (2.13) obtenemos el circuito equivalente de la figura 2.6

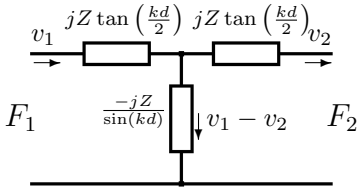

Figura 2.6: *Modelo de impedancias acústicas para una lámina de material no piezoeléctrico.*

2.3.1.3. Solución unideminsonal para una lámina de material piezoeléctrico

Como los materiales piezoeléctricos presentan un momento dipolar ante la presencia de un campo eléctrico, estos materiales presentaran un momento dipolar no nulo ante la presencia de fenómenos mecánicos que provocarán un aumento del flujo de campo eléctrico a través del material.

Con esta premisa, podemos plantear que el vector desplazamiento eléctrico para un material piezoeléctrico, tiene una componente debida al campo eléctrico y otra debida a las deformaciones cumpliéndose la siguiente relación:

$$D = \varepsilon^S E + eS \tag{2.14}$$

Donde e es la constante piezoeléctrica parar el *stress*, y la permitividad se encuentra medida en términos de *strain* constante. Como esta medida puede ser complicada, podemos utilizar la siguiente relación:

$$\varepsilon^S = \varepsilon^T - de \tag{2.15}$$

Donde d es la constante piezoeléctrica para la deformación y ε^T es la constante de permitividad dieléctrica para deformación constante.

Además, las constantes piezoeléctricas para la deformación y para el *stress* están relacionadas a través del *stiffnes* medido para un campo eléctrico constante:

$$e = c^E d \tag{2.16}$$

Finalmente redefinimos la expresión (2.5) de la forma siguiente:

$$T = c^E \frac{\partial u}{\partial z} - eE \tag{2.17}$$

Procediendo del mismo modo que con la búsqueda de la solución para una lámina unidimensional no piezoeléctrica y añadiendo las nuevas expresiones obtenidas,

encontramos la siguiente solución:

$$\begin{aligned} F_1 &= \frac{Z}{j\sin(kd)}(v_1 - v_2) + jZ\tan\left(\tfrac{kd}{2}\right)v_1 + \frac{h}{j\omega}I \\ F_2 &= \frac{Z}{j\sin(kd)}(v_1 - v_2) - jZ\tan\left(\tfrac{kd}{2}\right)v_2 + \frac{h}{j\omega}I \end{aligned} \tag{2.18}$$

Con las soluciones de (2.18) obtenemos el circuito equivalente de la figura 2.7

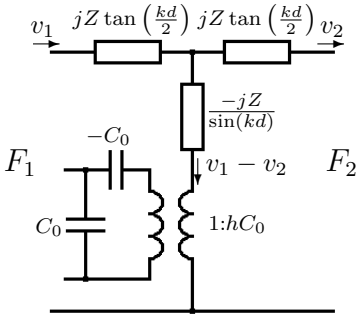

Figura 2.7: *Modelo de impedancias acústicas para una lámina de material piezoeléctrico.*

2.3.1.4. Generalización del modelo de Mason a multicapa

Después de hallar el modelo circuital eléctrico equivalente de la propagación de la onda acústica a través de un material piezoeléctrico, o no, podemos extender el modelo de Mason a una estructura multicapa más compleja, que tenga en cuenta todas las capas del resonador. Utilizando el resonador con reflexion por abertura de aire de la figura 2.3, podemos escribir el modelo equivalente circuital. Este modelo lo encontramos en la figura 2.8.

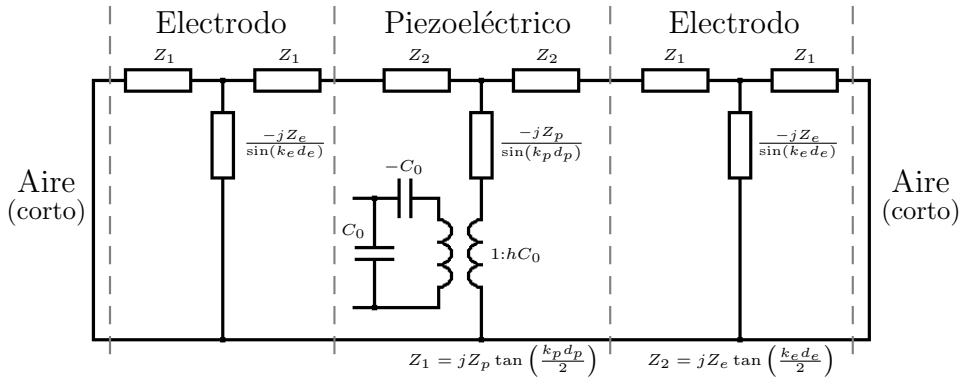

Figura 2.8: *Resonador FBAR con reflexión de onda mediante aire en las dos superficies.*

La impedancia acústica de entrada Z_{in} de una línea de transmisión puede ser calculada como función de la frecuencia f usando la ecuación 2.19 donde Z_0 es la impedancia acústica característica, v la velocidad del sonido en el material, h el grosor de la capa y Z_t la impedancia de terminación. Para múltiples capas, podemos usar dicha ecuación tomando Z_{in} como Z_t para la siguiente capa [1].

$$Z_{in} = Z_0 \frac{Z_t \cos\left(\frac{2\pi}{v} fh\right) + jZ_0 \sin\left(\frac{2\pi}{v} fh\right)}{Z_0 \cos\left(\frac{2\pi}{v} fh\right) + jZ_t \sin\left(\frac{2\pi}{v} fh\right)} \tag{2.19}$$

2.3.1.5. Expresión de la impedancia simplificada para el caso de aire en las dos superficies

Debido a la gran complejidad que conlleva la búsqueda de una expresión cerrada de la impedancia del resonador FBAR mediante el modelo de Mason, cuando contemplamos todo el conjunto multicapa, estudiaremos la aproximación de tener tan solo una capa piezoeléctrica con reflexión total en las superficies del piezoeléctrico. Hecho que conseguiremos mediante la aproximación de aire en sus dos superficies. De esta manera se puede demostrar que la impedancia del resonador es[1]:

$$Z = \frac{1}{j\omega C_0}\left(1 - k_t^2 \frac{\tan(kd/2)}{kd/2}\right) \tag{2.20}$$

Por último, buscaremos las frecuencias serie y paralelo mediante los casos extremos de la impedancia. Para el caso de la frecuencia paralelo f_p, deberemos analizar la impedancia cuando esta vale infinito.

$$Z = \infty \Rightarrow \tan(\phi) = \infty \Rightarrow \phi|_{f_p} = \frac{\pi}{2} \Rightarrow f_p = \frac{v_p}{2d} \tag{2.21}$$

Encontraremos la frecuencia serie f_s, cuando obtengamos un cero en la impedancia.

$$Z = 0 \Rightarrow k_t^2 \frac{\tan(\phi)}{\phi} = 1 \Rightarrow f_s = f_p\left(1 - \frac{4k_t^2}{\pi^2}\right) \tag{2.22}$$

Si graficamos la impedancia en función de la frecuencia obtenemos la grafica de la figura 2.10 (a). Donde también podemos apreciar los picos de resonancia de las frecuencias serie y paralelo.

2.3.2. Modelo de Butterworth Van-Dyke

La respuesta del circuito equivalente del resonador FBAR sin perdidas, es idéntica a la del circuito conocido como modelo de Butterworth Van-Dyke. Este modelo fue desarrollado para modelar eléctricamente el comportamiento de otros materiales piezoeléctricos, como el cristal de cuarzo. A diferencia del modelo de Mason, el modelo de BVD utiliza elementos concentrados. Debido a la sencillez de este modelo, es muy adecuado para hacer diseños y análisis de circuitos donde los resonadores no son el objetivo directo, sino una herramienta para conseguir otro dispositivo como por ejemplo el diseño de filtros o duplexores.

El modelo propuesto, que podemos ver en la figura 2.9 consta de dos ramas. La rama formada por C_m, R_m y L_m se denomina rama motora, y a la rama formada tan solo por la capacidad C_0 se le llama rama estática.

En un resonador FBAR, el área nos determina la capacidad C_0, que esta ligada a la geometría del diseño del resonador (Capacidad geométrica). El cociente C_0/C_m esta definido por la topología y el material piezoeléctrico del resonador. La inductancia L_m hace resonar a C_m a la frecuencia de resonancia determinada por el grosor y R_m esta determinada por las perdidas debido a la vibración del dispositivo [12].

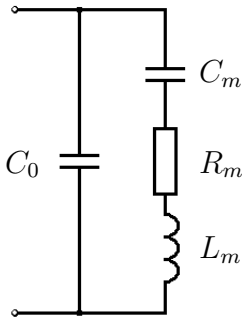

Figura 2.9: *Circuito equivalente del modelo de Butterworth Van-Dyke.*

Del análisis de este circuito, obtenemos que sus frecuencias de resonancia serie y paralelo son:

$$f_s = \frac{1}{2\pi}\sqrt{\frac{1}{L_m C_m}} \tag{2.23}$$

$$f_p = \frac{1}{2\pi}\sqrt{\frac{C_0 + C_m}{L_m C_m C_0}} \tag{2.24}$$

Para el caso del modelo BVD obtenemos la siguiente impedancia:

$$Z = \frac{\left(2\pi j f L_m + R_m + \frac{1}{j2\pi f C_m}\right)\frac{1}{j2\pi f C_0}}{2\pi j f L_m + R_m + \frac{1}{j2\pi f} + \frac{1}{j2\pi f C_0}} \tag{2.25}$$

Como podemos observar en las figuras 2.10 (a) y (b), la impedancia característica del modelo BVD es idéntica a la del modelo de Mason.

2.3.3. Modelo de Butterworth Van-Dyke Modificado

Para simular efectos parásitos con el modelo de BVD, debemos modificar dicho modelo añadiendo parámetros adicionales según los efectos que queramos simular.

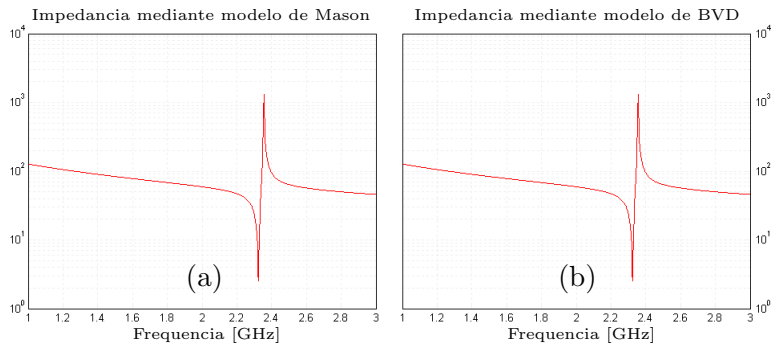

Figura 2.10: *Impedancia hallada mediante el modelo de Mason(a). Impedancia hallada mediante el modelo de Butterworth Van-Dyke(b).*

Como este trabajo trata sobre la extracción de parámetros de unas medidas reales, es necesario que el modelo se ajuste lo mejor posible al dispositivo físico real. Esto lo conseguiremos añadiendo los siguientes parámetros al circuito inicial de BVD:

R_{sub}	Perdidas resistivas del substrato.
C_{sub}	Capacidad del substrato a masa del chip.
C_{ox}	Capacidad entre el electrodo inferior y la superficie del substrato.
R_s	Resistencia serie de un resonador.
R_p	Perdidas asociadas a la rama OTRA del resonador.

El modelo obtenido mediante este método se denomina modelo de Butterworth Van-Dyke Modificado (MBVD) y se muestra en la figura 2.11.

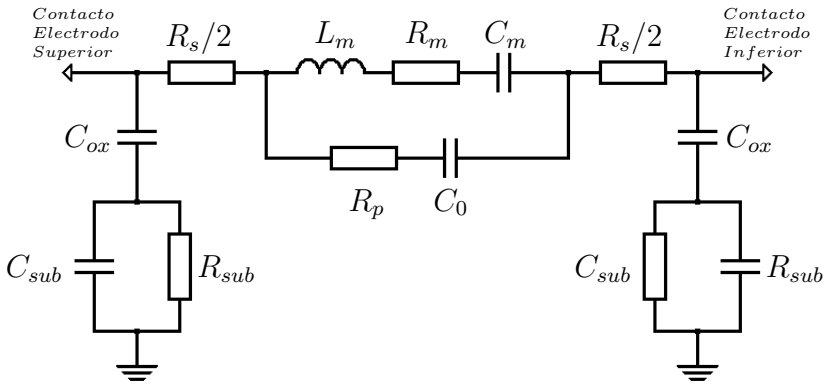

Figura 2.11: *Circuito del modelo de Butterworth Van-Dyke Modificado.*

2.3.4. Impedancia del modelo de MBVD

Para hallar la impedancia de este modelo, podríamos utilizar cualquier tipo de análisis como por ejemplo el uso de matrices de impedancia o admitancia. No ob-

stante, se puede observar en el circuito que este, esta formado por un subconjunto de bipuertos. La utilización de la matriz de transmisión ABCD es ideal para el análisis de bipuertos en cascada puesto el cálculo se convierte en una simple multiplicación de matrices 2×2. La matriz ABCD de un bipuerto queda definida por las siguientes expresiones en función de los voltajes e intensidades de la red [13]:

$$\begin{array}{rcl} V_1 & = & AV_2 + BI_2 \\ I_1 & = & CV_2 + DI_2 \end{array} \tag{2.26}$$

Pero la forma más usada para su representación es en forma de matriz:

$$\left[\begin{array}{c} V_1 \\ I_1 \end{array} \right] = \left[\begin{array}{cc} A & B \\ C & D \end{array} \right] \left[\begin{array}{c} V_2 \\ I_2 \end{array} \right] \tag{2.27}$$

Otro motivo por el cual las matrices ABCD son tan utilizadas es por el hecho de que existen toda una serie de tablas con diferentes bipuertos que pueden ser utilizadas para hallar fácilmente la matriz del circuito deseado. En nuestro caso utilizaremos la conversión del circuito en π.

$$\left[\begin{array}{cc} A & B \\ C & D \end{array} \right] = \left[\begin{array}{cc} 1 + \frac{Y_S}{Y_R} & \frac{1}{Y_R} \\ 2Y_S + Y_R + \frac{Y_S^2}{Y_R} & 1 + \frac{Y_S}{Y_R} \end{array} \right] \tag{2.28}$$

Donde Y_R es la admitancia del bipuerto formado por el resonador e Y_S la admitancia de una de las ramas de perdidas introducidas por el modelo MBVD.

$$Y_R^{-1} = Z_R = \frac{R_{sub}\frac{1}{j\omega C_{sub}}}{R_{sub} + \frac{1}{j\omega C_{sub}}} + \frac{1}{j\omega C_{ox}} \tag{2.29}$$

$$Y_R^{-1} = Z_R = \frac{\left(L_m j\omega + R_m + \frac{1}{j\omega C_m}\right)\left(R_P + \frac{1}{j\omega C_0}\right)}{L_m j\omega + R_m + \frac{1}{j\omega C_m} + R_P + \frac{1}{j\omega C_0}} + R_S \tag{2.30}$$

Finalmente mediante la transformación de la matriz ABCD a la matriz de impedancias obtenemos la impedancia del resonador:

$$Z = \frac{A}{C} = \frac{1 + \frac{Y_S}{Y_R}}{2Y_S + Y_R + \frac{Y_S^2}{Y_R}} \tag{2.31}$$

Podemos apreciar, en la figura 2.12 que esta vez la respuesta de la impedancia del resonador es diferente al de los modelos discutidos anteriormente. Asemejándose más a la respuesta real de un resonador FBAR.

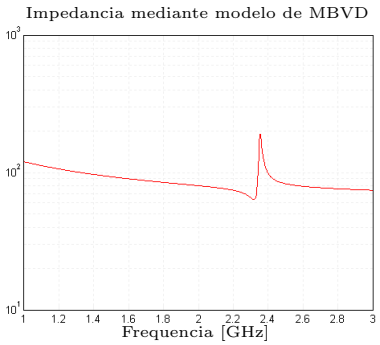

Figura 2.12: *Impedancia hallada mediante el modelo de BVD Modificado.*

2.4. Conclusiones

En este capítulo se ha explicado el funcionamiento, la estructura general y el modelado de los resonadores FBAR.

Hemos visto como un resonador FBAR basa su funcionamiento en el efecto inverso piezoeléctrico para generar una onda acústica que viaja entre los electrodos del resonador a la frecuencia de resonancia definida por los parámetros de diseño del propio resonador.

La estructura básica que se ha presentado para lograr obtener el resonador FBAR, ha sido explicada y ampliada con las dos geometrías que se usan habitualmente en la fabricación de resonadores FBAR. La primera de ellas, el resonador de membrana, usa la aproximación de aire en las dos superficies, llevada a cabo mediante una abertura de aire en la estructura, que permite a la onda acústica reflejarse casi por completo. El otro sistema que hemos estudiado ha sido el refractor de Bragg, que consiste en la inserción de sucesivas capas con impedancia acústica alta y baja, para conseguir la reflexión acústica deseada.

En referéncia al modelado del resonador, se han presentado los tres modelos más utilizados en el estudio de los resonadores FBAR.

El primer modelo que se ha introducido ha sido el modelo de Mason que nos ha permitido obtener un circuito equivalente electromagnético a partir del comportamiento acústico. Este modelo circuital equivalente esta constituido por elementos distribuidos que dificultan los cálculos a la hora de trabajar con el modelo, pero a cambio hemos obtenido un modelo muy eficiente del comportamiento de las capas de un resonador. Gracias a esto, podemos estudiar el efecto de la carga mecánica sobre cualquier combinación de capas, concatenando los circuitos equivalentes hallados en 2.3.1.2 y 2.3.1.3.

En segundo lugar se ha hecho referencia al modelo de Butterworth Van-Dyke que permite modelar resonadores basados en piezoeléctricos. Hemos visto que este modelo es más sencillo que el modelo de Mason, ya que contiene elementos concentrados. Este motivo le hace muy adecuado para diseñar y analizar eficientemente circuitos complejos que contengan resonadores.

En último lugar se ha introducido el modelo Modificado de Butterworth Van-Dyke que consiste en añadir parámetros de efectos parásitos o de perdidas, mediante la adición de elementos concentrados. Este modelo es una respuesta más real que los anteriores modelos, puesto tiene en cuenta las perdidas debidas a la tecnología de diseño con que se ha fabricado.

Como hemos podido ver en las soluciones presentadas de los diferentes modelos, los resonadores tienen dos frecuencias de resonancia muy próximas entre las cuales obtenemos un efecto inductivo muy fuerte, o un cambio de fase muy abrupto que nos da información sobre la buena calidad de los resonador FBAR.

Capítulo 3

Algoritmos de optimización

3.1. Introducción

La optimización de modelos intenta expresar, en términos matemáticos, la mejor solución posible de un problema. Esto puede significar mejorar la producción de una empresa para maximizar su producción o minimizar los gastos para obtener mayor beneficio. El deseo de solucionar problemas de forma óptima, es tan común que se puede aplicar en cualquier tipo de entorno, ya sea en áreas ingenieriles, como en entornos empresariales, donde por ejemplo una pequeña mejora puede suponer beneficios millonarios.

Aunque la optimización de modelos ha sido utilizada desde hace siglos, la expansión y mejora, en las últimas décadas, de los ordenadores, ha permitido que la optimización de modelos sea una herramienta real y práctica en negocios, ciencia e ingeniería.

Los diferentes modelos con que podemos encontrarnos, podemos clasificarlos según la relación que tienen los parámetros con el propio modelo. De esta manera podemos agruparlos en dos grandes grupos, los modelos con ecuaciones lineales y los modelos con ecuaciones no-lineales.

3.2. Programación Lineal

Los sistemas de ecuaciones lineales son la parte central de casi todos los algoritmos de optimización, y se encuentran en una gran cantidad de modelos. Además, los sistemas de ecuaciones lineales también se usan para representar constricciones a las variables dentro de un modelo.

Un problema de programación lineal implica la optimización de una función lineal sujeta a constricciones lineales en las variables. Aunque las funciones lineales son sencillas, podemos encontrarlas con frecuencia en problemas económicos, de redes, de programación horaria y otras muchas aplicaciones.

La programación lineal, puede ser estudiada tanto algebraicamente, como geométricamente. Las dos formas son equivalentes pero en determinados problemas una puede funcionar mejor que la otra. Para solucionar problemas lineales con una gran cantidad de variables y constricciones, es aconsejable proceder mediante la división del problema en subproblemas.

3.3. Programación No-Lineal

Aunque los problemas de programación lineal son muy comunes, y cubren un gran número de aplicaciones, en el mundo real nos enfrentamos a una gran cantidad de problemas que no lo son. Un modelo de programación no-lineal consiste en la optimización de una función sujeta a posibles constricciones, donde cualquiera de las funciones ya sea el propio modelo, las constricciones o ambas, puede ser no-lineales. Este es el tipo más general e incluye todos los demás problemas como casos particulares. Problemas de programación no-lineal aparecen comúnmente en ingenierías y ciencias, como es el caso de este trabajo, donde se ha llevado a cabo una optimización de parámetros mediante un modelo no-lineal. Por ello en los siguientes apartados se introducirán algunas de las técnicas para solucionar estos problemas.

3.4. Concepto de optimización

Tal y comom podemos extraer de [16], la optimización trata de buscar la mejor solución a un problema determinado. Matemáticamente podemos explicar dicho concepto mediante la minimización, o maximización, de una determinada función $f(x)$ en un espació n-dimensional.

$$\min_{x \in S} \quad f(x) \tag{3.1}$$

El conjunto de posibles soluciones que pueden satisfacer la anterior ecuación, pueden depender, o no, de un conjunto de constricciones. Por ejemplo podríamos establecer que el conjunto de soluciones tan solo formara parte del espacio no negativo ($x \geq 0 \quad \forall x \in S$).

Sin tener en cuenta ninguna constricción, la definición más básica de solución es x_* que minimiza f si

$$f(x_*) \leq f(x) \quad \forall x \in S \tag{3.2}$$

3.4.1. Algoritmo general de optimización

El hecho de que la solución de problemas de optimización pueda ser resuelta de muchas maneras distintas, ha propiciado que existan una gran cantidad de algoritmos específicos para cada aplicación o conjunto de aplicaciones. Aun así, la gran mayoría de estos algoritmos, incluido el creado para este trabajo, comparten

la misma base común explicada a continuación en pseudocódigo:

Algoritmo general de optimización
Especificar una conjetura inicial de la solución x_0.
Para $k = 0, 1, \ldots$
    Si x_k es óptima acaba.
    Determina x_{k+1}, una nueva estimación de la solución

Aunque el algoritmo nos sugiere que la pregunta de la optimización y la determinación de un nuevo valor x_{k+1} son conceptos separados, esto no acostumbra a ser así, ya que a veces la propia información de la condición de optimización es la base de la siguiente iteración.

3.5. Ajuste de datos por mínimos cuadrados

Para obtener un conjunto de variables de un modelo teórico, mediante una serie de muestras o datos recogidos, tan solo son necesarios un número de puntos igual a la cantidad de variables que tenemos en el modelo. Sin embargo normalmente se cogen más datos que cantidad de variables, como es el caso de encuestas políticas o experimentos científicos. Esto es debido a que se supone que cada muestra contiene un cierto error y que la colección completa de muestras recogidas, serán utilizadas de forma colectiva, con la esperanza de encontrar mejores resultados que con un pequeño conjunto de muestras. Debido a que consideramos que todas las muestras contendrán algún tipo de error, ya no consideraremos que los modelos utilizados sean resueltos de manera exacta. En vez de ello intentaremos encontrar aproximaciones mediante el cálculo del *vector residuo*. La aproximación más utilizada es denominada ajuste de datos por mínimos cuadrados, donde trataremos de minimizar la suma de los cuadrados de las componentes del *vector residuo*:

$$\min_x \quad r_1^2 + \cdots + r_m^2 = \sum_{i=1}^{m} \left[b_i - m\left(t_i\right)\right]^2 \tag{3.3}$$

Donde b_i son los datos obtenidos experimentalmente y $m\left(t_i\right)$ el modelo evaluado en los instantes t_i. Si los residuos pueden expresarse como $r = b - Ax$, entonces el modelo es lineal. Que el modelo sea lineal no implica que la dependencia con otras variables deba ser también lineal. En las ecuaciones siguientes podemos ver ejemplos de modelos lineales:

$$\begin{aligned} m\left(t\right) &= x_1 + x_2 \sin(t) + x_3 \sin(2t) + x_{k+1} \sin(kt) \\ m\left(t\right) &= x_1 + \frac{x_2}{1 + t^2} \end{aligned} \tag{3.4}$$

De todas formas los modelos no-lineales son mucho mas comunes. En los siguientes ejemplos podemos ver un par de dichos problemas.

$$\begin{aligned} m\left(t\right) &= x_1 + x_2 e^{x_3 t} + x_4 e^{x_5 t} \\ m\left(t\right) &= x_1 + \frac{x_2}{1 + x_3 t^2} \end{aligned} \tag{3.5}$$

En estos modelos, hay relaciones no-lineales entre los coeficientes x_j. Un ajuste de datos no-lineal puede expresarse como:

$$\min_x \quad \sum_{i=1}^{m} f_i(x)^2 \tag{3.6}$$

Donde $f_i(x)$ representa el residuo evaluado en t_i. Por ejemplo utilizando el primer modelo de las ecuaciones en 3.5:

$$f_i(x) \equiv b_i - \left(x_1 + x_2 e^{x_3 t} + x_4 e^{x_5 t}\right) \tag{3.7}$$

Un modelo de ajuste de datos no-lineal es un ejemplo de problema de minimización sin constricciones. Este tipo es uno de los problemas de optimización más comunes que podemos encontrar. Concretamente, el problema a solucionar en este trabajo es precisamente un ajuste de parámetros no-lineal.

3.5.1. Método de Newton para ecuaciones no-lineales

El primer método que estudiaremos, basa su funcionamiento en aproximar la función no-lineal por tangentes en el punto x_k. De esta manera conseguimos transformar nuestro problema no-lineal en una secuencia de problemas lineales para así poder aplicar nuestro conocimiento de algebra lineal en la resolución del problema.

Dada una conjetura x_k, la función f es aproximada por la función lineal consistente en los dos primeros términos de la serie de Taylor de la función f evaluada en el punto de la estimación x_k:

$$f(x_k + p) \approx f(x_k) + pf'(x_k) \tag{3.8}$$

Si $f'(x_k) \neq 0$ entonces podemos solucionar la ecuación

$$f(x_*) \approx f(x_k) + pf'(x_k) = 0 \tag{3.9}$$

De lo que obtenemos p:

$$p = -\frac{f(x_k)}{f'(x_k)} \tag{3.10}$$

Y de esta manera obtenemos la formula de Newton. La nueva estimación de la solución es $x_{k+1} = x_k + p$ o bien:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{3.11}$$

Como ya se ha mencionado, el método de Newton consiste en aproximar la función f mediante su tangente en el punto x_k. Cogemos el punto donde cruza la tangente con la el eje x como la nueva estimación. Esta interpretación geométrica podemos verla en la figura 3.1. Típicamente el método de Newton converge de una forma rápida y una vez x_k esta cerca de la solución x_*, el error es reducido de forma cuadrática cada iteración.

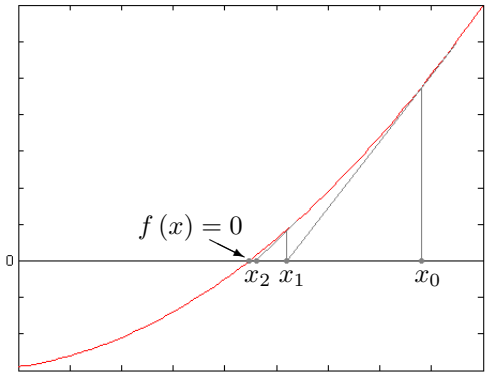

Figura 3.1: *Interpretación geométrica del método de Newton.*

3.5.2. Método de Gauss-Newton

Partimos de la ecuación 3.6 y reescribimos el problema de ajuste de datos de la forma:

$$\min_x f(x) = \frac{1}{2}\sum_{i=1}^{m} f_i(x)^2 \equiv \frac{1}{2}F(x)^T(x) \tag{3.12}$$

Donde F es el *vector residuo* y F^T su transpuesto.

$$F(x) = (f_1(x)\, f_2(x) \cdots f_m(x))^T \tag{3.13}$$

Adicionalmente hemos escalado el problema por $\frac{1}{2}$ para hacer más sencillas las derivadas. Las componentes de $\nabla f(x)$ se pueden derivar de la regla de la cadena:

$$\nabla f(x) = \nabla F(x)\, F(x) \tag{3.14}$$

Podemos encontrar $\nabla^2 f(x)$ derivando la formula anterior respecto x_j:

$$\nabla^2 f(x) = \nabla F(x)\, \nabla F(x)^T + \sum_{i=1}^{m} f_i(x)\, \nabla^2 f_i(x) \tag{3.15}$$

Las ecuaciones 3.14 y 3.15 son respectivamente el Jacobiano y el Hessiano de f.

Supongamos que x_* es la solución del problema de mínimos cuadrados, y que en la solución $f_i(x_*) = 0\ \forall i$, indicando que todos los residuos son cero y que el modelo ajusta perfectamente los datos sin error. Como resultado obtenemos $F(x_*) = 0$ y en consecuencia $\nabla f(x_*) = 0$. Este resultado implica también que:

$$\nabla^2 f(x_*) = \nabla F(x_*)\, \nabla F(x_*)^T \tag{3.16}$$

Por lo que el Hessiano en valores cercanos a la solución se puede aproximar por:

$$\nabla^2 f(x) = \nabla F(x)\, \nabla F(x)^T + \sum_{i=1}^{m} f_i(x)\, \nabla^2 f_i(x) \approx \nabla F(x)\, \nabla F(x)^T \tag{3.17}$$

Esta formula final, tan solo implica a las primeras derivadas de las funciones f_i y nos muestra que podemos aproximar el Hessiano mediante la utilización de estas primeras derivadas, cuando el modelo teórico ajusta bien los datos. Resultado que conlleva a una mejora sustancial del tiempo de cómputo y la dificultad de la implementación del Hessiano en lenguajes informáticos. Esta idea es la base para un gran numero de métodos especializados para ajuste de datos no lineales.

El método más simple es el llamado *método de Gauss-Newton* y utiliza esta aproximación directamente para calcular una dirección de búsqueda utilizando la formula para el método de Newton 3.8.

$$\nabla^2 f(x)\, p = -\nabla f(x) \tag{3.18}$$

Si reemplazamos el Hessiano por la aproximación anterior obtenemos:

$$\nabla F(x)\, \nabla F(x)^T\, p = -\nabla F(x)\, F(x) \tag{3.19}$$

Para el caso donde $F(x_*) = 0$ i $\nabla F(x_*)$ sea de rango entero, el método de Gauss-Newton se comporta como el método de Newton cerca de la solución, pero sin el coste de calcular las segundas derivadas.

Por otra parte, el método de Gauss-Newton tiene un comportamiento *pobre* cuando los residuos cerca de la solución no son pequeños, o cuando el modelo no ajusta bien los datos. También se comportara de forma *pobre* cuando el Jacobiano de F no sea de rango entero indicando así que la aproximación del Hessiano no es buena.

3.6. Conclusiones

En este capituló se ha introducido la teoría básica necesaria para comprender la optimización llevada a cabo en el trabajo. Se ha empezando por introducir el concepto de programación lineal y no-lineal y de ello podemos concluir que necesitamos un algoritmo no-lineal debido a que como se ha visto en capítulos anteriores, el modelo teórico que utilizamos es claramente no-lineal, puesto en general los parámetros tienen una relación no-lineal con el modelo.

Partiendo del hecho que necesitamos un algoritmo no-lineal para encontrar la solución al problema que nos concierne, se desarrolla a partir del método de Newton, el método de Gauss-Newton que consiste en aproximar la función mediante la serie de Taylor con una aproximación sencilla de la matriz de segundas derivadas o Hessiano.

Capítulo 4

Algoritmo

4.1. Introducción

En este capítulo se explica el algoritmo de optimización generado para la extracción de parámetros mediante el ajuste de datos con el modelo teórico de Butterworth Van-Dyke modificado. Este capítulo introducirá el código elaborado con Matlab.

Empezaremos introduciendo los diferentes códigos que componen nuestro programa. Una vez explicados, se introducirá el entorno gráfico desarrollado y por último se presentaran los resultados obtenidos de la optimización de los parámetros. En el apéndice podremos encontrar todos los códigos descritos en este capítulo.

4.2. Medidas disponibles

La información que tenemos para elaborar la extracción de parámetros, serán una serie de medidas extraídas de los resonadores FBAR fabricados en el laboratorio del CNM. El aparato utilizado para medir los datos, es un analizador de redes vectorial y nos proporciona la medida de los parámetros S, concretamente se ha proporcionado el parámetros S_{11} en módulo y el S_{21} en módulo y fase.

4.2.1. La matriz de Scattering

Podemos extraer de [13] que al igual que la matriz de impedancias o la matriz de admitancias, la matriz de Scattering proporciona una descripción completa de una red de N puertos. Esta matriz se utilizada ya que la medida de voltajes i intensidades se vuelve complicada a frecuencias de microondas. La matriz de parámetros S nos informa de las ondas que entran y salen por cada puerto, permitiendo así calcular los parámetros mediante un analizador de redes vectorial. Una vez se ha hallado la matriz de scattering, podemos convertir dichos parámetros a matrices de impedancias o admitancias si nos conviene. La matriz de Scattering

$[S]$ esta definida por:

$$\begin{bmatrix} V_1^- \\ V_2^- \\ \vdots \\ V_N^- \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} & \cdots & S_{1N} \\ S_{21} & S_{22} & & \vdots \\ \vdots & & \ddots & \\ S_{N1} & \cdots & & S_{NN} \end{bmatrix} \begin{bmatrix} V_1^+ \\ V_2^+ \\ \vdots \\ V_N^+ \end{bmatrix} \tag{4.1}$$

Donde podemos definir cada parámetro S_{ij} como:

$$S_{ij} = \left. \frac{V_i^-}{V_j^+} \right|_{V_k^+ = 0 \quad \forall \quad k \neq j} \tag{4.2}$$

Como los resonadores FBAR son una estructura bipuerto, tendremos la matriz de parámetros S siguiente:

$$\begin{bmatrix} V_1^- \\ V_2^- \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} V_1^+ \\ V_2^+ \end{bmatrix} \tag{4.3}$$

El parámetro S_{11} podemos hallarlo teniendo en cuenta que dicho parámetro es el coeficiente de reflexión a la entrada evitando que haya onda incidente por el puerto 2. Esto podemos conseguirlo mediante la terminación del puerto 2 con una carga Z_0 adaptada a la impedancia del resonador.

$$S_{11} = \left. \frac{V_1^-}{V_1^+} \right|_{V_2^+ = 0} \tag{4.4}$$

Por otra parte podemos deducir que el parámetro S_{21} es el coeficiente de transmisión del puerto 1 al puerto 2:

$$S_{21} = \left. \frac{V_2^-}{V_1^+} \right|_{V_2^+ = 0} \tag{4.5}$$

Aunque tan solo son necesarios estos dos parámetros para la elaboración del algoritmo, podemos hallar los otros dos fácilmente ya que el resonador es una red de dos puertos simétrica, por lo que los parámetros S_{11} y S_{22}, así como S_{21} y S_{12} son iguales.

4.3. El cuerpo del algoritmo

Como se ha introducido, el algoritmo que utilizaremos en este trabajo está basado en un ajuste por mínimos cuadrados no-lineal mediante el método de Gauss-Newton. Este algoritmo se puede dividir en las siguientes etapas:

- **Medidas S_{11} y S_{21} :** En esta etapa, se recogen las medidas del resonador a estudiar y se procesan para tener los datos deseados. En nuestro caso, mediante un analizador de redes vectorial, capturamos los parámetros S_{11} en módulo y S_{21} en módulo y fase.
- **Conjetura inicial :** Escogemos una conjetura inicial que introduciremos al algoritmo para facilitar la convergencia de este. En un caso general, se utiliza una conjetura estimada por el orden de magnitud de los parámetros según otros resonadores parecidos.
- **Modelado :** En esta etapa, se desarrolla el modelo que se utilizará para la optimización, en nuestro caso se han tenido que encontrar los modelos teóricos de los módulos y fases de los parámetros S medidos.
- **Optimización :** Aplicamos el algoritmo mediante el modelo y los datos. Si la función del modelo teórico se ajusta adecuadamente a los datos, este nos devolverá el resultado. Cabe destacar que se usa una estrategia por pasos para la ejecución del algoritmo.
- **Extracción de los parámetros :** Una vez ha finalizado el algoritmo, se extraen los parámetros que modelan teóricamente el comportamiento del resonador FBAR fabricado.

En la figurar 4.1 podemos observar un esquema de los pasos que se han a seguido para llevar a cabo la optimización.

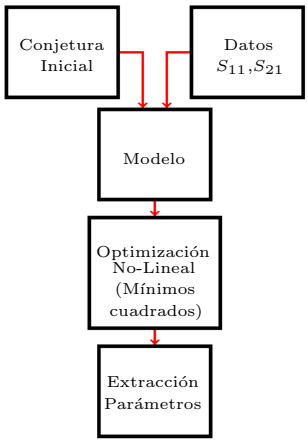

Figura 4.1: *Esquema del algoritmo.*

4.4. La implementación del código

Para implementar el código del algoritmo se ha utilizado el lenguaje de programación incorporado por Matlab. Este programa, desarrollado por "*The MathWorks*", es un programa de cálculo numérico orientado a matrices, ampliamente

extendido por universidades, centros de investigación e ingenieros. La potencia de dicho programa radica en la utilización de un lenguaje interpretado, muy sencillo de manejar que ha sido desarrollado para que el cálculo con vectores y matrices se procese de la forma más eficiente posible. Como nuestro algoritmo deberá trabajar con vectores de datos procedentes de las medidas del laboratorio y aplicar el algoritmo directamente sobre estos, Matlab ha sido escogido para realizar la implementación del código de este trabajo.

4.4.1. Organización del código

El código del proyecto, se ha dividido en diversas funciones encargadas de realizar una tarea especifica. Estas funciones se encuentran en ficheros separados para facilitar la lectura del código.

4.4.2. El modelo

El modelo teórico del resonador FBAR, utiliza la ecuación 2.31 para evaluar el modelo de Butterworth Van-Dyke Modificado según los parámetros de la iteración actual donde se halle el algoritmo. Mediante el pase de argumentos, podemos seleccionar que parámetro S queremos que la función retorne y si este ha de estar en módulo o en fase.

Esta función, es importante que este en un fichero aislada puesto si queremos añadir algún elemento al circuito equivalente del modelo MBVD, aquí es donde deberemos hacerlo. Esta es una de las razones por la que se han utilizado matrices ABCD (*ver 2.3.4*) en el calculo de los parámetros S, ya que facilitan la adición de elementos, siendo necesario recalcular tan solo aquellas matrices donde se hayan añadido los elementos.

Para convertir la impedancia obtenida del modelo MBVD a parámetros S, tan solo se han tenido que usar las transformaciones adecuadas extraídas de [13]:

$$S_{11} = \frac{A + B/Z_0 - CZ_0 - D}{A + B/Z_0 + CZ_0 + D} \tag{4.6}$$

$$S_{21} = \frac{2}{A + B/Z_0 + CZ_0 + D} \tag{4.7}$$

4.4.3. Función del Jacobiano

Debido a la incapacidad de Matlab para evaluar simbólicamente el valor absoluto de un número complejo, se descarto la posibilidad de utilizar la función de cálculo simbólico *jacobian* que incorpora Matlab para el cálculo del modelo. Se tuvo que implementar un Jacobiano basado en diferencias finitas para poderlo

aplicar al algoritmo. El Jacobiano de una función F compuesta de m funciones $Y_1(x_1,\ldots,x_n),\ldots,Y_m(x_1,\ldots,x_n)$ se puede interpretar como:

$$J_F(x_1,\ldots,x_n)=\frac{\partial(Y_1,\ldots,Y_m)}{\partial(x_1,\ldots,x_n)} \tag{4.8}$$

Aunque la forma más común de representar el Jacobiano es mediante su matriz:

$$J_F(x_1,\ldots,x_n)=\begin{bmatrix}\frac{\partial Y_1}{\partial x_1} & \cdots & \frac{\partial Y_1}{\partial x_n}\\ \vdots & \ddots & \vdots\\ \frac{\partial Y_m}{\partial x_1} & \cdots & \frac{\partial Y_m}{\partial x_n}\end{bmatrix} \tag{4.9}$$

El cálculo analítico de esta función en Matlab es muy costosa en cuanto a computo, por ello se ha preferido utilizar una estrategia de aproximación para dicha función. La aproximación de la matriz Jacobiana mediante diferencias finitas, consiste en hacer la diferencia de la función evaluada en el punto $(x_1,\ldots,x_n)$ y en una variación finita de este mediante la adición a los parámetros de un valor pequeño que denotamos como ε. De esta manera, obtenemos la función del Jacobiano tal y como se muestra en la siguiente ecuación:

$$J_{FD}=\frac{\Delta(x_1,\ldots,x_n)}{\varepsilon}=\frac{F(x_1+\varepsilon,\ldots,x_n+\varepsilon)-F(x_1,\ldots,x_n)}{\varepsilon} \tag{4.10}$$

Para el caso concreto de nuestro algoritmo, las variables $(x_1,\ldots,x_n)$ serán los parámetros ha extraer $(R_{sub},C_{sub},C_{ox},R_S,R_P,C_0,L_m,R_m,C_m)$, la función F será la función del modelo evaluada en dichos parámetros y ε será el valor de $\sqrt{eps}$ donde *eps* es la distancia positiva entre dos números consecutivos de la misma precisión. Ha esta función se le ha denominado *jacobFD*.

4.4.4. El núcleo del algoritmo

El algoritmo principal de la optimización, se halla en la función *gn_method*. Esta función es la encargada de ejecutar la optimización no-lineal mediante mínimos cuadrados. Consiste básicamente en la aplicación del método de Gauss-Newton, visto en el apartado 3.5.2, sobre los vectores de datos obtenidos de las medidas.

Mediante el pase de argumentos, podemos manipular algunos parámetros del algoritmo, como por ejemplo el número de iteraciones máximas que queremos que ejecute, la tolerancia que queremos que alcance o el número de muestras con que queremos aplicar el algoritmo. Según los valores que demos a estos parámetros, lograremos un resultado más preciso, a cambió de un mayor tiempo de cómputo.

Cabe destacar que si en cualquier momento la función no converge, el algoritmo lo detecta i detiene automáticamente la ejecución mostrando por la salida estándar el mensaje de error pertinente.

4.4.5. El algoritmo por etapas

Hasta este punto, el algoritmo puede funcionar para cualquier función que introduzcamos en el modelo, si este se ajusta a los datos proporcionados. No obstante, debido al gran número de parámetros de nuestro modelo, se hace necesario usar una estrategia de resolución segmentada en diferentes etapas. En cada una de estas etapas, se optimizará alguno o varios de estos parámetros con alguna de las mediciones disponibles. Para esto, se debe elaborar una función que ejecute el algoritmo de optimización de *gn_method* de forma secuencial para cada uno de los parámetros en un orden determinado con el fin de que la función converja. Se ha llamado a esta función *FBAR_optim*.

La secuencialización elaborada para nuestro problema no es única y en nuestro caso no hemos utilizado el argumento del parámetro S_{21}. Concretamente nuestra secuencia de optimización consta de pasos tal y como podemos ver en la tabla 4.1.

Paso	Elemento Optimizado	Parámetro S Ajustado	Propiedad
1	C_0	S_{11}	Módulo
2	L_m, C_m	S_{11}	Argumento
3	C_{ox}	S_{11}	Módulo
4	R_S	S_{11}	Argumento
5	R_P	S_{11}	Módulo
6	R_m	S_{11}	Módulo
7	C_{ox}	S_{21}	Módulo
8	C_{sub}	S_{21}	Módulo
9	R_{sub}	S_{11}	Módulo

Tabla 4.1: *Pasos del algoritmo secuencial.*

Esta función además de contener los pasos necesarios para desarrollar la optimización, también cuenta con una breve etapa para elaborar una conjetura inicial de alguno de los parámetros de nuestro modelo. Esta etapa es necesaria para calcular una aproximación de L_m y C_m puesto una elección aleatoria aunque cercana, puede dar como resultado una translación de las frecuencias de resonancia. Este hecho hace que el algoritmo falle debido a que este intentará eliminar la región de resonancia en lugar de desplazarla hasta su sitio óptimo.

Para hallar la conjetura inicial de estos dos parámetros, primero ejecutamos el algoritmo de optimización para el parámetro C_0 puesto es necesario para la aproximación de C_m y L_m, tal y como podemos ver en las ecuaciones 2.23 y 2.24. Una vez hemos encontrado una primera aproximación para C_0, buscamos la frecuencia serie f_s y paralelo f_p. Es fácil encontrar dichas frecuencias hallando

el instante donde el signo de la reactancia de la respuesta del resonador cambia. Una vez hallados dichos valores, se busca el valor de los parámetros mediante las formulas analíticas de f_s y f_p.

La función también cuenta con el control de la tolerancia. Mediante la ecuación de error 4.11 comprueba si se ha logrado la tolerancia y en caso afirmativo termina la ejecución indicandolo por pantalla.

$$\epsilon = \sum_{k=1}^{m} f\left(x_k\right) \tag{4.11}$$

4.4.6. Control de errores y avisos

A lo largo del desarrollo del algoritmo, se pueden dar toda una serie de eventos que hagan que nuestro problema no pueda ser solucionado, una mal elección de conjetura inicial, que el modelo no ajuste bien las medidas... Cada programa de los anteriormente mencionados, cuenta con una serie de avisos que nos indicaran cuando y porque se ha producido un error.

Además de los errores que impidan el correcto funcionamiento del programa, se han añadido mensajes de información como el número actual de iteración, o el aviso concerniente a que una de las matrices sea singular (ver en [16]).

Se ha creado un sistema para poder activar i desactivar todos estos avisos según convengan en cada momento, que consisten en el paso mediante argumento de un vector con variables de control.

4.5. Interfaz gráfica de usuario

Para facilitar el uso del algoritmo, se ha añadido una interfaz gráfica fácil de utilizar, donde se puede controlar la optimización sin necesidad de tener que escribir ni recordar los comandos o los argumentos a pasar.

Esta interfaz gráfica de usuario (GUI), se abre mediante el comando *FBAR_GUI* y es necesario que le pasemos las medidas en forma de matriz como argumento. Una vez inicializado el entorno, hallaremos una ventana como la que se muestra en la figura 4.2. En ella podemos ver el circuito así como la aproximación inicial y una serie de elementos de control.

La interfaz gráfica tiene una serie de controles que se explican a continuación:

- **Parameters:** Este botón abrirá una nueva ventana que nos permitirá cambiar la conjetura inicial, así como visualizar de una forma más precisa los resultados de la optimización.

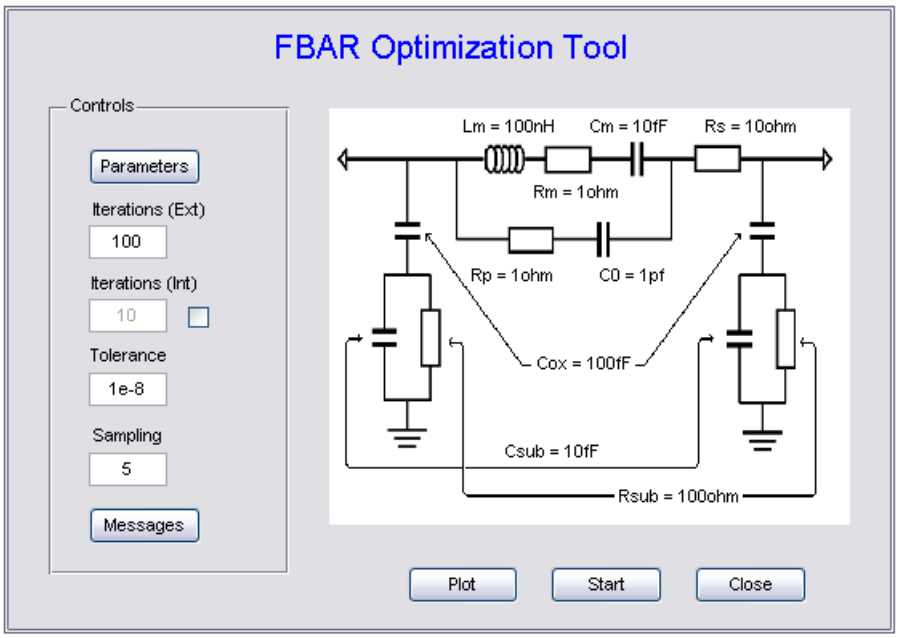

Figura 4.2: *Ventana principal del entorno gráfico.*

- **Messages:** Mediante este botón, accedemos a una ventana donde se puede indicar el tipo de mensajes que queremos mostrar por la pantalla principal de Matlab.

- **Plot:** Este botón muestra las gráficas de los parámetros S_{11} y S_{21} en módulo. Mediante unas casillas seleccionables, podemos agregar a la gráfica la conjetura inicial, las medidas reales y la extracción teórica. Esta ventana podemos verla en 4.3.

- **Start:** Este es el botón encargado de empezar el algoritmo de simulación una vez el usuario ha seleccionado todos los parámetros deseados.

- **Iterations (Ext):** Texto editable para el número de iteraciones del bucle externo del programa.

- **Iterations (Int):** Texto editable, por defecto desactivado, para el número de iteraciones de los bucles internos del algoritmo.

- **Tolerante:** Tolerancia a alcanzar por el algoritmo.

- **Sampling:** Factor que nos indica cada cuantos puntos del vector de medidas, debemos tomar una muestra, por defecto 1 muestra de cada 5 puntos.

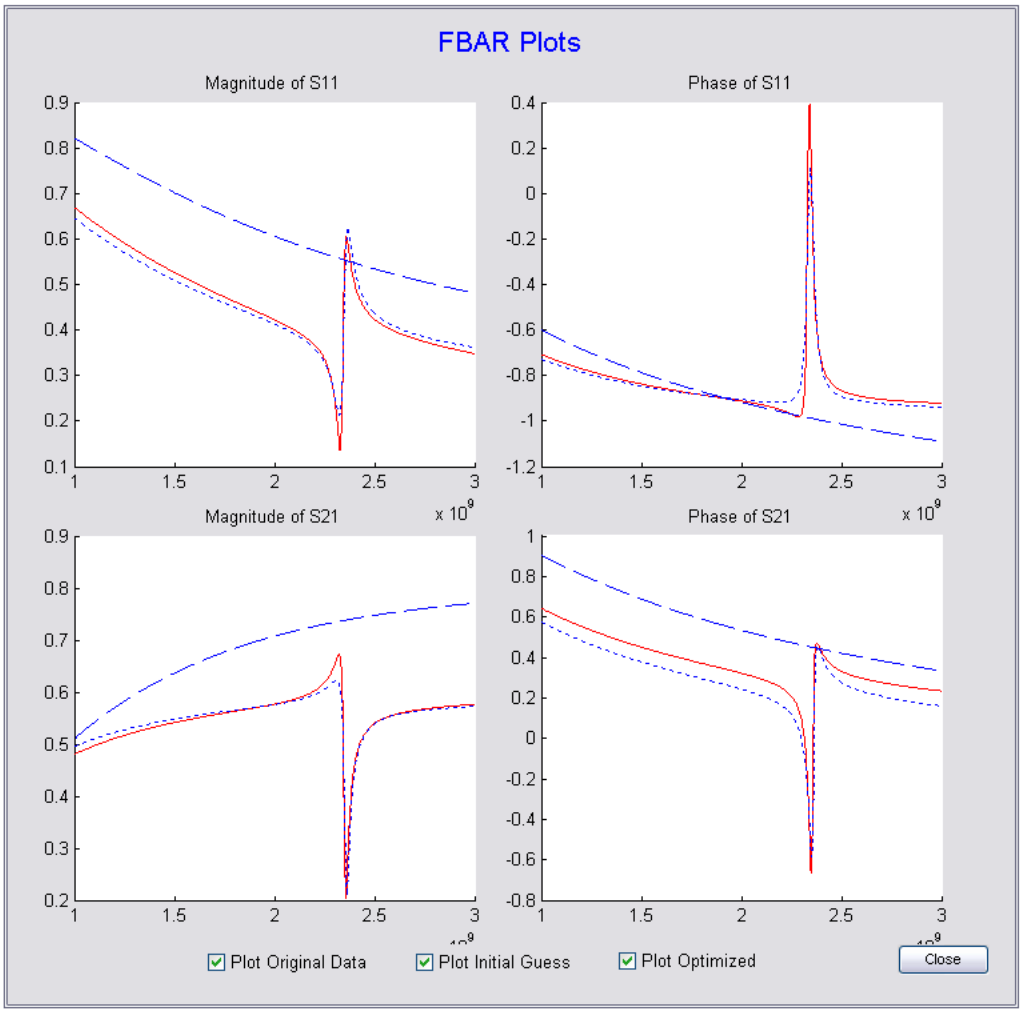

Figura 4.3: *Ventana de gráficas: en azul discontinuo conjetura inicial, en rojo medida real y en azul punteado aproximación actual (5 iteraciones).*

4.6. Pruebas y resultados

En este apartado, se presentan las diferentes pruebas realizadas con el fin de mostrar los resultados obtenidos mediante el algoritmo. Cada ejemplo de la ejecución del algoritmo está acompañada de gráficas, tanto de los datos como del resultado de la optimización, como de una tabla con los parámetros extraídos.

4.6.1. Ejemplo de optimización

Para este primer ejemplo se han generado unas medidas teóricas de los parámetros S mediante un modelo de prueba. En la figura 4.4 (a) y (b) podemos apreciar los parámetros S gráficos. Los parámetros utilizados para generar el modelo se encuentran en la tabla 4.2. Una vez creado el modelo que utilizaremos como prueba para nuestras optimizaciones, lo introducimos en el algoritmo.

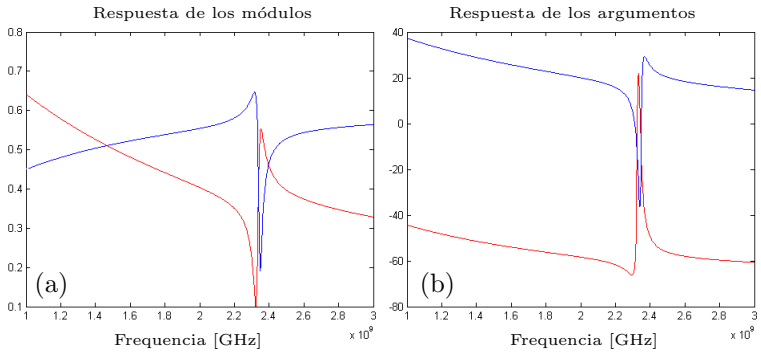

Figura 4.4: *Medidas graficadas de un resonador FBAR teórico: (a) Módulos de S_{11} (rojo) y S_{21} (azul). (b) Argumentos de S_{11} (rojo) y S_{21} (azul).*

Después de indicar al programa cuales son los datos con los que deberá hacer la optimización, debemos proporcionarle una conjetura inicial o utilizar la conjetura de base (por ordenes de magnitud). Para la optimización del primer ejemplo, utilizamos la conjetura base que podemos ver en la tabla 4.2.

Parámetro	R_{sub}	C_{sub}	C_{ox}	R_S	R_P	C_0	L_m	R_m	C_m
Parámetros del modelo	275Ω	40 fF	373 fF	27,9Ω	9,1Ω	1,21 pF	138 nH	2,37Ω	34 pF
Conjetura Inicial	100Ω	10 fF	100 fF	10Ω	1Ω	1 pF	100 nH	1Ω	1 pF

Tabla 4.2: *Parámetros usados para generar el modelo teórico y conjetura inicial para la optimización de las medidas.*

Una vez hemos introducido tanto las medidas como la conjetura inicial, indicamos al algoritmo el número de iteraciones que queremos ejecutar. En este caso se harán tres ejecuciones con 5, 10 y 100 iteraciones. También le indicamos al programa una tolerancia de uno entre mil (10^{-3}). El bucle interno lo dejamos en 10 iteraciones y el factor de muestreado en 5, indicando que tan solo usaremos una de cada cinco muestras proporcionadas. Podemos ver los resultados de los valores de los parámetros, así como el tiempo de computo en la tabla 4.3 y un detalle de la zona de resonancia del módulo de S_{11} en 4.5.

Iter.	T_C	R_{sub}	C_{sub}	C_{ox}	R_S	R_P	C_0	L_m	R_m	C_m
5	0,57s	237Ω	99,5 fF	388 fF	36,1Ω	4,2Ω	1,3 pF	102 nH	5,8Ω	45,9 pF
10	1,71s	262Ω	67,4 fF	383 fF	32,1Ω	6,6Ω	1,2 pF	126 nH	3,7Ω	37,4 pF
30	3,46s	274Ω	40,8 fF	373 fF	28Ω	9,0Ω	1,2 pF	138 nH	2,4Ω	34,1 pF

Tabla 4.3: *Resultados de la aplicación del algoritmo con 5, 10 y 100 iteraciones. Donde T_C es el tiempo de computo del algoritmo.*

Aunque una de las optimizaciones la hicimos con 100 iteraciones, tan solo fueron necesarias 30 para llegar a la tolerancia especificada. Si disminuyéramos dicha tolerancia, los parámetros se acercarían cada vez más a los valores reales de las medidas. De esta forma especificando una tolerancia muy pequeña, de cien entre

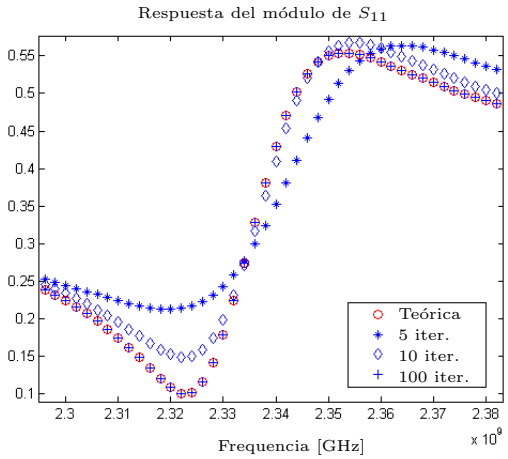

Figura 4.5: *Respuesta graficada de la zona de resonancia de la optimización con diferentes iteraciones.*

un millón (10^{-8}), alcanzamos la tolerancia en la iteración 89 utilizando un tiempo de computo de 9,64s. En esta ejecución, encontramos los valores de parámetros mostrados en la tabla 4.4, que como vemos son los que utilizamos para generar el modelo de prueba.

Parámetro	R_{sub}	C_{sub}	C_{ox}	R_S	R_P	C_0	L_m	R_m	C_m
Valor	275Ω	40 fF	373 fF	27,9Ω	9,1Ω	1,21 pF	138 nH	2,37Ω	34 pF

Tabla 4.4: *Resultados de la aplicación del algoritmo con una tolerancia de* 10^{-8} *y 100 iteraciones.*

Si hiciéramos un zoom de la figura 4.5 y añadieramos más graficas correspondientes a otras iteraciones del algoritmo, podríamos observar como el modelo obtenido se ajustaría cada vez más a la medida original. La figura 4.6 muestra dicho comportamiento con el pico de resonancia serie y diferentes iteraciones del módulo del parámetro S_{21}.

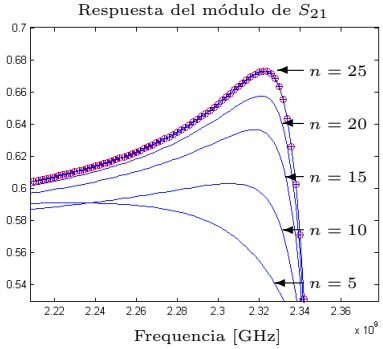

Figura 4.6: *pico de resonancia del módulo de* S_{21} *con diferentes iteraciones (azul) y medida real (rojo).*

4.6.2. Ejemplo de optimización con ruido

Para este segundo ejemplo, se ha añadido un ruido gaussiano blanco al modelo teórico del apartado anterior, con el fin de comprobar el comportamiento del algoritmo.

La adición de este ruido, puede ser interpretado de dos formas. En el primer caso podríamos suponer que el ruido ha sido añadido por efectos externos al resonador FBAR, mientras el segundo caso supondría que esta señal no contiene ruido, sino que esta forma parte de la respuesta del resonador.

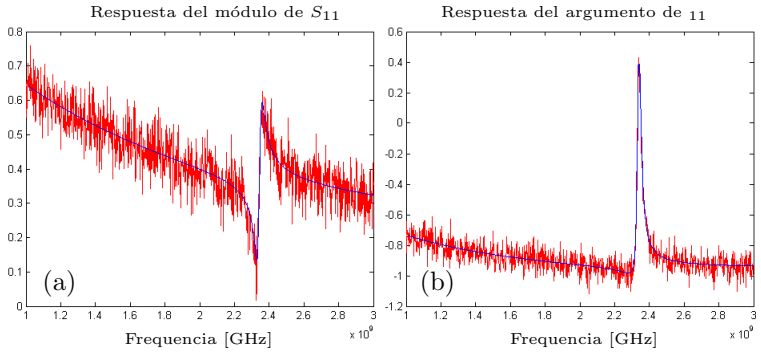

Figura 4.7: *Resultado de la optimización con ruido gaussiano blanco: (a) Módulo de la medida de S_{11} (rojo) y optimización (azul). (b) Argumento de la medida de S_{11} (rojo) y optimización (azul).*

Tal y como se observa en la figura 4.7, el algoritmo descarta una gran cantidad de ruido y aproxima la medida al modelo teórico de MBVD del resonador FBAR. La conclusión que obtenemos en los casos descritos es diferente. En el primer caso, donde consideramos un ruido añadido, obtenemos una buena aproximación del comportamiento del resonador ideal, ya que el algoritmo omite la mayor parte del ruido. En el segundo caso, se concluye que el modelo que estamos utilizando no ajusta correctamente los datos y deberíamos proceder a cambiar el modelo de forma conveniente para ajustar mejor las medidas.

En la tabla 4.5 se muestra el resultado de los parámetros obtenidos en la optimización.

Parámetro	R_{sub}	C_{sub}	C_{ox}	R_S	R_P	C_0	L_m	R_m	C_m
Valor	257Ω	24,9 fF	465 fF	31,5Ω	6,8Ω	1,23 pF	137 nH	2,52Ω	34,2 pF

Tabla 4.5: *Parámetros obtenidos de la optimación de las medidas con ruido gaussiano blanco.*

Es necesario comentar, que el ruido introducido en este ejemplo es desproporcionado con respecto al modelo, pero se ha procedido de esta manera para comprobar el potencial del algoritmo.

En el siguiente ejemplo, añadimos a la medida un rizado senoidal con un período grande que nos permitirá comprobar como se comporta el algoritmo en presencia de una perturbación con variaciones lentas y de gran amplitud.

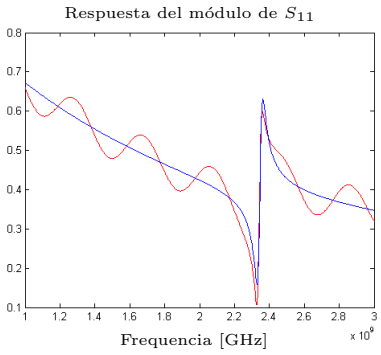

Figura 4.8: *Resultado de la optimización con ruido senoidal: Módulo de la medida de S_{11} (rojo) y optimización (azul).*

Tal y como observamos en la figura 4.8, podemos concluir lo mismo que en el caso del ruido gaussiano blanco, si la medida que hemos proporcionada contiene errores de medida o ruido, este es suavizado. No obstante, si esta respuesta es la del resonador, deberemos añadir algún tipo de elemento al modelo para mejorar el ajuste de los datos. Los parámetros obtenidos cos esta optimización se muestran en al tabla 4.6.

Parámetro	R_{sub}	C_{sub}	C_{ox}	R_S	R_P	C_0	L_m	R_m	C_m
Valor	304Ω	30 fF	413 fF	29,6Ω	4,6Ω	1,16 pF	148 nH	5,3Ω	31,6 pF

Tabla 4.6: *Parámetros obtenidos de la optimación de las medidas con ruido senoidal.*

4.6.3. Límite de la conjetura inicial

Aunque el núcleo del algoritmo, formado por la función *gn_method*, tiene una tolerancia enorme con respecto a la conjetura inicial que podeos introducirle, es capaz de encontrar la optimización con conjeturas muy alejadas del valor real, no es así el rango conseguido mediante el algoritmo entero. La gran cantidad de parámetros así como la secuencialización realizada, hace descender notablemente el rango de conjeturas iniciales que podemos introducir. Aproximadamente con el algoritmo actual, se puede introducir en él una conjetura que puede comprender entre el 12 % y el 160 % del valor del parámetro. Estos valores, dependen del tipo de medida que tengamos, del tipo de secuencia que hayamos utilizado y se debe tener en cuenta que no será igual para todos los parámetros. No obstante, el rango es suficiente como para englobar todo un subconjunto de resonadores diferentes con parámetros que tengan el mismo orden de magnitud.

4.7. Conclusiones

En este capítulo hemos visto la estructura del código implementado para la realización de la optimización mediante mínimos cuadrados. La primera parte del capítulo se ha usado para explicar las funciones que constituyen el algoritmo, así como sus particularidades.

Se ha visto que la estrategia del algoritmo generado cuenta con dos bucles, el interno que se encarga de aplicar el método de Gauss-Newton para un conjunto pequeño de parámetros (uno o dos por vez en nuestro caso), y el bucle externo que se hace cargo de secuenciar las diferentes optimizaciones para facilitar la optimización mediante el algoritmo.

Dos funciones necesarias para el algoritmo, son la función que evalúa el modelo en los parámetros específicos y la función que calcula el Jacobiano del modelo. La función del modelo esta construida mediante matrices ABCD que facilitan el cálculo y el cambió de parámetros en el modelo teórico. El Jacobiano ha tenido que implementarse para poderlo ejecutar sobre datos en vez de sobre expresiones analíticas, por ello se ha utilizado una estrategia de diferencias finitas para su computo.

La segunda parte del capítulo hace referencia a la interfaz gráfica de usuario desarrollada para la aplicación. Una interfaz sencilla de utilizar que permite elaborar y visualizar las optimizaciones de una forma rápida y eficaz mediante el uso de ventanas y controles por ratón.

Finalmente en la última parte del capítulo se han visto varios ejemplos de utilización del algoritmo. Para poder elaborar los resultados, se han generado varios ejemplos de resonadores FBAR teóricos con los que se ha trabajado para poder obtener toda una serie de pruebas y resultados a lo largo del desarrollo del programa.

En primera instancia se presenta la ejecución del algoritmo directamente sobre las medidas teóricas obtenidas del modelo. En este ejemplo se ha demostrado el funcionamiento básico del algoritmo y como con un modelo que ajuste perfectamente los datos, el algoritmo se comporta de una forma muy eficiente extrayendo los parámetros esperados con una baja tolerancia.

En los ejemplos siguientes, se ha añadido diferentes tipos de ruido a las medidas con el propósito de comprobar el comportamiento del algoritmo. De estos ejemplos se ha deducido que si se introducen en el algoritmo medidas con ruido, este las puede tolerar y elaborar una aproximación de los parámetros. No obstante, también hemos comprado que si interpretamos el ruido añadido como parte de la respuesta real del resonador, la optimización no representará correctamente los datos y por lo tanto se deberá proceder a cambiar el modelo.

En algunos ejemplos, hemos podido ver el tiempo de computo del algoritmo para la extracción de los parámetros. Como se ha podido comprobar dicho tiempo es del orden de $8 - 20$ segundos con 100 iteraciones externas, 10 internas y con un factor de muestreado de 5. Este tiempo aumenta a medida que le indicamos al programa que haga más iteraciones, tanto internas como externas, así como con el número de puntos que utiliza para hacer los cálculos.

Capítulo 5

Conclusiones y líneas futuras

5.1. Conclusiones

En este trabajo se ha presentado un algoritmo de optimización capaz de extraer los parámetros de un resonador FBAR a través del modelo teórico mediante un conjunto de medidas. Estos dispositivos, están siendo estudiados debido a las grandes prestaciones que presentan si los comparamos con los dispositivos que actualmente se están utilizando. Mientras la tecnología FBAR es capaz de proporcionar factores de calidad de hasta 67000, (factores de calidad tan buenos como los obtenidos con resonadores de cristal de cuarzo) otras tecnologías tan solo son capaces de conseguir factores de calidad de alrededor de 400, como es el caso de la tecnología SAW. Otra de las prestaciones que hacen de la tecnología FBAR una buena aspirante a dominar el mercado de la tecnología inalámbrica es la capacidad de operar en rangos de frecuencia comprendidos entre los 600MHz y los 12GHz, muy por encima de los rangos de otras tecnologías como la de cristal de cuarzo (hasta 250MHz) o la tecnología SAW (hasta 2-2.5GHz). Otros puntos a favor de la tecnología FBAR, son su alto grado de miniaturización y la posibilidad de implementarse en substratos convencionales tales como el silicio.

En el segundo capítulo de este trabajo hemos visto la teoría necesaria para entender el principio de funcionamiento de un resonador FBAR, así como su estructura y su modelado.

Los resonadores FBAR basan su funcionamiento en la propagación de una onda mecánica por un material piezoeléctrico, donde esta es transformada al dominio eléctrico mediante una transducción, conocida como el efecto piezoeléctrico inverso.

La estructura general de un resonador FBAR, consiste en una delgada lámina de material piezoeléctrico situada entre dos electrodos. Para confinar la onda acústica dentro de la estructura, esta deberá estar aislada del entorno con una interfície electrodo-aire o mediante un espejo de impedancias.

La teoría más importante para la elaboración del proyecto, es el modelado de los resonadores. Utilizado para describir el comportamiento del los resonadores. Se han visto los tres modelos más conocidos. El primer modelo que se ha estudiado es el modelo de Mason que nos permite encontrar un circuito equivalente eléctrico con componentes distribuidos a partir de las ecuaciones de onda acústica. Debido a las dificultades que presenta dicho modelo a la hora de simularlo, se procede al estudio de un modelo más sencillo denominado modelo de Butterworth Van-Dyke. Este modelo fue desarrollado para estudiar el comportamiento de los resonadores de cristal de cuarzo, pero se ha visto que también modela el comportamiento de otros resonadores y en concreto el de los resonadores FBAR. El circuito equivalente de dicho modelo, esta elaborado mediante elementos concentrados y es muy sencillo de simular con las herramientas que tenemos.

Para estudiar con detalle el comportamiento de un resonador más real, se introducen en el modelo de BVD, una serie de elementos que permiten estudiar los efectos parásitos y las perdidas que tienen los resonadores implementados físicamente. La adición de estos elementos da lugar al denominado modelo de Butterworth Van-Dyke Modificado. Este es el modelo que se ha utilizado para la elaboración del algoritmo que tiene como objetivo este proyecto.

Una vez estudiado el resonador FBAR, se procede al estudio del algoritmo que ejecutará la optimización. De este modo, en el tercer capítulo se introduce la teoría necesaria para su comprensión.

De la teoría de programación lineal y no lineal, y observando el modelo teórico MBVD se desprende que el algoritmo que deberemos utilizar será un algoritmo no-lineal y por ello se decidió utilizar un algoritmo basado en el método de los mínimos cuadrados para el ajuste de los datos.

El algoritmo que utilizamos para desempeñar la optimización por mínimos cuadrados, esta basado en el método de Gauss-Newton. Este método, que a su vez esta basado en el método de Newton, basa su funcionamiento en la aproximación de la serie de Taylor del modelo mediante el uso de matrices de derivadas parciales tales como el Jacobiano y el Hessiano.

Con la teoría vista en el segundo y tercer capítulo, en el cuarto se procede a la explicación de la estructura del algoritmo generado. Primero se explican las funciones más importantes creadas, tales como por ejemplo la función que evalúa el modelo según los parámetros o la función del núcleo del propio algoritmo que ejecuta la optimización mediante el modelo de Gauss-Newton.

Adicionalmente a la generación del código, se ha creado también una interfaz gráfica que pretende elaborar y visualizar las optimizaciones de una forma rápida y sencilla mediante el uso de ventanas y controles por ratón.

Por último se han presentado las pruebas y resultados que se han llevado a cabo para comprobar el correcto funcionamiento del algoritmo. Las pruebas realizadas permiten comprobar la robustez del algoritmo, así como su eficiencia y rapidez. Se ha comprobado que si el modelo teórico ajusta a la perfección los datos, es posible conseguir una tolerancia muy buena en pocas iteraciones y en apenas unos segundos. Por el contrario, si el modelo no ajusta bien los resultados del resonador o bien le pasamos una conjetura inicial poco acertada, el algoritmo no convergirá a una solución o si lo hace, el resultado no será valido.

5.2. Líneas futuras

Debido a la gran cantidad de algoritmos diferentes que presenta el campo de la optimización de datos, sería posible buscar algoritmos diferentes o ampliar el actual con el fin de mejorar aún más las prestaciones que hemos obtenido.

Algunos de estos algoritmos podrían utilizar estrategias más avanzadas, como algoritmos basados en código genético o en redes neuronales, o bien la utilización de mejoras para el método de Gauss-Newton como es el caso del método de Levenberg-Marquardt donde hay una mejora de la aproximación del Hessiano, o la incorporación de direcciones de búsqueda de línea.

Por último, aunque hemos obtenido la respuesta esperada con las optimizaciones teóricas realizadas, todavía debería aplicarse el algoritmo de extracción a las medidas experimentales de los parámetros S que se llevarán a cabo en el laboratorio del CNM.

Apéndice A

Resumen

El objetivo de este proyecto consiste en la elaboración de un algoritmo de optimización que permita, mediante un ajuste de datos por mínimos cuadrados, la extracción de los parámetros del circuito equivalente que componen el modelo teórico de un resonador FBAR, a partir de las medidas de los parámetros S.

Para llevar a cabo dicho trabajo, se desarrolla en primer lugar toda la teoría necesaria de resonadores FBAR. Empezando por el funcionamiento y la estructura, y mostrando especial interés en el modelado de dichos resonadores mediante los modelos de Mason, Butterworth Van-Dyke y BVD Modificado. En segundo término, se estudia la teoría sobre optimización y programación No-Lineal.

Una vez se ha expuesto la teoría, se procede con la descripción del algoritmo implementado. Dicho algoritmo utiliza una estrategia de múltiples pasos que agiliza la extracción de los parámetros del resonador.

L'objectiu d'aquest projecte consistent en l'elaboració d'un algoritme d'optimització que permeti, mitjançant un ajust de dades per mínims quadrats, la extracció dels paràmetres del circuit equivalennt que composen el model teòric d'un ressonador FBAR, a partir de les mesures dels paràmetres S.

Per a dur a terme aquest treball, es desenvolupa en primer lloc tota la teoria necessària de ressonadors FBAR. Començant pel funcionament i l'estructura, i mostrant especial interès en el modelat d'aquests ressonadors mitjançant els models de Mason, Butterworth Van-Dyke y BVD Modificat. En segon terme, s'estudia la teoria sobre optimització i programació No-Lineal.

Un cop s'ha exposat la teoria, es procedeix amb la descripció de l'algoritme implementat. Aquest algoritme utilitza una estratègia de múltiples passos que agilitzen la extracció dels paràmetres del ressonador.

The objective of this project consists in the elaboration of an optimization algorithm that would extract the equivalent-circuit parameters that compound the model of an FBAR resonator from the measurement of its scattering-parameters.

In order to do that work, the necessary FBAR resonator theory is developed. First of all, we study the operation and the structure of the resonators putting special interest in the resonator modeling, by using the Mason model, the Butterworth Van-Dyke model and the Modified BVD model. In second term, we study de theory of optimization and Non-Linear programming.

After exposing the theory, we proceed with the description of the implemented algorithm. This algorithm uses a multiple-step strategy to make more agile the extraction of the resonator parameters.

Apéndice B

Código

B.1. Funciones del Algoritmo

B.1.1. Modelo de MBVD (*model*)

```
function [ fdata ] = model( xdata, param , S_PARAM, PROP)
%MODEL(X,param,S_PARAM,PROP) Retorna un vector amb el model numèric del
%   paràmetre S_PARAM del model Butterworth Van-Dyke Modificat d'un
%   resonador FBAR segons PROP evaluat amb els paràmetres del vector
param
%   i en funció de la variable independent X.
%
%   X :         Vector de variables independents del model (freqüència).
%   S_PARAM :   Cadena de caràcters que indica quin paràmetre volem que
%               retorni la funció: 'S11' o 'S21'.
%   PROP :      Cadena de caràcters que indica que volem retornar si la
%               magnitud o la fase.
%   param :     Conjunt de paràmetres del model MBVD,
%
%      Paràmetres:
%          Rsub =  param(1)
%          Csub =  param(2)
%          Cox  =  param(3)
%          Rs   =  param(4)
%          Rp   =  param(5)
%          C0   =  param(6)
%          Lm   =  param(7)
%          Rm   =  param(8)
%          Cm   =  param(9)
%

% Parametre de l'S11 del model de Butterworth Van-Dyke Modificat.
%
%                       --- Lm ---- Rm ---- Cm---
%          --- Rs/2---|                          |--- Rs/2 --- ----------
%         |            ------- Rp ---- C0 ------             |          |
%         |                                                  |         z0
%        Cox                                                Cox         |
%         |                                                  |          v
%        -----                                              -----
%      |     |                                            |     |
%    Csub   Rsub                                        Csub   Rsub
%      |     |                                            |     |
%       -----                                              -----
%         |                                                  |
%         v                                                  v
%
%    \___ A ___/                                      2  \___ A ___/
%            \_____________________ B ____________________________/
%                                                       \______ C ______/


% Impedància característica:
z0=50;

% Converitm f a w.
xdata = xdata*2*pi*i;
```

```
A =
(param(1)*1/param(2)./xdata)./(param(1)+1/param(2)./xdata)+1/param(3)./xd
ata;
B =
((param(7)*xdata+param(8)+1/param(9)./xdata).*(param(5)+1/param(6)./xdata
))./(param(7)*xdata+param(8)+1/param(9)./xdata+param(5)+1/param(6)./xdata
);

mat_A_A=ones(length(A),1);
mat_A_B=zeros(length(A),1);
mat_A_C=1./A;
mat_A_D=ones(length(A),1);
mat_B_A=ones(length(B),1);
mat_B_B=B;
mat_B_C=zeros(length(B),1);
mat_B_D=ones(length(B),1);
mat_AB_A=mat_A_A.*mat_B_A+mat_A_B.*mat_B_C;
mat_AB_B=mat_A_A.*mat_B_B+mat_A_B.*mat_B_D;
mat_AB_C=mat_A_C.*mat_B_A+mat_A_D.*mat_B_C;
mat_AB_D=mat_A_C.*mat_B_B+mat_A_D.*mat_B_D;
out_A=mat_AB_A.*mat_A_A+mat_AB_B.*mat_A_C;
out_B=mat_AB_A.*mat_A_B+mat_AB_B.*mat_A_D;
out_C=mat_AB_C.*mat_A_A+mat_AB_D.*mat_A_C;
out_D=mat_AB_C.*mat_A_B+mat_AB_D.*mat_A_D;

% Mirem que hem de retornar:
%   S11,MAG : Magnitud del paràmentre S11.
%   S11,PHASE : Fase del Paràmentre S11.
%   S21,MAG : Magnitud del paràmentre S21.
%   S21,PHASE : Fase del Paràmentre S21.

switch lower(S_PARAM)
    case 's11'
        E=(out_A+out_B/z0-out_C*z0-
out_D)./(out_A+out_B/z0+out_C*z0+out_D);
    case 's21'
        E = 2./(out_A+out_B/z0+out_C*z0+out_D);
    otherwise
        disp('Tipus de paràmetre S passat incorrecte. Utilitzi S11 o
S21).');
        fdata = 0;
end
switch lower(PROP)
    case 'mag'
        fdata = abs(E);
    case 'phase'
        fdata = angle(E);
    otherwise
        disp('Propietat incorrecte. Utilitzi MAG o PHASE');
        fdata = 0;
end
```

B.1.2. Jacobiano de diferencias finitas (*JacobFD*)

```
function [ J ] = jacobFD( xdata, param, paramOPT, S_PARAM, PROP )

param0=param;

for i=1:length( paramOPT )
    param=param0;
    diferencial=param(paramOPT(i))*sqrt(eps);
    param(paramOPT(i)) = param(paramOPT(i))+diferencial;
    ydata1 = model( xdata, param0, S_PARAM, PROP );
    ydata2 = model( xdata, param, S_PARAM, PROP );
    J(:,i) = (ydata2 - ydata1)/diferencial;
end
```

B.1.3. Función de primera aproximación de L_m y C_m (LC_seek)

```
function [ param ] = LC_seek ( xdata, ydata, C0)
%LC_seek(x,y,C0)
%   Extracció de d'una primera aproximació dels paràmentres Lm i Cm
%
% Retorna:
%   param(1) : Lm
%   param(2) : Cm
%
f = fsfp(xdata,ydata);
param = zeros(1,2);
param(2) = C0*((f(2)/f(1))^2-1);
param(1) = 1/((2*pi*f(1))^2*param(2));
```

B.1.4. Búsqueda de las frecuencias serie y paralelo (*fsfp*)

```
function [ zero ] = fsfp ( xdata, ydata )

% Extracció de les freqüències de resonancia:
%
%   f(1) : Freqüència de resonancia serie.
%   f(2) : Freqüència de resonancia paral·lel.
%
i=1;
ydata_temp = sign(ydata);
for N=1:(length(ydata)-1)
    if(ydata_temp(N)+ydata_temp(N+1)==0)
        m = (ydata(N+1)-ydata(N))/(xdata(N+1)-xdata(N));
        zero(i) = -(ydata(N)/m)+xdata(N);
        i=i+1;
     end
end
```

B.1.5. Núcleo del algoritmo (*gn_method*)

```
function [ param ] = gn_method( xdata, ydata, param0, paramOPT, S_PARAM,
PROP, max_iter, tol, sampling_factor, msg_control, varargin)
%GN_METHOD(X,Y,param,paramOPT,S_PARAM,PROP,max_iter,tol)
%
%   X :         Vector de les dades en abcisa.
%   Y :         Vector de les dades en ordenada.
%   param :     Vector de parametres coneguts.
%   paramOPT :  Vector dels index de conjectures inicials a optimitzar.
%   S_PARAM :   Cadena de caràcters que indica el paràmetre S amb el que
%               aplicar la optimització ('S11','S21').
%   PROP :      Cadena de caràcters que indica si volem optimitzar
magnitud
%               o fase del paràmetre S_PARAM ('MAG','PHASE').
%   max_iter:   Màxim numero d'iteracions del algoritme Gauss-Newton.
%   tol :       Màxima tolerància.
%
%GN_METHOD(X,Y,param,paramOPT,S_PARAM,PROP,max_iter,tol,step_modifier)
%
%   step_modifier :     Modificador del pas de l'algoritme (Opcional).
%
%   gn_method trova mitjançant el mètode de Gauss-Newton i els mínims
%   quadrats un vector de paràmetres que satisfagi els parells de dades
%   xdata i ydata mitjançant el model dintre de la funció "model".

%Inicialitzem el missatge de sortida (msg);
msg = 0;

samples=[1:sampling_factor:length(xdata)];
%samples = [1:1:length(xdata)];

%Ens assegurem que estiguin en columna;
xdata = xdata(:);
ydata = ydata(:);
xdata_few = xdata(samples);
ydata_few = ydata(samples);

% Inicialitzem el pas d'optimització i mirem si l'usuari n'ha
proporcionat
% algun, en cas afirmatiu modifiquem el pas d'optimització;
step_modifier = 1;
if (nargin == 11)
    step_modifier = varargin{1};
end

%Eliminació dels warnings per matrius singulars:
if (~msg_control(4))
    warningstate1 = warning('off', 'MATLAB:nearlySingularMatrix');
    warningstate2 = warning('off', 'MATLAB:singularMatrix');
else
    warningstate1 = warning('on', 'MATLAB:nearlySingularMatrix');
    warningstate2 = warning('on', 'MATLAB:singularMatrix');
end
param=param0;

for N=1:max_iter
    f = model(xdata_few,param,S_PARAM,PROP);
```

```
    F = f-ydata_few;
    max_actual = abs(max(F));
    if(isnan(max_actual))
        msg = sprintf('\nLactual funció no convergeix amb els valors
assignats.');
        disp(msg);
        param = 0;
        return;
        break;
    end
    if(max_actual<tol)
        msg = sprintf('\nHem arribat a la tolerància demanada en %d
iteracions.',N);
        if (msg_control(2))
            disp(msg);
            break;
        end
    end
    JacobF = jacobFD(xdata_few,param,paramOPT,S_PARAM,PROP);
    p = -((JacobF.'*JacobF)\(JacobF'*F))/step_modifier;
    param(paramOPT)=param((paramOPT))+p.';
    f = (model(xdata_few,param,S_PARAM,PROP));
end
if (msg_control(3))
    if (~msg)
        msg=sprintf('\nHem arribat al màxim de iteracions sense assolir
la tolerància.');
        disp(msg);
    end
end
```

B.1.6. Algoritmo multipaso (*FBAR_optim*)

```
function [param] = FBAR_optim( Sdata, param0, num_iter_ext, num_iter_int,
tol_int, sampling_factor, msg_control)
%FBAR_optim     Extracció de paràmetres d'un FBAR
%
%   Sdata:
%
%       Sdata(:,1) = f
%       Sdata(:,2) = MAG(S11)
%       Sdata(:,3) = PHASE(S11)
%       Sdata(:,4) = MAG(S21)
%       Sdata(:,5) = PHASE(S21)
%
%   Parametres:
%
%       Rsub =  param(1)
%       Csub =  param(2)
%       Cox  =  param(3)
%       Rs   =  param(4)
%       Rp   =  param(5)
%       C0   =  param(6)
%       Lm   =  param(7)
%       Rm   =  param(8)
```

```
%       Cm    =  param(9)
%

% Vector de conjectura inicial segons ordre de magnitud
%param0=[100, 10e-15, 100e-15, 10, 1, 1e-12, 100e-9, 1, 10e-15];
if(msg_control(5))
    tic;
end
for N=1:num_iter_ext
%    tic;
    if (msg_control(1)==1)
        msg=sprintf('Iteració número %d',N);
        disp(msg);
    end
    if (N==1)

param=gn_method(Sdata(:,1),Sdata(:,2),param0,[6],'S11','MAG',num_iter_int
,tol_int, sampling_factor,msg_control,10);
        if (~param)
            return;
        end
        LmCm=LC_seek(Sdata(:,1),Sdata(:,3),param(6));
        param([7 9])=LmCm;
    else

param=gn_method(Sdata(:,1),Sdata(:,2),param,[6],'S11','MAG',num_iter_int,
tol_int, sampling_factor,msg_control,10);
        if (~param)
            return;
        end
    end
    param=gn_method(Sdata(:,1),Sdata(:,3),param,[7
9],'S11','PHASE',num_iter_int,tol_int, sampling_factor,msg_control,10);
    if (~param)
        return;
    end

param=gn_method(Sdata(:,1),Sdata(:,2),param,[3],'S11','MAG',num_iter_int,
tol_int, sampling_factor,msg_control);
    if (~param)
        return;
    end

param=gn_method(Sdata(:,1),Sdata(:,3),param,[4],'S11','PHASE',num_iter_in
t,tol_int, sampling_factor,msg_control,10);
    if (~param)
        return;
    end

param=gn_method(Sdata(:,1),Sdata(:,2),param,[5],'S11','MAG',num_iter_int,
tol_int, sampling_factor,msg_control,10);
    if (~param)
        return;
    end

param=gn_method(Sdata(:,1),Sdata(:,2),param,[8],'S11','MAG',num_iter_int,
tol_int, sampling_factor,msg_control,10);
```

```
    if (~param)
        return;
    end

param=gn_method(Sdata(:,1),Sdata(:,4),param,[3],'S21','MAG',num_iter_int,
tol_int, sampling_factor,msg_control,10);
    if (~param)
        return;
    end

param=gn_method(Sdata(:,1),Sdata(:,4),param,[1],'S21','MAG',num_iter_int,
tol_int, sampling_factor,msg_control,10);
    if (~param)
        return;
    end

param=gn_method(Sdata(:,1),Sdata(:,2),param,[2],'S11','MAG',num_iter_int,
tol_int, sampling_factor,msg_control,10);
    if (~param)
        return;
    end

% Comprovació de l'error actual.
        error(1)=sum(Sdata(:,2)-model(Sdata(:,1),param,'S11','MAG'));
        error(2)=sum(Sdata(:,3)-model(Sdata(:,1),param,'S11','PHASE'));
        error(3)=sum(Sdata(:,4)-model(Sdata(:,1),param,'S21','MAG'));
        error(4)=sum(Sdata(:,5)-model(Sdata(:,1),param,'S21','PHASE'));

    if (error<tol_int)
        disp('Hem assolit la tolerància');
        break
    end
end


if(msg_control(5))
    toc;
end
```

B.2. Funciones de la interfaz gráfica

B.2.1. Ventana principal (*FBAR_GUI*)

```
function varargout = FBAR_GUI(varargin)
%FBAR_GUI M-file for FBAR_GUI.fig
%      FBAR_GUI, by itself, creates a new FBAR_GUI or raises the existing
%      singleton*.
%
%      H = FBAR_GUI returns the handle to a new FBAR_GUI or the handle to
%      the existing singleton*.
%
%      FBAR_GUI('Property','Value',...) creates a new FBAR_GUI using the
%      given property value pairs. Unrecognized properties are passed via
%      varargin to FBAR_GUI_OpeningFcn.  This calling syntax produces a
%      warning when there is an existing singleton*.
%
%      FBAR_GUI('CALLBACK') and FBAR_GUI('CALLBACK',hObject,...) call the
%      local function named CALLBACK in FBAR_GUI.M with the given input
%      arguments.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help FBAR_GUI

% Last Modified by GUIDE v2.5 15-May-2007 13:09:12

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @FBAR_GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @FBAR_GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before FBAR_GUI is made visible.
function FBAR_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% varargin   unrecognized PropertyName/PropertyValue pairs from the
%            command line (see VARARGIN)

% Choose default command line output for FBAR_GUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

[BVDM,BVDM_map] = imread('BVDM.bmp');
image(BVDM);
colormap(BVDM_map);
set(gca,'Visible','Off');


handles.msg=[1 0 0 0 1];
handles.param0 = [100, 10e-15, 100e-15, 10, 1, 1e-12, 100e-9, 1, 10e-15];
handles.param = [0 0 0 0 0 0 0 0 0];
handles.Sdata=varargin{1};
guidata(hObject,handles);
format_picture(handles.param0,handles)

% --- Outputs from this function are returned to the command line.
function varargout = FBAR_GUI_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;


% --- Executes on button press in box_enable_iter_int.
function box_enable_iter_int_Callback(hObject, eventdata, handles)

if(get(hObject,'Value'))
    set(handles.edit_iter_int,'Enable','On');
else
    set(handles.edit_iter_int,'Enable','Off');
end

function edit_iter_ext_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_iter_ext_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_iter_int_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_iter_int_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end

function edit_tol_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_tol_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in button_close.
function button_close_Callback(hObject, eventdata, handles)

user_response = exitFBAR('Title','Confirm Close');
switch user_response
case {'No'}
    % Sense acció
case 'Yes'
    % Tanca el programa
    delete(handles.figure1)
end

% --- Executes on button press in button_start.
function button_start_Callback(hObject, eventdata, handles)

iter_ext = get(handles.edit_iter_ext,'String');
iter_int = get(handles.edit_iter_int,'String');
tol = get(handles.edit_tol,'String');
sampling_factor = get(handles.edit_sample,'string');

param = FBAR_optim( handles.Sdata, ...
                    handles.param0, ...
                    str2num(iter_ext), ...
                    str2num(iter_int), ...
                    str2num(tol), ...
                    str2num(sampling_factor), ...
                    handles.msg);
if(~sum(param))
    disp('error');
else
    format_picture(param,handles);
    handles.param=param;
    guidata(hObject,handles);
end

% --- Executes on button press in msg.
function msg_Callback(hObject, eventdata, handles)

handles.msg = msg_GUI(handles.msg);
guidata(hObject,handles);

% --- Executes on button press in button_plot.
```

```
function button_plot_Callback(hObject, eventdata, handles)

plots_GUI(handles.Sdata,handles.param0,handles.param);

function format_picture(param,handles)
param_names = {'Rsub', 'Csub', 'Cox', 'Rs', 'Rp', 'C0', 'Lm', 'Rm',
'Cm'};
param_units = {'ohm', 'fF', 'fF', 'ohm', 'ohm', 'pf', 'nH', 'ohm', 'fF'};
param=param./[1, 1e-15, 1e-15, 1, 1, 1e-12, 1e-9, 1, 1e-15];
equal_sign={' = '};
param_string = strcat(param_names,equal_sign);
for N=1:9
    param_string(N)=strcat(param_string(N),num2str(param(N),3));
end
param_string=strcat(param_string,param_units);

set(handles.Rsub,'string',param_string(1));
set(handles.Csub,'string',param_string(2));
set(handles.Cox,'string',param_string(3));
set(handles.Rs,'string',param_string(4));
set(handles.Rp,'string',param_string(5));
set(handles.C0,'string',param_string(6));
set(handles.Lm,'string',param_string(7));
set(handles.Rm,'string',param_string(8));
set(handles.Cm,'string',param_string(9));

% --- Executes on button press in button_param.
function button_param_Callback(hObject, eventdata, handles)

handles.param0 = param_GUI(handles.param0,handles.param);
guidata(hObject,handles);
if(~sum(handles.param))
    format_picture(handles.param0,handles)
end

function edit_sample_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_sample_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

B.2.2. Elección de mensajes a mostrar (*msg_GUI*)

```
function varargout = msg_GUI(varargin)
% MSG_GUI M-file for msg_GUI.fig
%      MSG_GUI, by itself, creates a new MSG_GUI or raises the existing
%      singleton*.
%
%      H = MSG_GUI returns the handle to a new MSG_GUI or the handle to
%      the existing singleton*.
%
%      MSG_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in MSG_GUI.M with the given input
arguments.
%
%      MSG_GUI('Property','Value',...) creates a new MSG_GUI or raises
the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before msg_GUI_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to msg_GUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help msg_GUI

% Last Modified by GUIDE v2.5 13-May-2007 19:10:50

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @msg_GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @msg_GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before msg_GUI is made visible.
function msg_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
```

```
handles.output=varargin{1};
handles.aux=[0 0 0 0 0];
% Update handles structure
guidata(hObject, handles);

if (sum(handles.output))
    if (handles.output(1))
        set(handles.check_01,'Value',1);
    end
    if (handles.output(2))
        set(handles.check_02,'Value',1);
    end
    if (handles.output(3))
        set(handles.check_03,'Value',1);
    end
    if (handles.output(4))
        set(handles.check_04,'Value',1);
    end
    if (handles.output(5))
        set(handles.check_05,'Value',1);
    end
    set(handles.check_enable,'Value',0);
else
    set(handles.check_enable,'Value',1);
    set(handles.check_01,'Enable','Off');
    set(handles.check_02,'Enable','Off');
    set(handles.check_03,'Enable','Off');
    set(handles.check_04,'Enable','Off');
    set(handles.check_05,'Enable','Off');
end

% UIWAIT makes msg_GUI wait for user response (see UIRESUME)
set(handles.figure1,'WindowStyle','modal')
uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = msg_GUI_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;
delete(handles.figure1);


% --- Executes on button press in check_enable.
function check_enable_Callback(hObject, eventdata, handles)

if(get(hObject,'Value'))
    set(handles.check_01,'Enable','Off');
    set(handles.check_02,'Enable','Off');
    set(handles.check_03,'Enable','Off');
    set(handles.check_04,'Enable','Off');
    set(handles.check_05,'Enable','Off');
    handles.aux=handles.output;
    handles.output(:)=0;
else
```

```
    set(handles.check_01,'Enable','On');
    set(handles.check_02,'Enable','On');
    set(handles.check_03,'Enable','On');
    set(handles.check_04,'Enable','On');
    set(handles.check_05,'Enable','On');
    if (sum(handles.aux))
        handles.output=handles.aux;
    end
end
guidata(hObject, handles);

% --- Executes on button press in check_01.
function check_01_Callback(hObject, eventdata, handles)

if (get(hObject,'Value'))
    handles.output(1)=1;
else
    handles.output(1)=0;
end
guidata(hObject,handles);
%handles.output

% --- Executes on button press in check_02.
function check_02_Callback(hObject, eventdata, handles)

if (get(hObject,'Value'))
    handles.output(2)=1;
else
    handles.output(2)=0;
end
guidata(hObject,handles);
%handles.output

% --- Executes on button press in check_03.
function check_03_Callback(hObject, eventdata, handles)

if (get(hObject,'Value'))
    handles.output(3)=1;
else
    handles.output(3)=0;
end
guidata(hObject,handles);
%handles.output

% --- Executes on button press in check_04.
function check_04_Callback(hObject, eventdata, handles)

if (get(hObject,'Value'))
    handles.output(4)=1;
else
    handles.output(4)=0;
end
guidata(hObject,handles);
%handles.output
```

```
% --- Executes on button press in button_close.
function button_close_Callback(hObject, eventdata, handles)

uiresume(handles.figure1);

% --- Executes on button press in check_05.
function check_05_Callback(hObject, eventdata, handles)

if (get(hObject,'Value'))
    handles.output(5)=1;
else
    handles.output(5)=0;
end
guidata(hObject,handles);
```

B.2.3. Gestión de los parámetros (*param_GUI*)

```
function varargout = param_GUI(varargin)
% PARAM_GUI M-file for param_GUI.fig
%      PARAM_GUI, by itself, creates a new PARAM_GUI or raises the
existing
%      singleton*.
%
%      H = PARAM_GUI returns the handle to a new PARAM_GUI or the handle
to
%      the existing singleton*.
%
%      PARAM_GUI('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in PARAM_GUI.M with the given input
arguments.
%
%      PARAM_GUI('Property','Value',...) creates a new PARAM_GUI or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before param_GUI_OpeningFunction gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to param_GUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help param_GUI

% Last Modified by GUIDE v2.5 08-Jun-2007 12:29:04

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
```

```
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @param_GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @param_GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before param_GUI is made visible.
function param_GUI_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = varargin{1};
handles.param0 = varargin{1};
handles.param = varargin{2};
handles.param0=handles.param0./[1, 1e-15, 1e-15, 1, 1, 1e-12, 1e-9, 1,
1e-15];
% Update handles structure
guidata(hObject, handles);
if (~sum(handles.param))
    set_optimization(['-' '-' '-' '-' '-' '-' '-' '-' '-'],handles);
else
    param = handles.param./[1, 1e-15, 1e-15, 1, 1, 1e-12, 1e-9, 1, 1e-
15];
    set_optimization(param,handles);
end

set_initial_guess(handles.param0,handles);

% UIWAIT makes param_GUI wait for user response (see UIRESUME)
set(handles.figure1,'WindowStyle','modal')
uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = param_GUI_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;
delete(handles.figure1);

function edit_param_init_04_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_param_init_04_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end

function edit_param_init_07_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_param_init_07_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_param_init_09_Callback(hObject, eventdata, handles)

function edit_param_init_09_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_param_init_05_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_param_init_05_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_param_init_08_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_param_init_08_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_param_init_06_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_param_init_06_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function edit_param_init_01_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_param_init_01_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_param_init_02_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_param_init_02_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_param_init_03_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_param_init_03_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in button_default.
function button_default_Callback(hObject, eventdata, handles)

if (get(hObject,'Value'))
    handles.param0 = [100, 10, 100, 10, 1, 1, 100, 1, 10];
    guidata(hObject,handles);
    set_initial_guess(handles.param0,handles);

end

% --- Executes during object creation, after setting all properties.
function edit_param_04_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_param_07_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_param_09_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_param_05_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_param_08_CreateFcn(hObject, eventdata, handles)
```

```
% --- Executes during object creation, after setting all properties.
function edit_param_06_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_param_01_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_param_02_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_param_03_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in button_close.
function button_close_Callback(hObject, eventdata, handles)

handles.param0(1) = str2num(get(handles.edit_param_init_01,'String'));
handles.param0(2) = str2num(get(handles.edit_param_init_02,'String'));
handles.param0(3) = str2num(get(handles.edit_param_init_03,'String'));
handles.param0(4) = str2num(get(handles.edit_param_init_04,'String'));
handles.param0(5) = str2num(get(handles.edit_param_init_05,'String'));
handles.param0(6) = str2num(get(handles.edit_param_init_06,'String'));
handles.param0(7) = str2num(get(handles.edit_param_init_07,'String'));
handles.param0(8) = str2num(get(handles.edit_param_init_08,'String'));
handles.param0(9) = str2num(get(handles.edit_param_init_09,'String'));
handles.param0 = handles.param0.*[1, 1e-15, 1e-15, 1, 1, 1e-12, 1e-9, 1,
1e-15];
handles.output = handles.param0;
guidata(hObject, handles);
uiresume(handles.figure1);

% --- Executes on button press in save_to_workspace.
function save_to_workspace_Callback(hObject, eventdata, handles)

varname=get(handles.edit_varname,'String');
msg = sprintf('%s = [%d %d %d %d %d %d %d %d %d]',varname,...
                handles.param(1),handles.param(2),handles.param(3),...
                handles.param(4),handles.param(5),handles.param(6),...
                handles.param(7),handles.param(8),handles.param(9));
evalin('base',msg);

function edit_varname_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit_varname_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function set_initial_guess(param,handles)
set(handles.edit_param_init_01,'String',param(1));
set(handles.edit_param_init_02,'String',param(2));
set(handles.edit_param_init_03,'String',param(3));
```

```
set(handles.edit_param_init_04,'String',param(4));
set(handles.edit_param_init_05,'String',param(5));
set(handles.edit_param_init_06,'String',param(6));
set(handles.edit_param_init_07,'String',param(7));
set(handles.edit_param_init_08,'String',param(8));
set(handles.edit_param_init_09,'String',param(9));

function set_optimization(param,handles)
set(handles.edit_param_01,'String',param(1));
set(handles.edit_param_02,'String',param(2));
set(handles.edit_param_03,'String',param(3));
set(handles.edit_param_04,'String',param(4));
set(handles.edit_param_05,'String',param(5));
set(handles.edit_param_06,'String',param(6));
set(handles.edit_param_07,'String',param(7));
set(handles.edit_param_08,'String',param(8));
set(handles.edit_param_09,'String',param(9));
```

B.2.4. Ventana de gráficas (*plots_GUI*)

```
function varargout = plots_GUI(varargin)
% PLOTS_GUI M-file for plots_GUI.fig
%      PLOTS_GUI, by itself, creates a new PLOTS_GUI or raises the
existing
%      singleton*.
%
%      H = PLOTS_GUI returns the handle to a new PLOTS_GUI or the handle
to
%      the existing singleton*.
%
%      PLOTS_GUI('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in PLOTS_GUI.M with the given input
arguments.
%
%      PLOTS_GUI('Property','Value',...) creates a new PLOTS_GUI or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before plots_GUI_OpeningFunction gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to plots_GUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help plots_GUI

% Last Modified by GUIDE v2.5 14-May-2007 16:27:52
```

```
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @plots_GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @plots_GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before plots_GUI is made visible.
function plots_GUI_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;

if (nargin==6)
    handles.Sdata = varargin{1};
    handles.param0 = varargin{2};
    handles.param = varargin{3};
% Update handles structure
guidata(hObject, handles);
    if (~sum(handles.param))
       set(handles.check_optim,'Enable','Off');
    end
end

% --- Outputs from this function are returned to the command line.
function varargout = plots_GUI_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

% --- Executes on button press in button_close.
function button_close_Callback(hObject, eventdata, handles)
% hObject    handle to button_close (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.output = handles.param0;
guidata(hObject, handles);
delete(handles.figure1);

% --- Executes on button press in check_original.
function check_original_Callback(hObject, eventdata, handles)

plot_model(handles.Sdata, handles.param0, handles.param, handles)
```

```
% --- Executes on button press in check_initial.
function check_initial_Callback(hObject, eventdata, handles)

plot_model(handles.Sdata, handles.param0, handles.param, handles)


% --- Executes on button press in check_optim.
function check_optim_Callback(hObject, eventdata, handles)

plot_model(handles.Sdata, handles.param0, handles.param, handles)

function plot_model(Sdata, param0, param, handles)
cla(handles.axes1);
cla(handles.axes2);
cla(handles.axes3);
cla(handles.axes4);
hold(handles.axes1,'on');
hold(handles.axes2,'on');
hold(handles.axes3,'on');
hold(handles.axes4,'on');
if (get(handles.check_original,'Value'))
    axes(handles.axes1);
    plot(Sdata(:,1),Sdata(:,2),'r');
    axes(handles.axes2);
    plot(Sdata(:,1),Sdata(:,3),'r');
    axes(handles.axes3);
    plot(Sdata(:,1),Sdata(:,4),'r');
    axes(handles.axes4);
    plot(Sdata(:,1),Sdata(:,5),'r');
end
if (get(handles.check_initial,'Value'))
    axes(handles.axes1);
    plot(Sdata(:,1),model(Sdata(:,1),param0,'S11','MAG'),'b--');
    axes(handles.axes2);
    plot(Sdata(:,1),model(Sdata(:,1),param0,'S11','PHASE'),'b--');
    axes(handles.axes3);
    plot(Sdata(:,1),model(Sdata(:,1),param0,'S21','MAG'),'b--');
    axes(handles.axes4);
    plot(Sdata(:,1),model(Sdata(:,1),param0,'S21','PHASE'),'b--');
end
if (get(handles.check_optim,'Value'))
    axes(handles.axes1);
    plot(Sdata(:,1),model(Sdata(:,1),param,'S11','MAG'),'b:');
    axes(handles.axes2);
    plot(Sdata(:,1),model(Sdata(:,1),param,'S11','PHASE'),'b:');
    axes(handles.axes3);
    plot(Sdata(:,1),model(Sdata(:,1),param,'S21','MAG'),'b:');
    axes(handles.axes4);
    plot(Sdata(:,1),model(Sdata(:,1),param,'S21','PHASE'),'b:');
end
title(handles.axes1,'Magnitude of S11');
title(handles.axes2,'Phase of S11');
title(handles.axes3,'Magnitude of S21');
title(handles.axes4,'Phase of S21');
hold(handles.axes1,'off');
```

```
hold(handles.axes2,'off');
hold(handles.axes3,'off');
hold(handles.axes4,'off');
```

B.2.5. Confirmación de salida (*exitFBAR*)

```
function varargout = exitFBAR(varargin)
% EXITFBAR M-file for exitFBAR.fig
%      EXITFBAR by itself, creates a new EXITFBAR or raises the
%      existing singleton*.
%
%      H = EXITFBAR returns the handle to a new EXITFBAR or the handle to
%      the existing singleton*.
%
%      EXITFBAR('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in EXITFBAR.M with the given input
arguments.
%
%      EXITFBAR('Property','Value',...) creates a new EXITFBAR or raises
the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before exitFBAR_OpeningFunction gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to exitFBAR_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help exitFBAR

% Last Modified by GUIDE v2.5 13-May-2007 16:27:40

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @exitFBAR_OpeningFcn, ...
                   'gui_OutputFcn',  @exitFBAR_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
```

```
% End initialization code - DO NOT EDIT

% --- Executes just before exitFBAR is made visible.
function exitFBAR_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = 'Yes';

% Update handles structure
guidata(hObject, handles);

if(nargin > 3)
    for index = 1:2:(nargin-3),
        if nargin-3==index, break, end
        switch lower(varargin{index})
         case 'title'
          set(hObject, 'Name', varargin{index+1});
         case 'string'
          set(handles.text1, 'String', varargin{index+1});
        end
    end
end

% Determine the position of the dialog - centered on the callback figure
% if available, else, centered on the screen
FigPos=get(0,'DefaultFigurePosition');
OldUnits = get(hObject, 'Units');
set(hObject, 'Units', 'pixels');
OldPos = get(hObject,'Position');
FigWidth = OldPos(3);
FigHeight = OldPos(4);
if isempty(gcbf)
    ScreenUnits=get(0,'Units');
    set(0,'Units','pixels');
    ScreenSize=get(0,'ScreenSize');
    set(0,'Units',ScreenUnits);

    FigPos(1)=1/2*(ScreenSize(3)-FigWidth);
    FigPos(2)=2/3*(ScreenSize(4)-FigHeight);
else
    GCBFOldUnits = get(gcbf,'Units');
    set(gcbf,'Units','pixels');
    GCBFPos = get(gcbf,'Position');
    set(gcbf,'Units',GCBFOldUnits);
    FigPos(1:2) = [(GCBFPos(1) + GCBFPos(3) / 2) - FigWidth / 2, ...
                   (GCBFPos(2) + GCBFPos(4) / 2) - FigHeight / 2];
end
FigPos(3:4)=[FigWidth FigHeight];
set(hObject, 'Position', FigPos);
set(hObject, 'Units', OldUnits);

% Show a question icon from dialogicons.mat - variables questIconData
% and questIconMap
load dialogicons.mat

IconData=questIconData;
```

```
questIconMap(256,:) = get(handles.figure1, 'Color');
IconCMap=questIconMap;

Img=image(IconData, 'Parent', handles.axes1);
set(handles.figure1, 'Colormap', IconCMap);

set(handles.axes1, ...
    'Visible', 'off', ...
    'YDir'   , 'reverse'       , ...
    'XLim'   , get(Img,'XData'), ...
    'YLim'   , get(Img,'YData')  ...
    );

% Make the GUI modal
set(handles.figure1,'WindowStyle','modal')

% UIWAIT makes exitFBAR wait for user response (see UIRESUME)
uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = exitFBAR_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

% The figure can be deleted now
delete(handles.figure1);

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)

handles.output = get(hObject,'String');

% Update handles structure
guidata(hObject, handles);

% Use UIRESUME instead of delete because the OutputFcn needs
% to get the updated handles structure.
uiresume(handles.figure1);

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)

handles.output = get(hObject,'String');

% Update handles structure
guidata(hObject, handles);

uiresume(handles.figure1);

% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)

if isequal(get(handles.figure1, 'waitstatus'), 'waiting')
    % The GUI is still in UIWAIT, us UIRESUME
```

```
    uiresume(handles.figure1);
else
    % The GUI is no longer waiting, just close it
    delete(handles.figure1);
end

% --- Executes on key press over figure1 with no controls selected.
function figure1_KeyPressFcn(hObject, eventdata, handles)

% Check for "enter" or "escape"
if isequal(get(hObject,'CurrentKey'),'escape')
    % User said no by hitting escape
    handles.output = 'No';

    % Update handles structure
    guidata(hObject, handles);

    uiresume(handles.figure1);
end

if isequal(get(hObject,'CurrentKey'),'return')
    uiresume(handles.figure1);
end
```

Bibliografía

[1] R. Aigner, MEMS in RF Filter Applications: Thin-film Bulk Acoustic Wave Techonology, Infineon Technologies.

[2] K.M. Lakin, Thin Film Resonator Technology, IEEE Transactions on Ultrasonics, 2005.

[3] K.M. Lakin, T.K. McCarron, J. Belsick, R. Rose, Filter Banks Implemented With Integrated Thin Film Resonators, Ultrasonic Symposium, 2000.

[4] K.M. Lakin, R.G. Kline, T.K. McCarron, Development of Miniature Filters for Wireless Applications, IEEE Transactions on Microwave Theory and Techniques, 1995.

[5] K.M. Lakin, J.F. McDonald, T.K. McCarron, High Perfomance Stacked Crystal Filters for GPS and Wide Bandwidth Applications, IEEE Ultrasonic Symposium, 2001.

[6] K.M. Lakin, High-Q Microwave Acoustic Resonators and Filters, IEEE Transductions on Microwave Theory and Techniques 1993.

[7] H. Kondoh, An Accurate FET modeling from measured S-paramaters, IEEE MTT-S Digest, 1986.

[8] F. Seyfert, L. Baratchart, J.P. Mamorrat, S. Bila, J. Sombrin, Extraction of coupling parameters for microwave filters: determination of a stable rational model from scattering data, IEEE MTT-S Digest, 2003.

[9] M. Chao, Z. Huang, S. Pao, Z. Wang, C. S. Lam, Modified BVD-Equivalent circuit of FBAR by taking electrodes into account, IEEE Ultrasonic Simposium.

[10] R. Ruby, P. Merchant, Micromachined Thin Film Bulk Acoustic Resonators, IEEE International Frequency Control Symposium, 1996.

[11] J. Yang, An introduction to the theory of piezoelectricity, Springer.

[12] K.M. Lakin, A Review of Thin-Film Resonator Technology, IEEE Microwave magazine, 2003.

[13] D.M. Pozar, Microwave Engineering, Jhon Wiley & sons.

[14] R. Weigel, D.P. Morgan, J.M. Owens, A. Ballato, K.M. Lakin, K. Hashimoto, C.W. Clmenes, Microwave Acoustic Materials, Devices, and Applications, IEEE Transactions on Microwave Theory and Techniques, 2002.

[15] R.K. Wagness, Electromagnetic Fields, Jhon Wiley & sons.

[16] G. Stephen, A. Sofer, Linear and Nonlinear Programming, McGRAW HILL.