UNIVERSIDAD POLITECNICA DE VALENCIA

ESCUELA POLITECNICA SUPERIOR DE GANDIA

I.T. Telecomunicación (Sist. de Telecomunicación)

UNIVERSIDAD
POLITECNICA
DE VALENCIA

ESCUELA POLITECNICA
SUPERIOR DE GANDIA

"Implementación del algoritmo de mínimos cuadrados aplicado al diseño de dispositivos de microondas"

TRABAJO FINAL DE CARRERA

Autor/es:
Mª Dolors Puig Martí

Director/es:
D. José Vicente Morro Ros

GANDIA, 2010

Índice

CAPÍTULO 1: Introducción 3
1.1- Objetivo 5
1.2- Metodología 5
1.3- Estructura del documento 7
CAPÍTULO 2: teoría clásica de la optimización de funciones **9**
2.1- Introducción 9
2.2- Conceptos básicos de optimización 9
2.2.1- Definición del problema 9
2.2.2-Conceptos de mínimo global y mínimo local 10
2.2.3. Clasificación de las funciones 13
2.2.4. Condiciones necesarias y suficientes para la existencia de mínimo 14
2.2.5. Estructura general de los métodos 15
2.3- Algoritmos de optimización sin restricciones 17
2.3.1- Introducción 17
2.3.2- Búsqueda Lineal 19
2.3.2.1- Interpolaciones polinómicas 20
2.3.2.1.1- Interpolación cuadrática 20
2.3.2.1.2- Interpolación cúbica 21
2.3.2.2- Implementación sencilla del Line Search 23
2.3.3- Métodos de Mínimos Cuadrados 23
2.3.3.1- Método Gauss-Newton 25
2.3.3.2- Método Levenberg-Marquardt 27
2.3.3.2.1- Cálculo del parámetro de amortiguamiento λ 28
2.3.3.3- Método Quasi-Newton 30
2.3.3.4- Método Híbrido: Levenberg-Marquardt y Quasi-Newton 32
CAPÍTULO 3: Diseño de dispositivos pasivos de microondas **34**
3.1- Introducción 35
3.2. Diseño de filtros evanescentes con resonadores dieléctricos 39
CAPÍTULO 4: Resultados **44**
4.1- Resultado del método de los mínimos cuadrados 44
4.2- Resultado del diseño de filtros de microondas con postes dieléctricos 60
CAPÍTULO 5: Conclusiones **72**
ANEXO: Código MATLAB de los programas **75**
Bibliografía **86**

CAPÍTULO 1

Introducción

En la sociedad actual, hay una enorme demanda de todo tipo de sistemas de comunicaciones de gran capacidad, es decir, se necesitan anchos de banda elevados, tecnología que posibilite el dar servicio a un número cada vez mayor de usuarios, ofrecer nuevas aplicaciones, etc.

Todo ello requiere un uso extremadamente eficiente del espectro radioeléctrico. Las bandas usadas actualmente están prácticamente saturadas, de modo que la solución más práctica es la de extender las telecomunicaciones sobre frecuencias cada vez más elevadas, concretamente sobre la banda de *microondas* (de 300 MHz hasta 300 GHz).

Las ventajas de la banda de microondas son las siguientes:

- En estas frecuencias es más sencillo conseguir antenas de mayor ganancia, ya que ésta es proporcional a la longitud efectiva de la antena, por tanto, para un tamaño fijo, se tiene mayor ganancia conforme aumenta la frecuencia de operación.

- La banda de frecuencias de microondas ofrece mucha directividad, lo que hace que sea idónea para las comunicaciones punto a punto, ya que la onda viaja desde el transmisor hasta el receptor en línea recta, creando poca interferencia a otros usuarios.

- La atmósfera presenta un comportamiento bastante bueno en este rango de frecuencias, ya que la atenuación debida a la ionosfera es prácticamente plana en frecuencia y no hay absorción por ozono ni radiaciones ionizantes, aunque sí existen dos picos de absorción por moléculas de oxígeno (a 60 y a 120 GHz), y otros dos por moléculas de agua (a 22.2 y a 183.3 MHz), que hay que evitar.

- Es un hecho que en este rango hay muchas frecuencias disponibles, de modo que no se solapan con las aplicaciones ya existentes.

- En estas frecuencias tienen lugar diferentes reacciones moleculares, atómicas y nucleares, que permiten el desarrollo de numerosas aplicaciones en el ámbito industrial como el calentamiento de alimentos y productos industriales; en la agricultura con las novedosas aplicaciones en el tratamiento de los suelos cultivables, para eliminar el crecimiento indeseado de algunas hierbas, o en el control de plagas: y en la ecología con el tratamiento de residuos radioactivos o la purificación del carbón. También, existen aplicaciones médicas que utilizan esta banda de frecuencias, como la elaboración de diagnósticos y tratamientos médicos.

- Por último, la mayor ventaja de todas: al aumentar la frecuencia sobre la que se transmite, es más sencillo proporcionar *anchos de banda relativos elevados* en los dispositivos, lo que equivale a *más capacidad y velocidad* para el usuario final.

No obstante, la transmisión en la banda de microondas también presenta inconvenientes:

- La frecuencia máxima de operación nunca es infinita, sino que viene marcada por el estado de la tecnología que se esté usando en ese momento (que no siempre coincide con la tecnología que está siendo investigada).

- La alta directividad de las transmisiones en la banda de microondas puede suponer una desventaja para las comunicaciones punto a multipunto.

- Se debe tener un cuidado especial a la hora de multiplexar distintos canales adyacentes en frecuencia, ya que al trabajar en altas frecuencias, es difícil conseguir unas pendientes abruptas que permitan un aprovechamiento óptimo del espectro, por lo que hay que prever unas bandas de guarda mayores de lo habitual.

A pesar de estos inconvenientes, el uso de la banda de microondas está muy extendido en la actualidad. Además de los servicios ya existentes, como las *comunicaciones por satélite* o las innumerables aplicaciones de la *tecnología de radar* [13], están surgiendo nuevos servicios que hacen uso de esta banda de frecuencias, como son los *bucles de acceso inalámbricos* (Wireless LAN, WiMAX) o los sistemas de posicionamiento global como *GPS* o el futuro *GALILEO*.

Por ello, y debido a las dificultades anteriormente comentadas, es muy importante diseñar bien todos los componentes de un sistema de comunicaciones que opere en este rango de frecuencias. Teniendo como objetivo que el comportamiento de todos los dispositivos sea lo más cercano posible al ideal.

El proceso de diseño que tradicionalmente se sigue, se basa en un procedimiento de ensayo-error, es decir, conociendo la respuesta ideal del dispositivo, el diseñador modifica dicha estructura y vuelve a analizarla. Este proceso se repite hasta conseguir una respuesta suficientemente buena, lo cual, en ocasiones, es muy difícil.

El problema de este proceso surge cuando la complejidad de los dispositivos aumenta, como ocurre en la actualidad, ya que para satisfacer las necesidades de los usuarios es imprescindible el uso de una tecnología compleja de diseñar, que plantea problemas a los métodos tradicionales y que requiere de una alta precisión para que su uso sea satisfactorio.

Por ello, actualmente el proceso de diseño de dispositivos de microondas se basa en minimizar una función de error entre la *respuesta ideal* y la *respuesta real* del dispositivo bajo estudio. Esta función dependerá de las dimensiones físicas del dispositivo y de las características de algunos materiales utilizados.

El problema que se plantea es que no basta sólo con minimizar esa función de error, sino que además hay que asegurar que las variables tomen unos valores adecuados, físicamente realizables y coherentes. Por lo tanto, es *imprescindible* añadir ciertas restricciones al problema de minimización.

De esta forma, se construye una herramienta CAD de diseño automatizado, en la que la intervención humana es mínima, tan sólo es necesario introducir la topología del dispositivo, el objetivo de diseño, las restricciones a tener en cuenta y un punto inicial.

Una vez establecido esto, se obtiene la solución óptima de forma iterativa. Así, el proceso es más eficaz y eficiente que el proceso clásico, ya que se evitan los errores que puede cometer el diseñador en los cambios a realizar en la estructura durante el proceso de ensayo-error, pudiendo hacer imposible la obtención de una configuración razonablemente buena.

1.1- Objetivo

El *objetivo* principal de este trabajo es el *diseño e implementación de algoritmos que permitan la optimización automatizada de una función matemática, así como la aplicación de los mismos al diseño de dispositivos pasivos de microondas*.

En el *Grupo de Aplicaciones de las Microondas*, del *i-TEAM*, ya habían sido programados distintos métodos sin tener en cuenta restricciones capaces de diseñar dispositivos bastante complejos.

El método fundamental que se va a programar es el *Método de Mínimos Cuadrados*, además en el interior de éste se hace uso de otros algoritmos que resuelven problemas más sencillos. Todos ellos han sido también programados.

En definitiva, el programar e implementar estos métodos y su inclusión en el proceso de diseño de dispositivos de microondas me planteaba un reto por varios motivos: uno porque suponía la puesta en práctica de conocimientos y destrezas adquiridos en esta Escuela; otro por hacer frente a nuevos imprevistos y darles solución.

1.2- Metodología

Este proyecto se ha programado en *MATLAB*, bajo el sistema operativo *Windows*.

Durante la realización de este proyecto se ha llevado a cabo un proceso a dos niveles: *individual* y *tutelado por el director del proyecto*.

La idea inicial del proyecto me fue sugerida por el director del mismo, José Vicente Morro Ros. Lo primero que hizo fue proporcionarme bibliografía y guiarme en el estudio. Posteriormente, me ha facilitado aclaraciones y sugerencias siempre que lo he necesitado, resultando ser una ayuda imprescindible en el desarrollo de este proyecto.

Mi trabajo individual inicialmente se dirigió a entender los diferentes aspectos teóricos de la bibliografía. Posteriormente, comencé a programar los diferentes algoritmos que dan forma a este proyecto, para terminar dedicando tiempo a su puesta en práctica.

Durante las pruebas de estos algoritmos, gracias a mi tutor pude superar algunas dificultades que surgieron en la puesta en práctica.

Por otro lado, el proceso de trabajo ha constado de diferentes fases, en las cuales han coexistido los niveles anteriormente mencionados. Las fases han sido las siguientes:

- **Formación teórica:** Se han estudiado diferentes métodos de minimización de funciones no sujetas a restricciones, que consideran una sola función objetivo, comprendiendo bien su funcionamiento y el significado de todas sus variables para más tarde poderlos programar.

  Además, ha sido necesario perfeccionar el conocimiento y uso de distintos programas y librerías. Éste ha sido el caso del lenguaje *MATLAB* [7].

- **Programación de los algoritmos:** Programar un algoritmo consiste en describir su funcionamiento de una manera sistemática y organizada, mediante un lenguaje de programación, que en este caso ha sido *MATLAB*. La programación ha sido vital en este proyecto, de hecho los métodos han sido depurados muchas veces para conseguir una mejora, tanto en los resultados como en el tiempo de cómputo.

- **Validación de los algoritmos:** Tras la programación de cada algoritmo, se han intentado detectar los distintos errores presentes en él, comprobando, al mismo tiempo, si el algoritmo desarrollado obtiene unos resultados cercanos a los pasos del algoritmo. Esta fase ha sido una de las más costosas.

- **Obtención de resultados:** Se han realizado diferentes pruebas para comprobar los algoritmos y métodos utilizados, aplicándose después al diseño de un filtro de microondas. De este modo, los resultados obtenidos se presentan de la siguiente manera:

  - Prueba de los distintos métodos que se han programado para resolver sistemas de ecuaciones lineales. Para la realización de estas pruebas se han tenido que hacer varias implementaciones del algoritmo desarrollado, con el fin de que los valores obtenidos fueran lo más parecido posible a los valores reales.

  - Pruebas de los distintos métodos de optimización de funciones en el programa *MATLAB*, comparando los resultados ofrecidos por el programa con los obtenidos anteriormente.

  - Finalmente, se han aplicado los cuatro algoritmos de optimización sin restricciones al diseño de un dispositivo de microondas real, como es un filtro evanescente con cuatro cavidades resonantes con postes dieléctricos en su interior. Para esto ha sido necesario utilizar el software de diseño del GAM del i-TEAM.

1.3- Estructura del documento

El trabajo se ha estructurado en capítulos de la siguiente manera:

- **Capítulo 2:** En este capítulo se presenta la teoría fundamental de la minimización de funciones, definiéndose conceptos fundamentales como *mínimo global* y *mínimo local, condiciones suficientes y necesarias* para la existencia de mínimos, etc. Seguidamente, se explica la *función de Rosenbrock* que es la función de test utilizada en las pruebas. Hay una breve introducción a la optimización sin restricciones, tras explicar las características generales de ésta se explica en que consiste la búsqueda lineal. Finalmente, se presentarán varios conceptos básicos para entender matemáticamente el método de mínimos cuadrados, explicando en que consisten los métodos de Gauss-Newton, Levenberg-Marquardt, Quasi-Newton con dos técnicas de actualización de la hessiana y el método híbrido.

- **Capítulo 3:** En este capítulo se explican, de forma general, las estrategias y procesos más comunes en el diseño de cualquier dispositivo de microondas. A continuación, se expone el diseño de los filtros evanescentes con resonadores dieléctricos, analizando, primero, las ventajas que presenta este tipo de filtros respecto a otros. Posteriormente, se explica la estrategia de diseño aplicada para el diseño de este tipo de filtros.

- **Capítulo 4:** Se evalúa el algoritmo ya programado en MATLAB presentando los resultados obtenidos comparados con los proporcionados por el programa. Gracias a estas pruebas, se han hecho modificaciones al algoritmo para mejorar su funcionamiento.

  Por último, se muestran los resultados obtenidos al aplicar los diferentes algoritmos al proceso de diseño de un dispositivo de microondas real, en concreto un filtro evanescente con resonadores dieléctricos.

- **Capítulo 5:** En este último capítulo se exponen las principales conclusiones a las que se llega tras la realización del proyecto.

Además, al final del proyecto se ha incluido un apéndice:

- **Apéndice A:** En este apéndice se hace una revisión de los distintos métodos implementados, indicando cómo invocarlos y cuáles son sus parámetros de entrada y de salida. Pretende ser un manual de referencia para el usuario que desee utilizar cualquiera de los algoritmos aquí programados.

CAPÍTULO 2

Teoría clásica de la optimización de funciones

2.1- Introducción

Este capítulo comienza presentando los conceptos básicos de optimización de funciones, los cuales sirven de soporte para el resto del proyecto.

Posteriormente, se explican algunos algoritmos de optimización sin restricciones del tipo de *Búsqueda Lineal* mediante *interpolaciones polinómicas*, como es el método de *Mínimos Cuadrados*, según Gauss-Newton, Levenberg-Marquardt, el método *Quasi-Newton* y un método híbrido.

2.2- Conceptos básicos de optimización

2.2.1- Definición del problema

La optimización de funciones es una herramienta matemática cuyo objetivo es encontrar la mejor solución a un problema matemático expresado, en la mayoría de los casos, mediante una ecuación. Esta *mejor solución* puede ser un punto donde la función presente un máximo o un mínimo, según el problema dado. Sin embargo, ambos casos son equivalentes, ya que encontrar el mínimo de una función f es lo mismo que encontrar el máximo de la función $f_1 = -f$. Por ello, a partir de este momento, se hablará de optimización como *minimización de funciones* sin pérdida de generalidad.

Las variables de las que depende la función que define el problema, conocida como función objetivo, pueden estar sujetas a unas restricciones, limitando así la zona de búsqueda de la solución. Este hecho, cambia la forma de tratar el problema, distinguiéndose de este modo dos tipos de optimización, con y sin restricciones, para los cuales los métodos a aplicar son distintos.

El problema de minimización de una función se define matemáticamente así:

$$\min_{x \in \Re^n} f(x)$$

donde

$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ es el vector de las variables reales del problema

$f : \Re^n \to \Re$ es la *función objetivo*.

Es importante remarcar que, además de encontrar el punto en el que la función objetivo toma el valor mínimo, interesa hacerlo de manera rápida y con el menor coste computacional posible. Este coste computacional está totalmente marcado por el coste de evaluar la función f, por lo que se interesará hallar el mínimo con el *menor número posible de evaluaciones* de la función.

En este estudio se asume que la función f tiene derivadas primera y segunda y ambas son continuas (es decir, f es de clase C^2). Si alguna de dichas derivadas no fuese continua, las condiciones necesarias para que un punto sea mínimo cambiarían drásticamente. Este tipo de problemas requiere un tratamiento diferente al que aquí se plantea. Para mayor detalle, consultar [1].

2.2.2-Conceptos de mínimo global y mínimo local

Al valor de x que minimiza dicha función se le va a denominar a partir de aquí como $x*$. Normalmente, al abordar un problema de minimización se suele pensar que dicho punto mínimo $x*$ existe y es único. Sin embargo, aunque en bastantes situaciones será así, estas condiciones tan favorables no tienen por qué darse siempre.

En primer lugar, conviene establecer la diferencia entre *mínimo global* y *mínimo local*. Veamos las definiciones de cada uno de estos conceptos:

Definición: Sea una función $f : \Re^n \to \Re$. Dicha función presenta un mínimo global en un punto $x*$ si y solo si

$$f(x) \geq f(x*), \ \forall x \neq x*$$

Es decir, en todo el dominio de la función no hay ningún otro punto x en el que dicha función tome un valor menor.

Generalmente, al abordar un problema de minimización, se suele pensar que el punto obtenido como solución $x*$, es el mínimo global de la función objetivo, es decir, el punto donde la función toma el menor valor. Pero esto no siempre es así, ya que el

método de optimización aplicado tiene una visión local, y no global, de la función, obtenida a partir de los puntos que analiza. Por lo que el punto obtenido será el mínimo de la zona analizada, que no tiene por qué ser el mínimo global de la función. Este mínimo recibe el nombre de mínimo local. Formalmente, estos mínimos se definen de la siguiente manera:

Definición: Sea una función $f : \Re^n \to \Re$. Dicha función presenta un mínimo local en un punto $x*$ si y solo si

$$f(x) \geq f(x*), \ \forall x \neq x* \text{ en un cierto entorno de } x*$$

Es decir, si en el espacio n-dimensional de x se pueden definir regiones de las cuales sólo hay un punto mínimo, cada uno de ellos será un mínimo local.

Un ejemplo en el que se ilustra estos dos tipos de mínimos puede ser la función

$$f(x) = (1-x)\sin(x), \qquad \text{con } x \in [-4,4]$$

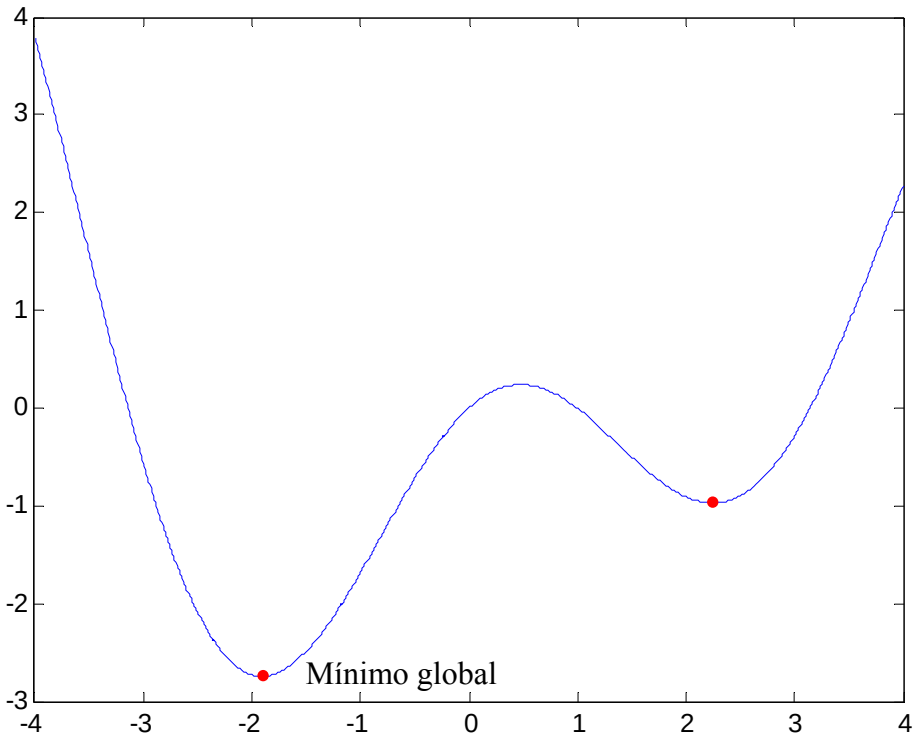


Figura 2.1: Función con mínimo global y mínimo local

A partir de estas definiciones, se deduce la dificultad de encontrar el mínimo global y, por lo tanto, garantizar que el punto obtenido tras el proceso de optimización sea dicho mínimo, ya que el método aplicado puede quedar *atrapado* en un mínimo local. Por ello, se necesita un mayor conocimiento de la función objetivo, muchas veces difícil de conseguir, para identificar el mínimo global.

2.2.3. Clasificación de las funciones

Una posible clasificación de las funciones se puede realizar atendiendo al número de mínimos que representan en la región bajo estudio. De este modo, se dice que una función es *unimodal* cuando tiene un solo mínimo local en la región permitida. De manera análoga, la función es *multimodal* cuando hay varios mínimos locales en dicha región.

Si la función a minimizar es multimodal, tras el proceso de optimización es bastante difícil garantizar que el punto obtenido sea un mínimo global.

Otra forma de clasificar las funciones viene dada en función de sus derivadas parciales de primer y segundo orden (en este caso, se supone que existen y son continuas, como se dijo anteriormente). De este modo, se tiene lo siguiente

- **Derivadas parciales de primer orden:** Las derivadas parciales de primer orden forman el vector *gradiente*. El gradiente de una función en un punto x viene expresado por

$$g(x) = \nabla f(x) = \begin{bmatrix} \dfrac{\partial f(x)}{\partial x_1} \\ \dfrac{\partial f(x)}{\partial x_2} \\ \vdots \\ \dfrac{\partial f(x)}{\partial x_n} \end{bmatrix}$$

  Este vector indica la dirección de máxima variación.

  De esta forma, se puede hablar de funciones *crecientes* o *decrecientes*. Cuando el gradiente es positivo, la función es *creciente* en esa dirección. Análogamente, cuando el gradiente es negativo, la función es decreciente en dicha dirección.

- **Derivadas parciales de segundo orden:** Como se supone que la función bajo estudio presenta derivadas segundas continuas, se puede definir la matriz de derivadas parciales de segundo orden o *Matriz Hessiana*, la cual se define como:

$$G(x) = \nabla^2 f(x) = \begin{bmatrix} \dfrac{\partial^2 f(x)}{\partial x_1^2} & \dfrac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \dfrac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \dfrac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 f(x)}{\partial x_n \partial x_1} & \dfrac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \dfrac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}$$

  Gracias a esta matriz, podemos hallar la *curvatura* de la función en un punto x a lo largo de una dirección definida por s. Dicha curvatura viene dada por la expresión $s^T G(x) s$.

Si la curvatura que se acaba de hallar es positiva, indica que la derivada de la función en esa dirección es creciente, o lo que es lo mismo, que la función es *convexa*. De manera que si dicha curvatura es negativa, la derivada de la función es decreciente, es decir, la función es *cóncava*.

Como se puede intuir, estos conceptos van a ser de gran utilidad a la hora de describir lo que debe suceder en un punto en el que la función toma un valor mínimo.

2.2.4. Condiciones necesarias y suficientes para la existencia de mínimo

Una vez explicados los conceptos, se procede a establecer las condiciones que se deben cumplir para que un cierto punto sea el que minimice la función objetivo.

En primer lugar, se definen las *Condiciones Necesarias de Primer y Segundo Orden*, que cumple todo punto x^*, el cual es un mínimo local de una cierta función f. por el simple hecho de ser un mínimo local:

Sea una función $f : \Re^n \to \Re$ y un punto $x \in \Re^n$.

- **Condición Necesaria de Primer Orden**: La derivada direccional de f a lo largo de *cualquier* dirección s, evaluada en ese punto x^*, será 0 (es decir, el punto x^* es un extremo de la función f). Matemáticamente se puede expresar de la siguiente manera:

$$s^T \nabla f(x^*) = 0 \quad \forall s$$

  O de manera equivalente:

$$\nabla f(x^*) = g^* = 0$$

- **Condición Necesaria de Segundo Orden**: La función *nunca va a ser cóncava* en ese punto, o lo que es lo mismo, la curvatura de la función en dicho punto a lo largo de cualquier dirección es *no negativa*. Es importante puntualizar que el hecho de estar en un mínimo local no implica que la función sea convexa, sino que implica que la función no es cóncava. Se puede expresar matemáticamente de la manera siguiente:

$$s^T G^* s \geq 0 \quad \forall s$$

  donde $G^* = G(x^*) = \nabla^2 f(x^*)$

  Esta condición necesaria es equivalente a decir que la matriz Hessiana en el punto mínimo, G^*, es una *matriz semidefinida positiva*.

Las condiciones necesarias las cumple todo punto que sea mínimo. Si ahora se realiza el análisis en sentido inverso, se tienen las *Condiciones Suficientes* que implican que un punto cualquiera que las cumpla será un mínimo de la función.

Condiciones Suficientes: Sea una función $f : \Re^n \to \Re$. Un punto $x*$ es un mínimo local de f si cumple con las siguientes condiciones matemáticas:

$$\nabla f(x*) = g* = 0$$
$$s^T G* s > 0 \qquad \forall s \neq 0$$

2.2.5. Estructura general de los métodos

Los métodos para minimizar funciones cuando no están sometidas a ningún tipo de restricción son muchos y muy variados, ya que las posibilidades para encontrar el mínimo de una función son muy amplias.

Sin embargo todos estos métodos tienen la misma estructura: partiendo de un punto inicial x_0, introducido por el usuario, se van tomando sucesivas aproximaciones de la solución, basándose en información tanto del valor de la función como del valor de alguna de sus derivadas. De esta forma, se obtiene una sucesión de puntos $x^{(k)}$ que convergen en el mínimo de la función (donde el índice *k* indica el número de la iteración actual).

Este proceso iterativo finalizará cuando se cumple algún *criterio de terminación* previamente establecido por el usuario. Hay muchas posibilidades en cuanto a criterios de terminación, pudiéndose utilizar la que más convenga en cada caso.

Si se conoce el valor de la función en el punto óptimo $x*$, el criterio de terminación que se aplica será el siguiente:

$$f(x^{(k)}) - f(x*) \leq f_{tol}$$

donde f_{tol} es un valor introducido por el usuario.

Si, por el contrario, no se conoce el valor de la función en el punto óptimo, es posible definir hasta cuatro criterios de terminación diferentes:

- Se parará si el gradiente de la función en el punto actual es *suficientemente* cercano a cero. Matemáticamente, se puede expresar así:
  - $\left\| g(x^{(k)}) \right\| \leq g_{tol}$
- Este criterio no se suele emplear en la práctica ya que es difícil determinar un umbral adecuado.

- El algoritmo terminará cuando, entre dos iteraciones consecutivas, en cada eje se haya avanzado una cantidad menor a una cierta tolerancia x_{tol}. Dicha tolerancia puede ser distinta para cada eje. Es decir:

  - $\left\| x_i^{(k+1)} - x_i^{(k)} \right\| \le x_{tol} \quad , \quad \forall i$

- El algoritmo finalizará si, entre dos iteraciones sucesivas, la distancia entre dos puntos **x** es menor que un cierto umbral x_{tol}, esta es la opción más usual. Es decir:

$$\left\| x^{(k+1)} - x^{(k)} \right\| \le x_{tol}$$

- Se terminará la búsqueda cuando, entre dos iteraciones consecutivas, la diferencia entre los valores que toma la función es menor que una cierta tolerancia f_{tol} (definida por el usuario), es decir:

$$\left\| f^{(k+1)} - f^{(k)} \right\| \le f_{tol}$$

En la práctica, es muy común la utilización de varios de estos criterios a la vez. Además, se suele limitar el número de evaluaciones de la función objetivo o el número de iteraciones, por si no se produce la convergencia.

Hasta aquí se ha presentado la estructura básica más común de los diferentes métodos de minimización. Sin embargo, falta por explicar la forma en la que se puede *medir* la eficiencia de los mismos, la cual se basa en el concepto de *orden de convergencia* de un algoritmo, que es una medida del grado de *rapidez* o *lentitud* con la que la sucesión de puntos $x^{(k)}$ converge al punto óptimo $x*$ en un entorno del mismo.

Partiendo de la definición del vector $h^{(k)} = x^{(k)} - x*$, podemos hablar de *convergencia de orden* p si

$$\frac{\left\| h^{(k+1)} \right\|}{\left\| h^{(k)} \right\|} \to a \quad , \quad a > 0$$

Hay varios casos particulares de gran interés:

$$\frac{\left\| h^{(k+1)} \right\|}{\left\| h^{(k)} \right\|} \to a \quad , \qquad \textit{Convergencia Lineal}$$

$$\frac{\left\| h^{(k+1)} \right\|}{\left\| h^{(k)} \right\|^2} \to a \quad , \qquad \textit{Convergencia Cuadrática}$$

$$\frac{\left\| h^{(k+1)} \right\|}{\left\| h^{(k)} \right\|} \to 0 \quad , \qquad \textit{Convergencia Superlineal}$$

En la práctica, la existencia de convergencia y de un determinado orden de la misma no es garantía de un buen comportamiento del algoritmo. Eso es debido a muchos factores, como por ejemplo:

- Al estudiar teóricamente la convergencia de un algoritmo, no se tienen en cuenta los errores de redondeo del ordenador.

- El orden de convergencia teórico se demuestra sólo para puntos cercanos al óptimo.

- Para determinar dicho orden de convergencia, se hacen suposiciones sobre la función objetivo que pueden no cumplirse en la práctica (como por ejemplo la convexidad).

Por esto, el mejor indicador del buen comportamiento de un algoritmo es la prueba experimental, realizada normalmente con diversas funciones de test. Un ejemplo de estas funciones es la *Función de Rosenbrock* o *Función "banana"*, cuya expresión, para $n = 2$ es la siguiente:

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2$$

Esta función suele ser muy usada como función de prueba en optimización por su lenta convergencia para la mayoría de los métodos. Su mínimo está en $x = [1,1]$, donde $f(x) = 0$.

Podemos ver la representación gráfica de la función de Rosenbrock en la siguiente imagen:

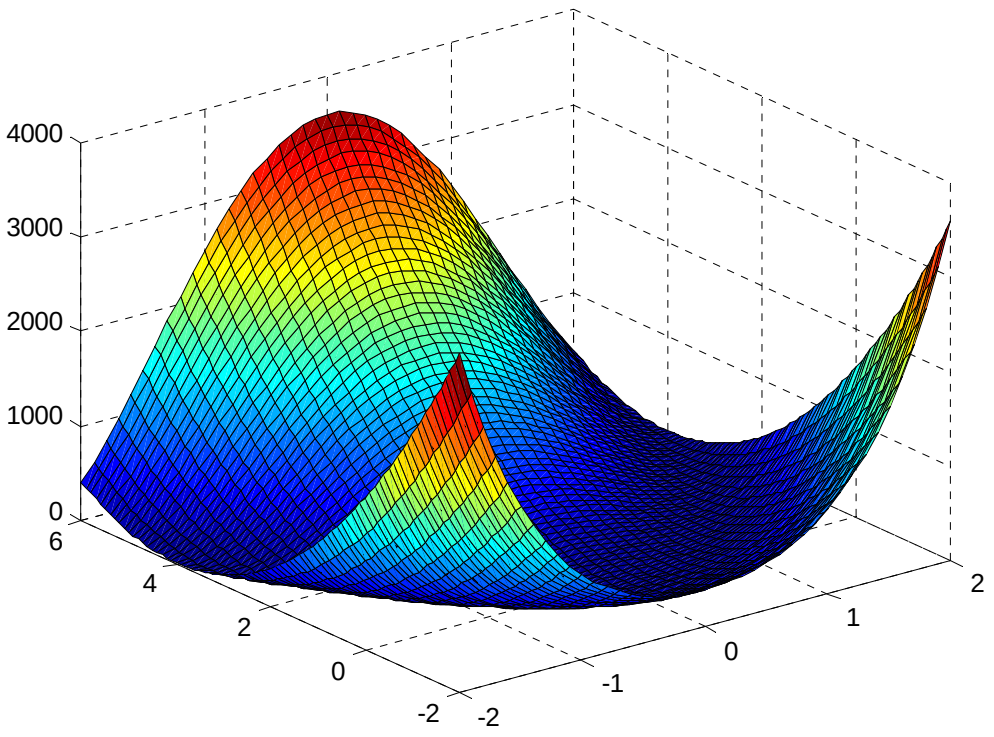
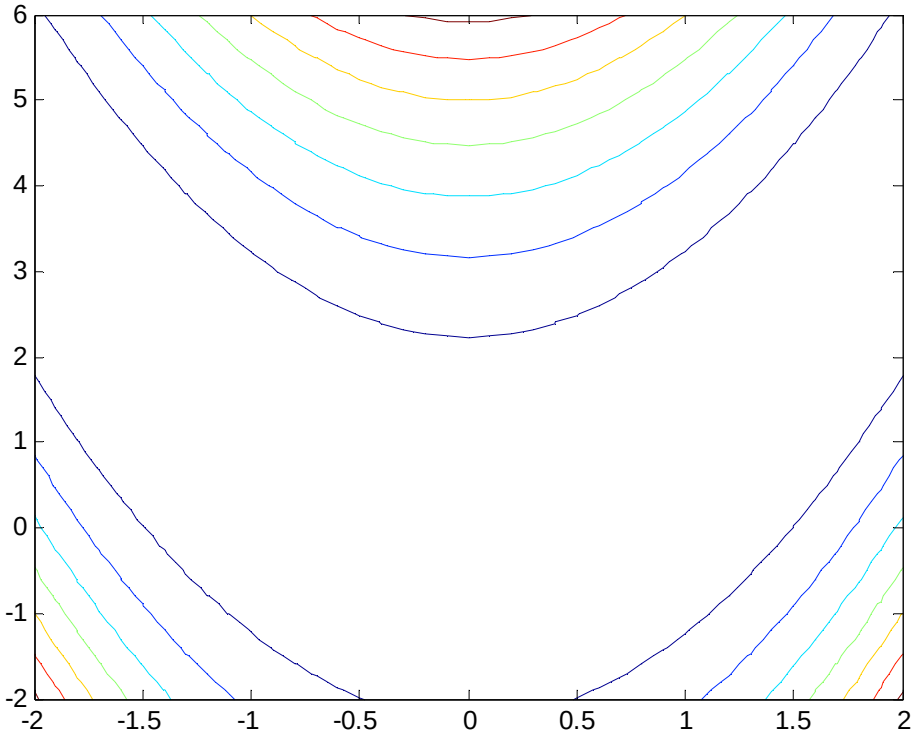


Figura 2.2: Función de Rosenbrock

A continuación se muestran las curvas de nivel de la función de Rosenbrock:



2.3- Algoritmos de optimización sin restricciones

2.3.1- Introducción

Una vez situados en el problema, es interesante comprender los métodos más utilizados para minimizar funciones cuando no están sometidas a ningún tipo de restricción.

Las posibilidades para encontrar el mínimo de una función son muy amplias, abarcando un enorme abanico de métodos.

Sin embargo, esto depende de la información de la función y de las derivadas que utilice el método para aproximarse a la solución. En base a esto, todos los métodos se pueden clasificar en los tres tipos siguientes:

- **Métodos de búsqueda directa:** Estos métodos sólo evalúan la función objetivo. Son adecuados para funciones con una alta no linealidad o con discontinuidades. Son un tipo de métodos muy robustos, pero presentan una convergencia más lenta que los otros.

- **Métodos de Gradiente:** Estos métodos utilizan el valor del gradiente de la función (primera derivada). Su eficiencia es mucho mayor cuando el gradiente es continuo.

- **Métodos de orden superior:** Estos métodos necesitan calcular derivadas de orden superior, La gran desventaja de este método es el alto coste computacional que supone este cálculo (ya que exige bastantes evaluaciones de la función objetivo). Sin embargo, en la mayoría de los casos, posee la ventaja de tener rápida convergencia.

Todos los algoritmos de minimización necesitan un punto de partida x_0, este punto lo debe proporcionar un usuario con cierto conocimiento de la aplicación. Cuanto más cerca esté x_0 de la solución óptima menos tiempo necesitará el algoritmo para converger y habrá menos riesgo de mínimos locales.

Se generan varios puntos hasta que se cumple el criterio de terminación, para generar un nuevo punto hay dos estrategias posibles: *Región de confianza* y *Búsqueda lineal*.

En la estrategia de *búsqueda lineal*, se determina una dirección de avance d_k y si se esta en el punto x_k, el nuevo punto será $x_{k+1} = x_k + \alpha d_k$

A la búsqueda de α se le llama *búsqueda lineal* (Line Search). Consiste en buscar el mínimo de una función *unidimensional* $g(\alpha)$:

$$g(\alpha) = f(x_k + \alpha d_k)$$

En este caso hay que determinar a cada paso la dirección de avance.

En la estrategia de *región de confianza*, en cada iteración, se aproxima la función en el entorno de x_k (punto origen) por otra función más sencilla utilizando la *Serie de Taylor*, es decir:

$$m_k(x_k + s) = f(x_k) + \nabla f^T s + \frac{1}{2} s^T H s \,,\ \|s\|_2 \le \Delta \,,\ \Delta > 0$$

donde Δ es el radio de la región de confianza.

Se obtiene el mínimo de m_k dentro de la región de confianza, si no decrece f suficientemente, se reduce Δ y se vuelve a buscar el mínimo de m_k en la nueva región de confianza.

La diferencia entre estas dos estrategias está en cómo se selecciona la dirección y la distancia del nuevo paso:

- En la búsqueda lineal primero se elige la dirección y luego la distancia.
- En la región de confianza primero se elige la distancia y luego la dirección.

La estrategia de la búsqueda lineal es la utilizada en la mayoría de algoritmos clásicos, y la utilizada en este proyecto.

Tras explicar las características generales de los métodos para minimizar una función sin restricciones, se va a explicar en que consiste la búsqueda lineal.

2.3.2- Búsqueda Lineal

Las técnicas de búsqueda lineal realmente son procedimientos de optimización para una sola variable, y que se realiza repetidamente en problemas de varias variables.

La elección de un método adecuado de búsqueda lineal es de gran importancia en un algoritmo de optimización ya que es un procedimiento muy costoso.

En la búsqueda lineal se buscan dos cosas:

- **Acotación:** determina un intervalo que acote los valores de α deseables.
- **Bisección o interpolación:** determina un valor de α dentro del intervalo.

Se prueban unos cuantos valores de α y se para cuando se satisfacen dos condiciones, llamadas *Condiciones de Wolfe*, éstas son condición de suficiente descenso y condición de curvatura:

$$\text{condición de suficiente descenso} \rightarrow f\left(x_k + \alpha p_k\right) \leq f\left(x_k\right) + c_1 \alpha \nabla f_k{}^T \; p_k$$

$$\text{condición de curvatura} \rightarrow \nabla f\left(x_k + \alpha p_k\right)^T \; p_k \geq c_2 \nabla f_k{}^T \; p_k$$

O también:

$$\text{condición de suficiente descenso} \rightarrow g\left(\alpha\right) \leq g\left(0\right) + c_1 \alpha g'\left(0\right) = l\left(\alpha\right)$$

$$\text{condición de curvatura} \rightarrow g'\left(\alpha\right) \geq c_2 g'\left(0\right)$$

donde $0 < c_1 < c_2 < 1$

La *Condición de Curvatura* se añade para evitar que se puedan tomar valores de α muy pequeños.

Como un punto puede satisfacer las condiciones de Wolfe sin estar muy próximo a un mínimo, se usan las *condiciones de Wolfe estrictas*:

$$\text{condición de suficiente descenso} \rightarrow f\left(x_k + \alpha p_k\right) \leq f\left(x_k\right) + c_1 \alpha \nabla f_k{}^T \; p_k$$

$$\text{condición de curvatura} \rightarrow \left|\nabla f\left(x_k + \alpha p_k\right)^T \; p_k\right| \geq c_2 \nabla f_k{}^T \; p_k$$

O también:

$$\text{condición de suficiente descenso} \rightarrow g\left(\alpha\right) \leq g\left(0\right) + c_1 \alpha g'\left(0\right) = l\left(\alpha\right)$$

$$\text{condición de curvatura} \rightarrow \left|g'\left(\alpha\right)\right| \geq c_2 g'\left(0\right)$$

Con esto se evita que $g'\left(\alpha\right)$ este muy alejado de un mínimo.

Hay dos tipos de algoritmos de búsqueda lineal, los que no calculan derivadas y los que sí calculan derivadas (interpolaciones polinómicas).

En este proyecto para hallar el mínimo a lo largo de la dirección de avance d se ha utilizado un método polinómico que calcula derivadas, usando una combinación de interpolación cuadrática y cúbica

2.3.2.1- Interpolaciones polinómicas

Consiste en hacer aproximaciones polinómicas con la información de la función y las derivadas en los puntos por los que se va pasando.

2.3.2.1.1- Interpolación cuadrática

Consiste en ajustar una función de una variable de la forma:

$$g(\alpha) = f(x_k + \alpha \cdot d) \approx a \cdot \alpha^2 + b \cdot \alpha + c$$

Un extremo de la función (mínimo o máximo) tiene lugar en

$$g'(\alpha) = 2a\alpha + b = 0 \rightarrow \alpha^* = \frac{-b}{2a} \rightarrow x_{k+1} = x_k + \alpha^* \cdot d$$

Para que haya un mínimo se tiene que cumplir que:

- a es positivo (la segunda derivada es positiva).
- b siempre es negativo.

Los coeficientes a y b se pueden obtener a partir de tres puntos conocidos, o el valor del gradiente en dos puntos. Se pueden hacer simplificaciones si, por ejemplo, el primer punto se toma para $\alpha = 0$.

Si se conoce el valor de g en tres puntos no equiespaciados:

$$\alpha_1 \rightarrow g(\alpha_1)$$

$$\alpha_2 \rightarrow g(\alpha_2)$$

$$\alpha_3 \rightarrow g(\alpha_3)$$

$$\alpha^* = \frac{1}{2} \frac{\beta_{23} g(\alpha_1) + \beta_{31} g(\alpha_2) + \beta_{12} g(\alpha_3)}{2\gamma_{23} g(\alpha_1) + \gamma_{31} g(\alpha_2) + \gamma_{12} g(\alpha_3)}$$

Donde: $\beta_{ij} = \alpha_i^2 - \alpha_j^2$

$$\gamma_{ij} = \alpha_i - \alpha_j$$

Para realizar interpolación, y asegurar que encontramos un mínimo, buscaremos tres puntos cercanos al mínimo. Eso se garantiza si:

$$g(\alpha_2) < g(\alpha_1) \quad \text{y} \quad g(\alpha_2) < g(\alpha_3)$$

Un caso muy frecuente es disponer de la siguiente información:

$$\alpha_1 = 0 \begin{cases} g(0) = f(x_k) \\ g'(0) = \nabla f(x_k) \cdot d \end{cases}$$

$$\alpha_2 \rightarrow g(\alpha_2)$$

En este caso:

$$g(\alpha) = a \cdot \alpha^2 + b \cdot \alpha + c$$

$$g'(\alpha) = 2a\alpha + b \rightarrow \alpha^* = \frac{-b}{2a}$$

$$c = g(0)$$

$$b = g'(0)$$

$$g(\alpha_2) = a \cdot \alpha_2^2 + b \cdot \alpha_2 + c \rightarrow a = \frac{g(\alpha_2) - g'(0)\alpha_2 - g(0)}{\alpha_2^2}$$

$$\boxed{\alpha^* = \frac{-g'(0)\alpha_2^2}{2(g(\alpha_2) - g'(0)\alpha_2 - g(0))}}$$

2.3.2.1.2- Interpolación cúbica

La interpolación cúbica resulta adecuada cuando se dispone de información del gradiente o de más de tres puntos.

Se ajusta una función de una variable de la forma:

$$g(\alpha) = a \cdot \alpha^3 + b \cdot \alpha^2 + c \cdot \alpha + d$$

Los extremos son las raíces de:

$$g'(\alpha) = 3a \cdot \alpha^2 + 2b \cdot \alpha + c = 0 \rightarrow \alpha^* = \frac{-b + \sqrt{b^2 - 3ac}}{3a}$$

El mínimo debe cumplir que $g''(\alpha) = 6a \cdot \alpha + 2b > 0$

Se puede hallar una solución con la información de g en cuatro puntos, o con sólo el valor del gradiente en tres puntos.

Si disponemos de la información de la función y el gradiente en dos puntos:

$$\alpha_i \begin{cases} g(\alpha_i) = f(x_k + \alpha_i d) \\ g'(\alpha_i) = \nabla f(x_k + \alpha_i d) \cdot d \end{cases} \qquad i \in [1,2]$$

$$\alpha^* = \alpha_2 - (\alpha_2 - \alpha_1) \frac{g'(\alpha_2) + \beta_2 - \beta_1}{g'(\alpha_2) - g'(\alpha_1) + 2\beta_2}$$

donde: $\beta_1 = g'(\alpha_1) + g'(\alpha_2) - 3\frac{g(\alpha_1) - g(\alpha_2)}{\alpha_1 - \alpha_2}$

$$\beta_2 = \left(\beta_1^2 - g'(\alpha_1) g'(\alpha_2)\right)^{1/2}$$

Un caso muy frecuente es disponer de la siguiente información:

$$\alpha_1 = 0 \begin{cases} g(0) = f(x_k) \\ g'(0) = \nabla f(x_k) \cdot d \end{cases}$$

$$\alpha_2 \rightarrow g(\alpha_2)$$

$$\alpha_3 \rightarrow g(\alpha_3)$$

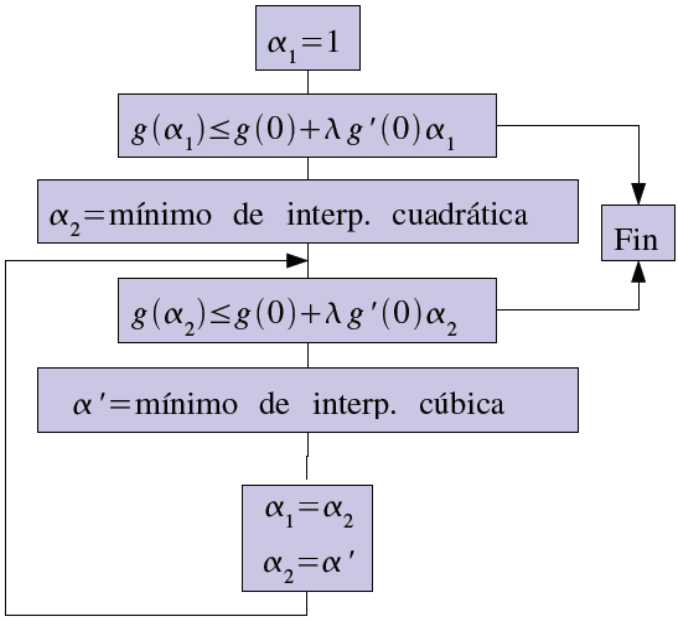
En este caso: $g'(\alpha) = 3a \cdot \alpha^2 + 2b \cdot \alpha + c = 0 \rightarrow \alpha^* = \frac{-b + \sqrt{b^2 - 3ac}}{3a}$

$$c = g'(0)$$

$$\begin{bmatrix} a \\ b \end{bmatrix} = \frac{1}{\alpha_2 - \alpha_3} \begin{bmatrix} \frac{1}{\alpha_2^2} & \frac{-1}{\alpha_3^2} \\ \frac{-\alpha_3}{\alpha_2^2} & \frac{\alpha_2}{\alpha_3^2} \end{bmatrix} \cdot \begin{bmatrix} g(\alpha_2) - g'(0)\alpha_2 - g(0) \\ g(\alpha_3) - g'(0)\alpha_3 - g(0) \end{bmatrix}$$

2.3.2.2- Implementación sencilla del Line Search

El siguiente viene a ser un esquema de la implementación del Line Search que se ha programado en el presente proyecto:



2.3.3- Métodos de Mínimos Cuadrados

El problema de mínimos cuadrados consiste en encontrar el mínimo de una función que es una suma de cuadrados:

$$\min_{x \in \Re^n} f(x) = \| K(x) \|_2^2 = \sum_{i=1}^{m} K_i(x)^2$$

Si se trata de un problema de optimización de una red de microondas utilizamos la norma 2:

$$f(x) = \| \omega(\psi)[F(x,\psi) - S(\psi)] \|_2^2 = \sum_{i=1}^{m} | \omega(\psi_i)[F(x,\psi_i) - S(\psi_i)] |^2 = \sum_{i=1}^{m} | K_i(x) |^2$$

donde F es la respuesta de la red, S la respuesta ideal, ω los pesos, ψ puede ser la frecuencia o el tiempo y $K(x) = \omega(\psi_i)[F(x,\psi_i) - S(\psi_i)]$.

Aunque se puede abordar la optimización con cualquier método, esta función objetivo presenta determinadas características que pueden ser explotadas para aumentar la eficiencia.

En *MATLAB* lo implementa la función *lsqnonlin*.

Para la descripción del método en este punto se necesitan fórmulas para las derivadas de la función, por lo que se cumple:

$$\text{La función es: } f(x) = K(x)^T K(x)$$

$$\text{El gradiente es: } \nabla f(x) = 2J(x)^T K(x)$$

$$\text{La hessiana es: } H(x) = 2J(x)^T J(x) + 2Q(x)$$

Donde:

$J(x)$ es la *Jacobiana*, que es una matriz de $m \times n$ de $k(x)$ que contiene las derivadas primeras parciales de los componentes de la función,

$$J(x) = \begin{bmatrix} \frac{\partial K_1}{\partial x_1} & \frac{\partial K_1}{\partial x_2} & \cdots & \frac{\partial K_1}{\partial x_n} \\ \frac{\partial K_2}{\partial x_1} & \frac{\partial K_2}{\partial x_2} & \cdots & \frac{\partial K_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial K_m}{\partial x_1} & \frac{\partial K_m}{\partial x_2} & \cdots & \frac{\partial K_m}{\partial x_n} \end{bmatrix} = \left[\nabla K_1, \nabla K_2, \cdots, \nabla K_m\right]^T$$

$H(x)$ es la matriz *Hessiana* de $f(x)$, que es una matriz que contiene las derivadas segundas parciales de los componentes de la función.

$$H(x) = \begin{bmatrix} \frac{\partial^2 K}{\partial^2 x_1} & \frac{\partial^2 K}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 K}{\partial x_1 \partial x_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 K}{\partial x_N \partial x_1} & \frac{\partial^2 K}{\partial x_N \partial x_2} & \cdots & \frac{\partial^2 K}{\partial^2 x_N} \end{bmatrix} \equiv \text{ es la matriz } \textit{Hessiana} \text{ de } K(x)$$

$$Q(x) = \sum_{i=1}^{m} \left(K_i(x) \cdot H_i(x)\right)$$

$H_i(x) \equiv$ es la matriz *Hessiana* de $K_i(x)$

Buscamos que el gradiente de $f(x + \Delta x)$ sea cero:

$$K(x + \Delta x) \cong K(x) + J(x)\Delta x$$

$$\begin{aligned} \nabla f(x + \Delta x) &= 2J(x + \Delta x)^T K(x + \Delta x) \\ &= 2J(x + \Delta x)^T \left[K(x) + J(x)\,\Delta x\right] \\ &\cong 2J(x)^T \left[K(x) + J(x)\,\Delta x\right] = 0 \end{aligned}$$

Por tanto hay que resolver:

$$J(x)^T K(x) + J(x)^T J(x)\Delta x = 0 \rightarrow \Delta x = -\left[J(x)^T J(x)\right]^{-1} J(x)^T K(x)$$

En cada iteración se hace una búsqueda lineal a lo largo de la dirección: $x_{k+1} = x_k + \alpha d$

En este caso

$$d = -\left[J(x_k)^T J(x_k)\right]^{-1} J(x_k)^T K(x_k)$$

La principal ventaja de este método se da cuando el mínimo de $f(x)$ tiende a cero. En este caso:

$$H(x) = 2J(x)^T J(x) + 2Q(x) \approx 2J(x)^T J(x)$$

Por tanto:

$$\begin{aligned} d &= -\left[J(x_k)^T J(x_k)\right]^{-1} J(x_k)^T K(x_k) \\ &\approx -\left(\frac{1}{2}H(x_k)\right)^{-1} \frac{1}{2}\nabla f(x_k) \\ &\approx -H(x_k)^{-1}\nabla f(x_k) \end{aligned}$$

2.3.3.1- Método Gauss-Newton

Este método es la base de los métodos que se describirán más adelante. Se basa en la implementación de las primeras derivadas de la función.

El método Gauss-Newton se basa en una aproximación lineal de los componentes de f en la proximidad de x: Para pequeñas $\|h\|$ se tiene una serie de Taylor tal que:

$$f(x+h) \cong l(h) \equiv f(x) + J(x)h$$

Insertando esto en la definición de F se puede ver que:

$$\begin{aligned} F(x+h) \cong L(h) &\equiv \frac{1}{2} l(h)^T l(h) \\ &= \frac{1}{2} f^T f + h^T J^T f + \frac{1}{2} h^T J^T J h \\ &= F(x) + h^T J^T f + \frac{1}{2} h^T J^T J h \end{aligned}$$

(con $f = f(x)$ y $J = J(x)$). El paso de Gauss-Newton h_{GN} minimiza $L(h)$, $h_{GN} = \arg\min_h \{L(h)\}$.

El gradiente y la Hessiana de L son:

$$L'(h) = J^T f + J^T J h$$

$$L''(h) = J^T J$$

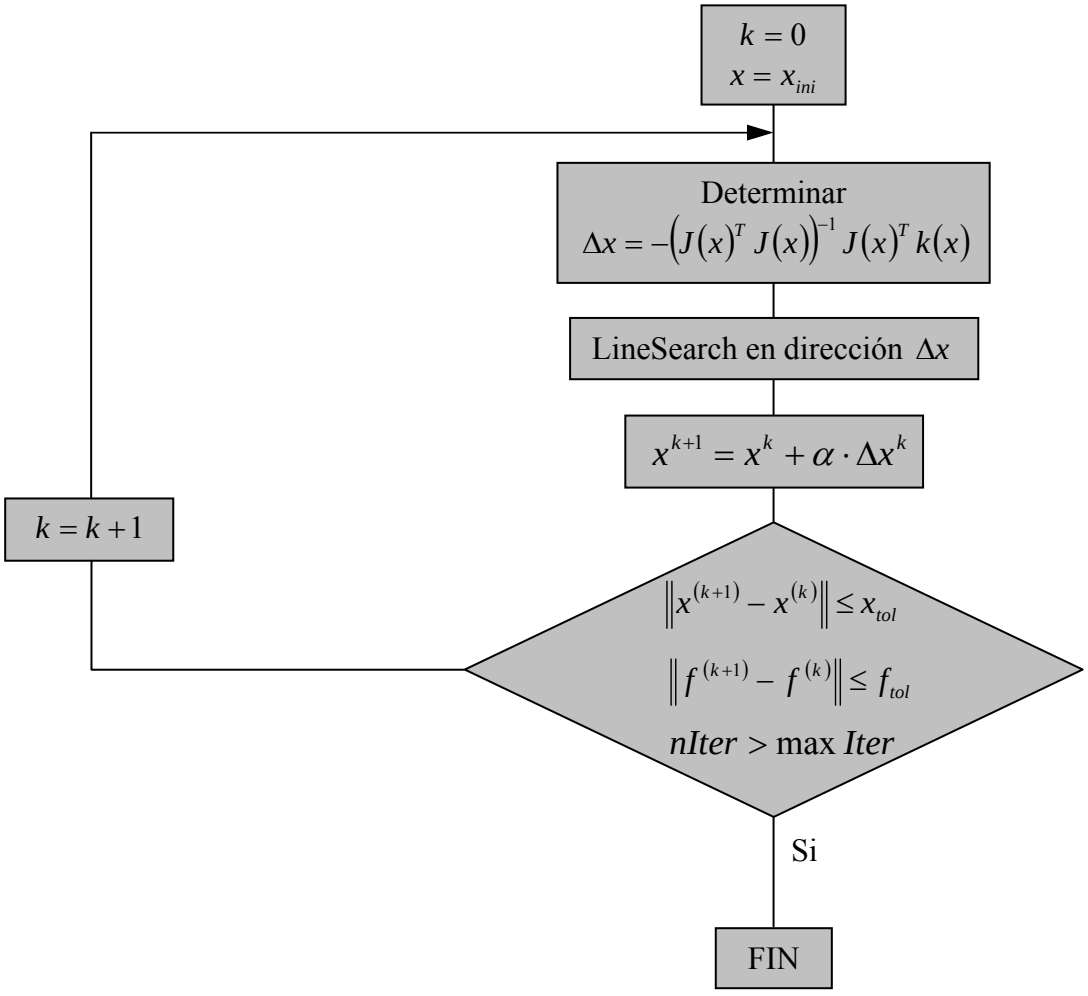
Se puede observar que la matriz $L''(h)$ es independiente de h, simétrica y se define positiva si J tiene las columnas linealmente independientes. Esto implica que $L(h)$ tiene un único mínimo que se encuentra resolviendo en cada iteración:

$$\left(J(x)^J\, J(x)\right)\Delta x = -J(x)^T\, K(x) \rightarrow \Delta x = -\left[J(x)^J\, J(x)\right]^{-1} J(x)^T\, K(x)$$

Donde se actualiza $x = x + \alpha h_{GN}$ en cada iteración y α se encuentra por *Búsqueda Lineal*. El método *Gauss-Newton* usa $\alpha = 1$ en todos los pasos.

La dirección Δx se usa en un proceso de LineSearch para asegurar que $f(x)$ decrece.

El diagrama de flujo de este método es el siguiente:



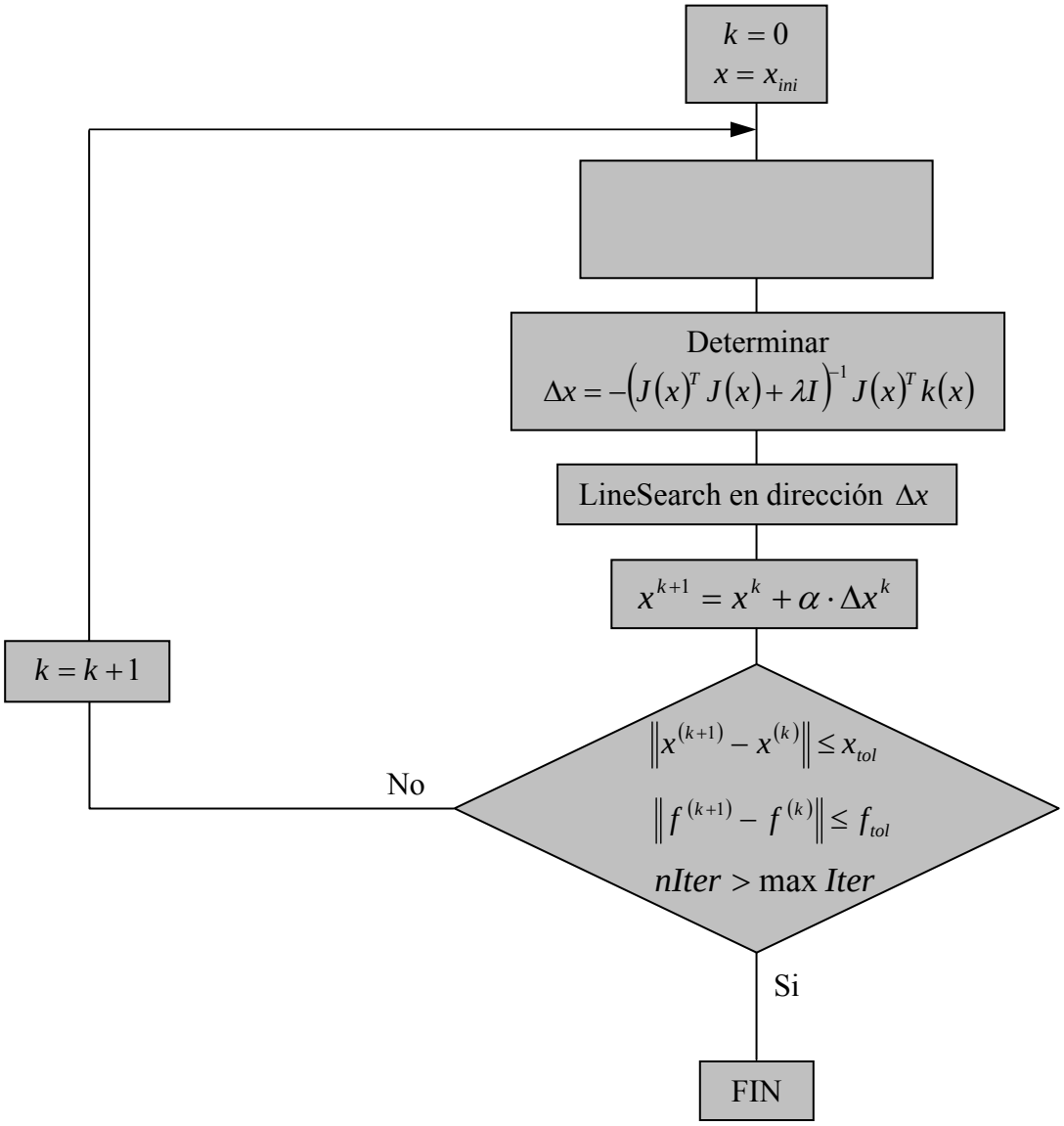
2.3.3.2- Método Levenberg-Marquardt

El método Levenberg-Marquardt es una variante del método Gauss-Newton, por lo que el paso h_{LM} se define modificando el paso del método Gauss-Newton en cada iteración de la siguiente manera:

$$\left(J(x)^J\, J(x)+\lambda I\right)\Delta x=-J(x)^T\, K(x)\rightarrow \Delta x=-\left(\left(J(x)^J\, J(x)+\lambda I\right)\right)^{-1}\cdot J(x)^T\, K(x)$$

con $\lambda \geq 0$

El diagrama de flujo del método Levenberg-Marquardt es el siguiente:



2.3.3.2.1- Cálculo del parámetro de amortiguamiento λ

Usualmente se eligen valores iniciales grandes de λ, los cuales se disminuyen a medida que se aproxima al mínimo.

Los términos "grande" y "pequeño" deben ir relacionados con el tamaño de los elementos en $J(x)^T J(x)$. Esta matriz es simétrica y positiva, implicando que los valores propios de $\lambda(\lambda_i)$ deben ser reales y no negativos; y los correspondientes vectores propios (v_i) pueden ser elegidos como una base ortonormal de $\Re^{ns}$.

La elección del parámetro λ influye tanto en la dirección de búsqueda como en el tamaño del paso. Por lo que si λ tiende a cero, se tiene la misma dirección de avance que en el método de Gauss-Newton., y si λ tiende a ∞, Δx tiende a un vector de ceros en la dirección de máxima pendiente. Esto asegura que para valores altos de λ conseguiremos disminuir $f(x)$.

La solución para calcular el paso por el método de Levenberg- Marquardt puede escribirse de la forma:

$$\Delta x = \frac{-J(x)^T K(x)}{\left(J(x)^T J(x) + \lambda I\right)}$$

Como consecuencia, es razonable relacionar el valor inicial de $\lambda(\lambda_o)$ con el tamaño de los valores propios. El máximo de los elementos de la diagonal en el inicial $J(x)^T J(x)$ tiene el mismo orden de magnitud que $\max(\lambda_i)$, así que una simple estrategia para elegir λ_o vendría dada por:

$$\lambda_0 = \tau \cdot \max\left\{\left(J(x_0)^T J(x_0)\right)_{ii}\right\}$$

donde un pequeño valor de τ sería usado si pensamos que x_0 está próximo a la solución. Los valores de τ normalmente suelen estar en el intervalo $\left[10^{-8}, 1\right]$. En las simulaciones que he realizado he tomado un valor de τ de 10^{-3}, que es el valor estándar que suele tomarse.

A la hora de actualizar el parámetro λ, básicamente hay 2 tipos de estrategias: la primera está relacionada con la solución para calcular el paso por el método de Levenberg- Marquardt que se acaba de describir. Se encuentra un valor del paso mediante Line Search y se usa esta información para actualizar el valor de λ. Una búsqueda lineal puede implicar bastantes evaluaciones extra de $f(x)$ que no se acerquen a la solución.

El otro tipo de estrategias está basada en el hecho de que la variable λ influye tanto en la dirección d como en el paso Δx, con lo cual se puede implementar el método sin necesidad de una búsqueda lineal explícita.

El método de Gauss-Newton está basado en el modelo obtenido de la expansión de Taylor de $K(x)$:

$$K(x + \Delta x) \approx l(\Delta x) \equiv K(x) + J(x) \cdot \Delta x$$

Si $f(x+\Delta x) < f(x)$, entonces, como $x_{k+1} = x_k + \alpha \cdot d$, se tiene:

- Para $\alpha = 1 \rightarrow x_{k+1} = x_k + d$.
- Para $\alpha = 0 \rightarrow x_{k+1} = x_k$, por lo que λ aumenta.

Con esta estrategia se usaran los siguientes criterios de parada del algoritmo:

$$\| f'(x) \| \leq \varepsilon_1,$$
$$\| \Delta x \| \leq \varepsilon_2 \| x \|,$$
$$k \geq k_{\max}$$

Por otra parte, lo que se busca es que λ disminuya cuando el punto x está cerca de la solución, lo cual se hará suponiendo que cuando el paso es muy pequeño se puede dar por buena la aproximación siguiente:

$$f(x+\Delta x) \approx L(\Delta x) \equiv l(\Delta x)^T \cdot l(\Delta x)$$

La actualización de λ se controla mediante un "factor ganancia":

$$G = \frac{f(x) - f(x+\Delta x)}{L(0) - L(\Delta x)}$$

De la fórmula de Δx y la aproximación realizada en el modelo obtenido de la expansión de Taylor de $K(x)$ se obtiene:

$$\begin{aligned} L(0) - L(\Delta x) &= -2(\Delta x)^T J(x)^T K(x) - (\Delta x)^T J(x)^T J(x) \cdot \Delta x \\ &= -(\Delta x)^T \cdot \left[2 f'(x) + \left(J(x)^T J(x) + \lambda I - \lambda I \right) \cdot \Delta x \right] \\ &= (\Delta x)^T \cdot \left[\lambda \Delta x - f'(x) \right] \end{aligned}$$

Como tanto $\Delta x \cdot \lambda \cdot \Delta x \cdot (\Delta x)^T$ como $-(\Delta x)^T \cdot f'(x)$ son positivas, se garantiza que el denominador de G también lo es y como $f(x)$ es decreciente, si se tiene un valor elevado de G se puede disminuir el valor de λ para que en la siguiente iteración el paso Levenberg-Marquardt se parezca al paso Gauss-Newton.

Si $G \leq 0$, entonces se incrementa λ con el fin de acercarse al método de máxima pendiente y reducir la longitud del paso. Del mismo modo, si G es mayor que 0 pero muy pequeña, podría ser mejor utilizar un λ mayor al calcular el paso en la siguiente iteración.

Tal y como propone Mardquardt en [12], muchas implementaciones de esta estrategia se realizan del siguiente modo:

- Si $G < G_1$, entonces $\lambda = \beta\lambda$
- Si $G > G_2$, entonces $\lambda_{new} = \frac{\lambda}{\gamma}$
- Si $G > 0$, entonces $x = x + \Delta x$

Donde $0 < G_1 < G_2$ y β, $\gamma > 1$.

Este método es muy sensible a pequeños cambios en los valores de estos parámetros, unos valores bastante populares y utilizados en este proyecto son:

$$G_1 = 0.25$$

$$G_2 = 0.75$$

$$\beta = 2$$

$$\gamma = 3$$

Finalmente se ha optado por una nueva estrategia basada en la propuesta de Mardquardt, pero que evita los saltos en $\frac{\lambda_{new}}{\lambda}$ entre G_1 y G_2. Además, si $G < 0$, al calcular pasos consecutivos se puede permitir que λ crezca más rápidamente:

- Si $G > 0$, entonces: $x = x + \Delta x$, $\lambda = \lambda \cdot \max\left\{\frac{1}{\gamma}, 1 - (\beta - 1)(2G - 1)^p\right\}$ y $\eta = \beta$
- Si $G \leq 0$, entonces: $\lambda = \lambda\eta$ y $\eta = 2\eta$

Con η inizializada a β y siendo p un entero impar (se ha tomado $p = 3$).

2.3.3.3- Método Quasi-Newton

Los métodos Quasi-Newton hallan la matriz Hessiana (H) a partir de la evolución de $f(x)$ y del gradiente, en vez de numéricamente como se hace en el método Gauss-Newton, evitándose así un número excesivo de evaluaciones de la función objetivo.

En este método, las expresiones de los gradientes se aproximan por diferencias finitas y la matriz origen, $H^{(0)}$, utilizada en la primera iteración, se suele tomar la matriz identidad, aunque también se puede usar cualquier matriz definida positiva, esto asegura avanzar en sentido descendente.

La técnica más eficiente de actualización de H es la de *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) que propone aproximar la Hessiana en la siguiente iteración por la matriz definida positiva H_{k+1}, cuya expresión es la siguiente:

$$H_{k+1} = H_k + \frac{q_k q_k^T}{q_k^T s_k} - \frac{H_k s_k s_k^T H_k}{s_k^T H_k s_k}$$

donde:

$$s_k = x_{k+1} - x_k$$

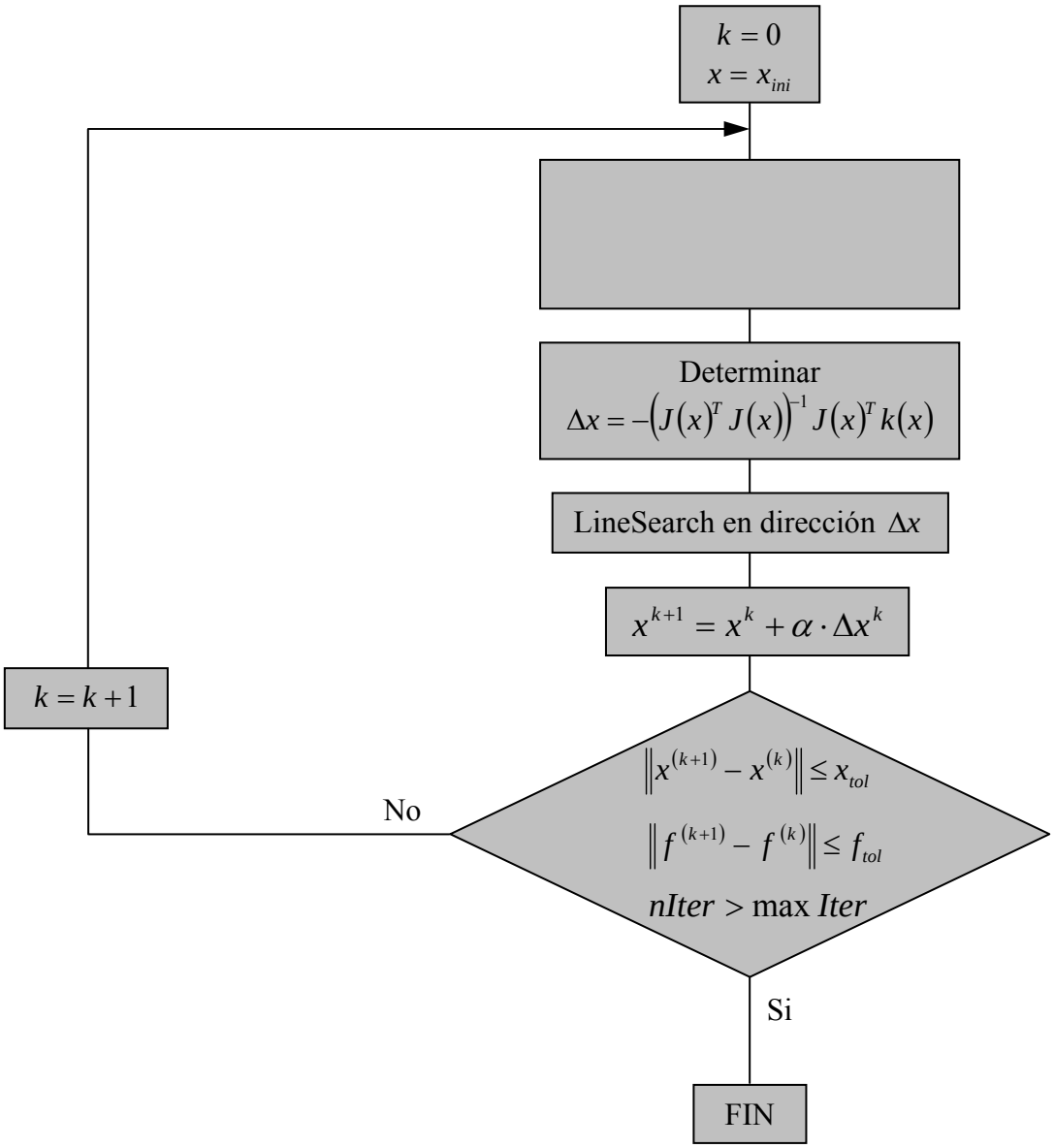
$$q_k = \nabla f(x_{k+1}) - \nabla f(x_k)$$

Como en esta expresión es necesario invertir dicha matriz H, se puede optar por aproximar H^{-1}, esto es lo que hace el *Método de Davidon-Fletcher-Powell*, y la expresión de dicha aproximación es la siguiente:

$$H_{k+1}^{(-1)} = H_k^{(-1)} + \frac{s_k s_k^{\ T}}{s_k^{\ T} q_k} - \frac{H_k^{(-1)^T} q_k^{\ T} q_k H_k^{(-1)}}{q_k^{\ T} H_k^{(-1)} q_k}$$

Experimentalmente se ha comprobado que la recurrencia de *BFGS* funciona mejor que la de *DFP*.

El diagrama de flujo de este método con la actualización *BFGS* es el siguiente:



La técnica de actualización de H defendida por *Al-Baali* y *Fletcher* propone aproximar la Hessiana en la siguiente iteración por la expresión siguiente si se cumple que $s_k{}^T y_k > 0$:

$$H_{k+1} = H_k + \left(\frac{1}{s_k{}^T y_k} y_k \right) y_k{}^T - \left(\frac{1}{s_k{}^T v_k} v_k \right) v_k{}^T$$

donde:

$$s_k = x_{k+1} - x_k$$

$$y_k = J_{k+1}{}^T J_{k+1} s_k + \left(J_{k+1} - J_k \right)^T k\left(x_{k+1} \right)$$

$$v_k = H_k s_k$$

2.3.3.4- Método Híbrido: Levenberg-Marquardt y Quasi-Newton

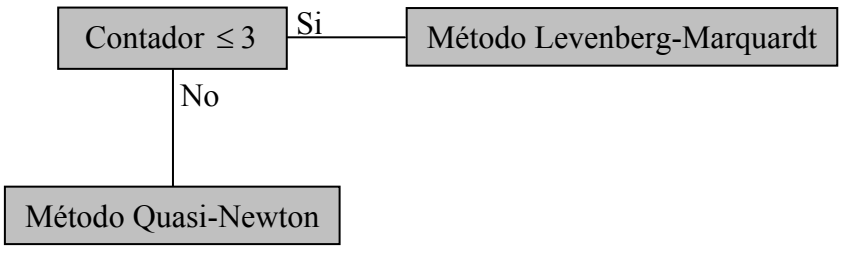
Un método híbrido es una combinación de varias estrategias de optimización. La idea es utilizar en cada caso el algoritmo más adecuado.

La mejor combinación de los algoritmos de optimización depende de la proximidad del punto inicial a los valores óptimos, como se ha comentado. Así, por ejemplo, se puede empezar con un método robusto que no tenga una gran dependencia con el punto inicial, como los métodos de búsqueda directa, y cuando el punto obtenido esté cerca del mínimo, se conmuta a un algoritmo de gradiente para acelerar la convergencia.

Además de seleccionar los algoritmos a utilizar, hay que establecer mecanismos para conmutar de un tipo de algoritmo a otro, los cuales se basan en fijar un número máximo de evaluaciones de la función objetivo o una tolerancia para los parámetros o para la función de error. Con la estrategia de optimización híbrida se consigue aumentar la robustez y la rapidez en la convergencia.

En este trabajo se presenta un método híbrido combinando el método Levenberg-Marquardt con el método Quasi-Newton. Para empezar la iteración suponemos que el punto está lejos de la solución por lo que se empieza con la robustez del método Levenberg-Marquardt. Si en tres pasos la representación indica que $F(x^*) \neq 0$, entonces se conmuta al método Quasi-Newton para mejorar la representación.

El siguiente diagrama de flujo es el del método híbrido:



CAPÍTULO 3

Diseño de dispositivos pasivos de microondas

Los dispositivos pasivos de microondas son los elementos clave en la mayoría de los sistemas de comunicaciones. En concreto, los filtros en alta frecuencia son de gran importancia debido a su empleo en muchas de las aplicaciones de las telecomunicaciones, como, por ejemplo, los sistemas de acceso inalámbrico, sistemas de comunicaciones móviles o sistemas de comunicaciones vía satélite. Por ello, el diseño de estos filtros es uno de los temas fundamentales en el desarrollo de cualquier sistema de comunicaciones; pero no es una tarea fácil, ya que la aparición de nuevas tecnologías y servicios han incrementado las necesidades de estos dispositivos, haciendo su diseño cada vez más complejo.

De este modo, el desarrollo de nuevas estructuras para la implementación de filtros en altas frecuencias se basa en cumplir una serie de especificaciones, entre las que destacan: una masa y volumen reducidos, una buena estabilidad térmica para su uso en aplicaciones con altas potencias, y un incremento de la banda libre de frecuencias espúreas o banda de rechazo. Todo ello, junto con la necesidad de una reducción del esfuerzo de fabricación y la disponibilidad de herramientas para su análisis y diseño.

Los filtros evanescentes en guía rectangular cargados con resonadores dieléctricos proporcionan una buena respuesta para las necesidades comentadas, ya que ofrecen una reducción en masa y volumen de alrededor del 50% y una mayor banda libre de espúreas, respecto a los filtros en tecnología guiada compuestos de sólo metal, así como una gran estabilidad térmica para aplicaciones de alta potencia. Además, si los resonadores dieléctricos son circulares, los filtros presentan una fabricación sencilla y una mayor capacidad de manejo de potencia. Sin embargo, el modelado exacto de este tipo de filtros es mucho más complejo que el de las estructuras metálicas convencionales, puesto que la geometría incluye partes circulares y rectangulares, metálicas y dieléctricas. Por esta razón, el diseño automatizado CAD (Diseño Asistido por Computador) de estos filtros está siendo objeto de una intensa investigación en los últimos años.

En este capítulo se explican, de forma general, las estrategias y métodos más comunes en el diseño de dispositivos pasivos de comunicaciones en alta frecuencia, en especial los filtros de microondas.

3.1- Introducción

Una vez que se verifica el funcionamiento de los algoritmos programados con funciones de test, se insertan en el software de diseño del grupo de investigación, para proceder a aplicarlo en las distintas fases del proceso de diseño, de modo que se minimice la función de error entre la respuesta ideal del dispositivo y su respuesta real.

Antes de exponer los resultados obtenidos, conviene explicar brevemente en qué se basa este proceso de diseño, para así comprenderse bien todas las medidas realizadas.

Las técnicas clásicas de diseño de dispositivos de microondas se basan en herramientas de análisis, en las que el diseñador propone una determinada estructura y la analiza mediante alguna de estas herramientas. Si el resultado no es el deseado, entonces el diseñador modifica la estructura, según su intuición, y la vuelve a analizar, esto provoca que el proceso sea muy lento. Este proceso iterativo finaliza cuando los resultados obtenidos cumplen las especificaciones iniciales, lo cual, en ocasiones, es muy difícil.

Por ello, actualmente, los dispositivos de microondas se obtienen mediante un diseño automatizado, realizado a través de una herramienta CAD que intenta evitar al máximo la intervención humana durante el proceso de diseño, mediante el uso de algoritmos de optimización. De todas formas, cualquier herramienta CAD necesita una herramienta de análisis exacta y eficiente para simular la estructura, como pueden ser los métodos de análisis aplicados en las técnicas clásicas.

Los métodos para el análisis de los dispositivos de microondas existentes en la actualidad, pueden clasificarse en tres grandes grupos:

- Los métodos *analíticos* o *modales:* proporcionan resultados muy exactos y de forma eficiente desde un punto de vista computacional. No obstante, son sólo aplicables a unos pocos problemas canónicos con geometrías regulares.
- Los métodos *numéricos* o *de discretización espacial:* son capaces de analizar problemas con geometrías arbitrarias, pero su desventaja es el elevado consumo de memoria y de tiempo de CPU.
- Los métodos *híbridos:* intentan aprovechar las ventajas de cada uno de los grupos anteriores, es decir, la eficiencia de los métodos modales y la flexibilidad de los métodos numéricos. Aunque su eficiencia es mayor que la de los métodos numéricos, dichos métodos resultan demasiado lentos para analizar una estructura tan compleja como los filtros con resonadores dieléctricos.

Por tanto, ninguno de los tres tipos de métodos puede por sí solo analizar estructuras complicadas, como las que contienen resonadores dieléctricos, de forma suficientemente eficiente para abordar costosos procesos de diseño computacional.

Por ello, el análisis de estas estructuras se hace normalmente distinguiendo las partes en las que están formadas, para aplicar en cada una de ellas el método de análisis que más se ajusta a sus características, es decir, las partes correspondientes a estructuras con geometrías regulares, se analizan de forma eficiente con un método analítico, y en las partes con geometrías arbitrarias o demasiado complejas para los métodos analíticos, como los postes dieléctricos, se aplica un método híbrido.

Este análisis sirve para simular la estructura durante el proceso de diseño y, a partir de los datos proporcionados, construir la función de error utilizada en los algoritmos de optimización. Esta función de error depende, normalmente, de las dimensiones físicas del dispositivo, las cuales constituyen los parámetros de diseño considerados en la optimización. Cuando se diseña una estructura compleja, el número de parámetros aumenta, haciendo poco eficiente la optimización de todos los parámetros a la vez, por lo que habrá que aplicar una estrategia de segmentación.

Las *estrategias de segmentación* consisten en dividir la estructura bajo análisis en un conjunto de bloques. De este modo, en cada paso se diseñan los parámetros de cada bloque utilizando los valores obtenidos en los anteriores pasos, es decir, se van añadiendo los bloques sucesivamente. Tras haber optimizado la estructura por bloques, se optimizan todos los parámetros de diseño partiendo del punto inicial dado por las dimensiones obtenidas en los pasos anteriores. Estos pasos se repiten para todos los bloques, salvo si la estructura es simétrica, en cuyo caso sólo se optimiza la mitad del filtro. De este modo, una vez se han obtenido las dimensiones de la mitad del filtro, se construye el filtro completo aplicando la propiedad de simetría, y realizando, posteriormente, una optimización en la que sólo los parámetros del bloque central se ajustan ligeramente.

Por tanto, realizando segmentación se aumenta la eficiencia del proceso, la cual puede incrementarse aún más si la estructura presenta simetría, ya que así se puede reducir el número de pasos del proceso.

Sin embargo, si el diseño contiene cavidades o elementos resonantes, existe el riesgo de que el acoplo entre todas las cavidades, no se diseñe adecuadamente, haciendo que este acoplo no se tenga en cuenta. Esto provoca que el último paso de refinamiento de las dimensiones sea muy costoso debido a que el punto inicial proporcionado por el diseño de los distintos bloques está lejos de los valores óptimos, al no considerar todos los acoplos existentes. Por ello, se suelen hacer pasos intermedios de refinamiento, con el fin de mejorar el punto inicial de la última optimización.

En cada paso de las estrategias seguidas, se aplican algoritmos de optimización durante el proceso de diseño. La idea fundamental de estas técnicas de diseño es la definición de una función de error entre la respuesta ideal y la respuesta real del dispositivo. Una vez definida esta función de error, el objetivo es encontrar la topología física del dispositivo para la cual la respuesta real se parece lo máximo posible a la ideal.

Existen varias formas de construir esta función de error, que será la función objetivo a minimizar. La elección de cualquiera de ellas dependerá de la geometría del dispositivo, de su margen dinámico y de otros factores, pues hay que tener en cuenta que, en un diseño real no todas las configuraciones son posibles. La función objetivo tipo para la optimización de redes de microondas (como se ha dicho anteriormente) es:

$$f(x) = \max_{\{\psi_l, \psi_u\}} \{\omega_u(\psi)[F(x,\psi) - S_u(\psi)], \omega_l(\psi)[F(x,\psi) - S_l(\psi)]\}$$

donde:

x : Vector de parámetros del dispositivo

ψ : Variable independiente, normalmente *frecuencia* o *tiempo*

$F(x,\psi)$: Respuesta real del dispositivo

$S_u(\psi)$: Especificación superior

$S_l(\psi)$: Especificación inferior

$\omega_u(\psi)$: Ponderación o peso para $S_u(\psi)$

$\omega_l(\psi)$: Ponderación o peso para $S_l(\psi)$

ψ_u : Límite superior para ψ (frecuencia o tiempo)

ψ_l : Límite inferior para ψ

Un caso especial de esta formulación ocurre cuando se cumple:

$$S_u(\psi) = S_l(\psi) = S(\psi)$$
$$\omega_u(\psi) = \omega_l(\psi) = \omega(\psi)$$

Con lo que se tiene la función objetivo de tipo Chebyshev, cuya formulación es

$$f(x) = \max_{\{\psi_l,\psi_u\}} \left\{ \left| \omega(\psi)\left[F(x,\psi) - S(\psi)\right] \right| \right\} = \max_{\{\psi_l,\psi_u\}} \left\{ \left| e(x) \right| \right\}$$

donde $e(x) = \omega(\psi)\left[F(x,\psi) - S(\psi)\right]$

Normalmente, estas dos funciones objetivo no se suelen utilizar ya que pueden presentar derivadas discontinuas, al tener las especificaciones $S_u(\psi)$ y $S_l(\psi)$ una variación muy abrupta.

A parte de estas funciones, que utilizan una norma de tipo mínimas, existen otras basadas en, por ejemplo, la *norma p-ésima* o una de las más utilizadas en el diseño de dispositivos de microondas, la *norma Huber*, la cual viene definida por:

$$f(x) = H\left(\omega(\psi)\left[F(x,\psi) - S(\psi)\right]\right) = H(e_i(x)) = \sum_{i=1}^{m} \rho_k(e_i(x))$$

donde:

$$\rho_k(e_i) = \begin{cases} \dfrac{e_i^2}{2} & si \ \ |e_i| \le k \\ k|e_i| - \dfrac{k^2}{2} & si \ \ |e_i| > k \end{cases}$$

Partiendo de la expresión anterior, se observa que la norma *Huber* es una combinación de una función lineal o de norma uno, cuando el error es grande, y una función de segundo orden o norma cuadrática para errores pequeños, habiendo una transición suave entre las dos normas. La diferencia entre errores pequeños y grandes la marca la variable *k*.

A parte de las funciones de error comentadas, existen otras que pueden consultarse en [2].

Una vez seleccionada la función de error a aplicar en el diseño, hay que minimizarla mediante algún algoritmo de optimización, con el fin de hallar los valores de sus variables, normalmente las dimensiones del filtro, para las que la diferencia entre la respuesta real e ideal es minima. Sin embargo, en este proceso de optimización hay que tener en cuenta que las dimensiones del dispositivo no pueden tomar cualquier valor, sino que estarán acotadas. Esto implica la necesidad de algoritmos en los que se puedan introducir restricciones como los *algoritmos genéticos* o *evolutivos* [5], que permiten establecer límites inferior y superior a los parámetros de diseño, pero que convergen muy lentamente.

Los algoritmos de optimización que no consideran restricciones se utilizan cuando se dispone de una idea inicial de los parámetros lo suficientemente cercana a los valores óptimos para mantener al algoritmo dentro de la región permitida, ya que, al no considerar restricciones, siempre buscan el mínimo global que puede estar fuera de esta región, dando valores sin sentido físico, como, por ejemplo, longitudes negativas.

Con esta se puede observar que no hay un algoritmo perfecto para minimizar la función de error del dispositivo, sino que, tienen sus ventajas e inconvenientes. Por ello, se suele aplicar una *optimización híbrida*, como por ejemplo, el Mapeado espacial agresivo [5] (ASM).

En resumen, el proceso de diseño de dispositivos de microondas utilizado en la actualidad, se basa, en general, en seleccionar tres tipos de estrategias:

- En primer lugar se tiene que decidir si se aplica la técnica de mapeado espacial agresivo o una optimización directa.
- En segundo lugar se tiene que decidir si se considera toda la estructura o se aplica una estrategia de segmentación.
- Y por último, si se utiliza uno o varios algoritmos de optimización, aplicando así una optimización híbrida.

Por lo tanto, como ya se ha dicho anteriormente, el diseño sólo trata de conseguir la respuesta del dispositivo lo más parecida a la ideal, lo cual, en la mayoría de casos, se basa en la comparación del coeficiente de transmisión (S_{21}) o el de reflexión (S_{11}) de las respuestas ideal y real.

Tras haber presentado el proceso de diseño de forma genérica, se explica, a continuación, el proceso aplicado en este trabajo para el diseño de filtros evanescentes en guía rectangular cargados con resonadores dieléctricos. Otros ejemplos de diseño de dispositivos de microondas se pueden consultar en [2].

3.2. Diseño de filtros evanescentes con resonadores dieléctricos

Como se ha comentado al inicio, los filtros de alta frecuencia constituyen una parte fundamental en la mayoría de las aplicaciones de las telecomunicaciones. Las necesidades de los actuales sistemas de comunicaciones han aumentado la complejidad del diseño de estos filtros, motivando el desarrollo de nuevas estructuras para su implementación. Estas necesidades se centran en conseguir una masa y volumen reducidos, un incremento de la banda libre de frecuencias espurias o banda de rechazo, y una buena estabilidad térmica en aplicaciones de alta potencia.

El uso de resonadores dieléctricos en filtros de microondas está aumentando debido a que proporcionan una importante reducción en la masa y el volumen del filtro, comparado con la tecnología sólo metal, además de conseguir una mayor estabilidad térmica en aplicaciones de alta potencia; con lo que cumplen dos de las necesidades comentadas. No obstante, el análisis de los campos eléctricos en estas estructuras es más complejo debido a la presencia de material dieléctrico, por lo que, actualmente, se están desarrollando diferentes herramientas de análisis por ordenador para obtener un modelado exacto de estas estructuras.

Existen diferentes tipos de filtros cargados con resonadores dieléctricos que cumplen las especificaciones establecidas anteriormente y que están basadas en las estructuras metálicas utilizadas para implementar filtros. Entre estas estructuras, se encuentran los *filtros de cavidades acopladas sin resonadores*, los *filtros evanescentes con postes metálicos*, los *filtros de cavidades acopladas con resonadores dieléctricos* y los *filtros evanescentes con postes dieléctricos*. Este último tipo de filtro es el que mejores características presenta.

Los *filtros evanescentes en plano H con resonadores dieléctricos* tienen una longitud y volumen más reducidos comparado con los filtros plano H de cavidades acopladas todo metal y los de cavidades acopladas con resonadores dieléctricos circulares. Además, este tipo de filtros presenta un mejor rechazo y una banda de espurias más estrecha que los filtros con cavidades acopladas con y sin resonadores dieléctricos.

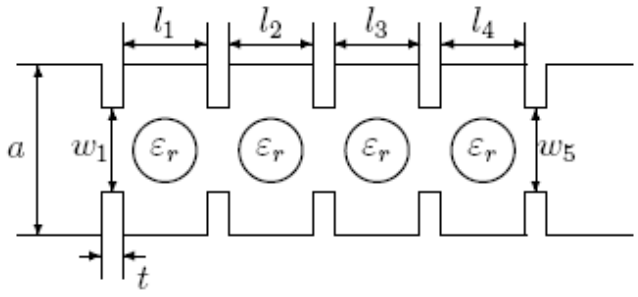


Figura 5.11: Filtro plano H de cuatro cavidades con resonadores dieléctricos. Vista superior

Otra ventaja de los filtros evanescentes con resonadores dieléctricos, a parte de una buena estabilidad térmica por el uso de estos resonadores, es un menor riesgo de multipactor, lo que los hace muy útiles para las aplicaciones de alta potencia. Esto se debe a que los postes dieléctricos se sitúan normalmente en el centro de la guía rectangular donde el campo eléctrico es máximo, aumentando de este modo la potencia máxima. De hecho, al ser la posición de los postes muy influyente en la distribución de los campos eléctricos y magnéticos, se han estudiado varias estructuras como los filtros con postes dieléctricos descentrados o los filtros periódicos con postes dieléctricos.

Por otra parte, los resonadores dieléctricos de estos filtros evanescentes pueden ser circulares o cuadrados, siendo mejor el uso de resonadores circulares por reducir drásticamente el esfuerzo de fabricación y por presentar una longitud y volumen un poco inferiores respecto a los resonadores cuadrados.

Sin embargo, los filtros evanescentes cargados con resonadores dieléctricos circulares están compuestos por dos guías rectangulares, de entrada y de salida, y una guía rectangular al corte que alberga los resonadores dieléctricos, como se puede observar en la figura 4.3. Por lo que deben analizarse geometrías circulares y cuadradas juntas, lo cual hace que el modelado de los resonadores dieléctricos circulares es mucho más complejo que el de los cuadrados.

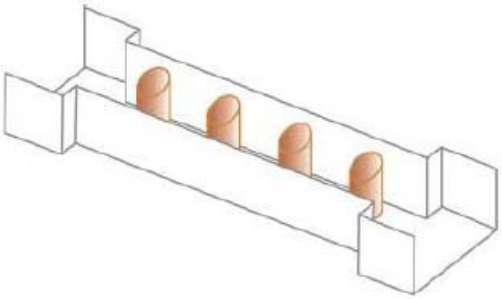


Figura 4.3: Filtro evanescente en plano H
con resonadores dieléctricos circulares

El primer paso en el proceso de diseño de estos filtros, es determinar los parámetros de diseño, que son: la anchura de la guía central (w), el radio de los postes dieléctricos (r_i), las distancias entre la guía de entrada y el primer resonador (l_1), y entre el último y la guía de salida (l_{n+1}), y las distancias entre los resonadores (l_i). Para ello, se fijan los parámetros que no son objeto de diseño, es decir, la altura de toda la estructura, la anchura de las guías de entrada y salida y la permitividad relativa de los resonadores dieléctricos.

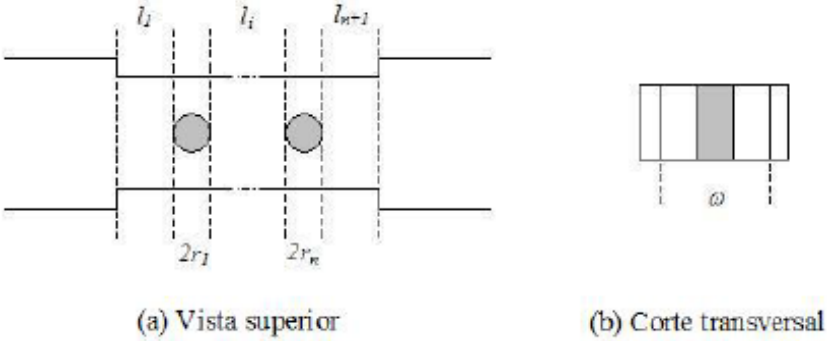


Figura 4.4: Parámetros de diseño

Cada uno de estos parámetros tiene su efecto sobre la respuesta del filtro. Así, la anchura de la guía central, w, determina principalmente el ancho de banda del filtro y, en segundo orden, la banda libre de frecuencias espurias y la capacidad de manejo de potencia, es decir, la potencia máxima de operación. El ancho de banda también puede ser controlado con las ventanas de acoplo dadas las distancias l_i, las cuales controlan, a su vez, las pérdidas de inserción. Por otra parte, modificando el tamaño de los postes dieléctricos se varían las frecuencias de resonancia.

En el análisis de la estructura en el proceso de diseño se distinguen las dos partes de estos filtros, por un lado se analizan con un método modal los tramos de guía existentes entre los resonadores dieléctricos y por otro mediante un método híbrido se analizan los resonadores. A partir de los datos obtenidos en este análisis, se forman las funciones objetivo que se utilizarán en los algoritmos de optimización, los cuales modificarán los parámetros de diseño para cumplir en todo lo posible las especificaciones establecidas.

Si el número de postes dieléctricos necesario es elevado, el número de parámetros considerados en la optimización también será elevado, por lo que se tendrá que utilizar una estrategia de segmentación en la fase de optimización. Esta estrategia se basa en la explicada en el apartado anterior, de este modo, en primer lugar, se optimizarán las dimensiones del primer resonador y las guías de alrededor, comparando la respuesta del filtro real y la respuesta ideal de un filtro evanescente con un solo resonador. Tras este paso, el proceso continúa añadiendo sucesivamente los resonadores dieléctricos necesarios, utilizando los valores ya obtenidos. También se pueden aplicar pasos intermedios de refinamiento de las dimensiones halladas para tener en cuenta los acoplos existentes. Por último se optimiza toda la estructura para conseguir las características del filtro ideal ajustando ligeramente todos los parámetros del filtro.

El proceso de diseño de los filtros evanescentes en plano H con postes dieléctricos se puede beneficiar de la simetría que presenta esta estructura. De este modo, una vez se han obtenido las dimensiones de la mitad del filtro, se construye el filtro completo aplicando esta propiedad de simetría, según la cual $l_{n+1} = l_1$, $l_n = l_2$, …, y $r_n = r_1$, $r_{n-1} = r_2$, …, y se compara la respuesta que proporciona la herramienta de simulación con la respuesta ideal que se pretende conseguir. Para ajustar ambas respuestas, en un principio se optimizan ligeramente los parámetros geométricos del filtro situados en las posiciones centrales del mismo, y si fuera necesario se continúan ajustando los parámetros más alejados de dicho centro. Todo ello, antes del paso final de refinamiento.

Tras este proceso de diseño, el filtro obtenido ofrece los parámetros $|S_{11}|$ y $|S_{21}|$ que se muestran en la figura 5.12. La curva azul corresponde al comportamiento de la estructura diseñada, es decir, al comportamiento real del dispositivo, la verde corresponde a la respuesta ideal del dispositivo y la roja son las pérdidas de retorno en el caso del parámetro S_{11} y las pérdidas de inserción en el caso del parámetro S_{21}.

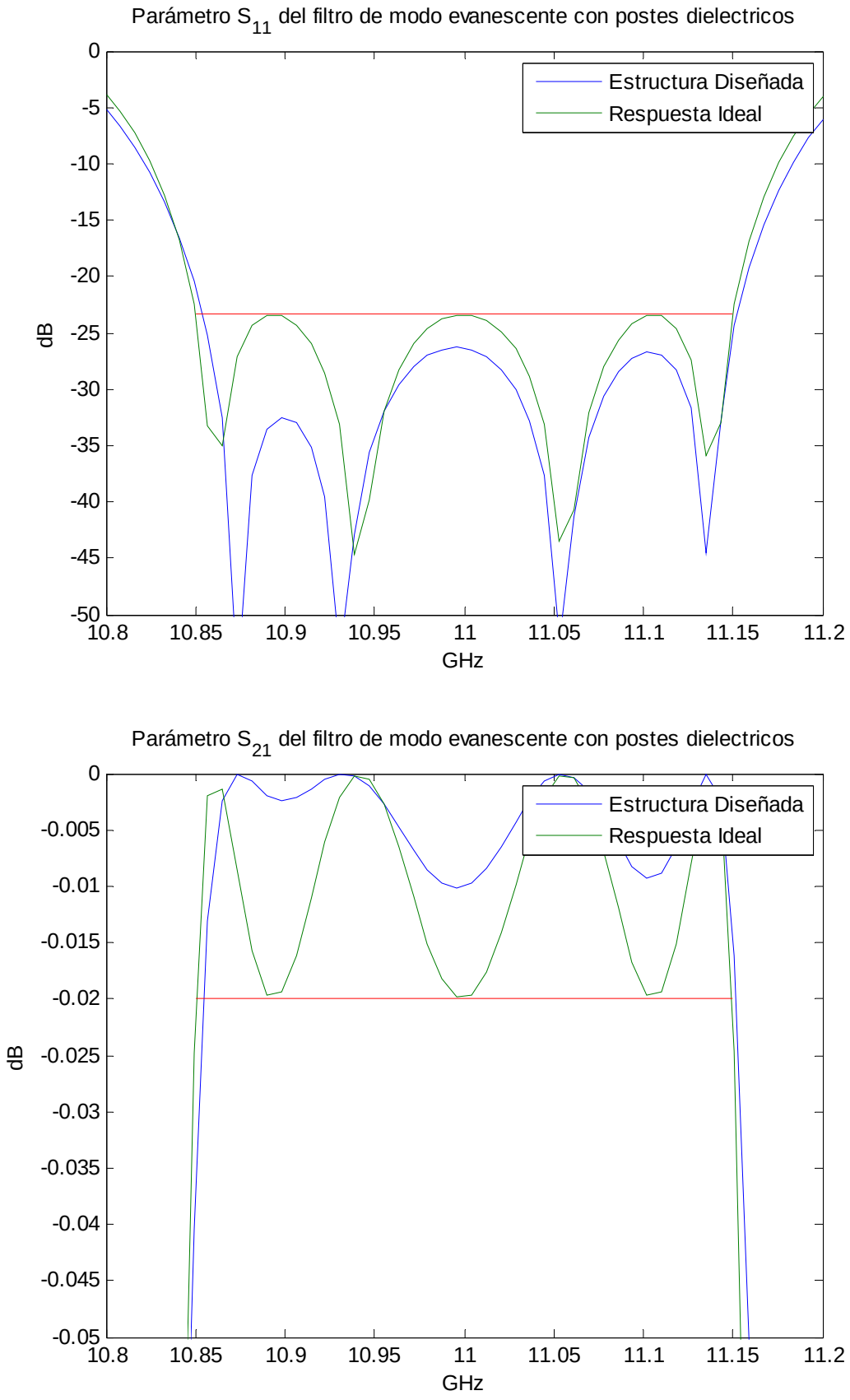


Figura 5.12: Parámetros S_{11} y S_{21}

CAPÍTULO 4

Resultados

En este capítulo los resultados se dividen en dos apartados, por un lado se muestran los resultados obtenidos por los programas realizados para los métodos de mínimos cuadrados y por otro se presentan los resultados obtenidos en el diseño de filtros evanescentes con postes dieléctricos.

4.1- Resultado del método de los mínimos cuadrados

A continuación se muestran las pruebas de cada uno de los métodos de *mínimos cuadrados* programados para minimizar una función cuadrática no sujeta a ninguna restricción, estos son los métodos Gauss-Newton, Levenberg Mardquardt, Quasi-Newton e híbrido. Prestando un poco más de atención en el método Quasi-Newton ya que se han realizados dos formas de actualizar la *hessiana*, según *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) y según *Al-Baali y Fletcher* (AF), utilizando la actualización más eficiente, es decir, la que mejores resultados ha obtenido, para usarla en el método Híbrido, el cual combina el método Levenberg-Marquardtt con el Quasi-Newton con la actualización BFGS.

Para ello, se ha utilizado como función de prueba la función de Rosenbrock (explicada anteriormente).

En Matlab existe la función *lsqnonlin* que resuelve problemas de mínimos cuadrados no lineales. Se han validado los resultados de los métodos Gauss-Newton y Levenberg-Marquardtt obtenidos con los obtenidos con esta función.

Para hacer las simulaciones pertinentes con los métodos programados se han tomado los siguientes valores para los parámetros de entrada:

Método	Xtol	Ftol	maxIter	maxEval	K	x0
Gauss-Newton	10^{-6}	10^{-6}	100	500	{'f1' 'f2'}	Varia
Levenberg-Marquardtt	10^{-6}	10^{-6}	100	500	{'f1' 'f2'}	Varia
Quasi-Newton (BFGS)	10^{-15}	10^{-15}	10000	100000	{'f1' 'f2'}	Varia
Quasi-Newton (AF)	10^{-15}	10^{-15}	10^{8}	10^{8}	{'f1' 'f2'}	Varia
Híbrido	10^{-12}	10^{-12}	10000	100000	{'f1' 'f2'}	Varia

Los parámetros se han ido modificando mientras se probaban los programas según la velocidad de convergencia de cada método, por lo que se han tenido que modificar también el número de evaluaciones y de iteraciones de la función objetivo, y la tolerancia de x y de F .

Donde:

xtol es la tolerancia en el valor de x.

ftol es la tolerancia en el valor de la función F.

maxIter es el número máximo de iteraciones.

maxEval es el número máximo de evaluaciones.

$x0$ es el valor inicial tomado para la estimación del valor de x, el cual se ve encabezando cada tabla de resultados.

k es la función que se va a evaluar

$f1 = 10\left(x_2 - x_1^2\right)$ y $f2 = 1 - x_1$ son los coeficientes de la función Rosenbrock

Los parámetros calculados son:

xfin $\rightarrow$ el valor de x en que la función se hace mínima.

kfin $\rightarrow$ el valor de k evaluado en x.

$F(x) \rightarrow$ el valor de la función de Rosenbrock evaluada en x.

nEval $\rightarrow$ el número de evaluaciones de la función F.

nIter $\rightarrow$ el número de iteraciones.

Así mismo, para hacer las simulaciones con MATLAB se han tomado los siguientes valores para los parámetros de entrada, en la función LSQNONLIN.M:

Método	**Options**	**ub**	**lb**	**Fun**	**x0**
Gauss-Newton	optimset(options, 'TolX', 1e-6, 'TolF', 1e-6, 'MaxIter', 100, 'MaxFunEvals', 500, 'LargeScale', 'off', 'LevenbergMarquardtt', 'off')	[ ]	[ ]	{'coefbanana'}	Varia
Levenberg-Marquardt	optimset(options, 'TolX', 1e-6, 'TolF', 1e-6, 'MaxIter', 100, 'MaxFunEvals', 500, 'LargeScale', 'off', 'LevenbergMarquardtt', 'on')	[ ]	[ ]	{'coefbanana'}	Varia
Quasi-Newton (BFGS)	No se puede realizar				
Quasi-Newton (AF)	No se puede realizar				
Híbrido	No se puede realizar				

Donde:

fun es la función F.

options es un parámetro que proporciona las opciones internas del algoritmo

ub es el límite superior de x, también se deja vacío.

lb es el límite inferior de x, se deja vacío ya que el problema que trata de solucionar no tiene cotas.

$x0$ es el valor inicial tomado para la estimación del valor de x, el cual podemos ver encabezando cada tabla de resultados.

coefbanana son los coeficientes de la función de Rosenbrock, $\left(10\left(x_2 - x_1^2\right)\right)^2$ y $\left(1 - x_1\right)^2$.

Los parámetros calculados son:

$xfin \rightarrow$ el valor de x donde la función se hace mínima.

$Resnorm \rightarrow$ el valor de f evaluado en x.

$Fval \rightarrow$ el valor de k evaluado en x.

$ExitFlag \rightarrow$ devuelve un valor que describe la condición de salida. Puede tener los valores:

1→ Incremento muy pequeño en el valor de la función

2→ Incremento muy pequeño en el valor de x.

3→ Número máximo de iteraciones excedido.

-1→ Formato incorrecto de x.

-2→ Formato incorrecto de la función.

-3→ Formato incorrecto del Jacobiano.

-4→ Error de dimensiones en x, K, J.

-5→ Desbordamiento.

$output \rightarrow$ estructura que contiene información acerca de la optimización, como por ejemplo el número de evaluaciones y de iteraciones.

A continuación se muestran las tablas con los resultados obtenidos:

X_0=[2, 3]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	$[2.4425 \cdot 10^{-14}, 0]$	$5.9658 \cdot 10^{-28}$	10	4
Gauss-Newton MATLAB	[1.0027, 1.0046]	$[6.5191 \cdot 10^{-6}, 7.2040 \cdot 10^{-6}]$	$9.4396 \cdot 10^{-11}$	36	6
Levenberg-Marquardt	[1.0000, 1.0000]	$[-1.1835 \cdot 10^{-6}, -7.3711 \cdot 10^{-6}]$	$5.5733 \cdot 10^{-11}$	18	10
Levenberg-Marquardt MATLAB	[1.0003, 1.0000]	$[2.3859 \cdot 10^{-6}, 6.4284 \cdot 10^{-8}]$	$5.6965 \cdot 10^{-12}$	39	12
Quasi-Newton (BFGS)	[1.0000, 1.0000]	$[-4.2188 \cdot 10^{-14}, -2.3981 \cdot 10^{-14}]$	$2.3549 \cdot 10^{-27}$	135	33
Quasi-Newton (AF)	[1.0000, 1.0000]	$[-4.4409 \cdot 10^{-15}, 7.6827 \cdot 10^{-14}]$	$5.9222 \cdot 10^{-27}$	788	187
Híbrido	[1.0000, 1.0000]	$[4.4142 \cdot 10^{-12}, -1.0972 \cdot 10^{-10}]$	$1.2059 \cdot 10^{-20}$	22	12

Como se observa, todos los métodos llegan al punto esperado $([1,1])$*, obteniendo los métodos de MATLAB buenos resultados pero no tan buenos como los obtenidos en los métodos implementados por mí.*

El método más rápido en converger es el método Gauss-Newton haciéndolo con 4 iteraciones, además se obtiene un error mínimo de $5.9658 \cdot 10^{-28}$*.*

El método Quasi-Newton con la actualización BFGS obtiene mejores resultados que este método con la actualización AF.

El método híbrido mejora los resultados obtenidos tanto con el método Levenberg-Marquardt como con el método Quasi-Newton.

X_0=[0, 3]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	$[7.7716 \cdot 10^{-14}, 0]$	$6.0397 \cdot 10^{-27}$	10	4
Gauss-Newton MATLAB	[0.9944, 0.9876]	$[1.6787 \cdot 10^{-5}, 3.0559 \cdot 10^{-5}]$	$1.2157 \cdot 10^{-9}$	22	4
Levenberg-Marquardt	[0.9999, 0.9999]	$[-9.0077 \cdot 10^{-7}, 5.2507 \cdot 10^{-6}]$	$2.8382 \cdot 10^{-11}$	10	6
Levenberg-Marquardt MATLAB	[0.9995, 1.0000]	$[9.5412 \cdot 10^{-6}, 2.3941 \cdot 10^{-7}]$	$9.1091 \cdot 10^{-11}$	36	11
Quasi-Newton (BFGS)	[1.0000, 1.0000]	$[1.9318 \cdot 10^{-13}, 3.0309 \cdot 10^{-14}]$	$3.8237 \cdot 10^{-26}$	1219	251
Quasi-Newton (AF)	[1.0000, 1.0000]	$[-2.2204 \cdot 10^{-15}, -7.2609 \cdot 10^{-14}]$	$5.2769 \cdot 10^{-27}$	9766	1998
Híbrido	[1.0000, 1.0000]	$[-3.0353 \cdot 10^{-12}, 7.5201 \cdot 10^{-11}]$	$5.6643 \cdot 10^{-21}$	14	8

Con los métodos implementados por mí se han obtenido mejores resultados en cuanto a número de evaluaciones e iteraciones y error obtenido.

El método que mejores resultados obtiene es el método Gauss-Newton, convergiendo en 4 iteraciones y obteniendo un error de $6.0397 \cdot 10^{-27}$*.*

El método que peores resultados ha obtenido es el Quasi-Newton con la actualización AF.

El método híbrido ha obtenido un error de $5.6643 \cdot 10^{-21}$ *que es menor que el obtenido con el método Levenberg-Marquardt (*$2.8382 \cdot 10^{-11}$*) y ha convergido con 12 iteraciones que es mucho menos que las iteraciones obtenidas con el método Quasi-Newton (251).*

X_0=[-1, 2]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	$[2.2204 \cdot 10^{-15}, 0]$	$4.9304 \cdot 10^{-30}$	90	20
Gauss-Newton MATLAB	[0.9993, 0.9977]	$[6.7258 \cdot 10^{-6}, 4.8838 \cdot 10^{-7}]$	$4.5476 \cdot 10^{-11}$	53	9
Levenberg-Marquardt	[1.0000, 1.0000]	$[-2.0759 \cdot 10^{-8}, 3.1183 \cdot 10^{-7}]$	$9.7668 \cdot 10^{-14}$	24	12
Levenberg-Marquardt MATLAB	[0.9997, 0.9991]	$[1.3403 \cdot 10^{-6}, 6.0159 \cdot 10^{-8}]$	$1.8001 \cdot 10^{-12}$	42	13
Quasi-Newton (BFGS)	[1.0000, 1.0000]	$[-1.3043 \cdot 10^{-11}, -5.7698 \cdot 10^{-12}]$	$2.0341 \cdot 10^{-22}$	5384	1082
Quasi-Newton (AF)	[1.0000, 1.0000]	$[1.5543 \cdot 10^{-14}, 3.6660 \cdot 10^{-13}]$	$1.3463 \cdot 10^{-25}$	55835	12648
Híbrido	[1.0000, 1.0000]	$[-4.5683 \cdot 10^{-11}, 1.1139 \cdot 10^{-9}]$	$1.2429 \cdot 10^{-18}$	26	13

El método más rápido en converger es el método Levenberg-Marquardt que lo ha hecho con 12 iteraciones.

El método que menor error ha obtenido es el Gauss-Newton, el cual ha obtenido un error de $4.9304 \cdot 10^{-30}$

Se han obtenido mejores resultados con la actualización BFGS del método Quasi-Newton que con la actualización AF.

Tal como se ha mencionado en la teoría, con el método Híbrido se ha mejorado el error obtenido con el método Levenberg-Marquardt ($1.2429 \cdot 10^{-18}$ frente al $9.7668 \cdot 10^{-14}$ del método LM) y se ha mejorado el número de iteraciones que con el método Quasi-Newton (13 iteraciones frente las 1082 iteraciones del método Quasi-Newton).

X_0=[2, 7]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	$[2.4425 \cdot 10^{-14}, 0]$	$5.9658 \cdot 10^{-28}$	10	4
Gauss-Newton MATLAB	[1.0055, 1.0098]	$[1.6782 \cdot 10^{-5}, 3.0560 \cdot 10^{-5}]$	$1.2155 \cdot 10^{-9}$	22	4
Levenberg-Marquardt	[1.0000, 1.0000]	$[-9.1205 \cdot 10^{-7}, -6.4036 \cdot 10^{-6}]$	$4.1837 \cdot 10^{-11}$	24	13
Levenberg-Marquardt MATLAB	[1.0005, 1.0020]	$[9.4958 \cdot 10^{-6}, 2.8494 \cdot 10^{-7}]$	$9.0252 \cdot 10^{-11}$	36	11
Quasi-Newton (BFGS)	[1.0000, 1.0000]	$[-5.3957 \cdot 10^{-13}, -1.1500 \cdot 10^{-12}]$	$1.6136 \cdot 10^{-24}$	219	50
Quasi-Newton (AF)	[1.0000, 1.0000]	$[4.8850 \cdot 10^{-14}, -5.3135 \cdot 10^{-13}]$	$2.8472 \cdot 10^{-25}$	20084	4057
Híbrido	[1.0000, 1.0000]	$[3.6393 \cdot 10^{-12}, -9.0532 \cdot 10^{-11}]$	$8.2093 \cdot 10^{-21}$	28	15

Todos los métodos convergen en el punto $([1,1])$*, que es el esperado según la teoría. Obteniendo mejores resultados de error, número de evaluaciones e iteraciones con los métodos míos.*

Con sólo 4 iteraciones y un error de $5.9658 \cdot 10^{-28}$ *el método Gauss-Newton converge, siendo éste el método más rápido.*

Se obtienen mejores resultados actualizando el método Quasi-Newton según BFGS que haciéndolo según AF.

Con el método híbrido se mejoran los resultados obtenidos con el método Levenberg-Marquardt y con el método Quasi-Newton.

X_0=[0.9, 1.1]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1, 1]	[0, 0]	0	10	4
Gauss-Newton MATLAB	[0.9998, 1.0001]	$[2.2415 \cdot 10^{-6}, 3.6052 \cdot 10^{-8}]$	$5.0254 \cdot 10^{-12}$	22	4
Levenberg-Marquardt	[1.0000, 1.0000]	$[-7.8383 \cdot 10^{-7}, -2.5448 \cdot 10^{-5}]$	$6.4820 \cdot 10^{-10}$	8	5
Levenberg-Marquardt MATLAB	[0.9999, 1.0000]	$[4.6670 \cdot 10^{-8}, 1.8008 \cdot 10^{-10}]$	$2.1781 \cdot 10^{-15}$	39	12
Quasi-Newton (BFGS)	[1.0000, 1.0000]	$[2.0819 \cdot 10^{-11}, 1.5107 \cdot 10^{-12}]$	$4.3571 \cdot 10^{-22}$	61	15
Quasi-Newton (AF)	[1.0000, 1.0000]	$[-2.2204 \cdot 10^{-14}, -3.7526 \cdot 10^{-14}]$	$1.9012 \cdot 10^{-27}$	1401	310
Híbrido	[1.0000, 1.0000]	$[1.2999 \cdot 10^{-10}, -3.3242 \cdot 10^{-9}]$	$1.1068 \cdot 10^{-17}$	12	7

Todos los métodos convergen en el punto visto en teoría $([1,1])$*, por lo que convergen correctamente. Además, el funcionamiento de los métodos implementados es más correcto que el funcionamiento de los métodos de MATLAB.*

El método que converge antes es el método Levenberg-Marquardt, mientras que el método Gauss-Newton obtiene un error nulo.

Se obtienen mejores resultados del método Quasi-Newton con la actualización BFGS que con la actualización AF, que son los resultados esperados según lo visto en teoría.

Con el método híbrido se obtiene un error de $1.1068 \cdot 10^{-17}$*, que es mucho menor que el error obtenido con el método Levenberg-Marquardt (*$6.4820 \cdot 10^{-10}$*) y converge con 7 iteraciones, por lo tanto ha convergido más rápido que el método Quasi-Newton.*

X_0=[-1.9, 2]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	[$1.1102 \cdot 10^{-14}$, 0]	$1.2326 \cdot 10^{-28}$	85	19
Gauss-Newton MATLAB	[0.9935, 0.9876]	[$3.4268 \cdot 10^{-6}$, $4.2103 \cdot 10^{-5}$]	$1.7844 \cdot 10^{-9}$	71	11
Levenberg-Marquardt	[0.9999, 0.9999]	[$-7.3440 \cdot 10^{-7}$, $7.9072 \cdot 10^{-6}$]	$6.3062 \cdot 10^{-11}$	38	18
Levenberg-Marquardt MATLAB	[0.9992, 0.9979]	[$3.1097 \cdot 10^{-6}$, $5.4425 \cdot 10^{-7}$]	$9.9663 \cdot 10^{-12}$	59	17
Quasi-Newton (BFGS)	[1.0000, 1.0000]	[$-2.9554 \cdot 10^{-12}$, $-1.4480 \cdot 10^{-12}$]	$1.0831 \cdot 10^{-23}$	454	96
Quasi-Newton (AF)	[1.0000, 1.0000]	[$1.3323 \cdot 10^{-14}$, $-8.8818 \cdot 10^{-16}$]	$1.7828 \cdot 10^{-28}$	28292	5686
Híbrido	[1.0000, 1.0000]	[$-1.7211 \cdot 10^{-11}$, $4.2707 \cdot 10^{-10}$]	$1.8269 \cdot 10^{-19}$	42	20

Como se puede observar todos los métodos convergen, obteniendo con los métodos implementados por mí, mejores resultados en cuanto a número de evaluaciones e iteraciones y error obtenido.

El método que ha convergido antes es el método Levenberg-Marquardt, con 18 iteraciones, y el que ha obtenido menor error es el Gauss-Newton ($1.2326 \cdot 10^{-28}$).

El método que peores resultados obtiene es el Quasi-Newton con la actualización AF.

El método híbrido ha mejorado los resultados obtenidos tanto con el método Levenberg-Marquardt ($1.8269 \cdot 10^{-19}$ frente a $9.9663 \cdot 10^{-12}$ del método LM), como con el Quasi-Newton (20 iteraciones frente a las 96 del método QN).

X_0=[3, 6]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	[$5.7732 \cdot 10^{-14}$, 0]	$3.3329 \cdot 10^{-27}$	15	5
Gauss-Newton MATLAB	[1.0008, 1.0021]	[$2.5168 \cdot 10^{-6}$, $6.5334 \cdot 10^{-7}$]	$6.7611 \cdot 10^{-12}$	41	7
Levenberg-Marquardt	[1.0000, 1.0000]	[$-3.0659 \cdot 10^{-6}$, $-1.4130 \cdot 10^{-5}$]	$2.0904 \cdot 10^{-10}$	25	13
Levenberg-Marquardt MATLAB	[1.0005, 0.9992]	[$2.9208 \cdot 10^{-5}$, $2.4757 \cdot 10^{-7}$]	$8.5315 \cdot 10^{-10}$	39	12
Quasi-Newton (BUGS)	[1.0000, 1.0000]	[$-2.4403 \cdot 10^{-12}$, $-5.3610 \cdot 10^{-12}$]	$3.4696 \cdot 10^{-23}$	286	65
Quasi-Newton (AF)	[1.0000, 1.0000]	[$-4.4409 \cdot 10^{-15}$, $7.6827 \cdot 10^{-14}$]	$5.9222 \cdot 10^{-27}$	788	187
Híbrido	[1.0000, 1.0000]	[$1.2426 \cdot 10^{-11}$, $-3.1364 \cdot 10^{-10}$]	$9.8523 \cdot 10^{-20}$	29	15

Con los métodos implementados por mí se han obtenido mejores resultados en cuanto a número de evaluaciones e iteraciones y error obtenido.

El método que mejores resultados ha obtenido es el método Gauss-Newton, convergiendo en 5 iteraciones y obteniendo un error de $3.3329 \cdot 10^{-27}$.

El método Quasi-Newton con la actualización BFGS obtiene mejores resultados que este método con la actualización AF.

El método híbrido ha mejorado los resultados obtenidos tanto con el método Levenberg-Marquardt como con el Quasi-Newton.

X_0=[0.5, 0.5]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	$[2.2204 \cdot 10^{-15}, 0]$	$4.9304 \cdot 10^{-30}$	10	4
Gauss-Newton MATLAB	[1.0000, 1.0000]	$[6.7199 \cdot 10^{-15}, 5.8609 \cdot 10^{-17}]$	$4.5161 \cdot 10^{-29}$	29	5
Levenberg-Marquardt	[0.9999, 0.9999]	$[-1.4674 \cdot 10^{-6}, 6.7321 \cdot 10^{-6}]$	$4.7474 \cdot 10^{-11}$	10	6
Levenberg-Marquardt MATLAB	[0.9994, 0.9989]	$[2.5083 \cdot 10^{-8}$, 2.5607e-$\cdot 10^{-7}]$	$6.6200 \cdot 10^{-14}$	33	10
Quasi-Newton (BFGS)	[1.0000, 1.0000]	$[0, -4.4409 \cdot 10^{-16}]$	$1.9722 \cdot 10^{-31}$	116	27
Quasi-Newton (AF)	[1.0000, 1.0000]	$[2.4425 \cdot 10^{-14}, 2.1705 \cdot 10^{-13}]$	$4.7707 \cdot 10^{-26}$	5942	1336
Híbrido	[1.0000, 1.0000]	$[-3.5416 \cdot 10^{-12}, 8.7760 \cdot 10^{-11}]$	$7.7143 \cdot 10^{-21}$	14	8

Como se puede observar todos los métodos convergen, obteniendo mejores resultados con los métodos implementados por mí.

El método que ha convergido más rápido es el método Gauss-Newton, haciéndolo con 4 iteraciones y obteniendo un error de $4.9304 \cdot 10^{-30}$, que es el menor error obtenido.

Se obtienen mejores resultados del método Quasi-Newton con la actualización BFGS que con la actualización AF, que son los resultados esperados según lo visto en t

El método híbrido ha convergido más rápido que el método Quasi-Newton, ya que ha convergido con 8 iteraciones y ha mejorado el error obtenido con el método Levenberg-Marquardt, ya que se ha obtenido $4.9304 \cdot 10^{-30}$.

X_0=[2, 0]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	[$2.4425\cdot10^{-14}$, 0]	$5.9658\cdot10^{-28}$	10	4
Gauss-Newton MATLAB	[0.9993, 0.9977]	[$6.4293\cdot10^{-6}$, $4.9502\cdot10^{-7}$]	$4.1581\cdot10^{-11}$	49	8
Levenberg-Marquardt	[1.0000, 1.0000]	[$-1.3692\cdot10^{-7}$, $1.2588\cdot10^{-6}$]	$1.6033\cdot10^{-12}$	14	8
Levenberg-Marquardt MATLAB	[1.0002, 0.9992]	[$1.4900\cdot10^{-5}$, $5.9109\cdot10^{-8}$]	$2.2201\cdot10^{-10}$	39	12
Quasi-Newton (BFGS)	[1.0000, 1.0000]	[-2.8122e-012, 3.3399e-012]	$1.9063\cdot10^{-23}$	155	37
Quasi-Newton (AF)	[1.0000, 1.0000]	[3.7748e-014, -6.5370e-013]	$4.2875\cdot10^{-25}$	161484	36074
Híbrido	[1.0000, 1.0000]	[-3.2641e-013, 8.1806e-012]	$6.7028\cdot10^{-23}$	18	10

Todos los métodos convergen en el punto visto en teoría $([1,1])$*, por lo que convergen correctamente. Además, el funcionamiento de los métodos implementados es más correcto que el funcionamiento de los métodos de MATLAB.*

El método que antes ha convergido, con 4 iteraciones, y que menos error ha obtenido, $5.9658\cdot10^{-28}$*, es el Gauss-Newton.*

El método Quasi-Newton con la actualización BFGS obtiene mejores resultados que este método con la actualización AF.

El método híbrido ha mejorado los resultados obtenidos tanto con el método Levenberg-Marquardt, (menos error) como con el Quasi-Newton (menos iteraciones).

x_0=[2, 6]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	[$2.4425\cdot10^{-14}$, 0]	$5.9658\cdot10^{-28}$	10	4
Gauss-Newton MATLAB	[1.0041, 1.0057]	[$5.8608\cdot10^{-5}$, $1.6615\cdot10^{-5}$]	$3.7109\cdot10^{-9}$	29	5
Levenberg-Marquardt	[1.0000, 1.0000]	[$-1.5722\cdot10^{-9}$, $-5.1543\cdot10^{-7}$]	$2.6567\cdot10^{-13}$	24	13
Levenberg-Marquardt MATLAB	[1.0011, 1.0031]	[$9.4905\cdot10^{-6}$, $1.1118\cdot10^{-6}$]	$9.1307\cdot10^{-11}$	33	10
Quasi-Newton (BUGS)	[1.0000, 1.0000]	[$-1.0519\cdot10^{-11}$, $2.9611\cdot10^{-11}$]	$9.8749\cdot10^{-22}$	222	50
Quasi-Newton (AF)	[1.0000, 1.0000]	[$4.4409\cdot10^{-14}$, $-5.6866\cdot10^{-13}$]	$3.2534\cdot10^{-25}$	64516	15865
Híbrido	[1.0000, 1.0000]	[$7.3417\cdot10^{-11}$, $-1.9117\cdot10^{-9}$]	$3.6599\cdot10^{-18}$	26	14

Todos los métodos convergen, obteniendo con mis métodos mejores resultados de error ($5.9658 \cdot 10^{-28}$), número de evaluaciones (10) e iteraciones (4).

El método que antes ha llegado al punto $[1,1]$ *es el método Gauss-Newton, además éste obtiene un error menor.*

El método Quasi-Newton con la actualización BFGS obtiene mejores resultados que este método con la actualización AF.

El método híbrido ha obtenido un error de $3.6599 \cdot 10^{-18}$ que es menor que el obtenido con el método Levenberg-Marquardt ($2.6567 \cdot 10^{-13}$) y ha convergido con 14 iteraciones que es mucho menos que las iteraciones obtenidas con el método Quasi-Newton (50).

X_0=[-2, -1]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	$[5.5511 \cdot 10^{-14}, 0]$	$3.0815 \cdot 10^{-27}$	25	7
Gauss-Newton MATLAB	[1.0036, 1.0090]	$[3.0105 \cdot 10^{-5}, 1.3308 \cdot 10^{-5}]$	$1.0834 \cdot 10^{-9}$	43	7
Levenberg-Marquardt	[0.9999, 0.9999]	$[-1.7190 \cdot 10^{-6}, 9.1583 \cdot 10^{-6}]$	$8.6830 \cdot 10^{-11}$	21	11
Levenberg-Marquardt MATLAB	[0.9996, 0.9975]	$[2.9201 \cdot 10^{-5}, 1.3495 \cdot 10^{-7}]$	$8.5274 \cdot 10^{-10}$	42	13
Quasi-Newton (BFGS)	[1.0000, 1.0000]	$[4.4076 \cdot 10^{-13}, 2.7867 \cdot 10^{-14}]$	$1.9504 \cdot 10^{-25}$	417	92
Quasi-Newton (AF)	[1.0054, 1.0109]	$[-4.4409 \cdot 10^{-15}, -2.8859 \cdot 10^{-12}]$	$8.3285 \cdot 10^{-24}$	4472431	1007159
Híbrido	[1.0000, 1.0000]	$[-8.0003 \cdot 10^{-12}, 1.9921 \cdot 10^{-10}]$	$3.9747 \cdot 10^{-20}$	25	13

Todos los métodos convergen en el punto visto en teoría, y como se puede ver en la tabla resumen, se obtienen mejores resultados con los métodos implementados por mí.

El método que peores resultados ha obtenido es el Quasi-Newton con la actualización AF.

Con el método híbrido se ha obtenido un error de $3.9747 \cdot 10^{-20}$, que es mucho mejor que el error obtenido con el método Levenberg-Marquardt ($8.6830 \cdot 10^{-11}$) y ha convergido con 25 iteraciones, por lo tanto ha convergido más rápido que el método Quasi-Newton.

X_0=[4, -2]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	$[2.0184 \cdot 10^{-12}, 0]$	$4.0739 \cdot 10^{-24}$	10	4
Gauss-Newton MATLAB	[0.9998, 1.0023]	$[6.9276 \cdot 10^{-5}, 2.9270 \cdot 10^{-8}]$	$4.7991 \cdot 10^{-9}$	35	6
Levenberg-Marquardt	[0.9999, 0.9999]	$[-1.9068 \cdot 10^{-6}, 9.8289 \cdot 10^{-6}]$	$1.0024 \cdot 10^{-10}$	23	12
Levenberg-Marquardt MATLAB	[1.0015, 0.9897]	$[1.7375 \cdot 10^{-3}, 2.1556 \cdot 10^{-6}]$	$3.0190 \cdot 10^{-6}$	36	11
Quasi-Newton (BFGS)	[1.0000, 1.0000]	$[-4.8765 \cdot 10^{-11}, -5.2620 \cdot 10^{-11}]$	$5.1469 \cdot 10^{-21}$	240	56
Quasi-Newton (AF)	[1.0000, 1.0000]	$[-8.8818 \cdot 10^{-15}, 5.0249 \cdot 10^{-13}]$	$2.5257 \cdot 10^{-25}$	2904778	583749
Híbrido	[1.0000, 1.0000]	$[-8.8152 \cdot 10^{-12}, 2.1967 \cdot 10^{-10}]$	$4.8332 \cdot 10^{-20}$	27	14

Todos los métodos llegan al punto de convergencia visto en teoría, obteniendo con mis métodos mejores resultados de error, número de evaluaciones e iteraciones que con MATLAB.

El método que más rápido ha convergido con 4 iteraciones y que menos error ha obtenido, $4.0739 \cdot 10^{-24}$, es el Gauss-Newton.

El método Quasi-Newton con la actualización BFGS obtiene mejores resultados que este método con la actualización AF.

El método híbrido ha mejorado los resultados obtenidos tanto con el método Levenberg-Marquardt como con el Quasi-Newton.

X_0=[6, 11]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	$[3.1553 \cdot 10^{-12}, 0]$	$9.9556 \cdot 10^{-24}$	15	5
Gauss-Newton MATLAB	[1.0043, 1.0098]	$[1.4655 \cdot 10^{-5}, 1.8294 \cdot 10^{-5}]$	$5.4944 \cdot 10^{-10}$	82	13
Levenberg-Marquardt	[1.0000, 1.0000]	$[-1.7035 \cdot 10^{-8}, -8.3948 \cdot 10^{-7}]$	$7.0502 \cdot 10^{-13}$	32	17
Levenberg-Marquardt MATLAB	[1.0012, 0.9902]	$[1.4899 \cdot 10^{-3}, 1.5136 \cdot 10^{-6}]$	$2.2197 \cdot 10^{-6}$	39	12
Quasi-Newton (BFGS)	[1.0000, 1.0000]	$[1.4845 \cdot 10^{-11}, 9.8850 \cdot 10^{-12}]$	$3.1808 \cdot 10^{-22}$	489	105
Quasi-Newton (AF)	[1.0003, 1.0006]	$[-1.1102 \cdot 10^{-15}, 7.5051 \cdot 10^{-14}]$	$5.6339 \cdot 10^{-27}$	31111	6228
Híbrido	[1.0000, 1.0000]	$[1.2906 \cdot 10^{-10}, -3.3760 \cdot 10^{-9}]$	$1.1414 \cdot 10^{-17}$	34	18

Todos los métodos convergen en el punto $([1,1])$*, que es el esperado según la teoría. Obteniendo mejores resultados de error, número de evaluaciones e iteraciones con los métodos míos.*

Con sólo 4 iteraciones y un error de $5.9658 \cdot 10^{-28}$ *el método Gauss-Newton converge, siendo éste el método más rápido.*

Se obtienen mejores resultados actualizando el método Quasi-Newton según BFGS que haciéndolo según AF.

Con el método híbrido se ha mejorado el error obtenido con el método Levenberg-Marquardt ($1.1414 \cdot 10^{-17}$ *frente al* $7.0502 \cdot 10^{-13}$ *del método LM) y el número de iteraciones es de 18 que ha mejorado las 105 iteraciones del método Quasi-Newton.*

X_0=[-3, 3]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	$[7.1054 \cdot 10^{-14}, 0]$	$5.0487 \cdot 10^{-27}$	35	9
Gauss-Newton MATLAB	[0.9994, 0.9996]	$[5.1672 \cdot 10^{-6}, 2.8479 \cdot 10^{-7}]$	$2.6781 \cdot 10^{-11}$	55	9
Levenberg-Marquardt	[0.9999, 0.9999]	$[-1.3741 \cdot 10^{-6}, 7.4670 \cdot 10^{-6}]$	$5.7645 \cdot 10^{-11}$	41	20
Levenberg-Marquardt MATLAB	[0.9993, 0.9957]	$[8.1727 \cdot 10^{-5}, 5.2615 \cdot 10^{-7}]$	$6.6795 \cdot 10^{-9}$	44	13
Quasi-Newton (BFGS)	[1.0000, 1.0000]	$[-7.7716 \cdot 10^{-15}, 2.8644 \cdot 10^{-14}]$	$8.8086 \cdot 10^{-28}$	681	143
Quasi-Newton (AF)	[1.0000, 1.0000]	$[-2.2204 \cdot 10^{-15}, -5.3735 \cdot 10^{-14}]$	$2.8924 \cdot 10^{-27}$	887589	177681
Híbrido	[1.0000, 1.0000]	$[-5.4567 \cdot 10^{-12}, 1.3556 \cdot 10^{-10}]$	$1.8406 \cdot 10^{-20}$	45	22

Todos los métodos convergen en el punto esperado, obteniendo con los métodos implementados por mí mejores resultados de error, número de evaluaciones e iteraciones.

El método que mejores resultados obtiene es el Gauss-Newton, convergiendo en 9 iteraciones y obteniendo un error de $5.0487 \cdot 10^{-27}$*.*

El método Quasi-Newton con la actualización BFGS obtiene mejores resultados que este método con la actualización AF.

El método híbrido ha mejorado los resultados obtenidos tanto con el método Levenberg-Marquardt, en cuanto a error obtenido, como con el Quasi-Newton, convergiendo en 22 iteraciones en vez de en 143.

X_0=[0.2, 4.8]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	$[2.2204 \cdot 10^{-15}, 0]$	$4.9304 \cdot 10^{-30}$	10	4
Gauss-Newton MATLAB	[1.0009, 0.9989]	$[8.0345 \cdot 10^{-5}, 7.2492 \cdot 10^{-7}]$	$6.4559 \cdot 10^{-9}$	23	4
Levenberg-Marquardt	[1.0000, 1.0000]	$[-2.9567 \cdot 10^{-8}, 2.7520 \cdot 10^{-7}]$	$7.6611 \cdot 10^{-14}$	10	6
Levenberg-Marquardt MATLAB	[0.9998, 1.0006]	$[1.0118 \cdot 10^{-5}, 3.6866 \cdot 10^{-8}]$	$1.0238 \cdot 10^{-10}$	39	12
Quasi-Newton (BUGS)	[1.0000, 1.0000]	$[-1.9063 \cdot 10^{-10}, 1.8749 \cdot 10^{-10}]$	$7.1493 \cdot 10^{-20}$	7425	1492
Quasi-Newton (AF)	[1.0000, 1.0000]	$[-8.6597 \cdot 10^{-14}, -3.1064 \cdot 10^{-12}]$	$9.6572 \cdot 10^{-24}$	2942418	679787
Híbrido	[1.0000, 1.0000]	$[-2.3492 \cdot 10^{-11}, 5.6752 \cdot 10^{-10}]$	$3.2263 \cdot 10^{-19}$	12	7

Todos los métodos convergen en el punto esperado según la teoría. Obteniendo mejores resultados de error, número de evaluaciones e iteraciones con los métodos míos.

El método más rápido en converger es el Gauss-Newton, haciéndolo en 4 iteraciones, además ha obtenido un error de $4.9304 \cdot 10^{-30}$, que es el menor error obtenido.

Se obtienen mejores resultados del método Quasi-Newton con la actualización BFGS que con la AF, que son los resultados esperados según lo visto en teoría.

El método híbrido ha convergido en 7 iteraciones, por lo que lo ha hecho más rápido que el método Quasi-Newton. Con este método también se ha obtenido un error menor que con el Levenberg-Marquardt, obteniendo un error de $3.2263 \cdot 10^{-19}$.

X_0=[0, 1]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	$[7.7716 \cdot 10^{-14}, 0]$	$6.0397 \cdot 10^{-27}$	10	4
Gauss-Newton MATLAB	[0.9993, 0.9981]	$[3.3780 \cdot 10^{-6}, 4.5598 \cdot 10^{-7}]$	$1.1619 \cdot 10^{-11}$	35	6
Levenberg-Marquardt	[0.9999, 0.9999]	$[-3.3535 \cdot 10^{-6}, 2.7810 \cdot 10^{-5}]$	$7.8466 \cdot 10^{-10}$	13	7
Levenberg-Marquardt MATLAB	[0.9980, 0.9961]	$[2.2524 \cdot 10^{-9}, 3.8622 \cdot 10^{-6}]$	$1.4917 \cdot 10^{-11}$	30	9
Quasi-Newton (BUGS)	[1.0000, 1.0000]	$[-8.7264 \cdot 10^{-12}, 1.3524 \cdot 10^{-11}]$	$2.5904 \cdot 10^{-22}$	41523	8314
Quasi-Newton (AF)	[1.0000, 1.0000]	$[2.2204 \cdot 10^{-15}, 2.3981 \cdot 10^{-14}]$	$5.8001 \cdot 10^{-28}$	1987436	397509
Híbrido	[1.0000, 1.0000]	$[-1.4014 \cdot 10^{-10}, 3.4404 \cdot 10^{-9}]$	$1.1856 \cdot 10^{-17}$	17	9

Como se esperaba todos los métodos convergen.

El método Gauss-Newton es el método que menor error ha obtenido ($6.0397 \cdot 10^{-27}$) y el más rápido en converger, haciéndolo en 4 iteraciones.

Se han obtenido mejores resultados con la actualización BFGS del método Quasi-Newton que con la actualización AF.

Tal como se ha mencionado en la teoría, con el método Híbrido se ha mejorado el error obtenido con el método Levenberg-Marquardt ($1.1856 \cdot 10^{-17}$ frente al $7.8466 \cdot 10^{-10}$ del método LM) y se ha mejorado el número de iteraciones que con el método Quasi-Newton (17 iteraciones frente las 8314 iteraciones del método Quasi-Newton).

X_0=[-3, 5]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	$[2.9532 \cdot 10^{-13}, 0]$	$8.7214 \cdot 10^{-26}$	60	14
Gauss-Newton MATLAB	[0.9966, 0.9960]	$[7.5129 \cdot 10^{-5}, 1.1365 \cdot 10^{-5}]$	$5.7735 \cdot 10^{-9}$	62	10
Levenberg-Marquardt	[1.2006, 0.5550]	$[-8.8637, -2.0056 \cdot 10^{-1}]$	$7.8599 \cdot 10^{+1}$	34	14
Levenberg-Marquardt MATLAB	[0.9992, 0.9958]	$[7.1427 \cdot 10^{-5}, 5.9319 \cdot 10^{-7}]$	$5.1022 \cdot 10^{-9}$	44	13
Quasi-Newton (BFGS)	[1.0000, 1.0000]	$[6.3061 \cdot 10^{-13}, -6.8834 \cdot 10^{-15}]$	$3.9771 \cdot 10^{-25}$	669	141
Quasi-Newton (AF)	[-0.7642, 0.5841]	$[-5.3750 \cdot 10^{-9}, 1.7643]$	3.1126	84115950	16823214
Híbrido	[1.2006, 0.5549]	$[-8.8637, -2.0056 \cdot 10^{-1}]$	$7.8606 \cdot 10^{+1}$	37	15

Todos los métodos convergen en el punto $([1,1])$, que es el esperado según la teoría. Obteniendo mejores resultados de error, número de evaluaciones e iteraciones con los métodos míos.

El método más rápido en converger y el que menor error obtiene es el método Gauss-Newton.

El método Quasi-Newton con la actualización de la hessiana según AF no converge en el punto esperado ([1, 1]) ya que el incremento del valor de la función es muy pequeño. Algo similar ocurre con los métodos Levenberg-Marquardt e Híbrido, pero estos no convergen debido a que el incremento del valor de x es muy pequeño.

El método Quasi-Newton con la actualización BFGS obtiene mejores resultados que este método con la actualización AF.

X_0=[0.5, 1.5]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	$[1.1102 \cdot 10^{-14}, 0]$	$1.2326 \cdot 10^{-28}$	10	4
Gauss-Newton MATLAB	[1.0008, 1.0011]	$[1.8241 \cdot 10^{-6}, 6.1561 \cdot 10^{-7}]$	$3.7062 \cdot 10^{-12}$	23	4
Levenberg-Marquardt	[1.0000, 1.0000]	$[-5.3206 \cdot 10^{-9}, 8.6289 \cdot 10^{-8}]$	$7.4741 \cdot 10^{-15}$	10	6
Levenberg-Marquardt MATLAB	[0.9999, 1.0000]	$[5.9606 \cdot 10^{-7}, 1.4776 \cdot 10^{-8}]$	$3.5550 \cdot 10^{-13}$	39	12
Quasi-Newton (BFGS)	[1.0000, 1.0000]	$[-1.1514 \cdot 10^{-11}, 8.3352 \cdot 10^{-12}]$	$2.0205 \cdot 10^{-22}$	119	28
Quasi-Newton (AF)	[1.1556, 1.3355]	$[1.2412 \cdot 10^{-9}, -1.5565 \cdot 10^{-1}]$	$2.4226 \cdot 10^{-2}$	20707427	4141486
Híbrido	[1.0000, 1.0000]	$[-7.2020 \cdot 10^{-12}, 1.7939 \cdot 10^{-10}]$	$3.2233 \cdot 10^{-20}$	12	7

Mis métodos obtienen mejores resultados que los de MATLAB.

El método que mejores resultados obtiene al converger es el método Gauss-Newton.

Se obtienen mejores resultados del método Quasi-Newton con la actualización BFGS que con la actualización AF, que son los resultados esperados según lo visto en teoría.

Con el método híbrido se obtiene un error de $3.2233 \cdot 10^{-20}$ *que es mejor que el obtenido con el método Levenberg-Marquardt y converge en 7 iteraciones, con lo que este método es más rápido que el método Quasi-Newton.*

X_0=[-1, -1]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	$[2.8866 \cdot 10^{-14}, 0]$	$8.3323 \cdot 10^{-28}$	25	7
Gauss-Newton MATLAB	[0.9952, 0.9901]	$[6.4464 \cdot 10^{-7}, 2.3299 \cdot 10^{-5}]$	$5.4325 \cdot 10^{-10}$	42	7
Levenberg-Marquardt	[0.9999, 0.9999]	$[-3.6166 \cdot 10^{-6}, 1.3025 \cdot 10^{-5}]$	$1.8273 \cdot 10^{-10}$	14	8
Levenberg-Marquardt MATLAB	[0.9998, 0.9987]	$[5.3632 \cdot 10^{-6}, 5.9972 \cdot 10^{-8}]$	$2.8768 \cdot 10^{-11}$	42	13
Quasi-Newton (BUGS)	[1.0000, 1.0000]	$[-6.6613 \cdot 10^{-15}, 4.4409 \cdot 10^{-15}]$	$6.4095 \cdot 10^{-29}$	309	68
Quasi-Newton (AF)	[0.9493, 0.9012]	$[-4.9101 \cdot 10^{-10}, 5.0664 \cdot 10^{-2}]$	$2.5669 \cdot 10^{-3}$	9961390	1992820
Híbrido	[1.0000, 1.0000]	$[-9.8710 \cdot 10^{-12}, 2.4361 \cdot 10^{-10}]$	$5.9443 \cdot 10^{-20}$	18	10

Todos los métodos convergen, obteniendo con mis métodos mejores resultados de error, número de evaluaciones e iteraciones.

El método que ha convergido antes es el método Gauss-Newton haciéndolo 4 iteraciones, además se ha obtenido menor error, siendo éste $8.3323 \cdot 10^{-28}$.

El método Quasi-Newton con la actualización BFGS obtiene mejores resultados que este método con la actualización AF.

El método híbrido ha convergido en 10 iteraciones, por lo que ha convergido más rápido que con el método Quasi-Newton, que lo ha hecho en 68 iteraciones. Con este método también se ha obtenido un error menor que con el método Levenberg-Marquardt, obteniendo un error de $5.9443 \cdot 10^{-20}$ *respecto un error de* $6.4095 \cdot 10^{-29}$.

X_0=[5, 2]					
MÉTODO	**X**	**K(x)**	**F(x)**	**nEval**	**nIter**
Gauss-Newton	[1.0000, 1.0000]	$[3.5905 \cdot 10^{-12}, 0]$	$1.2891 \cdot 10^{-23}$	10	4
Gauss-Newton MATLAB	[0.9999, 1.0011]	$[1.5227 \cdot 10^{-5}, 5.1813 \cdot 10^{-9}]$	$2.3185 \cdot 10^{-10}$	42	7
Levenberg-Marquardt	[1.0000, 1.0000]	$[-1.0226 \cdot 10^{-6}, -7.8176 \cdot 10^{-6}]$	$6.2161 \cdot 10^{-11}$	21	11
Levenberg-Marquardt MATLAB	[1.0010, 0.9924]	$[9.0644 \cdot 10^{-4}, 9.6405 \cdot 10^{-7}$	$8.2163 \cdot 10^{-7}$	39	12
Quasi-Newton (BUGS)	[1.0000, 1.0000]	$[3.4972 \cdot 10^{-13}, 9.7937 \cdot 10^{-12}]$	$9.6039 \cdot 10^{-23}$	975	204
Quasi-Newton (AF)	[1.0000, 1.0000]	$[2.2649 \cdot 10^{-13}, -3.6147 \cdot 10^{-10}]$	$1.3066 \cdot 10^{-19}$	26821666	5365503
Híbrido	[1.0000, 1.0000]	$[5.6732 \cdot 10^{-12}, -1.4100 \cdot 10^{-10}]$	$1.9914 \cdot 10^{-20}$	25	13

Todos los métodos convergen, obteniendo con mis métodos mejores resultados de error, número de evaluaciones e iteraciones.

El método Gauss-Newton ha convergido en 4 iteraciones siendo el método más rápido en converger, además de ser el método que menor error ha obtenido, obteniendo un error de $1.2891 \cdot 10^{-23}$.

El método Quasi-Newton con la actualización BFGS obtiene mejores resultados que este método con la actualización AF.

El método híbrido ha obtenido un error de $1.9914 \cdot 10^{-20}$ *que es menor que el obtenido con el método Levenberg-Marquardt (*$6.2161 \cdot 10^{-11}$*) y ha convergido en 13 iteraciones que es mucho menos que las iteraciones obtenidas con el método Quasi-Newton (204).*

4.2- Resultado del diseño de filtros de microondas con postes dieléctricos

Una vez que se ha explicado el proceso de diseño y tras la programación y validación de los algoritmos, se introducen en el programa de diseño elaborado en el grupo de investigación antes citado, para aplicarlos en el diseño de dispositivos pasivos de altas frecuencias, en concreto, filtros a frecuencias de microondas, se presentan los distintos resultados obtenidos:

Además se ha utilizado el programa *MATLAB* para obtener las gráficas de las pruebas realizadas.

Para hacer las simulaciones pertinentes se han tomado los siguientes parámetros de optimización:

Método	**xtol**	**Ftol**	**maxIter**	**maxEval**
Gauss-Newton	10^{-6}	10^{-6}	100	500
Levenberg-Marquardt	10^{-6}	10^{-6}	100	500
Quasi-Newton (BFGS)	10^{-15}	10^{-15}	10000	100000
Quasi-Newton (AF)	10^{-15}	10^{-15}	10^{8}	10^{8}
Híbrido	10^{-6}	10^{-6}	100	500

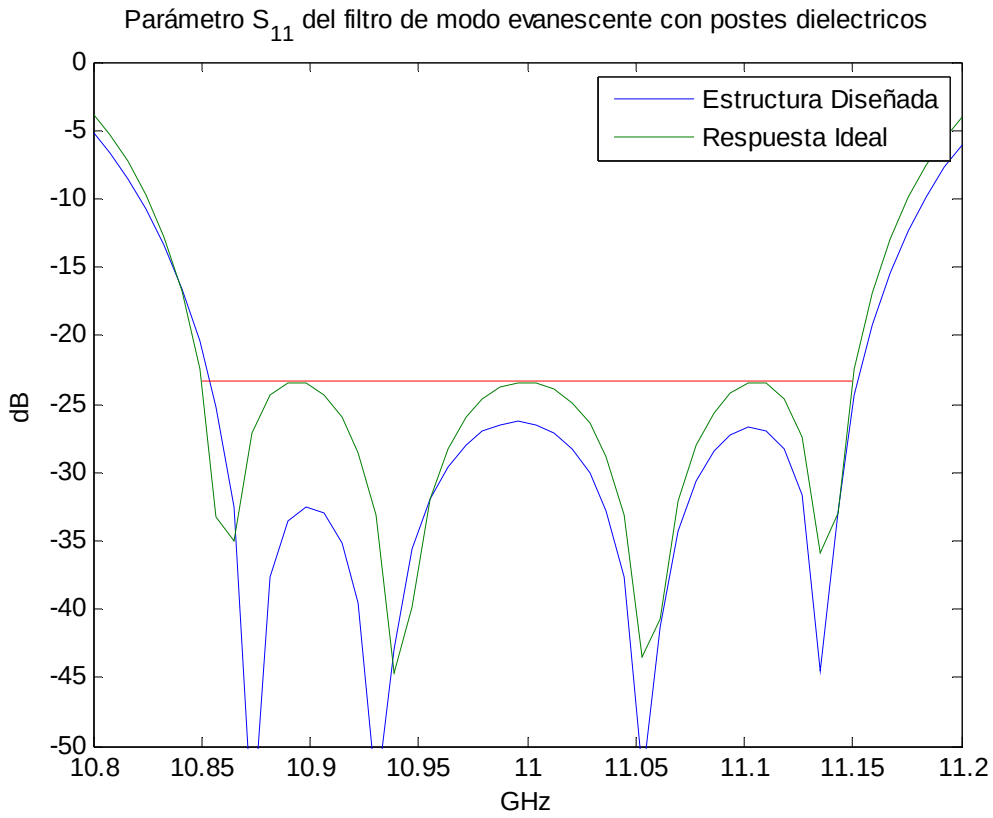
Los parámetros calculados son *xfin*, *kfin*, $F(x)$, *nEval* y *nIter*, explicados en el punto anterior.

Así mismo, para hacer las simulaciones con la función *lsqnonlin* de MATLAB se han tomado los mismos valores de entrada y de salida que en las simulaciones del punto anterior.

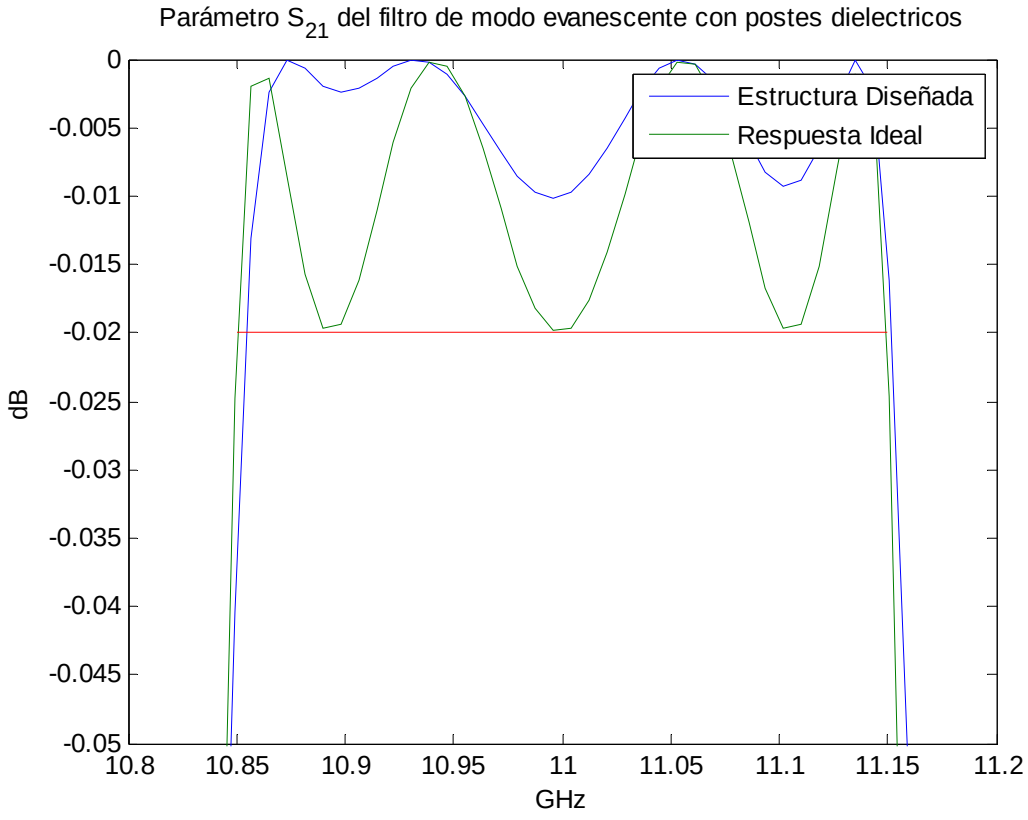
Seguidamente se exponen los resultados obtenidos en cada método.

- Método Gauss-Newton:

Xfin	[2.4179, 0.7777, 10.1820, 2.1696, 10.9919]
Kfin	[0.2970, 0.2970]
F(x)	0.1764
nEval	501
nIter	2



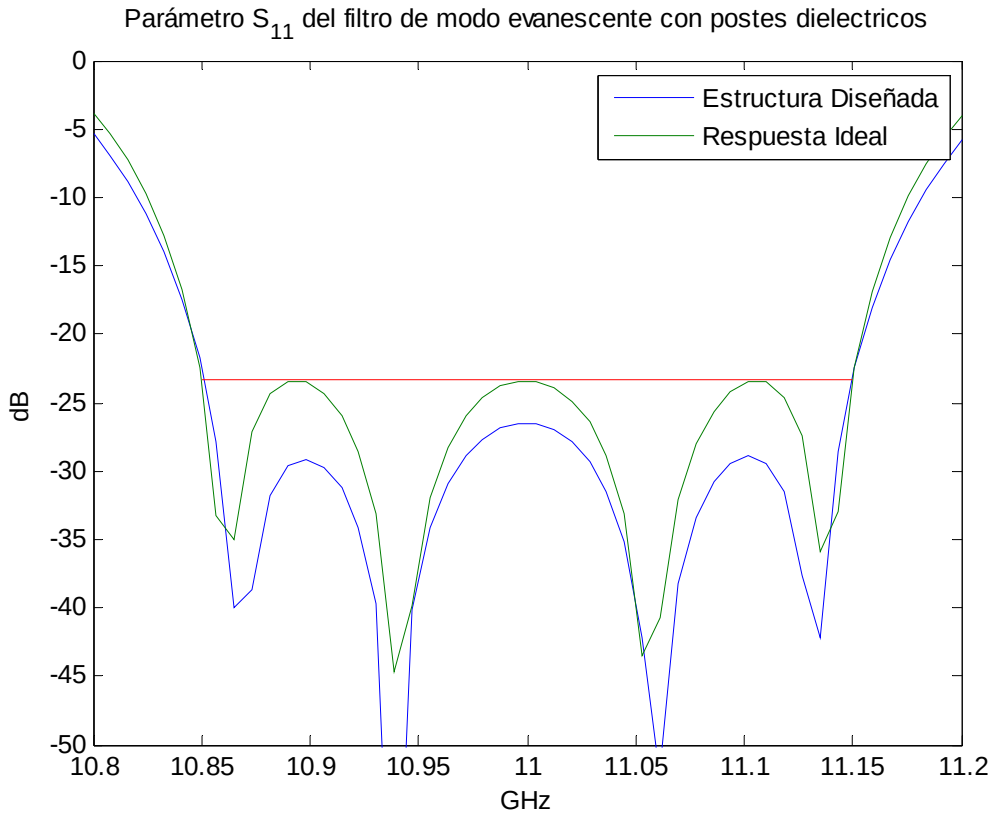
La curva de la estructura diseñada (la azul) esta por debajo de la curva de la respuesta ideal (verde) por lo que el comportamiento es correcto. Ambas están por debajo de la recta de las pérdidas de retorno (rojo).



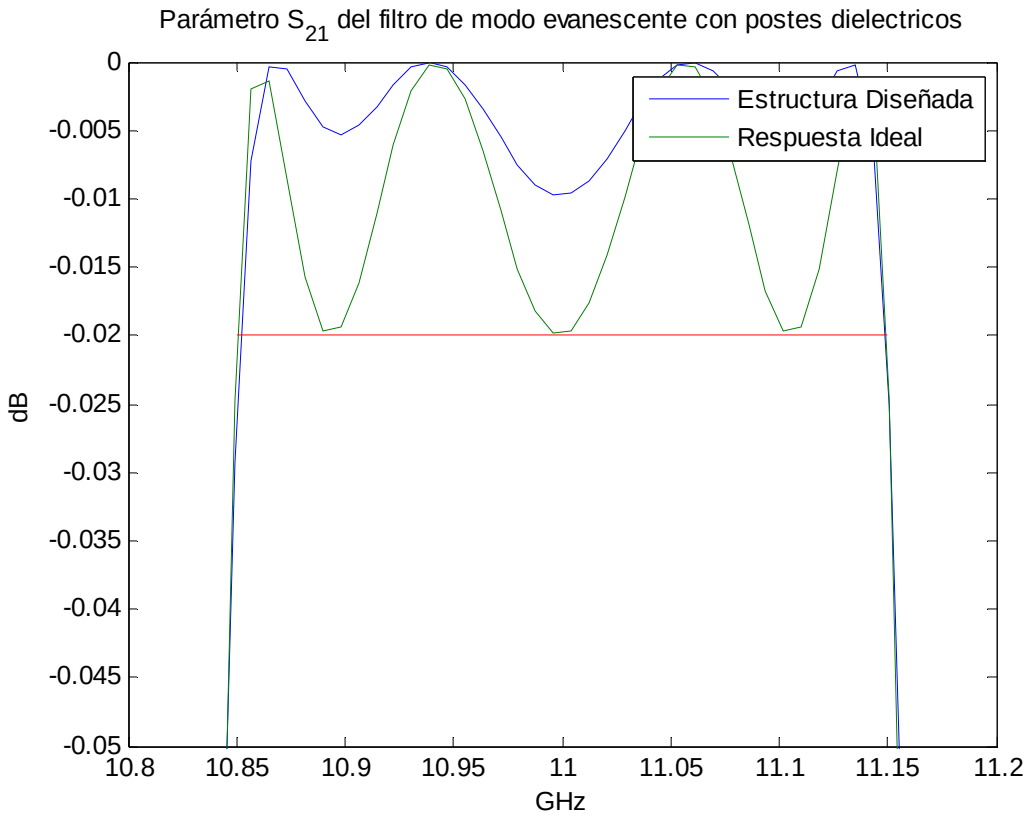
La curva de la estructura diseñada (la azul) esta por encima de la curva de la respuesta ideal (la verde) por lo que el comportamiento es correcto. Ambas curvas están por encima de la recta de las pérdidas de inserción (la roja).

- Método Levenberg-Marquardt:

xfin	[2.4175, 0.7775, 10.1824, 2.1698, 10.9924]
kfin	[0.2832, 0.2832]
F(x)	0.1603
nEval	54
nIter	23



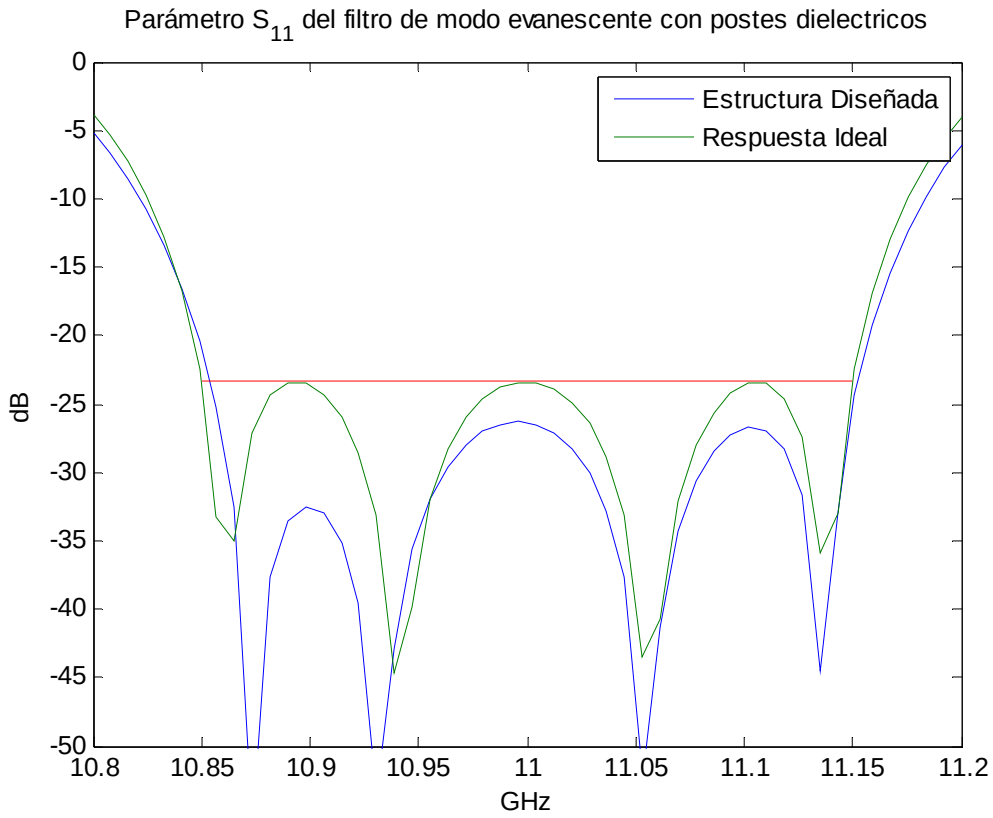
Tanto la estructura diseñada (azul) como la respuesta ideal (verde) están por debajo de la recta de las pérdidas de retorno (rojo). El comportamiento de la estructura diseñada esta por debajo del comportamiento ideal del dispositivo.



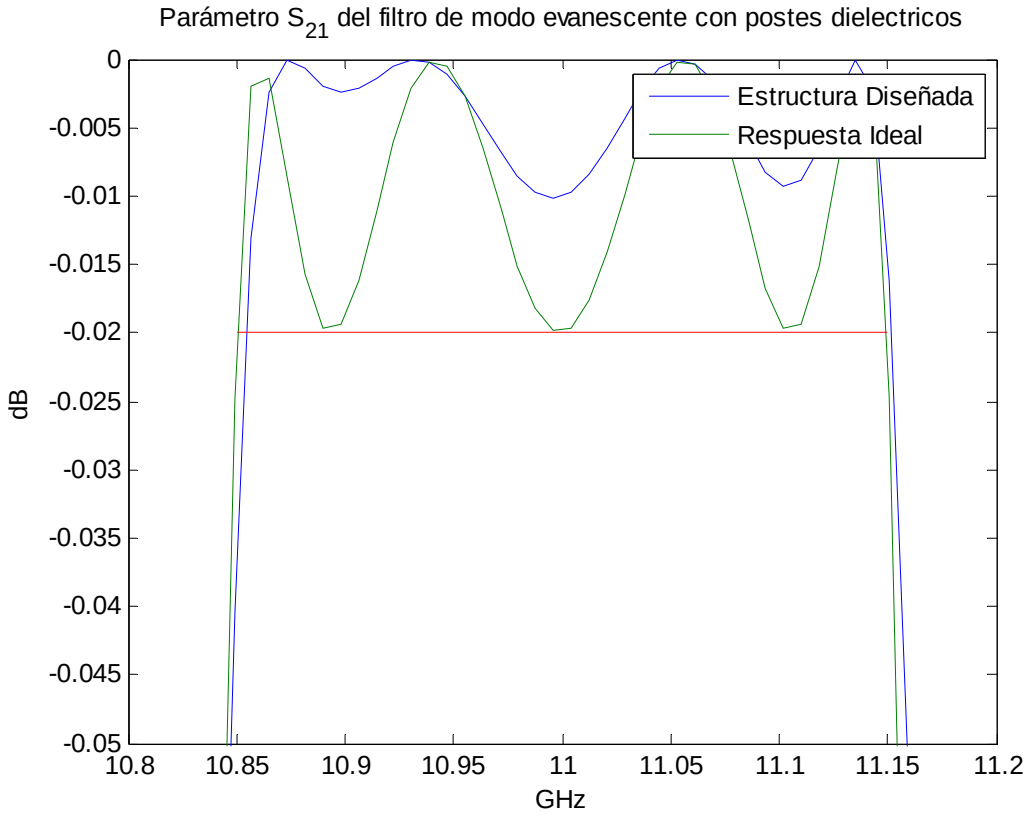
La curva del comportamiento real del dispositivo (curva azul) está por encima de la curva del comportamiento ideal de éste (curva verde). Ambas curvas están por encima de la recta de las pérdidas de inserción (recta roja).

- Método Quasi-Newton según Broyden-Fletcher-Goldfarb-Shanno:

xfin	[2.4179, 0.7777, 10.1820, 2.1696, 10.9919]
kfin	[0.2970, 0.2970]
F(x)	0.1764
nEval	3
nIter	1



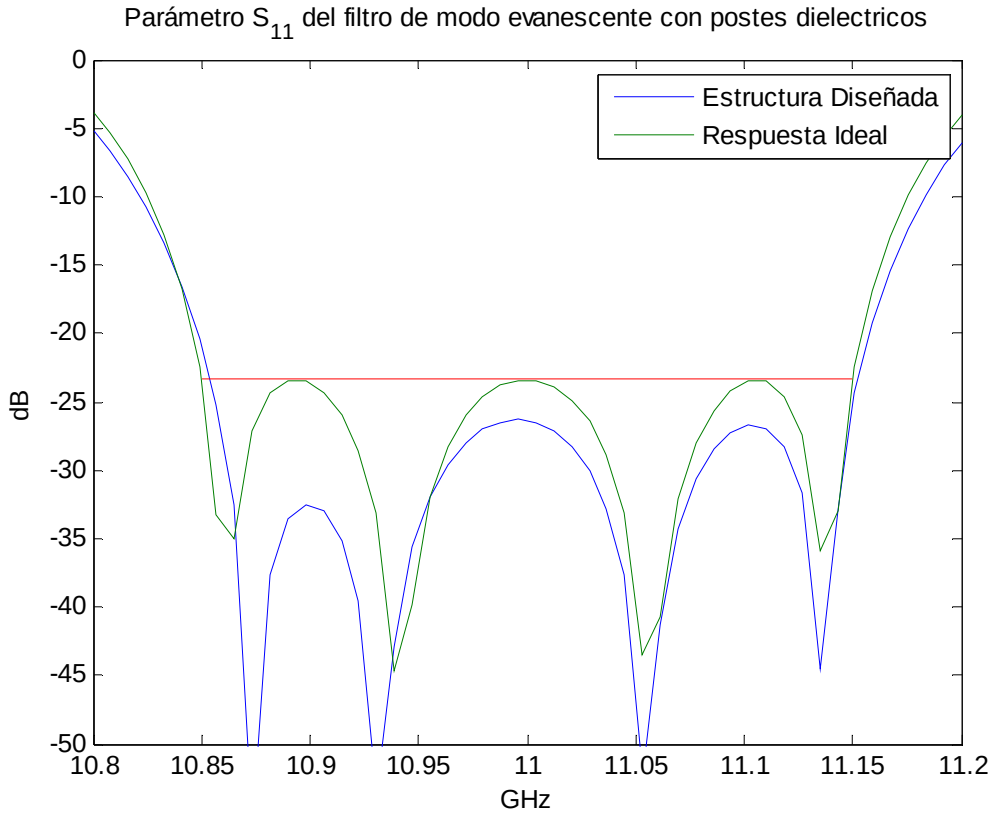
La curva del comportamiento real del dispositivo (la azul) esta por debajo de la curva de la respuesta ideal de éste (verde) por lo que el comportamiento es correcto. Ambas están por debajo de la recta de las pérdidas de retorno (rojo).



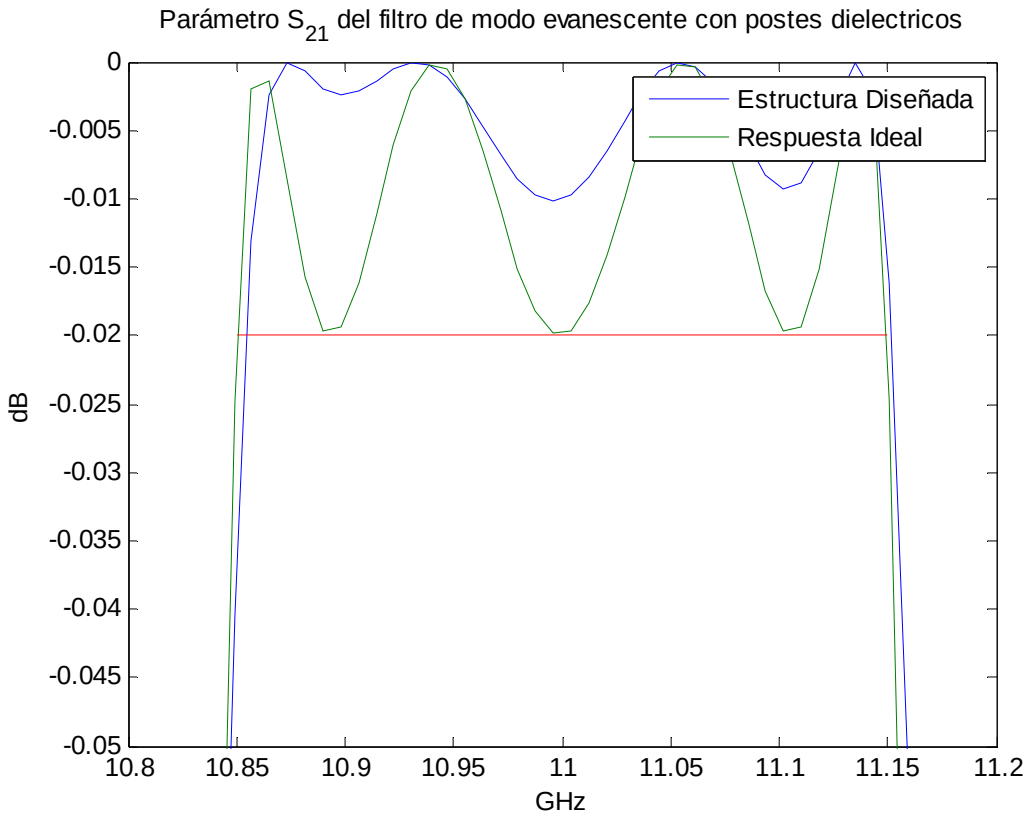
Tanto la estructura diseñada (azul) como la respuesta ideal (verde) están por encima de la recta de las pérdidas de inserción (rojo). El comportamiento de la estructura diseñada esta por encima del comportamiento ideal del dispositivo.

- Método Quasi-Newton según Al-Baali y Fletcher:

xfin	[2.4179, 0.7777, 10.1820, 2.1696, 10.9919]
kfin	[0.2970, 0.2970]
F(x)	0.1764
nEval	3
nIter	1



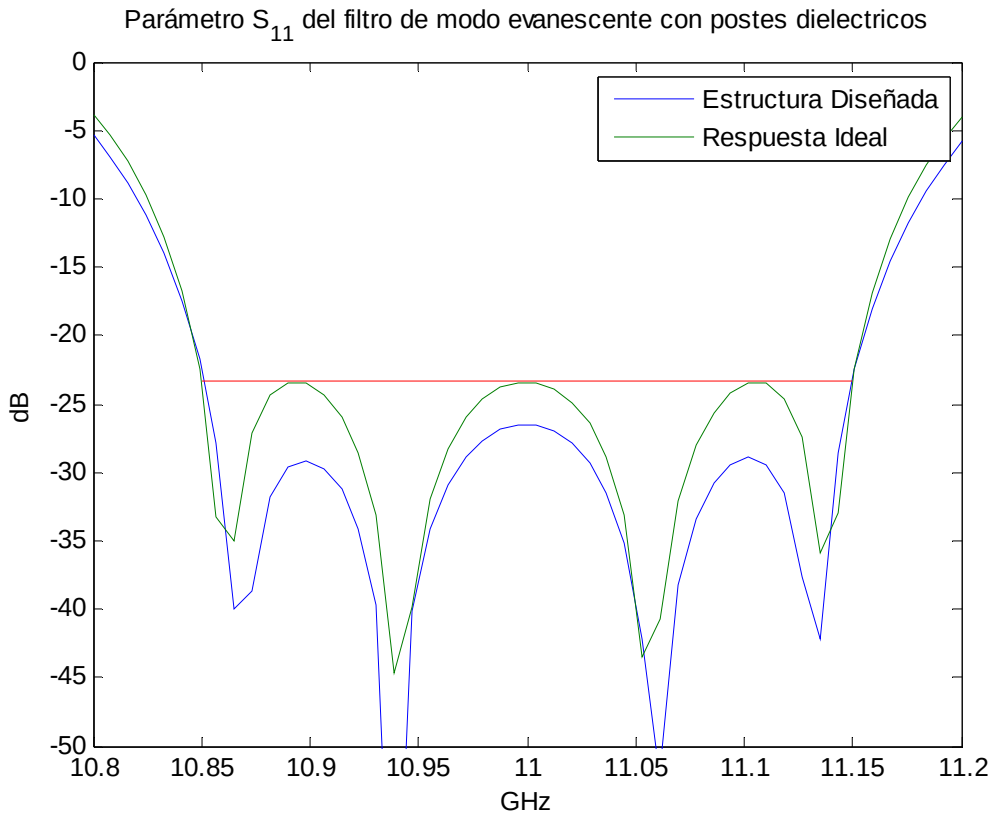
El comportamiento del dispositivo es correcto ya que la estructura diseñada tiene la curva por debajo de la de la estructura ideal y ambas están por debajo de las pérdidas de retorno.



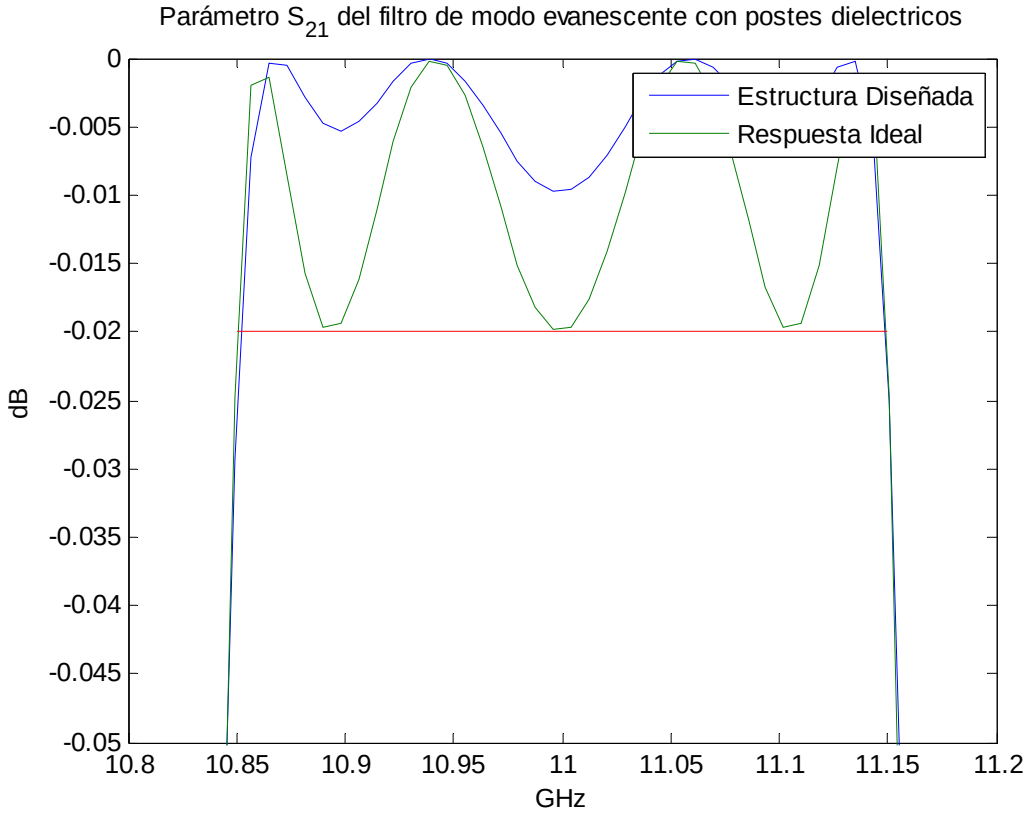
Tanto el comportamiento real del dispositivo (curva azul) como el comportamiento ideal de éste (curva verde) están por encima de la recta de las pérdidas de inserción (recta roja). El comportamiento real del dispositivo esta por encima del comportamiento ideal.

- Método híbrido:

Xfin	[2.4175, 0.7775, 10.1824, 2.1698, 10.9924]
Kfin	[0.2832, 0.2832]
F(x)	0.1603
nEval	54
nIter	23



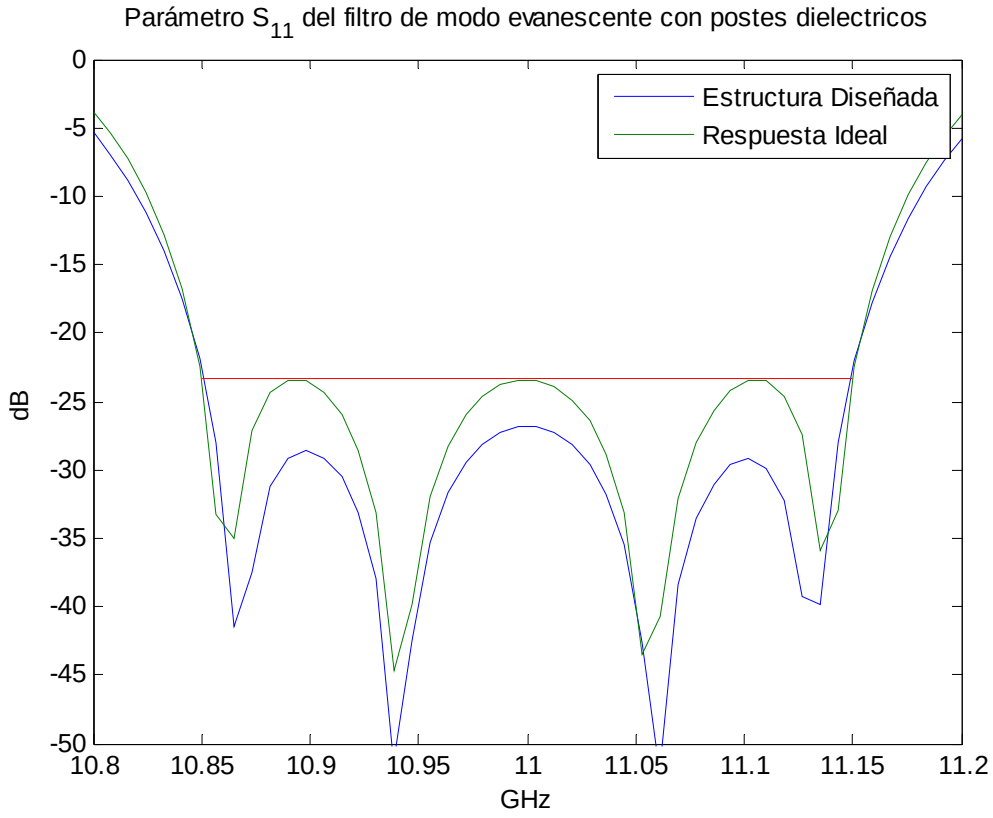
Tanto el comportamiento real del dispositivo (curva azul) como el comportamiento ideal de éste (curva verde) están por debajo de la recta de las pérdidas de retorno (curva roja). El comportamiento real del dispositivo esta por debajo del comportamiento ideal.



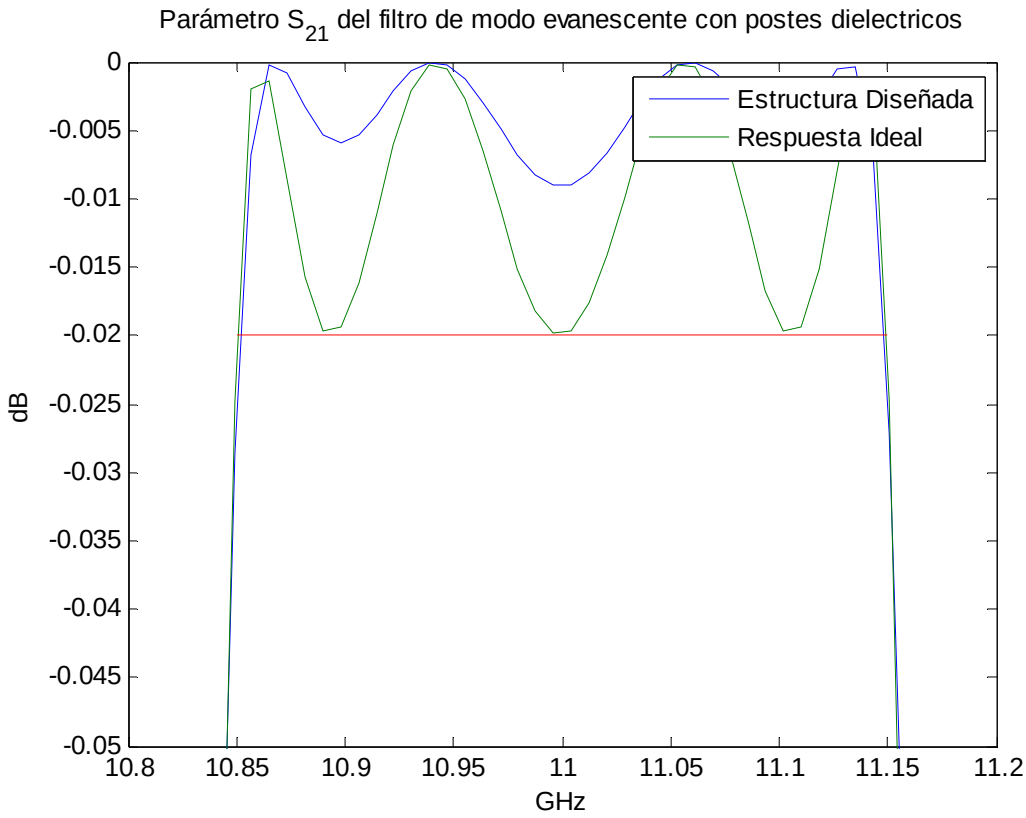
La curva de la estructura diseñada (la azul) esta por encima de la curva de la respuesta ideal (la verde) por lo que el comportamiento es correcto. Ambas curvas están por encima de la recta de las pérdidas de inserción (la roja).

- Método Gauss-Newton de MATLAB:

xfin	[2.4160, 0.7775, 10.1844, 2.1698, 10.9938]
kfin	[0.2777, 0.2777]
F(x)	0.0771
nEval	503
nIter	56



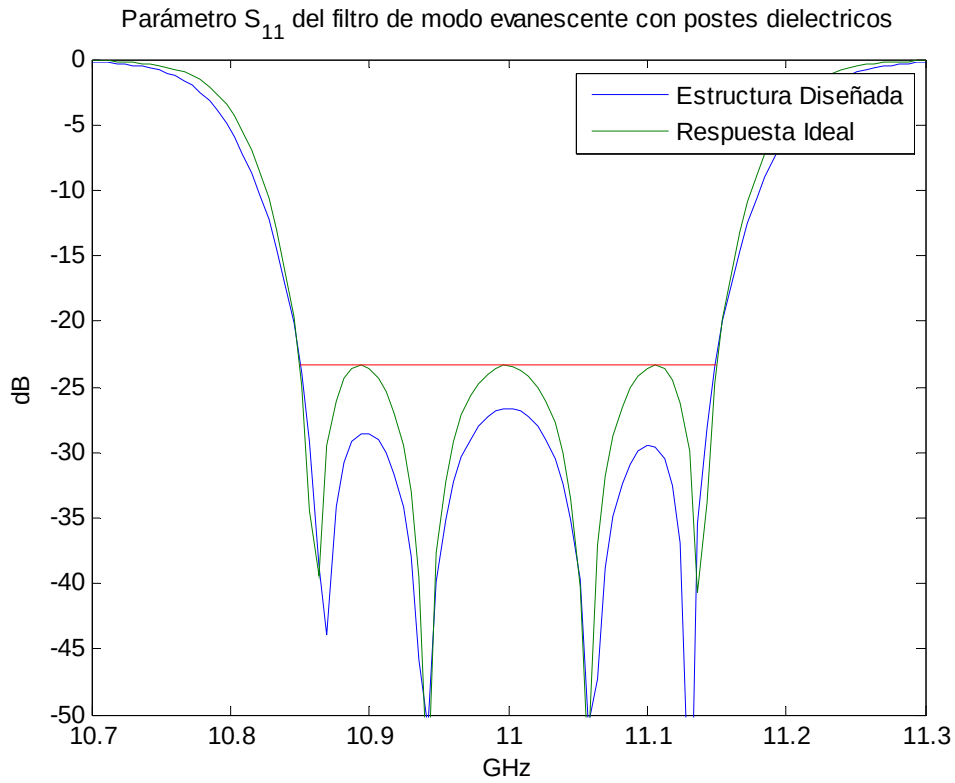
Tanto la estructura diseñada (azul) como la respuesta ideal (verde) están por debajo de la recta de las pérdidas de retorno (rojo). El comportamiento de la estructura diseñada esta por debajo del comportamiento ideal del dispositivo.



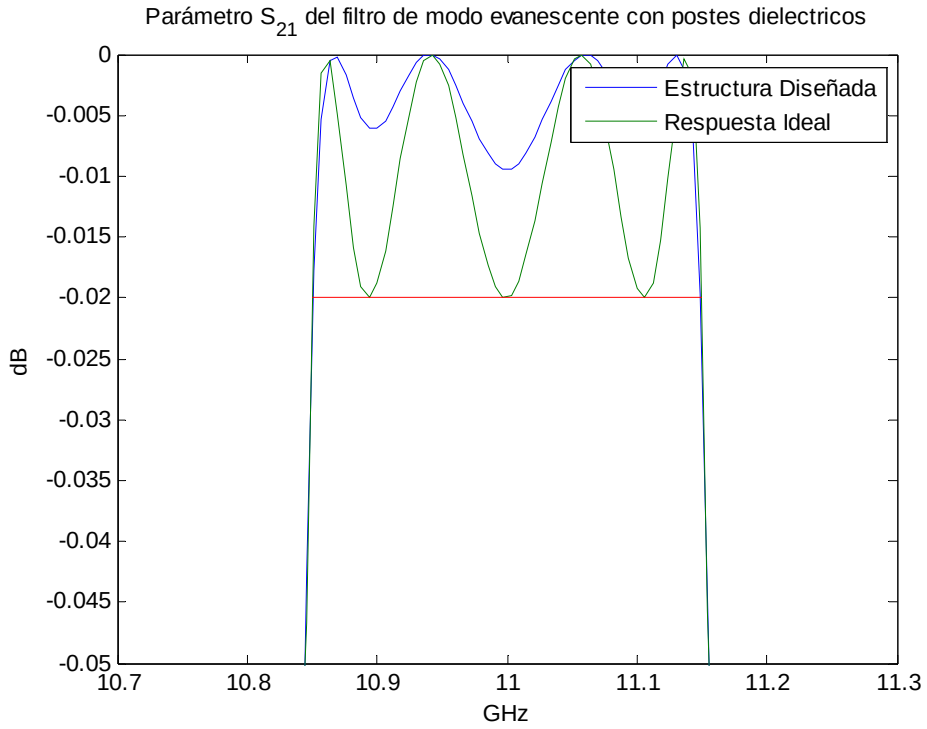
La curva del comportamiento real del dispositivo (curva azul) está por encima de la curva del comportamiento ideal de éste (curva verde). Ambas curvas están por encima de la recta de las pérdidas de inserción (recta roja).

- Método Levenberg-Marquardt de MATLAB:

xfin	[2.4171, 0.7775, 10.1831, 2.1699, 10.9928]
kfin	0.4714
F(x)	0.2222
nEval	26
nIter	183



La curva del comportamiento real del dispositivo (la azul) esta por debajo de la curva de la respuesta ideal de éste (verde) por lo que el comportamiento es correcto. Ambas están por debajo de la recta de las pérdidas de retorno (rojo).



Tanto el comportamiento real del dispositivo (curva azul) como el comportamiento ideal de éste (curva verde) están por encima de la recta de las pérdidas de inserción (recta roja). El comportamiento real del dispositivo esta por encima del comportamiento ideal.

CAPÍTULO 5

Conclusiones

En este capítulo se exponen las principales conclusiones a las que se llega tras el análisis de los resultados obtenidos por los programas realizados para los métodos de mínimos cuadrados y por los resultados obtenidos en el diseño de filtros evanescentes con postes dieléctricos.

Estas conclusiones son las siguientes:

1- En primer lugar, se ve que los cinco métodos evaluados de *Mínimos Cuadrados*: *Método de Gauss-Newton, Método de Levenberg-Marquardt, Método Quasi-Newton* (con las dos actualizaciones de la Hessiana) *y Método Híbrido* han convergido en el punto esperado de la función de Rosenbrock $(x = [1,1])$ con una precisión muy buena, llegando en algunos casos a ser prácticamente 0 el error, aunque el número de evaluaciones ha variado bastante, lo cual se ha verificado que se han implementado bien los métodos y que funcionan bien.

2- En cuanto a máxima precisión se refiere, el método con el que mejores resultados se ha obtenido es sin duda el de Gauss-Newton, con el cual el error más elevado que se ha obtenido es del orden de 10^{-23}, y se ha obtenido un número de evaluaciones menor en todos las pruebas realizadas. Puede considerarse un comportamiento excelente de este método.

3- Se ha comparado el funcionamiento del método Gauss-Newton con el de la función *lsqnonlin* del programa MATLAB, viéndose que el método programado necesita menos evaluaciones de la función objetivo y que tiene un error máximo menor que el de MATLAB.

4- Respecto a la comparación del método Levenberg-Marquardt programado con el obtenido con la función *lsqnonlin*, se ha obtenido un error máximo del orden 10^{-10} con el programa implementado y un error del orden 10^{-6} con el programa MATLAB, siendo el número de evaluaciones de la función menor para el implementado que para el de MATLAB, lo cual significa que el funcionamiento del programa implementado se puede considerar excelente.

5- Se ha comprobado que la actualización de la Hessiana según *Broyden-Fletcher-Goldfarb-Shanno* presenta mejor comportamiento que según *Al-Baali y Fletcher*, ya que aunque se obtiene un error máximo del orden de 10^{-20} frente al 10^{-19} que se obtiene en *AF*, el número de evaluaciones varía bastante, según *BFGS* son 41523 frente a las 84115950 evaluaciones del método según *AF*. Por lo que se ha determinado que la actualización según *BFGS*, es mucho más rápida y robusta que la actualización *AF*. Por lo que ésta se ha utilizado en el método Híbrido.

6- El método híbrido ha conseguido mejores resultados en cuanto a error se refiere consiguiendo un error máximo de 10^{-17}, que es menor que el obtenido con el método Levenberg-Marquardt, y un número menor de evaluaciones comparado con el método Quasi-Newton. Por tanto se ha conseguido lo que se esperaba con este método.

7- Por último, se han aplicado las cuatro versiones del método *Mínimos Cuadrados* al diseño de filtros de microondas, en concreto en el proceso de diseño de un filtro evanescente cargado con dos resonadores dieléctricos, considerando como objetivo: conseguir una diferencia mínima entre la respuesta real e ideal, intentando obtener la mayor banda de rechazo posible. Como se aprecia en las pruebas realizadas se ha llegado al objetivo deseado.

8- En base a las conclusiones anteriores, se observa que se ha conseguido el objetivo principal de este trabajo, ya que se ha introducido la optimización en el proceso de diseño de dispositivos pasivos de microondas, concretamente filtros evanescentes, desarrollando para ello, un algoritmo que implementa el método de mínimos cuadrados con el programa MATLAB.

ANEXO

Código MATLAB de los programas

En este apéndice se muestran los métodos implementados, indicando como llamarlo y sus parámetros de entrada y salida.

- Método Gaus-Newton: función que resuelve el problema de mínimos cuadrados con el método de Gauss-Newton.

```
function [xfin,kfin,ffin,nEval,nIter,stop]=metodoGN (k,x0,xtol,ftol,maxIter,maxEval)
    stop=0;
    m=length(x0);
    n=length(k);
    Eval=0;
    nIter=0;
    h=1e-9;
    J=Jacobiana(k,x0,h);
    k0=zeros(1,n);
    kfin=zeros(1,n);
    for i=1:n
        k0(i)=feval(char(k(i)),x0);
    end %for, calculo k0
    f0=k0*k0';
    d=pasoGN(k0,J);
    while ~stop
        nIter=nIter+1;
        if any(isinf(d(:))) %Se busca si hay algún infinito en el vector d
            stop=-5; %Si hay un infinito hay desbordamiento (-5)
        else
            Evals=maxEval-Eval;
            [xfin,stop,numEval]=LineSearch(k,x0,k0,d,h,Evals);
            for i=1:n
                kfin(i)=feval(char(k(i)),xfin);
            end %for, calculo kfin
            ffin=kfin*kfin';
            Eval=Eval+numEval+1;
            if any(isinf(kfin(:)))||isinf(ffin)
                stop=-5;
            elseif abs(xfin-x0)<=xtol
                stop=2; %se ha logrado un incremento muy pequeño en x
            elseif abs(ffin-f0)<=ftol
                stop=1; %se ha logrado un incremento muy pequeño
                        en el valor de la función
            elseif nIter>maxIter
                stop=3; %se ha sobrepasado el número máximo de iteraciones
            end %elseif
```

```
            x0=xfin;
            k0=kfin;
            f0=ffin;
        end %if-else
        if ~stop
            J=Jacobiana(k,x0,h);
            d=pasoGN(k0,J);
            Eval=Eval+m;
        end %if
    end %while
    nEval=Eval;
    nIter=nIter+1;
end
```

- Dirección método Gauss-Newton: esta función calcula la dirección de avance con el método Gauss-Newton, la cual se utilizará como entrada de la función LineSearch.

```
function d=pasoGN(k,J)
    d=-(((J'*J)^(-1))*J'*k')'; %Devuelve un vector fila
end
```

- Método Levenberg-Marquardt: función que resuelve el problema de mínimos cuadrados con el método de Levenberg-Marquardt.

```
function [xfin,kfin,ffin,nEval,nIter,stop]=
metodoLM(k,x0,xtol,ftol,maxIter,maxEval)
    stop=0;
    n=length(k);
    Eval=0;
    nIter=0;
    h=1e-9;
    k0=zeros(1,n);
    kfin=zeros(1,n);
    for i=1:n
        k0(i)=feval(char(k(i)),x0);
    end %for, calculo k0
    f0=k0*k0';
    J=Jacobiana(k,x0,h);
    grad=2*J'*k0'; %gradF0=2*(k0*J)
    normagrad=norm(grad,inf); %ngradF0=norm(gradF0,inf)
    H=J'*J; %aproximación de la matriz Hessiana que se hace un mínimo cuadrado
    if isinf(normagrad)||isinf(norm(A(:),inf))
        stop=-5;
    end %if
```

```
if stop~=0
    xfin=x0;
    kfin=NaN;
    ffin=NaN;
    nIter=0;
else
    tau=1e-3;
    lambda=tau*max(diag(H)); %Inicialización del valor de lambda
    d=pasoLM(k0,J,lambda);
    nu=2;
    while ~stop
        nIter=nIter+1;
        if any(isinf(d(:))) %Se busca si hay algún infinito en el vector d
            stop=-5; %Si hay un infinito hay desbordamiento (-5)
        else
            Evals=maxEval-Eval;
            [xfin,stop,numEval]=LineSearch(k,x0,k0,d,h,Evals/10);
            for i=1:n
                kfin(i)=feval(char(k(i)),xfin);
            end %for, calculo kfin
            ffin=kfin*kfin';
            Eval=Eval+numEval+1;
            if any(isinf(kfin(:)))||isinf(ffin)
                stop=-5;
            elseif abs(xfin-x0)<=xtol
                stop=2; %se ha logrado un incremento muy pequeño en  x
            elseif abs(ffin-f0)<=ftol
                stop=1; %se ha logrado un incremento muy pequeño
                        en el valor de la función
            elseif nIter>maxIter
                stop=3; %se ha sobrepasado el número máximo de iteraciones
            end %elseif
            xfin=x0+d;
            for i=1:n
                kfin(i)=feval(char(k(i)),xfin);
            end %for, calculo kfin
            df=f0-ffin;
            dL=(1/2)*d*((lambda*d')-grad);
            rho=df/dL;
            x0=xfin;
            k0=kfin;
            f0=ffin;
        end %if-else
        if (~stop)&&(rho>0)
            J=Jacobiana(k,xfin,h);
            H=J'*J;
            grad=2*J'*k0';
            normagrad=norm(grad,inf);
            lambda=lambda*max((1/3),1-(2*rho-1)^3);
            nu=2;
```

```
                if isinf(normagrad)||isinf(norm(H(:),inf))
                    stop = -5;
                end %if
            else
                lambda=lambda*nu;
                nu=2*nu;
            end %if-else
            d=pasoLM(k0,J,lambda);
        end %while
        nEval=Eval;
        nIter=nIter+1;
    end %if-else
end
```

- Dirección método Levenberg-Marquardt: esta función calcula la dirección de avance con el método Levenberg-Marquardt, la cual se utilizará como entrada de la función LineSearch.

```
function d=pasoLM(k,J,lambda)
    n=length(J);
    I=eye(n);
    d=(-(((J'*J)+(lambda*I))^(-1))*(J'*k'))'; %Devuelve vector fila
end
```

- Método Quasi-Newton según Broyden-Fletcher-Goldfarb-Shanno: esta función resuelve el problema de mínimos cuadrados con el método Quasi-Newton con la actualización de la Hessiana según Broyden-Fletcher-Goldfarb-Shanno.

```
function [xfin,kfin,ffin,nEval,nIter,stop]=
metodoQN(k,x0,xtol,ftol,maxIter,maxEval)
    stop=0;
    m=length(x0);
    n=length(k);
    Eval=0;
    nIter=0;
    h=1e-9;
    B=eye(m);
    J=Jacobiana(k,x0,h);
    k0=zeros(1,n);
    kfin=zeros(1,n);
    for i=1:n
        k0(i)=feval(char(k(i)),x0);
    end %for, calculo k0
    f0=k0*k0';
    grad=2*J'*k0';
    d=pasoQN(B,grad);
```

```
        while ~stop
            nIter=nIter+1;
            if any(isinf(d(:))) %Se busca si hay algún infinito en el vector d
                stop=-5; %Si hay un infinito hay desbordamiento (-5)
            else
                Evals=maxEval-Eval;
                [xfin,stop,numEval]=LineSearch(k,x0,k0,d,h,Evals);
                for i=1:n
                    kfin(i)=feval(char(k(i)),xfin);
                end %for, calculo kfin
                ffin=kfin*kfin';
                Eval=Eval+numEval+1;
                if any(isinf(kfin(:)))||isinf(ffin)
                    stop=-5;
                elseif abs(xfin-x0)<=xtol
                    stop=2; %se ha logrado un incremento muy pequeño en  x
                elseif abs(ffin-f0)<=ftol
                    stop=1; %se ha logrado un incremento muy pequeño
                              en el valor de la función
                elseif nIter>maxIter
                    stop=3; %se ha sobrepasado el número máximo de iteraciones
                end %elseif
                s=xfin-x0;
                Jfin=Jacobiana(k,xfin,h);
                gradfin=2*Jfin'*kfin';
                q=gradfin-grad;
                x0=xfin;
                k0=kfin;
                f0=ffin;
                grad=gradfin;
            end %if-else
            if ~stop
                s=s';
                A=(q*q')/(q'*s);
                C=-(B*s*s'*B)/(s'*B*s);
                B=B+A+C;
                d=pasoQN(B^(-1),grad);
                Eval=Eval+m;
            end %if
        end %while
    nEval=Eval;
end
```

- Método Quasi-Newton según Al-Baali y Fletcher: esta función resuelve el problema de mínimos cuadrados con el método Quasi-Newton con la actualización de la Hessiana según Al-Baali y Fletcher.

```
function [xfin,kfin,ffin,nEval,nIter,stop]=
metodoQN_ingles(k,x0,xtol,ftol,maxIter,maxEval)
    stop=0;
    m=length(x0);
    n=length(k);
    Eval=0;
    nIter=0;
    h=1e-9;
    B=eye(m);
    J=Jacobiana(k,x0,h);
    k0=zeros(1,n);
    kfin=zeros(1,n);
        for i=1:n
            k0(i)=feval(char(k(i)),x0);
        end %for, calculo k0
        f0=k0*k0';
        grad=2*J'*k0';
        d=pasoQN(B,grad);
        while ~stop
            nIter=nIter+1;
            if any(isinf(d(:))) %Se busca si hay algún infinito en el vector d
                stop=-5; %Si hay un infinito hay desbordamiento (-5)
            else
                Evals=maxEval-Eval;
                [xfin,stop,numEval]=LineSearch(k,x0,k0,d,h,Evals);
                for i=1:n
                    kfin(i)=feval(char(k(i)),xfin);
                end %for, calculo kfin
                ffin=kfin*kfin';
                Eval=Eval+numEval+1;
                if any(isinf(kfin(:)))||isinf(ffin)
                    stop=-5;
                elseif abs(xfin-x0)<=xtol
                    stop=2; %se ha logrado un incremento muy pequeño en  x
                elseif abs(ffin-f0)<=ftol
                    stop=1; %se ha logrado un incremento muy pequeño
                            en el valor de la función
                elseif nIter>maxIter
                    stop=3; %se ha sobrepasado el número máximo de iteraciones
                end %elseif
                s=(xfin-x0)';
                Jfin=Jacobiana(k,xfin,h);
                y=Jfin'*Jfin*s+(Jfin-J)'*kfin';
                v=B*s;
                gradfin=2*Jfin'*kfin';
                x0=xfin;
                k0=kfin;
```

```
                f0=ffin;
                grad=gradfin;
            end %if-else
            if ~stop
                if (s'*y>0)
                    A=((1/(s'*y))*y)*y';
                    C=-(((1/(s'*v))*v)*v');
                    B=B+A+C;
                end
                d=pasoQN(B^(-1),grad);
                Eval=Eval+m;
            end %if
        end %while
        nEval=Eval;
    end
```

- Dirección método Quasi-Newton: esta función calcula la dirección de avance del método Quasi-Newton, independientemente la actualización de la Hessiana utilizada:

```
function d=pasoQN(B,grad)
    d=(-B*grad)';
end
```

- Método Híbrido: esta función resuelve el problema de mínimos cuadrados con la hibridación de los método Levenberg-Marquardt y Quasi-Newton según Broyden-Fletcher-Goldfarb-Shanno:

```
function [xfin,kfin,ffin,nEval,nIter,stop]=
metodoHibrido(k,x0,xtol,ftol,maxIter,maxEval)
    contador=1;
    stop=0;
    n=length(k);
    Eval=0;
    nIter=0;
    k0=zeros(1,n);
    kfin=zeros(1,n);
    for i=1:n
        k0(i)=feval(char(k(i)),x0);
    end %for, calcul k0
    f0=k0*k0';
    while ~stop
        if contador<=3
            [xfin,kfin,ffin,nEval,nIter,stop]=metodoLM(k,x0,xtol,ftol,maxIter,maxEval);
        else
            [xfin,kfin,ffin,nEval,nIter,stop]=metodoQN(k,x0,xtol,ftol,maxIter,maxEval);
        end %if-else
        contador=contador+1;
    end %while
end
```

- LineSearch: función que implementa el algoritmo de búsqueda lineal utilizada para resolver problemas de mínimos cuadrados.

```
function [x,stop,Eval]=LineSearch(k,x0,k0,d,h,maxEval)
    stop=0;
    [alfa,encontrado,Eval]=calculoalfa(k,x0,k0,d,h,maxEval);
    x=x0+(alfa*d);
    if ~encontrado
        x=x0;
        stop=3;
    end
end
```

- Calculo de α: el paso α es necesario para hallar el mínimo a lo largo de la dirección, se calcula mediante una combinación de interpolación cuadrática y cúbica.

```
function [alfa,encontrado,Eval]=calculoalfa(k,x0,k0,d,h,maxEval)
    J0=Jacobiana(k,x0,h);
    g0=k0*k0';
    gprima0=2*k0*J0*d';
    alfa1=1;
    alfa=alfa1;
    x1=x0+(alfa1*d)
    n=length(k);
    k1=zeros(1,n);
    k2=zeros(1,n);
    for i=1:n
        k1(i)=feval(char(k(i)),x1);
    end %for, calculo k1
    g1=k1*k1'; %para α1→g(α1)=g(x0+d)=f(x0+ α1·d)
    Eval=1;
    encontrado=0;
    lambda=1e-3; % valor por defecto de lambda
    if g1<=g0+(lambda*gprima0*alfa1)
        alfa=alfa1;
        encontrado=1;
    else
        alfa2=intercuadratica(g0,gprima0,alfa1,g1);
        x2=x0+(alfa2*d);
        for i=1:n
            k2(i)=feval(char(k(i)),x2);
        end %for, calculo k2
        g2=k2*k2';
        Eval=Eval+1;
        encontrado=0;
```

```
        while ~encontrado&& Eval<maxEval
            if g2<=g0+(lambda*gprima0*alfa2)
                alfa=alfa2;
                encontrado=1;
            else
                alfaprima=intercubica(g0,gprima0,alfa1,g1,alfa2,g2);
                alfa1=alfa2;
                g1=g2;
                alfa2=alfaprima;
                x2=x0+(alfa2*d);  %Se recalcula g2 con la nueva alfa2
                for i=1:n
                    k2(i)=feval(char(k(i)),x2);
                end %for, calculo k2
                g2=k2*k2';
                Eval=Eval+1;
            end %if-else
        end %while
    end %if-else
end
```

- Interpolación cuadrática:

```
function alfa2=intercuadratica (g0,gprima0,alfa1,g1)
    b=gprima0;
    c=g0;
    a=(g1-(b*alfa1-c))/(alfa1.^2);
    alfa2=-b/(2*a);
end
```

- Interpolación cúbica:

```
function alfaprima=intercubica(g0,gprima0,alfa1,g1,alfa2,g2)
    c=gprima0;
    factor1=1/(alfa1-alfa2);
    factor2=[1/(alfa1^2) -1/(alfa2^2);-alfa2/alfa1^2 alfa1/alfa2^2];
    factor3=[g1-(gprima0*alfa1)-gprima0;g2-(gprima0*alfa2)-g0];
    ab=factor1*factor2*factor3;
    a=ab(1);
    b=ab(2);
    alfaprima=(-b+(sqrt(b^2-(3*a*c))))/(3*a);
end
```

- Jacobiana: esta función realiza la aproximación numérica del Jacobiano de una función en un punto.

```
function J=Jacobiana(k,x0,h)
    n=length(k);
    m=length(x0);
    J=zeros(n,m);
    for i=1:n
        J(i,:)=gradiente(char(k(i)),x0,h);
    end
end
```

- Gradiente: esta función calcula el valor aproximado del gradiente de una función evaluada en un punto.

```
function grad=gradiente (f,x0,h)
    n=length (x0);
    f0=feval(f,x0);
    grad=zeros(1,n);
    aux=h*eye(n);
    for i=1:n
        xaux=x0+aux(i,:);
        faux=feval(f,xaux);
        grad(1,i)=(faux-f0)/h;
    end
end
```

- Coeficientes función Rosenbrock: función utilizada en cada método, cuyo mínimo es conocido:
  - Primer coeficiente:

```
function f=f1(x)
    f=10.*(x(2)-(x(1).^2));
end
```

  - Segundo coeficiente:

```
function f=f2(x)
    f=1-x(1);
end
```

Bibliografía

[1] Roger Fletcher. Practical Methods of Optimization. John Wiley & Sons, 2001.

[2] John W. Bandler. Optimization methods for compute-aided design. *IEEE Trans. Microwave Theory Tech*, 17:533-552, 1969

[3] Antonio Marco Aguilar. Optimización multiobjetivo aplicada al diseño de dispositivos pasivos de comunicaciones de alta frecuencia. Escuela Técnica Superior de Ingenieros de Telecomunicación. Universidad Politécnica de Valencia, 2007

[4] Pedro Gómez Martínez. Optimización de funciones multidimensionales sujetas a restricciones. Aplicación al diseño de filtros de microondas. Escuela Técnica Superior de Ingenieros de Telecomunicación. Universidad Politécnica de Valencia, 2005.

[5] Héctor Esteban. *Ingenería Avanzada de microondas*. Curso de doctorado, programa de comunicaciones, Universidad Politécnica de Valencia, 2003-2004.

[6] *Optimization Toolbox User's Guide*. The MathWorks, Inc. Version 3, September 2005.

[7] Darren Redfern and Colin Campbell, *The MATLAB 5 Handbook*. Springer, 1997.

[8] k. Madsen, H.B. Nielsen, O. Tingleff, *Methods for non-linear least squares problems*, 2nd Edition. IMM DTU (2004)

[9] C. Bachiller, H. Esteban, V.E. Boria, J.V. Morro, L. J. Roglá, M. Taroncher and A. Belenguer. Efficient CAD tool of Direct-Coupled-Cavities Filters with Dielectric Resonators. *IEEE*, 2005.

[10] C. Bachiller, H. Esteban, V.E. Boria, J.V. Morro, M. Taroncher and B. Gimeno. CAD of Evanescent Mode Waveguide Filters with Circular Dielectric Resonators. *IEEE*, pp. 1567-1570, 2006.